MicroEMACS 3.12 for MS-Windows
Index

Introduction

MicroEMACS is a tool for creating and changing documents, programs, and other text files. It is both relatively easy for the novice to use, but also very powerful in the hands of an expert. MicroEMACS can be extensively customized for the needs of the individual user.
MicroEMACS allows several files to be edited at the same time. The display can be split into different windows and screens, and text may be moved freely from one window on any screen to the next. Depending on the type of file being edited, MicroEMACS can change how it behaves to make editing simple. Editing standard text files, program files and word processing documents are all possible at the same time.

There are extensive capabilities to make word processing and editing easier. These include commands for string searching and replacing, paragraph reformatting and deleting, automatic word wrapping, word move and deletes, easy case controlling, and automatic word counts.

For complex and repetitive editing tasks editing macros can be written. These macros allow the user a great degree of flexibility in determining how MicroEMACS behaves. Also, any and all the commands can be used by any keystroke by changing, or rebinding, what commands various keys invoke.

Special features are also available to perform a diverse set of operations such as file encryption, automatic backup file generation, entabbing and detabbing lines, executing operating system commands and filtering of text through other programs (like SORT to allow sorting text).

History

EMACS was originally a text editor written by Richard Stallman at MIT in the early 1970s for Digital Equipment computers. Various versions, rewrites and clones have made an appearance since.

This version of MicroEMACS is derived from code written by David G. Conroy in 1985. Later modifications were performed by Steve Wilhite and George Jones. In December of 1985 Daniel Lawrence picked up the then current source (version 2.0) and made extensive modifications and additions to it over the course of the next six years.

In November 1990, Pierre Perret produced a port of MicroEMACS 3.10e to the Microsoft Windows 3.0 environment (BETA version 0.6a which never enjoyed a full release). The first public version, 1.0, based on MicroEMACS 3.11c, was released in April 1992.

Update 1.1 included bug fixes, port to Windows NT, support of scroll bars and drag and drop mechanism. It was the first release to include a complete help file.

The new version 3.12 of MicroEMACS incorporates the port of the editor to MS-Windows, adding many new features, among which the capabilities to highlight regions of text and handle mouse movements in the macro language.

Support

Updates and support for the current version are available. Commercial support and usage and resale licences are also available.

For questions regarding the official MicroEMACS editor, contact Daniel Lawrence. For technical questions specific to the port of MicroEMACS to the Microsoft Windows environment, contact Pierre Perret.

The home BBS of MicroEMACS is "The Programmer's Room".

Bulletin Board System

The latest executables, sources and documentations can be obtained from:

The Programmer's Room
Opus 201/10
300/1200/2400 and 9600 bps (US Robotics HST)
(317) 742-5533 no parity 8 data bits no stop bits

The current MicroEMACS author can be contacted by writing to:

USMAIL: **Daniel Lawrence**
617 New York Street
Lafayette, IN 47901

Internet: mdbs!dan@dynamo.ecn.purdue.edu

The Programmer's Room BBS:
Daniel Lawrence

The author of the port of MicroEMACS to the Microsoft Windows environment can be contacted by writing to:

USMAIL: **Pierre Perret**
4326 W Michigan Ave
Glendale, AZ 85308

Internet: P.Perret@az05.bull.com

CompuServe: 73757,2337

The Programmer's Room BBS:
Pierre Perret

Copyright

This help file was authored by Pierre Perret.

Keyboard

All the MicroEMACS documentation talks about commands and the keystrokes needed to use them. Each MicroEMACS command has a name, and most of them are bound to a sequence of keys. Some of them are bound to mouse actions. The following commands are useful when looking for a binding:

M-?	apropos	looks up commands
	describe-bindings	lists all the bindings
^X?	describe-key	displays the command bound to a keystroke

In this help file, when a command is mentioned, its default key binding is often shown. Note that these bindings can be modified, in particular by the start-up file.

Keystrokes for commands include one of several prefixes, and a command character. Command keystrokes look like these:

^A	hold down Ctrl, press 'A'
M-A	press the meta key, release it and press 'A'
^XA	hold down Ctrl, press 'X', release, press 'A'
^X^A	hold down Ctrl, press 'X', release, hold Ctrl, press 'A'
A-C	hold down Alt, press 'C'
S-FN1	hold down Shift, press function key F1
FN^1	hold down Ctrl, press function key F1

Under Microsoft Windows, key bindings are displayed in front of menu items, using a CUA type syntax instead of the above-mentioned one. Though this may seem inconsistent, it looks more familiar to inexperienced users and is far less cryptic when it comes to special keys (Ins, Del, Arrows...).

Procedures

The Basics

MicroEMACS is a very powerful tool to use for editing text files. It has many commands, options and features to let you do just about anything you can imagine with text. But don't let this apparent complexity keep you from using it.... MicroEMACS can also be very simple.

To start editing text, all the keys you really need to know are the arrow keys. These keys let you move around in your file.

MicroEMACS also works by using control keys. Here are a few basic commands:

^P	Move upward
^B	Move backward
^F	Move forward
^N	Move downward
^X^S	Save your file
^X^C	Exit MicroEMACS

A hat sign "**^**" before a key means to hold the Ctrl key down and press the next character. For example, to exit MicroEMACS, hold down the Ctrl key and strike X and then C.

Getting at Files

The way you edit a file within MicroEMACS is by first reading it into a buffer, altering it and then saving it. Therefore, the most commonly used commands to access files are:

^X^F	find-file	to read a file from disk for editing
^X^S	save-file	to save an edited file to disk

Other read commands are:

^X^I	insert-file	to insert at the point
^X^R	read-file	to replace the whole buffer contents
^X^V	view-file	to read for viewing, preventing any alterations

To save a buffer to another file than the one MicroEMACS would normally access, use:

^X^W	write-file	to overwrite the file's previous contents
^X^A	append-file	to append to the end of the file

Searching and Replacing

Commands for searching for and replacing strings come in a number of different flavors. The simplest command is:

^S	search-forward

Following that, you can search for yet another occurrence of the same string by using:

A-S	hunt-forward

You can also search towards the beginning of the file instead of towards the end by using:

^R	search-reverse
A-R	hunt-backward

To replace strings, use:

M-R	replace-string	to replace all occurrences
M-^R	query-replace-string	to get queried for each replacement

MicroEMACS also supports incremental searching:

^XS	incremental-search	towards the beginning
^XR	reverse-incremental-search	towards the end

Cutting and Pasting

MicroEMACS allows you to manipulate text by blocks of any size. You can copy or move text within MicroEMACS through the kill buffer.

Under Microsoft Windows, you can also exchange text with other Windows applications via the clipboard, using the cut-region, clip-region and insert-clip commands.

Moving Text

To move text from one place to another:

1. Move to the beginning of the text you want to move.
2. Set the mark there with the set-mark (M-) command.
3. Move the point to the end of the text.
4. Use the kill-region (^W) command to delete the region you just defined. The text will be saved in the kill buffer.
5. Move the point to the place you want the text to appear.
6. Use the yank (^Y) command to copy the text from the kill buffer to the current point.

Repeat steps 5 and 6 to insert more copies of the same text.

Copying Text

To copy text from one place to another:

1. Move to the beginning of the text you want to copy.
2. Set the mark there with the set-mark (M-) command.
3. Move the point to the end of the text.
4. Use the copy-region (M-W) command to copy the region to the kill buffer.
5. Move the point to the place you want the text to appear.
6. Use the yank (^Y) command to copy the text from the kill buffer to the current point.

Repeat steps 5 and 6 to insert more copies of the same text.

Using the Mouse

MicroEMACS can use the mouse to perform many basic editing tasks. Unless mouse behavior has been altered by a macro, you can perform the following actions:

- Copying a Region
- Killing a Region
- Moving a Mode Line
- Pasting Text
- Repositioning the Point
- Scrolling Text Inside a Window

Repositioning the Point with the Mouse

Move the mouse to where you want the point to be, and click once with the left mouse button and release. The point will move there, making any screen or window active to do so.

Scrolling Text Inside a Window with the Mouse

Position the mouse on the text to drag, press the left button, move the mouse to where to display the text (horizontally or vertically), and release the mouse button.

If you are using the CUA.CMD page (which is usually the case under Microsoft Windows), the above action is performed by pressing the **right** mouse button instead of the left one.

Note that if you drag diagonally and the $diagflag variable is set to FALSE (the default value), the text will move only in the vertical direction.

Moving a Mode Line with the Mouse

Position the mouse on a mode line (except the bottom one which cannot be moved), press the left button, move to another position and release the button. The mode line will move, resizing the windows which are above and below.

If you are using the CUA.CMD page (which is usually the case under Microsoft Windows), the above action is performed by pressing the **right** mouse button instead of the left one.

Copying a Region with the Mouse

Position the mouse at the beginning of the text to be copied, press the right button, move the mouse to the other end of the text, release the button. This actually makes the selected text the current region and then copies it into the kill buffer.

If you are using the CUA.CMD page (which is usually the case under Microsoft Windows), the above action is performed by pressing the **Shift** key and the **right** mouse button together instead of just the right mouse button.

Killing a Region with the Mouse

After copying a region, without moving the mouse, click the right mouse button once. The text will be deleted, but it will still be kept in the kill buffer.

If you are using the CUA.CMD page (which is usually the case under Microsoft Windows), the above action is performed by pressing the **Shift** key and the **right** mouse button together instead of just the right mouse button.

Pasting Text with the Mouse

Move anywhere away from the place the mouse was last clicked, and click the right button once. The last text placed in the kill buffer will be inserted there.

If you are using the CUA.CMD page (which is usually the case under Microsoft Windows), the above action is performed by pressing the **Shift** key and the **right** mouse button together instead of just the right mouse button.

Using menus

Under Microsoft Windows, MicroEMACS sports an extensive menu system. Menu items can point to a pop-up menu or directly invoke a command or a macro. A few menu items are not linked to any MicroEMACS commands or macro (for instance, the "About..." item in the "Help" menu).

The text of each menu item can contain the following hints:

Items that lead to the apparition of a dialog box are followed by an ellipsis "**...**".

Items that require the user to type additional information in the message line are followed by a colon "**:**".

Items that require a numeric argument are preceded by an equal sign "**=**".

Items that are equivalent to a key binding have the corresponding key sequence displayed on the right side of the menu.

The MicroEMACS menus can be modified by macros to add/remove menus or menu items. The initial menus on the menu bar are:

File
Edit
Search
Execute
Miscellaneous
Screen
Help

File menu

This menu contains the following items:

Open...	invokes the find-file command. If the MDI.CMD page is loaded, this menu item is modified and bound to the open-file macro
View...	invokes the view-file command
Insert...	invokes the insert-file command
Read over...	invokes the read-file command
Rename...	invokes the change-file-name command
Save	invokes the save-file command
Save as...	invokes the write-file command
Append...	invokes the append-file command
Encryption key :	invokes the set-encryption-key command
Buffer	submenu
Window	submenu
Mode...	brings up a dialog box to change the modes of operation for the current buffer.
Global mode...	brings up a dialog box to change the global modes of operation.
Save + exit	invokes the quick-exit command
Exit	invokes the exit-emacs command

Buffer submenu

This menu is accessed via the File menu. It contains the following items:

Next	invokes the next-buffer command
Select :	invokes the select-buffer command
Unmark	invokes the unmark-buffer command
Rename :	invokes the name-buffer command
Delete :	invokes the delete-buffer command
Narrow to region	invokes the narrow-to-region command
Widen from region	invokes the widen-from-region command
List	invokes the list-buffers command

Window submenu

This menu is accessed via the File menu. It contains the following items:

Split	invokes the split-current-window command
Delete	invokes the delete-window command
Delete others	invokes the delete-other-windows command
Next	invokes the next-window command
Previous	invokes the previous-window command
Scroll	submenu
Size	submenu

Window Scroll submenu

This menu is accessed via the Window submenu of the File menu. It contains the following items:

= Up	invokes the move-window-up command
= Down	invokes the move-window-down command
= Next up	invokes the scroll-next-up command
= Next down	invokes the scroll-next-down command

Window Size submenu

This menu is accessed via the Window submenu of the File menu. It contains the following items:

= Grow	invokes the grow-window command
= Shrink	invokes the shrink-window command
= Height	invokes the resize-window command

Edit menu

This menu contains the following items:

Clipboard	submenu
Mark	submenu
Yank	invokes the yank command
Region	submenu
Paragraph	submenu
Line	submenu
Word	submenu
Delete blank lines	invokes the delete-blank-lines command
Transpose characters	invokes the transpose-characters command
Tab	invokes the handle-tab command
Quote	invokes the quote-character command
= Fill column	invokes the set-fill-column command. The emacs.rc page modifies this menu item slightly so that it prompts you for the fill column value.

If the CUA.CMD page is loaded, the menu is modified by the addition of the following item (before "Region"):

Selection	submenu

Clipboard submenu

This menu is accessed via the Edit menu. It contains the following items:

Cut region	invokes the cut-region command
Copy region	invokes the clip-region command
Paste	invokes the insert-clip command

If the CUA.CMD page is loaded, the menu is modified and, instead, contains the following items:

Cut	deletes and copies to the clipboard the text contained in the current selection
Copy	copies (without deleting) to the clipboard the text contained in the selection
Paste	inserts the text from the clipboard at the point

Mark submenu

This menu is accessed via the Edit menu. It contains the following items:

Set	invokes the set-mark command
Remove	invokes the remove-mark command
Exchange	invokes the exchange-point-and-mark command

Selection submenu

This menu is accessed via the Edit menu when the CUA.CMD page is loaded. It contains the following items:

Upper case	converts all the selected text to upper case
Lower case	converts all the selected text to lower case
Count words	displays on the message line the number of words, characters and lines that compose the selected text
Flip	exchanges the point with the other end of the selection
Select region	makes the current region the current selection

Region submenu

This menu is accessed via the Edit menu. It contains the following items:

Kill	invokes the kill-region command
Copy	invokes the copy-region command
Upper case	invokes the case-region-upper command
Lower case	invokes the case-region-lower command
Entab	invokes the entab-region command
Detab	invokes the detab-region command
Trim	invokes the trim-region command
Indent	invokes the indent-region command
Undent	invokes the undent-region command
Count words	invokes the count-words command

Edit Paragraph submenu

This menu is accessed via the Edit menu. It contains the following items:

Kill	invokes the kill-paragraph command
Fill	invokes the fill-paragraph command

Edit Line submenu

This menu is accessed via the Edit menu. It contains the following items:

Kill to end	invokes the kill-to-end-of-line command
Open	invokes the open-line command

Edit Word submenu

This menu contains the following items:

Kill next	invokes the delete-next-word command
Kill previous	invokes the delete-previous-word command
Capitalize	invokes the case-word-capitalize command
Lower case	invokes the case-word-lower command
Upper case	invokes the case-word-upper command

Search menu

This menu contains the following items:

Search forward :	invokes the search-forward command
Search backward :	invokes the search-reverse command
Hunt forward	invokes the hunt-forward command
Hunt backward	invokes the hunt-backward command
Incremental search :	invokes the incremental-search command
Reverse incremental :	invokes the reverse-incremental-search command
Replace :	invokes the replace-string command
Query replace :	invokes the query-replace-string command
Goto	submenu
Page	submenu
Paragraph	submenu
Line	submenu
Word	submenu

Goto submenu

This menu is accessed via the Search menu. It contains the following items:

Mark	invokes the goto-mark command
Line	invokes the goto-line command
Matching fence	invokes the goto-matching-fence command
Beginning of file	invokes the beginning-of-file command
End of file	invokes the end-of-file command

Page submenu

This menu is accessed via the Search menu. It contains the following items:

Next	invokes the next-page command
Previous	invokes the previous-page command

Search Paragraph submenu

This menu is accessed via the Search menu. It contains the following items:

Next	invokes the next-paragraph command
Previous	invokes the previous-paragraph command

Search Line submenu

This menu is accessed via the Search menu. It contains the following items:

Next	invokes the next-line command
Previous	invokes the previous-line command
Beginning of	invokes the beginning-of-line command
End of	invokes the end-of-line command

Search Word submenu

This menu is accessed via the Search menu. It contains the following items:

Next	invokes the next-word command
Previous	invokes the previous-word command
End of	invokes the end-of-word command

Execute menu

This menu contains the following items:

Windows program :	invokes the execute-program command
Shell program :	invokes the shell-command command
Pipe-in :	invokes the pipe-command command
Filter :	invokes the filter-buffer command
Shell	invokes the i-shell command
EMACS command	submenu
Keyboard macro	submenu
Abort command	invokes the abort-command command

If the DEV.CMD page is loaded, the menu is modified by the addition of the following item:

Make	invokes the run-makefile macro.

EMACS command submenu

This menu is accessed via the Execute menu. It contains the following items:

Named command :	invokes the execute-named-command command
Command line :	invokes the execute-command-line command
Procedure :	invokes the execute-procedure command
Buffer :	invokes the execute-buffer command
File...	invokes the execute-file command

Keyboard macro submenu

This menu is accessed via the Execute menu. It contains the following items:

Play	invokes the execute-macro command
Start recording	invokes the begin-macro command
End recording	invokes the end-macro command

Miscellaneous menu

This menu contains the following items:

Key bindings	submenu
Menu bindings	submenu
Variable	submenu
Show position	invokes the buffer-position command

Key bindings submenu

This menu is accessed via the Miscellaneous menu. It contains the following items:

Bind to Command	invokes the bind-to-key command
Bind to Macro	invokes the macro-to-key command
Unbind	invokes the unbind-key command
Describe key	invokes the describe-key command
List	invokes the describe-bindings command

Menu bindings submenu

This menu is accessed via the Miscellaneous menu. It contains the following items:

Bind to Command	invokes the bind-to-menu command
Bind to Macro	invokes the macro-to-menu command
Unbind	invokes the unbind-menu command

Variable submenu

This menu is accessed via the Miscellaneous menu. It contains the following items:

Set	invokes the set command
Display	invokes the display command
List	invokes the describe-variables command

Screen menu

This menu contains the following items:

Cascade	invokes the cascade-screens command
Tile	submenu
Arrange Icons	causes iconized screens to be rearranged at the bottom left of the MicroEMACS frame window.
Open	invokes the find-screen command
Rename	invokes the rename-screen command
Size	submenu
Font...	brings up a dialog box to change the font used by MicroEMACS

If the MDI.CMD page is loaded, the menu is modified by the addition of the following items:

Rebuild	rebuilds the set of screens, to have a screen associated with each editing buffer
Kill	deletes the current screen and release the corresponding buffer.

Additional items are added dynamically at the end of the "Screen" menu, listing the available screens. This allows quick switching between those screens.

Tile submenu

This menu is accessed via the Screen menu. It contains the following items:

Horizontally	causes all non-iconic screens to be rearranged in a tiling scheme, side by side if possible
Vertically	causes all non-iconic screens to be rearranged in a tiling scheme, on top of each other if possible

Screen Size submenu

This menu is accessed via the Screen menu. It contains the following items:

= Height	invokes the change-screen-size command
= Width	invokes the change-screen-width command
Normalize	causes the current screens to be resized so that it is as small as possible while retaining the same height and width in characters.

If the MDI.CMD page is loaded, the menu is modified by the replacement of "= Height" and "= Width" by the following item:

Set:	prompts you for the width and height of the screen, supplying the current values as defaults.

Help menu

This menu contains the following items:

Index	brings up this help file, on the main index.
Keyboard	brings up this help file, on the keyboard topic
Commands	brings up this help file, on the commands topic
Procedures	brings up this help file, on the procedures topic
List	submenu
Apropos :	invokes the apropos command
Describe key :	invokes the describe-key command
Display variable :	invokes the display command
About...	brings up a dialog box giving some information about MicroEMACS and the people involved in its making.

If the DEV.CMD page is loaded, the menu is modified by the addition of items (before "List") that invoke the Windows help engine for, respectively, Windows 3.0, Windows 3.1 or Win32 Software Development Kits or for Turbo C++. Each of those attempt to select a help topic based on the word currently at the point. You can eliminate the undesired items among these by editing the `macro-to-menu` commands in the DEV.CMD file.

List submenu

This menu is accessed via the Help menu. It contains the following items:

Key bindings	invokes the describe-bindings command
Functions	invokes the describe-functions command
Variables	invokes the describe-variables command
Buffers	invokes the list-buffers command

Customizing Command Keys

MicroEMACS lets you decide what keys activate what command or macro through the use of:

M-K	bind-to-key
^X^K	macro-to-key
M-^K	unbind-key

These commands can be used to permanently change your key bindings by placing them in your start up file. For example, if you have one of those nasty keyboards with a tilde "**~**" in the upper left corner, where the Escape key should be, and you want the tilde to become the meta key, add this line to emacs.rc:

```
bind-to-key   meta-prefix   ~
```

You can use this to make MicroEMACS feel similar to another editor by changing what keys activate which commands.

The unbind-key command is useful if you have a function key you keep tripping over, or if you are trying to make MicroEMACS look like a much more minimalist editor.

You can get a list of all the key bindings that MicroEMACS uses by using the describe-bindings command. Just do M-X and type:

```
describe-bindings
```

Issuing Commands

Commands within MicroEMACS have descriptive names which you can use to invoke them, or bind them to a keystroke or a menu. To invoke one of these commands by name, you can use:

M-X execute-named-command

You can supply numeric arguments to a such a command by prefixing it. You can also use a command line invocation.

To get a list of all the commands in your current MicroEMACS, do M-X and type:

```
describe-bindings
```

The describe-bindings command will display a paged list of all legal commands and the keystrokes to use to invoke them.

Interactive Numeric Arguments

Some commands take a number as an argument. For example, to move to a particular line within a file, you use the goto-line (M-G) command. To go to a particular line, precede the command with a number by striking the meta key, typing a number, and then the keys bound to the command. To go to the 123rd line of a file, use:

```
Meta 123   Meta g
```

If a command does not need a numeric argument, it is usually taken as a repeat count. This also works when typing any character. To make a line of 50 dashes type:

```
Meta 50   -
```

Command Lines

execute-command-line (M-^X) lets you type in a full command line. MicroEMACS macros are made from sequences of these command lines. A command line has three parts:

```
Numeric argument   Command   Arguments
```

The numeric argument is optional and has the same effect as an interactive numeric argument prefixing an interactive invocation of the same command.

Arguments following the command are not always required. If needed arguments have been omitted, the user will be prompted for them on the message line.

To insert the string "<*><*><*>" at the point, do M-^X and then:

```
3   insert-string   "<*>"
```

or to set the current fill column to 64, do M-^X and then:

```
64   set-fill-column
```

The Outside World

The following commands let you interact with the Operating System or with other applications:

^X^C	exit-emacs	terminates MicroEMACS
M-Z	quick-exit	same as above, but saves all changed buffers first
^X!	shell-command	executes a program within an Operating System "shell"
^X$	execute-program	launches another application
^X@	pipe-command	pipes a program's output into a buffer
^X#	filter-buffer	filters a buffer through a program
^XC	i-shell	opens an Operating System "shell"

Synchronizing With Another Program

When the pipe-command or the filter-buffer commands are used under Microsoft Windows, MicroEMACS creates a DOS box (or "shell box" under Windows NT) and waits for it to terminate.Also, if the execute-program or the shell-command command is invoked with a numeric argument, MicroEMACS waits for the launched application to terminate.

You can cancel the wait by pressing the Esc key or clicking on the "Cancel" button. Note that doing so does not terminate the other program.

For synchronization to work with a DOS box, the DOSExec profile must be set properly. Under Windows NT, shell boxes can be parametrized by setting the Shell and the ShellExecOption profiles.

Buffers

A buffer is where MicroEMACS stores text. Normally that text is read from a file, and is visible in an editing window. But text stored in buffers can also be MicroEMACS macros, temporary storage for macros, or lists of screens, files, buffers, variables, commands or bindings created by MicroEMACS commands. Commands that deal with buffers include:

^XB	select-buffer
^XK	delete-buffer
^X^B	list-buffers
^XX	next-buffer

Regions

Regions are used in MicroEMACS to specify what text is acted on by many commands. A region is defined as all the text between the point, and the last placed mark. To define a region:

1. Move the point to the beginning of the text you want to effect
2. Use the set-mark (M-) command to position the mark at the current point
3. Move the point to the end of the text you want to affect

At this time, the text between the mark and the point is the current region which will be affected by many commands. Regions can be defined backwards as well as forwards, and can include the entire buffer, or as little as one character.

Paragraphs

MicroEMACS defines a paragraph as any group of lines of text surrounded by blank lines. A line starting with one of the characters in the $paralead variable is considered the first line of a paragraph. Also, if line starts with one of the characters in the $fmtlead variable, the following line is considered to be the beginning of a paragraph.

Commands that deal with paragraphs include:

M-N	next-paragraph
M-P	previous-paragraph
M-^W	kill-paragraph
M-Q	fill-paragraph

Words

Words are defined, by default, as a string of characters consisting of alphabetic, numeral and the underscore "_" character. You can change this by setting the $wchars variable to a list of all the characters you want considered as part of a word.

The commands that deal with words include:

M-F	next-word
M-B	previous-word
M-D	delete-next-word
M-^H	delete-previous-word
M-^C	count-words

Screens

A screen is a collection of windows which are displayed together. On some non-graphically oriented systems, only one screen is displayed at a time. Under other graphical oriented operating systems like Microsoft Windows, X-Windows, the Macintosh or the Amiga, each screen may be displayed in an operating system "window". Notice that the MicroEMACS usage of the word window is different from the meaning used in these graphical systems:

MicroEMACS	Operating System
Window	Pane
Screen	Window

Each screen has its own set of windows. Switching from one screen to another (for instance by clicking on that screen) will preserve the window setup, the colors and the buffers being displayed.

When MicroEMACS starts up, it displays a single screen named "MAIN". Extra screens can be created by the command:

A-F find-screen

Windows

MicroEMACS uses windows to display and allow you to edit the contents of buffers. A single screen will show one or more windows, separated by a mode line which describes the contents of the window above it.

You can scroll text vertically and horizontally within a window by using the arrow keys or the page-up, page-down, home and end keys. Note that if a line of text extends beyond the boundary of a window, a dollar "**$**" sign is displayed instead of the last visible character.

Here are some window-related commands:

^X2	split-current-window
^X1	delete-other-windows
^X0	delete-window
^XO	next-window
^XP	previous-window

Notice that the MicroEMACS usage of the word window is different from the meaning used in graphical systems:

MicroEMACS	Operating System
Window	Pane
Screen	Window

Setting Colors

On systems which are capable of displaying colors, the mode commands can be used to set the background and foreground character colors. Using add-mode (^XM) or delete-mode (^X^M) and typing a lowercase color will set the background color in the current window. An uppercase color will set the foreground color in the current window.

In a similar manner, add-global-mode (M-M) and delete-global-mode (M-^M) will set the background or foreground colors of future windows.

Colors that MicroEMACS knows about are: **white**, **gray** (dark grey), **grey** (light grey), **cyan**, **lcyan** (light cyan), **magenta**, **lmagenta** (light magenta), **yellow**, **lyellow** (light yellow), **blue**, **lblue** (light blue), **red**, **lred** (light red), **green**, **lgreen** (light green) and **black**. If the computer you are running on does not have enough colors, MicroEMACS will attempt to guess at what color to use when you ask for one which is not there (systems with only 8 colors support: white, cyan, magenta, yellow, blue, red, green and black).

Under Microsoft Windows, the whole 16 colors above are available if the display system supports them (depending on the value of the Colors profile). In that case, Mode lines are displayed as black characters on a light grey background. The message line and desktop colors can be modified through the Windows "control panel" as "window text", "window background" and "application workspace". The value of the $deskcolor variable is always irrelevant.

Setting the Font

Under Microsoft Windows, the font used by MicroEMACS to display text within the screens and the message line can be selected by using the **Font...** item in the **Screen** menu. This brings up a dialog box in which you can select:

The character set	"ANSI" is the usual default within Windows application. "OEM" is useful when displaying files that contain pseudo-graphics characters.
The face name	You can chose any of the available fixed-pitch faces.
The size of the font	You can either chose one of the font heights listed or type one if you have scalable fonts. All heights are expressed in pixels.
The font weight	Normal unless you check the "Bold" box.

A sample of the selected font is shown, specifying its height and width. The maximum screen size is calculated as the number of columns and rows (including mode lines) that would be displayed in a maximized screen when the MicroEMACS frame is maximized.

Pressing the Enter key or the **OK** button effects the change of font in MicroEMACS. Pressing the Alt+S keys or the **Save** button has the same effect, but also saves the font selection in the profiles so that next time MicroEMACS is started, it uses that font. Pressing the Escape key or the **Cancel** button returns to MicroEMACS without changing the font.

Case Control

The following commands let you change the case of the word at or following the point:

M-C	case-word-capitalize
M-L	case-word-lower
M-U	case-word-upper

Setting a mark, moving to the other end of the region and using one of these commands will change the case of all the words in the selected region:

^X^L	case-region-lower
^X^U	case-region-upper

Controlling Tabs

By default, MicroEMACS sets the default tab stops every eighth column. This behavior can be changed (usually within the start-up file).

The behavior of the handle-tab (^I or Tab key) command depends on the numeric argument that is supplied to it:

> With no argument, **handle-tab** inserts space characters or a single tab character to get to the next tab stop, depending on its configuration...
>
> With an non-zero argument *n*, tabs stops are reset to every *n*th column and **handle-tab** is reconfigured to insert space characters in sufficient number to get to the next tab stop. This also sets the $softtab variable to *n*.
>
> With an argument of zero, **handle-tab** is reconfigured so that it inserts true tab characters (its default behavior) and the tab stop interval is reset to its default value of 8.

The distance which a true tab character moves the cursor is reflected by the value of the $hardtab variable. Initially set to 8, this determines how far each tab stop is placed from the previous one.

Tab characters can be globally replaced by the appropriate number of spaces by the detab-region (^X^D) command. The reverse, entab-region (^X^E) changes multiple spaces to tab characters.

Repetitive Tasks

To perform any repetitive task, where you have a list of things that need to be changed, for instance one per line, follow these steps:

1) Position the point to the beginning of the line to change
2) Invoke begin-macro (^X()to start recording
3) make the change, staying on that line
4) move to the beginning of the next line
5) Invoke end-macro (^X)) to stop recording

Do execute-macro (^XE) once to test your change on the next line. If it is satisfactory, count how many lines need to yet be changed, strike the meta key followed by that number and ^XE. This causes your change to be made on all the lines.

Narrowing Your Scope

Many times you will want to do something to a part of the text when the command works on all the text. Also it is helpful to see or edit just a portion of the text.

This kind of editing can be performed by narrowing the buffer and later restoring the invisible portions, using the following commands:

^X< narrow-to-region
^X> widen-from-region

Creating New Commands

MicroEMACS lets you create your own macros to perform any editing tasks, simple or complex. These macros are written in the MicroEMACS macro language. Macros can be invoked by other macros and they can be bound to keystrokes by the macro-to-key (^X^K) command.

For examples of macros, look at the .CMD files supplied with MicroEMACS for Windows. In that package, EMACS.RC is the file which is executed automatically whenever MicroEMACS is started. and all the ???.CMD files contain the code for each page.

Customizing Menus

MicroEMACS menus can be modified by the following commands (usually employed in the start-up file):

bind-to-menu	creates a menu item bound to a command
macro-to-menu	creates a menu item bound to a macro
unbind-menu	deletes a menu item

With these three commands, menus are specified by using the MicroEMACS menu name syntax.

Menu Name Syntax

Menu names used by the bind-to-menu, macro-to-menu and unbind-menu commands follow a common syntax. A menu name is composed of menu item names separated by right brackets:

```
>item1>item2>item3
```

When a menu name begins by a right bracket "**>**", it means that the menu item immediately following this right bracket is located within the menu bar. A menu name can also be specified as:

```
item1>item2
```

In this case *item1* is located within the last accessed menu. One or more left brackets "**<**" can appear before the first item, meaning it is located as many levels up in the menu hierarchy:

```
<<item1>item2
```

Notes: The tilde character "**~**" cannot be used to escape the meaning of the brackets ("**<**" or "**>**") and ampersand "**&**" characters within menu names. The brackets simply cannot be escaped. The ampersand can be escaped (i.e. considered as a real ampersand instead of indicating the underscoring of a character) by using two consecutive ampersands: "**&&**".

It is good practice to enclose menu names in double quotes. This is necessary when there are embedded spaces within a name. Also, when a menu name begins by an ampersand, MicroEMACS may misinterpret it as a function name.

See the examples for a more practical explanation...

Menu Item Syntax

Menu item names are used as parts of menu names. They specify a single menu item within a given popup menu or within the menu bar. A menu item name can be formed of an *item text* and/or an *item index*:

```
item text@item index
```

or:

```
item text
```

or:

```
@item index
```

The *item text* specifies the text of the item that appears within the menu, using an ampersand "**&**" as a prefix for the underlined character. Note that the key binding description, if any, is automatically generated by MicroEMACS and should not be part of the *item text*.

The *item index* is a decimal number that specifies the index of the item within the menu. Indexes start at zero.

If the specified item is being created:

The *item text* is mandatory.

Separators (horizontal lines between parts of a popup menu) are specified by the *item text* being a single dash "**-**". Note that either bind-to-menu or macro-to-menu can be used for this, since the bound command or macro is irrelevant (although it has to be a valid one).

The *item index* can be used to specify the position where the new item will be placed

If the *item index* is not specified, the new item is placed at the end of the menu or just after the item that was used in a previous menu binding command.

If the specified item already exists:

If the item is not a separator, only one of *item text* or *item index* is needed (but both can be specified).

If the item is a separator, the *item index* should be specified but **not** the *item text*.

See the examples for a more practical explanation...

Menu Examples

```
bind-to-menu  forward-character  ">&Search>&Character@15>&Next"
bind-to-menu  nop                "-"
bind-to-menu  backward-character  "&Previous"
```

This creates a new popup menu named "Character" under the "Search" menu, containing the two items "Next" and "Previous", with a separator (for the sake of the demonstration) between the two.

```
unbind-menu    ">&Search>&Character>@1"
```

removes the above-created separator.

```
macro-to-menu  load-c-page    ">Code &page@4>&Load>&C"
macro-to-menu  load-cpp-page  "C&++"
macro-to-menu  load-p-page    "&Pascal"
macro-to-menu  remove-c-page  "<&Remove>&C"
macro-to-menu  remove-cpp-page   "C&++"
macro-to-menu  remove-p-page  "&Pascal"
bind-to-menu   nop            "<-"
macro-to-menu  remove-all-pages  "Remove &all"
```

This (assuming the specified macros actually exist) creates a new menu "Code page", located between the "Execute" and the "Miscellaneous" menus in the menu bar. This new menu contains the "Load", "Remove" and "Remove all" items, the later being preceded by a separator. Both the "Load" and "Remove" items actually lead to sub-menus that both contain "C", "C++" and "Pascal".

Drag and Drop

Under MS-Windows 3.1 and above, MicroEMACS supports a "drag and drop" file-selection mechanism. If you select one or more files in the Windows File Manager and drag them with the mouse, dropping them over MicroEMACS generates a pseudo mouse action: MS! that can be used by binding it to a macro.

For instance, the following command causes a macro named "drop-files" to be invoked every time a group of files is dropped on MicroEMACS:

```
macro-to-key  drop-files  MS!
```

The macro that handles the drag and drop mechanism acquires the necessary information from a buffer named "**Dropped files**":

> The first line of that buffer contains the name of the screen on which the drop occurred. It is empty if the files were not dropped on any specific screen (for instance if they were dropped on the message line).
>
> The second and following lines contain the list of dropped files, one pathname per line.

In addition, the $xpos and $ypos variables are set to the text coordinates where the drop occurred (or to the value 255 if the files were not dropped on any specific screen).

The MDI.CMD page contains a sample macro that handles drag and drop.

Modes of Operation

Modes determine how MicroEMACS will treat text. Modes affect the contents of a buffer. Global modes determine the modes of newly created buffers.

^XM	add-mode	Adds a mode to the current buffer
^X^M	delete-mode	Removes a mode from the current buffer
M-M	add-global-mode	Adds a global mode
M-^M	delete-global-mode	Removes a global mode

MicroEMACS's modes are:

ASAVE	Automatically Save
CMODE	Editing C programs
CRYPT	Encryption
EXACT	Character Case during Searches
MAGIC	Regular Expression Pattern Matching
OVER	Overstrike Mode
REP	Replace Mode
VIEW	No Changes Permitted
WRAP	Wrap entered text

ASAVE Mode

When this mode is on, MicroEMACS automatically saves the contents of your current buffer to disk every time you have typed 256 characters. The buffer is saved to the file named on the mode line of the buffer. This mode assures you that you will loose very little text should your computer crash while you are editing. Be sure you are willing to have your original file replaced automatically before you add this mode.

The frequency of saving can be altered by changing the contents of the $asave variable.Use the set (^XA) command like this:

```
^XA $asave 2048
```

to tell MicroEMACS to automatically save the current buffer after 2048 characters are typed.

Note: the $acount variable contains the count down to the next auto-save.

CMODE Mode

This mode is specifically for editing programs written in the C language. When CMODE is active, MicroEMACS will try to anticipate what indentation is needed when the newline (^M or Enter key) command is used. It will always bring a pound sign "#" with only leading white space back to the left margin. It will also attempt to flash the cursor over the proper opening fence character matching any closing fence character (one of "**)}]**") that is typed (the duration of this flashing can be controlled by setting the $tpause variable).

Note that the standard start-up files for MicroEMACS install a macro which checks any file being read into MicroEMACS and sets CMODE if the file ends with a .c or .h extension.

Related command:

M-^F goto-matching-fence

CRYPT Mode

For files of a sensitive nature, MicroEMACS can encrypt text as it is written or read. The encryption algorithm is a Beaufort Cipher with a variant key. This is reasonably difficult to decrypt.

When you write out text, if CRYPT mode is active and there is no encryption key, MicroEMACS will ask:

> Encryption String:

Type in a word or phrase of at least five and up to 128 characters for the encryption to use. If you look at the file which is then written out, all the printable characters have been scrambled.To read such a file later, you can use the **-k** switch when calling up MicroEMACS:

> emacs -k *filename*

and you will be asked the encryption key before the file is read.

You can modify the encryption key by using the set-encryption-key (M-E) command.

Note: previous versions of MicroEMACS used a defective encryption method. For compatibility, you can chose to use the older algorithm by setting the $oldcrypt variable to TRUE.

EXACT Mode

Normally, when using search or replace commands, MicroEMACS ignores the case of letters for comparisons. With EXACT mode set, the case of the characters must be the same for a match to occur.

MAGIC Mode

Normally, MicroEMACS uses the string you type in response to a search or replace command as the string to find. When magic mode is enabled, MicroEMACS considers the string you type as a pattern or template to use in finding a string to match. Many characters in this template have special meaning:

.	any single character, except newline.
[*set***]**	any single character from the bracketed *set*.
^	beginning of a line.
$	end of a line.
\	the next character has no special meaning, take the next character literally (unless it is a parenthesis)
?	the preceding character (or "**.**" or **[***set***]**) is optional.
*	the preceding character (or "**.**" or **[***set***]**) matches zero to many times.
+	the preceding character (or "**.**" or **[***set***]**) matches one to many times.
\(*group***\)**	define a group for the replacement string, or for the &group function.

Some characters in the replacement string can have special meanings:

&	insert all of the text matched by the search.
\	the next character has no special meaning (but see groups below...)
\1 to **\9**	insert the text defined by the *n*th group in the search string.

OVER Mode

MicroEMACS is normally in what many other editors consider "insert" mode. This means when you strike a character, MicroEMACS makes room for that character in the current line, inserting it between the existing characters. In OVER mode, MicroEMACS instead overwrites characters, replacing the existing character under the point with the character you type. OVER mode will maintain the position of text lined up using tabs while replacing existing text.

Be wary of editing Japanese KANJI characters while in this mode: it is possible to overwrite the first byte of the character, leaving the second byte meaningless and alone. REP mode is more appropriate for such files.

WRAP Mode

This mode causes the point and the previous word to jump down to the next line when you type a space and are beyond the current fill column. This is normally set to column 72, allowing you to enter text non-stop on a standard screen without bothering to use the return key.

To change the column that text is wrapped past, use the set (^XA) command to change the value of the $fillcol variable, like this:

    ^XA $fillcol *new_value*

MicroEMACS will then be set to wrap words past column *new_value*.

The $wraphook variable contains the command or macro used to perform word wrapping. By default, it is the wrap-word command.

VIEW Mode

When in VIEW mode, no command which would change the text is allowed. If you attempt any such command, or try to type in any text, MicroEMACS responds with:

    [Key Illegal in View Mode]

This mode is very useful when you want to just look at some existing text, as it will prevent you from changing that text. Also MicroEMACS will not attempt a file lock if a file is read in VIEW mode, allowing you to view files which you don't have write access to, or other people have locked. To launch MicroEMACS and read a file in VIEW mode, use the -v switch:

    emacs -v *filename*

REP Mode

MicroEMACS is normally in what many other editors consider "insert" mode. This means when you strike a character, MicroEMACS makes room for that character in the current line, inserting it between the existing characters. In REP mode, MicroEMACS instead replaces the existing character under the point with the character you type. REP mode will not maintain the position of text which takes up multiple columns using tabs since it will replace a single tab character with the typed character which will not take up the same space on screen. For this purpose, the OVER mode is more appropriate

However, Japanese KANJI characters will correctly replace and be replaced in this mode as the two bytes will be considered together when either style character is used.

Start-up

There are different things that can be specified on the MicroEMACS command line to control the way the editor operates. These can be switches, which are a dash "**-**" followed by a letter, and possible other parameters, or a start-up file specifier, which is an at sign "**@**" followed by a file name that overrides the default "EMACS.RC".

Under Microsoft Windows, MicroEMACS also uses some profiles from the WIN.INI file.

Start-up File

When MicroEMACS starts executing, it looks for a start-up file which it will execute as a macro before it reads in any other file. This start-up macro usually redefines some bindings (for instance to use function keys) and loads various useful macros.

The name of the start-up file can be specified on the MicroEMACS command line. By default, it is: EMACS.RC.

Unless the pathname of the start-up file is fully qualified, MicroEMACS searches for the file along the path.

Command Line Switches

The command line used to launch MicroEMACS looks like this:

```
EMACS.EXE   switches   files to edit
```

The following *switches* can be specified:

@*file*	This causes the named *file* to be executed instead of the standard EMACS.RC file before MicroEMACS reads in any other files. More than one of these can be placed on the command line, and they will be executed in the order that they appear.
-C	The following source files on the command line can be changed (as opposed to being in VIEW mode). This is mainly used to cancel the effects of the -v switch used previously in the same command line.
-E	This flag causes emacs to automatically run the start-up file "error.cmd" instead of emacs.rc. This can be used by compilers for error processing.
-G*num* or **+***num*	Upon entering MicroEMACS, position the cursor at the *num* line of the first file.
-I*var value*	Initialize a MicroEMACS variable with *value*.
-K*key*	This tells MicroEMACS to place the source files in CRYPT mode and read it in using *key* as the encryption key. If no key is listed after the **-K** switch, you will be prompted for a key, and it will not be echoed as it is typed.
-R	This places MicroEMACS in "restricted mode" where any commands allowing the user to read or write any files other than the ones listed on the command line are disabled. Also all commands allowing the user access to the operating system are disabled. This makes MicroEMACS a "safe" environment for use within other applications and especially used as a remote editor for an electronic Bulletin Board System (BBS).
-S*string*	After MicroEMACS is started, it automatically searches for *string* in the first source file.
-V	This tells MicroEMACS that all the following files on the command line should be in VIEW mode to prevent any changes being made to them.

Profiles

Profiles are entries in the WIN.INI file and are used only under Microsoft Windows. MicroEMACS uses a few profiles, all placed under the "[MicroEMACS]" section, to define the initial window size, the initial font and the path names of some files.

The following profiles can be modified by editing the WIN.INI file:

Colors	number of colors supported by the display.
DOSExec	path name of a PIF file for pipe-command, filter-buffer and i-shell
DOSBox	path name of a PIF file for shell-command
HelpFile	path name of this help file
InitialSize	keywords: "maximize", "minimize" or "optimize"
Shell	path name of the shell executable under Windows NT.
ShellExecOption	command execution option for the shell under Windows NT.
TimeSlice	number of milliseconds of processing before yielding to other applications

The font-related profiles (**FontName**, **FontWeight**, **FontWidth**, **FontHeight** and **CharSet**) are updated by MicroEMACS itself when a font selection is saved.

Colors Profile

The Colors profile is used to force MicroEMACS to run in either color or monochrome mode. In color mode, the mode lines display back text over a light grey background and editable text is displayed as white on black (these colors can be customized). In monochrome mode, MicroEMACS uses the colors specified by the system (configurable through the Windows Control Panel), using highlighted text for the mode lines.

The value associated to the colors profile is the number of colors supported by the system, or zero (to allow MicroEMACS to automatically determine the proper value). Monochrome mode is assumed for values 1 and 2. Values greater than 2 put MicroEMACS in color mode.

If this profile does not appear in the [MicroEMACS] section of the WIN.INI file, the default value is 0.

Setting this profile is particularly useful on monochrome displays that allow multiple shades of gray (in particular, laptop screens), as MicroEMACS mistakenly believes these to be actual color displays.

DOSExec Profile

The DOSExec profile specifies the path name of a PIF file used by the pipe-command, filter-buffer and i-shell commands under MS Windows 3.x. This profile is also used when the shell-command command is invoked with a numeric argument.

If this profile does not appear in the [MicroEMACS] section of the WIN.INI file, the file "DOSEXEC.PIF" is searched along the path. This is appropriate if, for instance, that file is located in the directory where the MicroEMACS executable resides.

DOSBox Profile

The DOSBox profile specifies the path name of a PIF file used when the shell-command is invoked without a numeric argument under MS Windows 3.x.

If this profile does not appear in the [MicroEMACS] section of the WIN.INI file, the file "DOSBOX.PIF" is searched along the path. This is appropriate if, for instance, that file is located in the directory where the MicroEMACS executable resides.

HelpFile Profile

The HelpFile profile specifies the path name of the Help file for MicroEMACS. It allows proper function of the menu items that call-up this Help file.

The default value is the file "MEWIN.HLP" within the directory where the MicroEMACS executable resides.

InitialSize Profile

The InitialSize profile specifies options for the sizing of the initial MicroEMACS frame window. It can be one of the following keywords:

maximize	the frame window fills the whole display
icon or **minimize**	MicroEMACS starts as an icon
optimize	the frame window fills the whole display, except a single row of icons at the bottom.

If the InitialSize profile is not used, the initial size of the MicroEMACS frame window is decided by the operating system.

Shell and ShellExecOption Profiles

The **Shell** profile specifies the path name of the shell executable used by the pipe-command, filter-buffer, i-shell and shell-command commands under Windows NT. If this profile does not appear in the [MicroEMACS] section of the WIN.INI file, the default path name is "CMD.EXE". This is appropriate if that file is located in a directory that appears in the system path.

The **ShellExecOption** profile specifies the string to be inserted between the string specified by the Shell profile and the actual command to be executed (for pipe-command, filter-buffer and shell-command). If this profile does not appear in the [MicroEMACS] section of the WIN.INI file, the default is " /c ". This is appropriate for "CMD.EXE".

TimeSlice Profile

Under Microsoft Windows 3.x, when MicroEMACS performs a long operation (reading or writing a large file, searching text, moving large chunks of text to/from the kill buffer or clipboard, killing a buffer, etc...), it allows other applications to run concurrently with itself.

The TimeSlice profile specifies how often MicroEMACS should relinquish the processor: when a long operation is in process, MicroEMACS does not yield to other applications until the number of milliseconds thus specified has elapsed.

The default value is 100 milliseconds.

Notes: Under Windows NT, the preemptive multitasking nature of the operating system alleviates the need for MicroEMACS to voluntarily yield to other applications. The TimeSlice profile is still used to determine how often input (like a command to exit the editor) is checked.

If the animated grinder (replacing the hourglass mouse cursor) is enabled, the TimeSlice profile also determines the time interval between each change of the cursor image.

Memory Usage

The only limit to the number of buffers is the memory of your computer. All the buffers, text, screens and windows use memory for storage.

Under Microsoft Windows, the accessible storage can be rather large, depending on the amount of extended memory installed on you system. If you are running in Windows 3.x 386-enhanced mode, MicroEMACS is able to use virtual memory, allowing you to edit very large files.

Under MSDOS, the AMIGA, the Atari ST, the HP150 and other microcomputers you can estimate the memory used by adding up the size of all the files you want to edit simultaneously, multiply by 1.4, and add 170K for the size of MicroEMACS. This results in the amount of free memory needed to edit these files. On an MSDOS machine with 574K of conventional memory available, you can edit files totaling about 288K in size. If you are using the DOS-extended version of MicroEMACS, the memory available for editing is determined by the amount of extended memory installed in your computer, up to 16 Megabytes.

On UNIX, Windows NT and other systems with large virtual memory there is almost no limit to the number and size of files you edit.

MS-Windows Specifics

The port of MicroEMACS to the Microsoft Windows environment exhibits a few particularities not encountered with other versions of the editor:

All the standard commands are available. Additional commands are available: they allow access to the clipboard, menu customization, invocation of the help engine and control of screens as MDI (Multiple Document Interface) windows.

In interactive mode, the file access commands use a dialog box instead of the message line prompt.

It is possible to drag files from the Windows File Manager onto MicroEMACS, providing a macro has been set-up to handle them.

MDI windows (aka screens) and the MicroEMACS frame window can be resized by dragging their border with the mouse or using the sizing buttons.

Text can be scrolled into view by using the scroll bars located at the right and bottom of each screen.

When MicroEMACS is running a macro, waiting for user input on the message line, or reading/writing a file, it is possible to input menu or other mouse commands, but only a subset of features is available. In particular, resizing is disabled and most menu options are grayed.

It is possible to terminate MicroEMACS at any time, using the "Close" (Alt+F4) item of the upper-left corner menu box. If there are modified buffers, or a file write operation is in progress, a confirmation is requested.

The amount of memory available for buffers is limited only by the actual (conventional and extended) memory available, including virtual memory when running Windows NT or Windows 3.x in 386-enhanced mode.

MicroEMACS can synchronize with other applications it launches.

MicroEMACS runs as a well-behaved Windows application, sharing the processor with other applications, even when a lengthy operation is in process.

Under Windows 3.x, MicroEMACS is a protected mode-only application: **it does not support real mode**, and runs only under standard or 386-enhanced mode.

The following page are distributed with MicroEMACS for Windows and loaded by the emacs.rc start-up file supplied in the distribution package:

CUA.CMD	Common User Access macros
DEV.CMD	example macro for software development
MDI.CMD	macros to map files to MDI windows

In addition, if a page named CUSTOM.CMD (to be supplied by the user) is found in the path, it is loaded after the three above.

CUA.CMD

This page is distributed with MicroEMACS for Windows and loaded by the emacs.rc start-up file. It contains a number of macros and rebinds many keys, in order make MicroEMACS more similar to other Windows applications that use the Common User Access standard.

To that end, a set of clipboard-related macros are supplied and you can select a piece of text by dragging the mouse across it while holding the left button held down or by moving around with the arrows or page keys while holding the Shift key down. That selection can then be **deleted** by pressing the Delete key, **copied** to the clipboard with the Ctrl+Insert keys, **cut** with Shift+Delete and **pasted** from the clipboard with Shift+Insert

Additionally, the following general purpose macros that work on the selection are supplied:

A-U	**CUA-case-upper**	converts all the selected text to upper case
A-L	**CUA-case-lower**	converts all the selected text to lower case
A-W	**CUA-count-words**	displays on the message line the number of words, characters and lines that compose the selected text
A-=	**CUA-flip-selection**	exchanges the point with the other end of the selection
A-^M	**CUA-select-region**	(Alt+Enter) makes the current region the current selection

DEV.CMD

This sample page is distributed with MicroEMACS for Windows and loaded by the emacs.rc start-up file. It contains a few of macros that demonstrate how some features of the macro language can be used to facilitate software development:

The **run-makefile** macro is added to the Execute menu. It spawns a shell to run the command specified by the **%make** user variable and synchronizes with it. When the make process is finished, its output is displayed in a buffer named "**Results**".

A series of macros are added to the Help menu. They search a specific help file for a topic matching the word under the point.

MDI.CMD

This page is distributed with MicroEMACS for Windows and loaded by the emacs.rc start-up file. It contains macros that make it easier to associate each buffer with a separate screen (i.e. an MDI window). To that end:

The **open-file** macro replaces the find-file command in the File menu and in key bindings (^X^F). Instead of reusing the current screen, it creates a new screen to house each newly opened file.

The **rebuild-screens** macro, invoked from the Screen menu, associates a screen to each buffer.

The **kill-screen** macro (A-K) deletes a screen and the associated buffer.

MDI.CMD also contains the **drop-files** macro that handles drag and drop actions by invoking the **open-file** macro for each dropped file.

Sorry, no help available on this topic

You have attempted to get Help for a term that the Help system does not recognize.

Here are some other ways to find Help for individual terms:

Help Search

1) Choose the **Search** button (Alt+S) from the top of this Help window (just below the menu bar).
2) In the Help Search dialog box, under Search For, type in the term you want Help for. If the term is indexed in the Help, you will go to that term in the upper list box. If the term is not indexed, you will go to the closest lexical match instead.
3) Press Enter or choose the dialog's **Search** button. You will see a list of 1 or more Help topics in the Topics Found

Alternatively, within the Help Search list box, scroll through the list to find a specific topic, then press Enter or choose the **Go To** button to jump to that Help topic.

Help Index

1) Use the **Index** button (Alt+I) and then choose the category that best fits your query.
2) Then traverse Help links through the topics until you find what you are looking for. If it is documented in the Help system, you should be able to find it within 4 or 5 topics.

Glossary

Argument
Binding
Buffer
Clipboard
Command
DOS Box
File Locking
Function
Group
Keyboard Macro
Keystroke
Kill Buffer
Kill Ring
Macro
Mark
Message Line
Meta Key
Mode Line
Mouse Action
Page
Path
Point
Popup Buffer
Region
Screen
Selection
Variable
Window

A **DOS Box** is a Microsoft Windows feature within which DOS programs are executed. In Windows "386-enhanced" mode, a DOS box can appear as an icon, a window or it can occupy the whole screen. In Windows "standard" mode, DOS programs can execute only when their DOS box occupies the whole screen.
Under Windows NT, the equivalent of DOS Boxes are named "shell boxes"

Commands are built in functions that represent basic things that MicroEMACS does. For example, the up arrow key activates the "previous-line" command which moves the cursor up to the line of text immediately before the current line.

A **binding** is a link between a sequence of keys and a command or macro. For instance, the command "previous-line" is bound to the up-arrow key, and to the ^P key. Pressing a key sequence causes the command to which it is bound to execute.

Under Microsoft Windows, commands and macros can also be bound to menu items.

The **meta key** is the key used to start many commands. On most keyboards this is the Escape key, but many times it is rebound/changed to the key in the upper left corner of the keyboard. This is often the grave accent symbol.

Interactively, a **numeric argument** is supplied by typing the meta key (usually the Escape key), followed by a decimal number, before invoking a command.

Within the macro language, a numeric argument is placed before the name of the associated command.

Buffers are areas of memory set aside to hold text. Each buffer has a buffer name which is used to refer to it, and a file name from which it has been read or where it will be written.

Popup Buffers are a way to display a buffer temporarily, without using a window. When a popup buffer is displayed, it occupies the whole screen. If more than one screenfull is needed, the text "-- more --" appears on the message line. The next screenfull can be viewed by pressing the space bar. Pressing any other key cancels the popup buffer and the keystroke is then processed by MicroEMACS.

Windows are sections of the current screen which display a portion of a buffer. More than one window may be visible at a time. Multiple windows split the screen horizontally.

Notice that the MicroEMACS usage of the word window is different from the meaning used in window-based systems:

MicroEMACS	Operating System
Window	Pane
Screen	Window

Screens are collections of windows. On a older text style system, one screen is displayed at a time. On a newer window based system, like OS/2, the Macintosh or Microsoft Windows, each operating system window can display a different MicroEMACS screen.

Notice that the MicroEMACS usage of the word window is different from the meaning used in window-based systems:

MicroEMACS	Operating System
Window	Pane
Screen	Window

The **mode line** is the line at the bottom of each window naming the buffer being displayed, along with its file name. Also the active modes of the window are shown.

The **command line** or **message line** is the line at the bottom of the screen where you give more information to some commands and also receive information or error messages.

Macros (also called **procedures**) are programs written in the MicroEMACS language which let you customize the editor and, in particular, automate repetitive editing tasks.

A **keyboard macro** is a remembered sequence of keystrokes which can be used to greatly speed quick and dirty repetitive editing.

Pages are groups of macros which have been written to handle a particular editing task, and which have been packaged to be available from the MicroEMACS startup file. These files usually have a filename extension of ".CMD".

The MS-Windows version of MicroEMACS is bundled with sample macro pages called CUA.CMD, DEV.CMD and MDI.CMD.

The **path** is a list of directories that MicroEMACS searches for the following files:

EMACS.RC (the startup file)
The argument of the execute-file command
The argument of the &find function
The default DOSEXEC.PIF and DOSBOX.PIF files
EMACS.HLP (for the help command)

The following items compose the **path** (in order of decreasing priority):

1. The directory specified by the HOME system variable (or, under MS-Windows, the directory where the MicroEMACS executable resides).
2. The current working directory.
3. The directories specified in the PATH system variable.
4. The following directories (MS-DOS-based or Windows NT systems only. Other implementations use different lists):

    \sys\public
    \usr\bin
    \bin
    \

The **point** is the position of the cursor in the text of the current window. The point can be considered to lie between the character the cursor rests on and the one immediately after it.

The **mark** is the position in the current buffer which delimits the beginning or the end of a region. Various commands operate on text from the mark to the point, or move the current point to the mark. The mark can be set by the set-mark command.

Each buffer contains 16 independent marks, numbered 0 to 15. Most region-related commands, however, only refer to mark 0.
Marks 10 to 15 are reserved for use by macros. Marks 0 to 9 are reserved for interactive use.
Marks 11 and 12 are used internally by MicroEMACS by the mouse-region-down and mouse-region-up commands.

A **region** is the text located between the point (i.e. the position of the cursor) and the mark number 0. The mark can be set by the set-mark command.

The **selection** is available only if the macros from the CUA.CMD page have been loaded. It is the piece of text that has been selected by dragging the mouse (with the left button held down) over it, or by moving (with the arrow or the page keys) through the text with the Shift key held down.

The CUA.CMD file is distributed as part of the MicroEMACS for Windows package.

In the current version of MicroEMACS, the selection is not highlighted.

The **clipboard** is a temporary storage area. Text can be cut or copied to the clipboard from a Windows application and be pasted into another application.

Variables are elements of the MicroEMACS macro language. They carry numeric, boolean or string values.

Variables that begin with a dollar sign "**$**" are called environmental variables. They control various aspects of the editor.

Functions are elements of the MicroEMACS macro language. Functions have arguments and return numeric, boolean or string values.

Function names begin by an ampersand "**&**". Only the first 3 characters of a function name are significant.

Groups can be used with text substitution commands or macros in MAGIC mode, to duplicate parts of the target into the result.

In the search string, a group is defined as a portion beginning by the characters backlash and opening parenthesis "**\(**" and ended by the characters backlash and closing parenthesis "**\)**". There can be up to nine such groups.

In the replace string, groups appear as a backlash followed by a decimal digit ("**\1**" to "**\9**"). The portion of the target string matched by the *n*th group is substituted to each occurrence of *n* to form the replacement string.

The function &group *n* can be used in macros to obtain the text matched by the *n*th group in a search.

MicroEMACS may implement **file locking** to prevent simultaneous access of the same file by different MicroEMACS instances. The method used for this is dependant on the base operating system.

File locking is active only if MicroEMACS was compiled with a specific "FILOCK" option. Standard release versions usually do not implement file locking.

The **kill buffer** accumulates any text which is "killed" by a number of delete commands. If more than one delete command is used in a row, all the text from all the commands will be in the kill buffer. Using any command between deletes causes the kill buffer to just hold the most recent deletions.

Using this feature and the yank command, you can switch between windows, screens and files and copy text from one file to another.There is no limit to the amount of text that can be stored in the kill buffer except that of the memory of the computer running MicroEMACS. Extremely large kills may take a few seconds.

The last 16 kill buffers are kept in the kill ring. You can retrieve their contents through the cycle-ring or the yank-pop commands.

The **kill ring** is a circular list of the last 16 kill buffers. The position of the current kill buffer can be changed by the cycle-ring and the yank-pop commands. The kill ring can be emptied (and thus the used memory reclaimed) by using the delete-kill-ring command.

Mouse Syntax
Key bindings can include mouse actions which are represented as follows:

	Press	Release
Left button:	**MSa**	**MSb**
Center button:	**MSc**	**MSd**
Right button:	**MSe**	**MSf**
Shift+Left button:	**S-MSa**	**S-MSb**
Shift+Center button:	**S-MSc**	**S-MSd**
Shift+Right button:	**S-MSe**	**S-MSf**
Ctrl+Left button:	**MS^a**	**MS^b**
Ctrl+Center button:	**MS^c**	**MS^d**
Ctrl+Right button:	**MS^e**	**MS^f**

Mouse movement: **MSm**
Dropping files dragged from the MS-Windows File Manager: **MS!**

Keystroke Syntax:
In key bindings, regular characters are represented by the corresponding uppercase, preceded by a hat "**^**" sign if the Ctrl key is depressed. For instance, for Ctrl+G: **^G**.

Ctrl+SPACE is represented by **^@**.

Function keys are represented as:

F1 to F9, F10: **FN1** to **FN9**, **FN0**
Arrows: up **FNP**, down **FNN**, left **FNB**, right **FNF**
Page keys: up **FNZ**, down **FNV**
Other keys: Home: **FN<**, End: **FN>**, Insert: **FNC**, Del: **FND** (or **^?**)

If the Ctrl key is depressed for a function key, the hat "**^**" is located before the last char. For instance, for Ctrl+F1: **FN^1**.

The prefix, if any, appears before the keystroke:

M- the meta key (usually the Escape key) is depressed and released.
^X the Ctrl+X keys are depressed and released.
A- the Alt key is depressed.
S- (function keys only) the Shift key is depressed.

Macro Language

The MicroEMACS macro language allows you to add extensions to the editor. Statements (one per line) are composed of the following elements:

- Commands — manipulate text, buffers, windows, etc... within the editor
- Directives — control the flow of execution within a macro
- Arguments:
  - Constants
  - Variables
  - Functions
- Comments

Macros are registered with MicroEMACS by the store-macro or store-procedure commands. They get executed through menus or keystrokes they have been bound to, or through the execute-macro-*n* or run commands.

Macros can also be executed directly from a buffer or a file by the execute-buffer or execute-file commands.

Commands

By topic:

Alphabetical lists:

Binding commands

Block of Text commands

Commands that affect regions, lines, words and paragraphs.

- case-region-lower
- case-region-upper
- case-word-capitalize
- case-word-lower
- case-word-upper
- copy-region
- count-words
- delete-blank-lines
- delete-next-word
- delete-previous-word
- detab-region
- entab-region
- fill-paragraph
- indent-region
- kill-paragraph
- kill-region
- kill-to-end-of-line
- narrow-to-region
- remove-mark
- set-fill-column
- set-mark
- trim-region
- undent-region
- widen-from-region
- wrap-word

Buffer, Window and Screen commands

add-global-mode
add-mode
cascade-screens
change-screen-column
change-screen-row
change-screen-size
change-screen-width
clear-and-redraw
cycle-screens
delete-buffer
delete-global-mode
delete-other-windows
delete-mode
delete-screen
delete-window
execute-buffer
filter-buffer
find-screen
grow-window
list-buffers
list-screens
maximize-screen
minimize-screen
move-window-down
move-window-up
name-buffer
narrow-to-region
next-buffer
next-window
pipe-command
pop-buffer
previous-window
rename-screen
resize-window
restore-screen
restore-window
save-window
scroll-next-up
scroll-next-down
select-buffer
shrink-window
split-current-window
tile-screens
unmark-buffer
update-screen
widen-from-region

Clipboard and Kill Buffer commands

clip-region
copy-region
cut-region
cycle-ring
delete-kill-ring
delete-next-character (with argument)
delete-next-word
delete-previous-character (with argument)
delete-previous-word
insert-clip
kill-paragraph
kill-region
kill-to-end-of-line
yank
yank-pop

Execution, Macro and Variable commands

- abort-command
- begin-macro
- describe-functions
- describe-variables
- display
- end-macro
- execute-buffer
- execute-command-line
- execute-file
- execute-macro
- execute-macro-*n*
- execute-named-command
- execute-procedure
- execute-program
- filter-buffer
- i-shell
- nop
- pipe-command
- run
- set
- shell-command
- source
- store-macro
- store-procedure
- help-engine

File Commands

- append-file
- change-file-name
- execute-file
- find-file
- insert-file
- read-file
- save-file
- show-files
- source
- view-file
- write-file

Mouse commands

- mouse-move
- mouse-move-down
- mouse-move-up
- mouse-region-down
- mouse-region-up
- mouse-resize-screen

Positioning commands

backward-character
beginning-of-file
beginning-of-line
buffer-position
end-of-file
end-of-line
end-of-word
exchange-point-and-mark
forward-character
goto-line
goto-mark
goto-matching-fence
next-line
next-page
next-paragraph
next-word
previous-line
previous-page
previous-paragraph
previous-word
redraw-display

Search and Replace commands

- hunt-backward
- hunt-forward
- incremental-search
- query-replace-string
- replace-string
- reverse-incremental-search
- search-forward
- search-reverse

Miscellaneous Commands

clear-message-line
exit-emacs
handle-tab
help
insert-space
insert-string
newline
newline-and-indent
nop
open-line
overwrite-string
print
quick-exit
quote-character
redraw-display
set-encryption-key
set-fill-column
transpose-characters
universal-argument
write-message

Standard commands

The following commands are available in all implementations of MicroEMACS:

abort-command	Allows the user to abort out of any command that is waiting for input
add-global-mode	Add a global mode for all new buffers
add-mode	Add a mode to the current buffer
append-file	Append a buffer to the end of a file
apropos	Lists commands and macros whose name contains the string specified
backward-character	Move one character to the left
begin-macro	Begin recording a keyboard macro
beginning-of-file	Move to the beginning of the file in the current buffer
beginning-of-line	Move to the beginning of the current line
bind-to-key	Bind a key to a command
buffer-position	List the position of the point on the message line
case-region-lower	Make a region all lower case
case-region-upper	Make a region all upper case
case-word-capitalize	Capitalize the following word
case-word-lower	Lower case the following word
case-word-upper	Upper case the following word
change-file-name	Change the name of the file in the current buffer
change-screen-column	change the column offset of the current screen
change-screen-row	change the row offset of the current screen
change-screen-size	Change the number of lines of the current screen
change-screen-width	Change the number of columns of the current screen
clear-and-redraw	Repaint all screens or center the point in the current window
clear-message-line	Clear the message line
copy-region	Copy the current region into the kill buffer
count-words	Count how many words, lines and characters are in the current region
ctlx-prefix	Bound to the key used as the ^X prefix
cycle-ring	moves the current position of the kill buffer within the kill ring
cycle-screens	Bring the rearmost screen to front
delete-blank-lines	Delete all blank lines around the point
delete-buffer	Delete a buffer which is not being currently displayed in a window
delete-kill-ring	Reclaim the memory used by the kill ring
delete-global-mode	Turn off a global mode
delete-mode	Turn off a mode in the current buffer
delete-next-character	Delete the character following the point
delete-next-word	Delete the word following the point

delete-other-windows	Make the current window cover the entire screen
delete-previous-character	Delete the character to the left of the point
delete-previous-word	Delete the word to the left of the point
delete-screen	Delete a screen (not the top one)
delete-window	Remove the current window from the screen
describe-bindings	List all commands and macros
describe-functions	List all functions
describe-variables	List all variables
describe-key	Describe what command or macro is bound to a keystroke sequence
detab-region	Change all tabs in a region to the equivalent spaces
display	Displays a variable's current value
end-macro	Stop recording a keyboard macro
end-of-file	Move to the end of the current buffer
end-of-line	Move to the end of the current line
end-of-word	Move just past the end of the current word
entab-region	Change multiple spaces to tabs where possible
exchange-point-and-mark	Move the point to the last marked spot, make the original position be marked
execute-buffer	Execute a buffer as a macro
execute-command-line	Execute a line typed on the command line as a macro
execute-file	Execute a file as a macro
execute-macro	Execute the keyboard macro (play back the recorded keystrokes)
execute-macro-*n*	Execute numbered macro *n* where *n* is an integer from 1 to 40
execute-named-command	Execute a command by name
execute-procedure	Execute a procedure by name
execute-program	Execute a program directly (not through an intervening shell)
exit-emacs	Exit MicroEMACS. If there are unwritten, changed buffers MicroEMACS will ask to confirm
fill-paragraph	Fill the current paragraph
filter-buffer	Filter the current buffer through an external filter
find-file	Find a file to edit in the current window
find-screen	Bring the named screen on top, creating it if needed
forward-character	Move one character to the right
goto-line	Goto a numbered line
goto-mark	Goto a numbered mark
goto-matching-fence	Goto the matching fence
grow-window	Make the current window larger
handle-tab	Insert a tab or set tab stops

hunt-backward	Hunt for the last match of the last search string
hunt-forward	Hunt for the next match of the last search string
help	Read EMACS.HLP into a buffer and display it
i-shell	Shell up to a new command processor
incremental-search	Search for a string, incrementally
indent-region	Indent the current region one tab
insert-file	Insert a file at the point in the current file
insert-space	Insert a space to the right of the point
insert-string	Insert a string at the point
kill-paragraph	Delete the current paragraph
kill-region	Delete the current region, moving it to the kill buffer
kill-to-end-of-line	Delete the rest of the current line
list-buffers	List all existing buffers
list-screens	List all existing screens
macro-to-key	Bind a key to a macro
meta-prefix	Key used to precede all META commands
mouse-move	Usually bound to the movement of the mouse
mouse-move-down	Usually bound to a press on the left mouse button
mouse-move-up	Usually bound to the release of the left mouse button
mouse-region-down	Usually bound to a press on the right mouse button
mouse-region-up	Usually bound to the release of the right mouse button
mouse-resize-screen	Resize the screen to bring the bottom-left corner where the mouse was clicked
move-window-down	Scroll the current window down
move-window-up	Scroll the current window up
name-buffer	Change the name of the current buffer
narrow-to-region	Hides all text not in the current region (see widen-from-region)
newline	Insert a newline
newline-and-indent	Insert a newline and indent the new line the same as the preceding line
next-buffer	Bring the next buffer in the list into the current window
next-line	Move down one line
next-page	Move down one page
next-paragraph	Move to the next paragraph
next-window	Move to the next window
next-word	Move to the beginning of the next word
nop	Does nothing
open-line	Open a line at the point
overwrite-string	Overwrite a string at the point

pipe-command	Execute an external command and place its output in a buffer
pop-buffer	Display a buffer temporarily, paging through it
previous-line	Move up one line
previous-page	Move up one page
previous-paragraph	Move back one paragraph
previous-window	Move to the last window
previous-word	Move to the beginning of the word to the left of the point
print	Display a string on the message line (synonym of write-message)
query-replace-string	Replace occurrences of a string with another string, interactively querying the user
quick-exit	Exit MicroEMACS, writing out all the changed buffers
quote-character	Insert the next character literally
read-file	Read a file into the current buffer
redraw-display	Reposition the current line in the window
remove-mark	Remove a numbered mark
replace-string	Replace all occurrences of a string with another string
resize-window	Change the number of lines in the current window
restore-window	Move to the last saved window (see save-window)
reverse-incremental-search	Search backwards, incrementally
run	Execute a named procedure
save-file	Save the current buffer if it is changed
save-window	Remember the current window (see restore-window)
scroll-next-up	Scroll the next window up
scroll-next-down	Scroll the next window down
search-forward	Search for a string
search-reverse	Search backwards for a string
select-buffer	Select a buffer to display in the current window
set	Set a variable to a value
set-encryption-key	Set the encryption key of the current buffer
set-fill-column	Set the current fill column
set-mark	Set a numbered mark
shell-command	Causes an external shell to execute a command
show-files	list files matching a pattern within a directory
shrink-window	Make the current window smaller
source	Execute a file as a macro
split-current-window	Split the current window in two
store-macro	Store the following macro lines as a numbered macro
store-procedure	Store the following macro lines in a named procedure

transpose-characters	Transpose the character at the point with the character immediately to the left
trim-region	Trim any trailing white space from a region
unbind-key	Unbind a key from a command or macro
undent-region	Remove a leading indent from a region
universal-argument	Execute the following command or macro 4 times
unmark-buffer	Unmark the current buffer (so it is no longer seen as changed)
update-screen	Force a display update during macro execution
view-file	Read a file in a buffer, in view mode
widen-from-region	Restores hidden text (see narrow-to-region)
wrap-word	Wrap the current word (internal command)
write-file	Write the current buffer under a new file name
write-message	Display a string on the message line
yank	Yank the kill buffer into the current buffer at the point
yank-pop	yank the kill buffer, subsequent invocations replacing the yanked text by the next one from the kill ring.

Additional commands

The following commands are available only from the Microsoft Windows version of MicroEMACS:

bind-to-menu	creates a menu item and binds it to a command
cascade-screens	arranges all non-iconic screens using a cascading scheme
clip-region	copies the region to the Windows clipboard
cut-region	moves the region to the Windows clipboard
help-engine	invokes the Microsoft Windows help engine
insert-clip	inserts the contents of the Windows clipboard at the point
macro-to-menu	creates a menu item and binds it to a macro
maximize-screen	makes the current screen occupy the whole MicroEMACS window
minimize-screen	iconizes the current screen
rename-screen	change the current screen's name
restore-screen	restores the current screen back from maximized or iconized state
tile-screens	arranges all non-iconic screens using a tiling scheme
unbind-menu	deletes a menu item

Directives

Directives are used within macros to control what lines are executed and in what order.

Directives always start with the exclamation mark "!" character and must be the first non-white text placed on a line. They are:

- !BREAK
- !ENDM
- !FORCE
- !GOTO
- !IF, !ELSE and !ENDIF
- !RETURN
- !WHILE and !ENDWHILE

Directives do not make sense as a single commands. As such, they cannot be called up singly or bound to keystrokes.Directives executed interactively (via the execute-command-line command) are ignored.

!BREAK

This directive lets you abort out of the most inner currently executing while loop, in a macro. It is often used to abort processing for error conditions. For example:

```
; Read in files and substitute "beginning" with "beginning"
set %filename #list
!while &not &seq %filename "<end>"

!force    find-file %filename
    !if &seq $status FALSE
        write-message "[File read error]"
        !break
    !endif
    beginning-of-file
    replace-string "beginning" "beginning"
    save-file
    set %filename #list
!endwhile
```

!ENDM

This directive is used to terminate a macro being stored. For example:

```
; Read in a file in view mode, and make the window red
store-procedure get-red-viewed-file
    view-file @"File to view: "
    add-mode "red"
!endm
```

Related commands:

store-procedure
store-macro.

!FORCE

When MicroEMACS executes a macro, if any command fails, the macro is terminated at that point. If a line is preceded by a !FORCE directive, execution continues whether the command succeeds or not.

This is often used together with the $status variable to test if a command succeeded. For example:

```
set %seekstring @"String to Find: "
!force search-forward %seekstring
!if $status
    print "Your string is Found"
!else
    print "No such string!"
!endif
```

!GOTO

The flow of execution within a MicroEMACS macro can be controlled using the !GOTO directive. It takes a label as argument. A label consists of a line starting with an asterisk "*" and then an alphanumeric label. Only labels in the currently executing macro can be jumped to, and trying to jump to a non-existing label terminates execution of a macro. For example:

```
; Create a block of DATA statements for a BASIC program
insert-string "1000 DATA "
set %linenum 1000
*nxtin
update-screen    ;make sure we see the changes
set %data @@"Next number: "
!if &equal %data 0
    !goto finish
!endif
!if &greater $curcol 60
    2 delete-previous-character
    newline
    set %linenum &add %linenum 10
    insert-string &cat %linenum " DATA "
!endif
insert-string &cat %data ", "
!goto nxtin
*finish
2 delete-previous-character
newline
```

Note that loops constructed with !WHILE are usually more efficient than those constructed purely by ! GOTOs.

!IF, !ELSE and !ENDIF

The !IF directive allows for conditional execution within a macro.

Lines following the !IF directive, until the corresponding !ELSE or !ENDIF, are executed only if the expression within the !IF line evaluates to a TRUE value. Lines following an !ELSE directive, until the corresponding !ENDIF, are executed only if the expression within the corresponding !IF line did not evaluate to a TRUE value.

For example, the following macro creates the portion of a text file automatically:

```
!if &sequal %curplace "timespace vortex"
    insert-string "First, rematerialize~n"
!endif
!if &sequal %planet "earth"    ;If we have landed on earth...
    !if &sequal %time "late 20th century"    ;and we are then
        write-message "Contact U.N.I.T."
    !else
        insert-string "Investigate the situation....~n"
        insert-string "(SAY 'stay here Sarah)~n"
    !endif
!else
    set %conditions @"Atmosphere conditions outside? "
    !if &sequal %conditions "safe"
        insert-string &cat "Go outside......" "~n"
        insert-string "lock the door~n"
    !else
        insert-string "Dematerialize..try somewhen else"
        newline
    !endif
!endif
```

!RETURN

This directive causes the current macro to exit, either returning to the caller (if any) or to interactive mode. For example:

```
; Check the display type and set %wintyp
!if &sequal $sres "MSWIN"
    set %wintyp 1
    !return
!endif
set %wintyp 0
write-message "You are not running under MS-Windows!"
!return
```

!WHILE and !ENDWHILE

This pair of directives facilitates repetitive execution within a macro. If a group of statements needs to be executed while a certain expression evaluates to TRUE, enclose them with a while loop. For example:

```
!while &less $curcol 70
    insert-string &cat &cat "[" #stuff "]"
!endwhile
```

While loops may be nested and can contain and be the targets of !GOTOs with no ill effects. Using a while loop to enclose a repeated task will run much faster than the corresponding construct using !IFs.

Arguments

In the MicroEMACS macro language, commands and functions often require arguments. There are three types of arguments and they are automatically converted to the proper type when used:

Numerical	An ASCII string of digits which is interpreted as a numeric value. Any string which does not start with a digit or a minus sign "**-**" will be considered zero.
String	An arbitrary string of characters. Strings are limited to 128 characters in length.
Boolean	A logical value consisting of the string "TRUE" or "FALSE". Numeric strings will also evaluate to "FALSE" if they are equal to zero, and "TRUE" if they are non-zero. Arbitrary text strings will be considered equivalent "FALSE".

While arguments usually follow the command or function that uses them, a single numerical argument can also be placed in front of a command, producing an effect similar to the numeric arguments used in interactive mode.

If a command needs more arguments than have be supplied on the line, the command fails.

Constants

Wherever macro language statements need to have arguments, it is legal to place constants. A constant is a double quote character, followed by a string of characters, and terminated by another double quote character.

The double quotes around constants are not needed if the constant contains no white space and it also does not happen to meet the rules for any other MicroEMACS commands, directives, variables, or functions. This is very practical for numeric constants.

To represent various special characters within a constant, the tilde "~" character is used. The character following the tilde is interpreted according to the following table:

Sequence	Meaning
~"	double quote
~~	tilde
~b	backspace (^H)
~f	formfeed (^L)
~l	linefeed (^J)
~n	newline
~r	carriage return (^M)
~t	tab (^I)

Any character not in the above table which follows a tilde will be passed unmodified. This action is similar to the quote-character (^Q) command available from the keyboard.

MicroEMACS may use different characters for line terminators on different computers. The "**~n**" combination will always get the proper line terminating sequence for the current system.

Variables

Variables are part of the MicroEMACS Macro language. They can be used wherever an argument (number, boolean or string) is needed.

Environmental variables both control and report on different aspects of the editor. User variables hold values which may be changed and inspected. Buffer variables allow lines from buffers to be used as values. Interactive variables allow macros to prompt the user for information.

Buffer Variables

Buffer variables are a way to take a line of text from a buffer and place it in a variable. They can only be queried and cannot be set. A buffer variable consists of the buffer name, preceded by a pound sign "#". Its value is the text between the point and the end of the line. Each use of a buffer variable advances the point to the beginning of the following line.

For example, if you have a buffer by the name of RIGEL2, and it contains the text (the point being on the "B" of "Bloomington"):

```
Richmond
Lafayette
Bloomington
Indianapolis
Gary
```

and within a command you reference #rigel2, like in:

```
insert-string #rigel2
```

MicroEMACS would start at the current point in the RIGEL2 buffer and grab all the text up to the end of that line and pass that back. Then it would advance the point to the beginning of the next line. Thus, after the insert-string command executes, the string "Bloomington" gets inserted into the current buffer, and the buffer RIGEL2 now looks like this (the point is on the "I" of "Indianapolis"):

```
Richmond
Lafayette
Bloomington
Indianapolis
Gary
```

When the end of a buffer variable is reached, the value returned is: `<END>`

Environmental Variables

These variables are used to change or get information about various aspects of the editor. They return a current setting if used as part of an expression. All environmental variable names begin with a dollar sign "$" and are in lower case:

$acount	Countdown until next auto-save
$asave	Auto-save frequency
$bufhook	Command/macro run when entering a buffer
$cbflags	Buffer attribute flags.
$cbufname	Buffer name
$cfname	File name
$cmdhook	Command/macro run before each keystroke
$cmode	Buffer modes
$curchar	ASCII value of character
$curcol	Current column
$curline	Current line
$curwidth	Number of columns
$curwind	Window index
$cwline	Line number in current window
$debug	Macro debugging flag
$deskcolor	Color for desktop
$diagflag	Diagonal dragging flag
$discmd	Prompt echo flag
$disinp	Input echo flag
$disphigh	High-bit characters display flag
$exbhook	Command/macro run when leaving a buffer.
$fcol	Line number at top of window
$fillcol	Fill column.
$flicker	Flicker flag (for CGA or animated grinder cursor)
$fmtlead	Text formatter command prefixes
$gflags	Global flags
$gmode	Global mode flags
$hardtab	Size of hard tabs
$hilight	Region to be highlighted
$hjump	Horizontal scrolling quantum
$hscroll	Horizontal scrolling flag
$hscrlbar	Horizontal scroll bar flag
$isterm	Incremental search string terminator key
$kill	Kill buffer contents
$language	National language used by MicroEMACS
$lastkey	Last keyboard character
$lastmesg	Last message
$line	Current line contents
$lterm	Line terminator string
$lwidth	Width of current line
$match	Last string matched in a search
$mmove	Controls the generation of mouse movements
$modeflag	Mode line display flag
$msflag	Mouse flag
$numwind	Number of windows
$oldcrypt	Encryption method flag
$orgrow	Row of current screen within desktop
$orgcol	Column of current screen within desktop
$os	Operating system (MSWIN under MS-Windows)
$overlap	Size of overlap during paging
$pagelen	Number of lines in screen

$palette	Color palette settings
$paralead	Paragraph start characters
$pending	Keystrokes pending flag
$popflag	Popup buffer flag
$posflag	Row&column display flag
$progname	"MicroEMACS"
$readhook	Command/macro run when a file is read
$region	Contents of current region
$replace	Default replace string.
$rval	Exit value from last invoked subprocess
$scrname	Screen name
$search	Default search string
$searchpnt	After-search-positioning flag
$seed	Random number generator seed
$softtab	Tab size for handle-tab command
$sres	Display resolution (MSWIN under MS-Windows)
$ssave	Safe-save flag
$sscroll	Smooth scroll flag
$status	Status from last command
$sterm	Search string terminator key
$target	Target for line moves
$time	Date and time
$timeflag	Time display flag
$tpause	Duration of fence matching pause
$version	MicroEMACS version
$vscrlbar	Vertical scroll bar flag
$wchars	List of characters that can be part of a word
$wline	Window height (lines)
$wraphook	Command/macro run when wrapping text
$writehook	Command/macro run when writing a file
$xpos	Column the mouse was in at last click
$yankflag	After-yank-positioning flag
$ypos	Line the mouse was in at last click

$acount

This variable is used in ASAVE mode. It contains the countdown on inserted character until the next auto-save. When it reaches zero, it is reset to the value of $asave.

Initial value: 256

$asave

This variable is used in ASAVE mode. It specifies the value used to reset $acount after an automatic save occurs.

Default value: 256

$bufhook

The command or macro named in this variable is run when a buffer is entered. This can be used to implement modes which are specific to a particular file or file type.

Default value: nop

$cbflags

This variable contains the current buffer's attribute flags, encoded as the sum of the following numbers:

1	Internal invisible buffer
2	Changed since last read or write
4	Buffer was truncated when read (due to lack of memory)
8	Buffer has been narrowed

Only the invisible (1) and changed (2) flags can be modified by setting $cbflags. The truncated file (4) and narrowed (8) flags are read-only.

$cbufname

This variable contains the name of the current buffer.

$cfname

This variable contains the file name associated to the current buffer.

$cmdhook

This variable contains the name of a command or macro to run before accepting a keystroke. This is by default set to the nop command.

Default value: nop

$cmode and **$gmode**

The two variables $cmode and $gmode contain a number that corresponds to the modes for the current buffer ($cmode) and the new buffers ($gmode). They are encoded as the sum of the following numbers for each of the possible modes:

WRAP	1	Word wrap
CMODE	2	C indentation and fence matching
SPELL	4	Interactive spell checking (Not implemented yet)
EXACT	8	Exact matching for searches
VIEW	16	Read-only buffer
OVER	32	Overwrite mode
MAGIC	64	Regular expressions in search
CRYPT	128	Encryption mode active
ASAVE	256	Auto-save mode

Thus, if you wished to set the current buffer to have CMODE, EXACT, and MAGIC on, and all the others off, you would add up the values for those three, CMODE 2 + EXACT 8 + MAGIC 64 = 74, and use a statement like:

```
set  $cmode  74
```

or, use the binary or operator to combine the different modes:

```
set  $cmode  &bor  &bor  2  8  64
```

Alternatively, you can also modify the modes one by one, using the add-mode and add-global-mode or delete-mode and delete-global-mode commands

$curchar

This variable contains the ASCII value of the character currently at the point.

$curcol

This variable contains the column (starting at 0) of the point in the current buffer.

$curline

This variable contains the line number (starting at 1) of the point in the current buffer.

$curwidth

This variable contains the number of columns displayed in the current screen.

Setting this variable is equivalent to using the change-screen-width command with a numeric argument.

$curwind

This variable contains the index of the current window within the screen. Windows are numbered from top to bottom, starting at 1. The number of windows within the current screen is held by the $numwind variable.

$cwline

This variable contains the number of lines displayed in the current window.

$debug

This boolean variable contains a flag used to trigger macro debugging. If it is set to TRUE, macros are executed step by step, and each statement and variable assignment is displayed on the message line.

Default value: FALSE

$deskcolor

This variable contains the color to use for the desktop. In the MS-Windows version of MicroEMACS, the value of this variable is irrelevant.

Default value: BLACK.

$diagflag

If this boolean variable is set to TRUE, diagonal dragging of text and mode lines is enabled. If it is FALSE, text and modelines can either be dragged horizontally or vertically but not both at the same time.

$discmd

If this boolean variable is set to TRUE, the echoing of command prompts and output on the message line is enabled. If it is FALSE, most messages and prompts are disabled (this is handy to avoid some cases of message line flashing while a macro is running).

Default value: TRUE.

$disinp

If this boolean variable is set to TRUE, the echoing of input at the command prompts is enabled.

Default value: TRUE.

$disphigh

If this boolean variable is set to TRUE, high-bit characters (single byte characters that are greater than 127 in value) will be displayed in a pseudo-control format. The characters "**^!**" will lead off the sequence, followed by the character stripped of its high bit.

Default value: FALSE.

$exbhook

This variable holds the name of a command or macro which is run whenever you are switching out of a buffer.

Default value: nop

$fcol

This variable contains the line position being displayed in the first column of the current window.

$fillcol

This variable contains the current fill column. It is used by the fill-paragraph command. It can be set through the set command or by using the set-fill-column command.

Default value: 72

$flicker

In the MS-DOS version of MicroEMACS, this variable contains a flicker flag that should be set to TRUE if the display is an IBM CGA and set to FALSE for most others.

In the MS-Windows version of MicroEMACS, this variable can be set to FALSE to allow an animated grinder to be displayed in place of the hourglass mouse cursor. Since this animation results, on many video displays, in an annoying flicker of the cursor, it is disabled when $flicker is set to TRUE.

Default value: TRUE

$fmtlead

A line starting with one of the characters in the $fmtlead variable is considered to be a text formatter command. Therefore, the following line is considered to be the start of a paragraph.

If you are editing text destined for use by a text formatter, set $fmtlead to the command character for that formatter. That will prevent MicroEMACS from formatting what should be lines of commands meant for the formatter. If, for example, you are editing SCRIBE source, use the set (^XA) command to set $fmtlead to "**@**".

Default value: empty string

$gflags

Some of the ways MicroEMACS controls its internal functions can be modified by the value in the $gflags variable. Each bit in this variable will be used to control a different function:

1	If this bit is set to zero, EMACS will not automatically switch to the buffer of the first file after executing the startup macros.
2	If this bit is set to one, suppress redraw events.

$hardtab

This variable contains the number of spaces between hard tab stops. This can be used to change the way tabs are displayed within the editor.

Default value: 8

$hilight

When this variable contains a value *n* between 0 and 14, it indicates that the text located between the marks *n* and *n*+1 should be highlighted. A value of 255 indicates that no highlighting is performed.

Default value: 10

$hjump

This variable contains the number of columns the editor should scroll the screen horizontally when a horizontal scroll is required.

Default value: 1

$hscroll

This variable is a flag that determines if MicroEMACS will scroll the entire window horizontally, or just the current line. The default value, TRUE, results in the entire window being shifted left or right when the cursor goes off the edge of the screen.

$hscrlbar

This boolean variable exists only under the MS-Windows version of MicroEMACS. If it is TRUE, an horizontal scroll bar is available at the bottom of each screen, allowing you to scroll the text in the current window right and left.

If $hscrlbar is FALSE, the horizontal scroll bar is not present.

Default value: TRUE

$isterm

This variable contains the character used to terminate incremental search string inputs.

Default value: the last key bound to meta-prefix (initially: Escape character)

$kill

This variable contains the first 127 characters currently in the kill buffer.

Attempts to set this variable are ignored.

$language

This variable contains the name of the national language in which MicroEMACS messages will be displayed. (Currently MicroEMACS is available in English, French, Spanish, Latin, Portuguese, Dutch, German, and Pig Latin).

The MS-Windows version of MicroEMACS is currently available in English only.

Attempts to set this variable are ignored. Changing the language used by MicroEMACS requires recompiling.

$lastkey

This variable contains a number representing the ASCII value of the last key press processed by MicroEMACS. This variable does not contain any indication that the last keystroke was prefixed by the Meta or the **Alt** keys. Further more, function or special keys are perceived as the last character of their keystroke representation.

Note that this variable does not change during playback of a keyboard macro.

Setting this variable does not have any effect on the editor beyond changing the variable's value.

$lastmesg

This variable contains the text of the last message which MicroEMACS wrote on the message line. Setting this variable does not have any effect on the editor beyond changing the variable's value.

$line

This variable contains the first 127 characters of the current line. Setting this variable overwrites the contents of the current line.

$/term

This variable contains the string of characters to use as a line terminator when writing a file to disk. By default, it is an empty string, which causes a newline to be written (under MS-DOS or MS-Windows, this translates into a carriage return character followed by a line feed character).

Under some operating systems, the value of this variable is irrelevant.

$lwidth

This variable contains the number of characters of the current line.

Attempts to set this variable are ignored.

$match

This variable contains the last string matched by a search operation.

Attempts to set this variable are ignored.

$mmove

If this variable it is equal to 2, any mouse movement results in a mouse action (MSm, S-MSm or MS^m).

If this variable is set to 1, some mouse movement that are of marginal interest (like while a popup buffer is being displayed or, under MS-Windows, while no mouse button is pressed) are ignored.

If $mmove is set to 0, all mouse movements are ignored.

Default value: 1

$modeflag

If this boolean variable is TRUE, mode lines are visible. If it is FALSE, mode lines are not displayed (thus allowing one more line per window).

Default value: TRUE

$msflag

Under some operating systems, this boolean variable can be used to control the use of the pointing device: when it is TRUE, the mouse (if present) is active. When it is FALSE, the mouse cursor is not displayed, and mouse actions are ignored.

Under MS-Windows, setting this variable to FALSE does not cause the cursor to be hidden, but mouse actions within text areas are ignored. However, the mouse remains useable to activate menus or select, move and resize screens.

Default value: TRUE

$numwind

This variable contains the number of windows displayed within the current screen.

Attempts to set this variable are ignored.

$oldcrypt

If this boolean variable is TRUE, the CRYPT mode uses the old method of encryption (which had a bug in it). This allows you to read files that were encrypted with a previous version of MicroEMACS.

Default value: FALSE.

$orgrow

This variable contains the position of the current screen's top row on the desktop, starting at 0.

Setting this variable is equivalent to invoking the change-screen-row command.

Under MS-Windows, the value of this variable is irrelevant.

Default value: 0

$os

This variable contains a string that identifies the operating system. It is set to MSWIN in the Microsoft Windows version of MicroEMACS.

Attempts to set this variable are ignored.

$orgcol

This variable contains the position of the current screen's left column on the desktop, starting at 0.

Setting this variable is equivalent to invoking the change-screen-column command.

Under MS-Windows, the value of this variable is irrelevant.

Default value: 0

$overlap

This variable contains the amount of overlapping, in number of lines, used when paging up and down (using the next-page and previous-page commands).

Default value: 2

$pagelen

This variable contains the number of lines (including mode lines) displayed by the current screen.

Setting this variable is equivalent to invoking the change-screen-size command with a numeric argument.

$palette

This variable contains a string that is used to control the color palette settings on graphics versions of MicroEMACS.

Under MS-Windows, $palette is composed of up to 48 octal digits. Each group of three digits redefines an entry of the palette, by specifying the red, green and blue levels of that color.

Default value: empty string

$paralead

A line starting with one of the characters in the $paralead variable is considered to be the first line of a paragraph.

Default value: Space and TAB characters

$pending

This boolean variable is TRUE if there are type ahead keystrokes waiting to be processed.

Attempts to set this variable are ignored.

$popflag

If this boolean variable is TRUE, popup buffers are used instead of opening a window for building completion lists and by the following commands:

- apropos
- describe-bindings
- describe-functions
- describe-variables
- list-buffers
- list-screens
- show-files

Default value: TRUE

$posflag

If this boolean variable is TRUE, the position of the point (row and column) is displayed in the current window's mode line.

Default value: FALSE

$progname

This variable contains the string "MicroEMACS" for standard MicroEMACS. It can be something else if MicroEMACS is incorporated as part of someone else's program.

Attempts to set this variable are ignored. Changing it requires recompiling.

$readhook

The command or macro named in this variable is run when a file is read into a buffer. This can be used to implement modes which are specific to a particular file or file type.

Default value: nop

$region

This variable contains the first 255 characters of the current region. If the region is not defined (because the mark is not set), this variable contains the string: "ERROR".

Attempts to set this variable are ignored.

$replace

This variable contains the current default replace string. That is the replace string that was specified in the last replace-string or query-replace-string command and will be used as default value for the next such command.

$rval

This variable contains the returned value from the last subprocess which was invoked from MicroEMACS's commands: execute-program, filter-buffer, i-shell, pipe-command.and shell-command.

Under MS-Windows, this variable always has the value 0.

Attempts to set this variable are ignored.

$scrname

This variable contains the current screen's name.

Setting this variable causes the specified screen to be made the current one. If that screen does not exist, nothing happens. To change the name of a screen, use the rename-screen command.

$search

This variable contains the current default search string. That is the search string that was specified in the last search-forward, search-reverse, incremental-search, reverse-incremental-search, replace-string or query-replace-string command and will be used as default value for the next such command or as the target for hunt-forward and hunt-backward.

$searchpnt

The value of this variable specifies the positioning of the of the point at the end of a successful search:

- If $searchpnt = 0, the cursor is placed at the end of the matched text on forward searches, and at the beginning of this text on reverse searches.
- If $searchpnt = 1, the cursor is placed at the beginning of the matched text regardless of the search direction.
- If $searchpnt = 2, the cursor is placed at the end of the matched text regardless of the search direction.

Setting this variable to a value other than one of the above causes the value 0 to be used.

Default value: 0

$seed

This variable contains the integer seed of the random number generator. This is used by the &rnd function and also to compute temporary file names (if $ssave is TRUE).

Initial value: 0

$softtab

The value of this variable relates to the number of spaces inserted by MicroEMACS when the handle-tab command (which is normally bound to the TAB key) is invoked:

> If $softtab is *n*, strictly positive, tabs stops are located at every *n*th column and the handle-tab command inserts space characters in sufficient number to move the point to the next tab stop.
>
> If $softtab is zero, the handle-tab command inserts true tab characters.
>
> If $softtab is strictly negative, the handle-tab command fails.

This variable can be set by passing a numeric argument to handle-tab or by directly using the set command.

Default value: 0

$sres

This variable contains a string that identifies the current screen resolution (CGA, MONO, EGA or VGA on the IBM-PC, LOW, MEDIUM, HIGH or DENSE on the Atari ST1040, MSWIN under Microsoft Windows and NORMAL on most others).

Depending on the hardware and operating system MicroEMACS is running on, setting this variable may allow you to change the screen resolution. Not that under MS-Windows, attempts to set this variable are ignored.

$ssave

If this boolean variable is TRUE, MicroEMACS perform "safe saves": when it is asked to save the current buffer to disk, it writes it out to a temporary file, deletes the original file, and then renames the temporary to the old file name.

If $ssave is FALSE, MicroEMACS performs saves by directly overwriting the original file, thus risking loss of data if a system crash occurs before the end of the save operation. On the other hand, this mode insures that the original file attributes (ownership and access rights) are preserved on systems that support these (like UNIX).

Default value: TRUE.

$sscroll

If this boolean variable is TRUE, MicroEMACS is configured for smooth vertical scrolling: when the cursor moves off the top or bottom of the current window, the window's contents scroll up or down one line at a time.

If $sscroll is FALSE, scrolling occurs by half pages.

Default value: FALSE

$status

This boolean variable contains the status returned by the last command. This is usually used with the ! FORCE directive to check on the success of a search, or a file operation.

Setting this variable can be used to return a FALSE status from a macro.

$sterm

This variable contains the character used to terminate search string inputs.

Default value: the last key bound to meta-prefix (initially: Escape character)

$target

This variable contains the column position where the point will attempt to move after a next-line or previous-line command. Unless the previous command was next-line or previous-line, the default value for this variable is the current column.

$time

This variable contains a string corresponding to the current date and time. Usually this is given in a form like to "Mon May 09 10:10:58 1988". Not all operating systems support this.

$timeflag

If this boolean variable is TRUE, the current time is displayed on the bottom mode line of each screen.

Default value: FALSE.

Note: Under MS-Windows, this feature currently does not operate properly because MicroEMACS makes incorrect assumptions about the format of the time string (see $time).

$tpause

This variable contains the length of the pause used to show a matched fence when the current buffer is in CMODE and a closing fence (a character among "**)}]**") has been typed.

On most systems, this pause is performed by a CPU loop and therefore, the value of $tpause may need to be adjusted to compensate for the processor's speed.

Under MS-Windows, the pause is performed by a bona-fide timer and $tpause is expressed in milliseconds. The default value is 1000.

$version

This variable contains the current MicroEMACS version number (i.e. "3.11c").

Attempts to set this variable are ignored.

$vscrlbar

This boolean variable exists only under the MS-Windows version of MicroEMACS. If it is TRUE, a vertical scroll bar is available at the right end of each screen, allowing you to scroll the text in the current window up and down.

If $vscrlbar is FALSE, the vertical scroll bar is not present.

Default value: TRUE

$wchars

This variable is used to define what a word is for MicroEMACS. It contains the list of all the characters that can be considered part of a word.

If $wchar is empty, a word is defined as composed of upper and lower case letters, numerals (0 to 9) and the underscore character.

Default value: empty

$wline

This variable contains the number of lines displayed in the current window, excluding the mode line. Setting this variable is equivalent to using the resize-window command with a numeric argument.

$wraphook

This variable contains the name of a command or macro which is executed when a buffer is in WRAP mode and it is time to wrap the current line.

Default value: wrap-word

$writehook

This variable contains the name of a command or macro which is invoked whenever MicroEMACS attempts to write a file out to disk. This is executed before the file is written, allowing you to process a file on the way out.

Default value: nop

$xpos

This variable contains the horizontal screen coordinate where the mouse was located the last time a mouse button was pressed or released.

The leftmost column is considered to be 0 in screen coordinates.

$yankflag

This boolean variable controls the placement of the point after a yank, yank-pop, insert-file or insert-clip command.

If $yankflag is FALSE, the point is moved to the end of the yanked or inserted text.

If $yankflag is TRUE, the cursor remains at the start of the yanked or inserted text.

Default value: FALSE

$ypos

This variable contains the vertical screen coordinate where the mouse was located the last time a mouse button was pressed or released.

The top row is considered to be 0 in screen coordinates.

Interactive Variables

Interactive variables are actually a method to prompt the user for a string. This is done by using an at sign "@" followed with a string argument. The string is displayed on the message line, and the editor waits for the user to type in a string which is then returned as the value of the interactive variable. For example:

```
find-file @"What file? "
```

will ask the user for a file name, and then attempt to find it. Note also that complex expressions can be built up with these operators, such as:

```
set %default "file1"
@&cat &cat "File to decode[" %default "]: "
```

which prompts the user with the string:

```
File to decode[file1]:
```

User Variables

User variables allow you to store strings and manipulate them. These strings can be pieces of text, numbers (in text form), or the logical values TRUE and FALSE. These variables can be combined, tested, inserted into buffers, and otherwise used to control the way your macros execute. Up to 512 user variables may be in use in one editing session. All user variable names must begin with a percent sign "**%**" and may contain any printing character. Only the first 10 characters are significant (i.e. differences beyond the tenth character are ignored).

When a user variable has not been set, it has the value: "ERROR".

Functions

Functions are part of the MicroEMACS Macro language. They can be used wherever an argument (number, string or boolean) is needed.

Function names always begin with the ampersand "&" character, and only the first three characters after the ampersand are significant. Functions are always used in lower case.

Functions can be used to act on variables in various ways. Functions can have one, two, or three arguments. These are always placed after the function, and they can include functions (with their own arguments).

By topic:

Boolean functions
Numeric functions
String functions
Miscellaneous functions

By returned value:

Boolean:	&and, &equal, &exist, &greater, &isnum, &less, &not, &or, &sequal, &sgreater and &sless
Numeric:	&abs, &add, &ascii, &band, &bnot, &bor, &bxor, &divide, &length, &mod, &negate, &rnd, &sindex, &sub and &times
String:	&bind, &cat, &chr, &env, &find, &group, &gtc, &gtk, &indirect, &left, &lower, &mid, &rev, &right, &slower, &supper, &trim, &upper and &xlate

Boolean Functions

These functions perform operations on boolean arguments:

&and	*log1*	*log2*	Returns TRUE if both boolean arguments are TRUE
&not	*log*		Returns the opposite boolean value
&or	*log1*	*log2*	Returns TRUE if either argument is TRUE

Numeric Functions

These functions perform operations on numerical arguments:

&abs	*num*		Returns the absolute value of *num*
&add	*num1*	*num2*	Adds two numbers
&band	*num1*	*num2*	Bitwise AND function
&bnot	*num*		Bitwise NOT function
&bor	*num1*	*num2*	Bitwise OR function
&bxor	*num1*	*num2*	Bitwise XOR function
&chr	*num*		Returns a string with the character represented by ASCII code *num*. This function is the opposite of &ascii
&divide	*num1*	*num2*	Divides *num1* by *num2*,giving an integer result
&equal	*num1*	*num2*	Returns TRUE if *num1* and *num2* are numerically equal
&greater	*num1*	*num2*	Returns TRUE if *num1* is greater than, or equal to *num2*
&isnum	*num*		Returns TRUE if the given argument is a legitimate number
&less	*num1*	*num2*	Returns TRUE if *num1* is less than *num2*
&mod	*num1*	*num2*	Returns the reminder of dividing *num1* by *num2*
&negate	*num*		Multiplies *num* by -1
&rnd	*num*		Returns a random integer between 1 and *num*
&sub	*num1*	*num2*	Subtracts *num2* from *num1*
&times	*num1*	*num2*	Multiplies *num1* by *num2*

String Functions

These functions perform operations related to strings. All of them have at least one string argument:

&ascii	*str*			Returns the ASCII code of the first character in *str*. This function is the opposite of &chr
&cat	*str1*	*str2*		Concatenates the two strings to form one
&indirect	*str*			Evaluate *str* as a variable.
&left	*str*	*num*		Returns the *num* leftmost characters from *str*
&length	*str*			Returns length of string
&lower	*str*			Transforms *str* to lowercase
&mid	*str*	*num1*	*num2*	Starting from *num1* position in *str*, returns *num2* characters
&rev	*str*			Reverses the order of characters in *str*
&right	*str*	*num*		Returns the *num* rightmost characters from *str*
&sequal	*str1*	*str2*		Returns TRUE if the two strings are the same
&sgreater				*str1* *str2* Returns TRUE if *str1* is alphabetically greater than or equal to *str2*
&sindex	*str1*	*str2*		Returns the position of *str2* within *str1*. Returns zero if not found
&sless	*str1*	*str2*		Returns TRUE if *str1* is less alphabetically than *str2*
&slower	*str1*	*str2*		Translate the first char in *str1* to the first char in *str2* when lowercasing.
&supper	*str1*	*str2*		Translate the first char in *str1* to the first char in *str2* when uppercasing.
&trim	*str*			Trims the trailing white space from a string
&upper	*str*			Transforms *str* to uppercase
&xlate	*source*	*lookup*	*trans*	Translate each character of *source* that appears in *lookup* to the corresponding character from *trans*

Miscellaneous Functions

&bind	*str*	Returns the name of the command bound to the keystroke *str*
&env	*str*	If the operating system has this capability, this returns the environment string associated with *str*
&exist	*str*	Returns TRUE if the named file *str* exists
&find	*str*	Finds the named file *str* along the path and return its full file specification or an empty string if no such file exists
&group	*num*	Return group *num* as set by a MAGIC mode search.
&gtc		Returns a string of characters containing a MicroEMACS command input from the user
&gtk		Returns a string containing a single keystroke from the user

&indirect

The &indirect function evaluates its argument, takes the resulting string, and then uses it as a variable name. For example, given the following piece of macro language:

```
; set up reference table
set %one "elephant"
set %two "giraffe"
set %three "donkey"
set %index "%two"
insert-string &ind %index
```

The string "giraffe" would have been inserted at the point in the current buffer. This indirection can be safely nested up to about 10 levels.

Comments

Within the macro language, a semicolon "**;**" signals the beginning of a comment. The text from the semicolon to the end of the line is ignored by MicroEMACS.

A comment can be the only content of a line, in which case the semicolon must be the first non-blank character on the line. A comment can also appear at the end of any statement.

Note that empty lines are legal (treated as comments).

abort-command

Default binding: ^G
This command is used interactively to abort out of any command that is waiting for input.

It can be used within a macro to sound a beep but, unless it is used with the !FORCE directive, it causes the macro to abort.

This command is unaffected by numeric arguments.

add-global-mode

Default binding: M-M

Syntax:

```
add-global-mode    mode
```

or:

```
add-global-mode    color
```

This command causes the specified mode to be inherited by future (not yet created) buffers (These global modes can later be revoked by the delete-global-mode command). It can also be used to specify the foreground or background color for future windows.
This command does not modify the modes/colors of the current buffer/window. To do so, use the add-mode command.

This command is unaffected by numeric arguments.

add-mode

Default binding: ^XM

Syntax:

```
add-mode  mode
```

or:

```
add-mode  color
```

This command adds the specified mode to the current buffer. It can also be used to specify the foreground or background color for the current window.

To set the default modes/colors for all future buffers/windows, use the add-global-mode command.

This command is unaffected by numeric arguments.

append-file

Default binding: ^X^A

Syntax:

```
append-file  file name
```

Similar to write-file, this command writes out the current buffer to the named file, but rather than replacing its contents, it appends the buffer to the end of the existing text in the file. This does not change the filename of the current buffer. It is especially handy for building log files.

This command is unaffected by numeric arguments.

apropos

Default binding: M-A

Syntax:

```
apropos  string
```

This command builds a list of all the MicroEMACS commands and macros whose name contains the specified *string*. The list is stored in a buffer named "**Binding list**" and is displayed either in a popup buffer or in a regular window, depending on the value of the $popflag variable.

Commands are listed first, followed by macros (macro names are enclosed in square brackets "**[**" and "**]**"). For each command or macro listed, the associated bindings are also listed.

This command is unaffected by numeric arguments.

backward-character

Default bindings: ^B and FNB (left arrow)

Syntax:

```
n  backward-character
```

This command moves the point backward by *n* characters. If *n* is a negative number, the point is moved forward. If no numeric arguments is specified, the point is moved backward by one character.

Note: end of lines count as one character.

If the move would take the point beyond the boundaries of the buffer, this command fails and the point is left at said boundary.

begin-macro

Default binding: ^X(

This command tells MicroEMACS to begin recording all keystrokes, commands and mouse clicks into the keyboard macro. MicroEMACS stops recording when the end-macro (^X)) command is given.

The recording can be replayed by execute-macro (^XE).

This command is unaffected by numeric arguments.

Note: mouse clicks are recorded with the screen (row/column) position they occurred at.

beginning-of-file

Default binding: M-<

This command causes the point to move to the beginning of the buffer.

It is unaffected by numeric arguments.

beginning-of-line

Default binding: ^A

This command causes the point to move to the beginning of the current line.

It is unaffected by numeric arguments.

bind-to-key

Default binding: M-K

Syntax:

```
bind-to-key  command name  keystroke
```

This command associates a command with a *keystroke*, thus creating a binding. A keystroke can be bound only to one command or macro at a time, so when you rebind it, the previous binding is forgotten. On the other hand, a command can have more than one keystroke bound to it.

The *keystroke* is specified using the keystroke syntax or the mouse syntax.

This command cannot be used to specify the key binding for a macro. That is performed by the macro-to-key command.

This command is unaffected by numeric arguments.

bind-to-menu

No default binding

Syntax:

```
bind-to-menu  command name  menu name
```

This command is available only under Microsoft Windows. It creates a menu item associated with the specified command. The *menu name* is specified using the menu name syntax.

If the *menu name* designates a menu item that already exists, the command fails.

If the *menu name* specifies menus that do not exist yet, they are created as part of the creation of the menu item.

This command cannot be used to bind a macro to a menu. That is performed by the macro-to-menu command.

This command is unaffected by numeric arguments.

buffer-position

Default binding: ^X=

This command displays, on the message line, the position of the point within the current window. It lists:

The line (starting at 1), followed by the total number of lines in the buffer

The column (starting at 0), followed by the length of the current line

The character offset (starting at 0, newlines counting as a single character) from the beginning of the buffer, followed by the total number of character in the buffer

The percentage of text before the point

The hexadecimal value of the current character

This command is unaffected by numeric arguments.

cascade-screens

No default binding

This command is available only under Microsoft Windows. It causes all non-iconic screens to be rearranged in a cascading scheme. If the current screen is maximized (see maximize-screen) at the time this command is invoked, it is restored to its non-maximized size first.

This command is unaffected by numeric arguments.

case-region-lower

Default binding: ^X^L

This command causes all the upper case characters in the region to be changed into their lower case counterpart.

The command fails if the mark is not defined in the current window.

This command is unaffected by numeric arguments.

case-region-upper

Default binding: ^X^U

This command causes all the lower case characters in the region to be changed into their upper case counterpart.

The command fails if the mark is not defined in the current window.

This command is unaffected by numeric arguments.

case-word-capitalize

Default binding: M-C

Syntax:

```
n  case-word-capitalize
```

This command capitalizes *n* words after the point: it causes the first character of each word to be forced to upper case and the other characters to be forced to lower case. After the command has executed, the point is located just after the last processed word.

Note that since it starts by capitalizing the first letter after the point, this command would normally be issued with the cursor positioned in front of the first letter of the word you wish to capitalize. If you issue it in the middle of a word, you can end up with some strAnge looking text.

The command fails if the numeric argument is negative or if it goes beyond the end of the buffer. If *n* is null, nothing happens. If the numeric argument is not specified, only one word is affected.

case-word-lower

Default binding: M-L

Syntax:

```
n  case-word-lower
```

This command forces to lower case *n* words after the point. After the command has executed, the point is located just after the last processed word.

Note that since it starts by processing the first letter after the point, this command would normally be issued with the cursor positioned in front of the first letter of the word you wish to make lower case.

The command fails if the numeric argument is negative or if it goes beyond the end of the buffer. If *n* is null, nothing happens. If the numeric argument is not specified, only one word is affected.

case-word-upper

Default binding: M-U

Syntax:

```
n  case-word-upper
```

This command forces to upper case *n* words after the point. After the command has executed, the point is located just after the last processed word.

Note that since it starts by processing the first letter after the point, this command would normally be issued with the cursor positioned in front of the first letter of the word you wish to make upper case.

The command fails if the numeric argument is negative or if it goes beyond the end of the buffer. If *n* is null, nothing happens. If the numeric argument is not specified, only one word is affected.

change-file-name

Default binding: ^XN

Syntax:

```
change-file-name  file name
```

This command lets you change the file name associated with the current buffer. It does not change the buffer name. The disk file is unaffected.

This command is unaffected by numeric arguments.

change-screen-column

No default binding.

Syntax:

```
n  change-screen-column
```

This command modifies the offset of the current screen's left column on the desktop. The numeric argument *n* specifies that offset in number of characters. If *n* is not specified, it is taken as zero.

Using this command is equivalent to setting the $orgcol variable.

If *n* is negative or if it is positive but would cause the right border of the screen to be moved off the desktop, the command fails.

Under Microsoft Windows, this command always resets $orgcol to zero and it has no other effect.

change-screen-row

No default binding.

Syntax:

```
n  change-screen-row
```

This command modifies the offset of the current screen's top row on the desktop. The numeric argument *n* specifies that offset in number of characters. If *n* is not specified, it is taken as zero.

Using this command is equivalent to setting the $orgrow variable.

If *n* is negative or if it is positive but would cause the bottom border of the screen to be moved off the desktop, the command fails.

Under Microsoft Windows, this command always resets $orgrow to zero and it has no other effect.

change-screen-size

No default binding.

Syntax:

```
n  change-screen-size
```

This command modifies the height of the current screen, causing it to be *n* lines. If the numeric argument *n* is not specified, it is taken to be the height of the whole desktop.

As the height of the screen changes, the bottom window is resized to fit. If the height is decreased, windows that do not fit any more are eliminated, starting from the bottom one.

Using this command is equivalent to setting the $pagelen variable.

If *n* is lower than 3 or if it is greater than the height of the desktop, the command fails.

Under Microsoft Windows:

The height of a screen does not include the message line.

If *n* is not specified, the command fails.

change-screen-width

No default binding.

Syntax:

```
n  change-screen-width
```

This command modifies the width of the current screen, causing it to be *n* characters. If the numeric argument *n* is not specified, it is taken to be the width of the whole desktop.

Using this command is equivalent to setting the $curwidth variable.

If *n* is lower than 10 or if it is greater than the width of the desktop, the command fails.

Under Microsoft Windows, if *n* is not specified, the command fails.

clear-and-redraw

Default binding: ^L

Syntax:

```
clear-and-redraw
```

or:

```
n  clear-and-redraw
```

This command performs two different functions, depending on the way it is invoked:

wether it is invoked with a or not:

If the command is invoked without a numeric argument, it causes all screens to be completely repainted.

If the command is invoked with a numeric argument, it centers the line containing the point in the current window. The value of the numeric argument is irrelevant.

clear-message-line

No default binding.

This command erases the text (if any) displayed on the message line.

This command is unaffected by numeric arguments.

clip-region

Default binding: FN^C (Control+Insert)

This command copies the contents of the current region into the clipboard, overwriting any previous clipboard data.

This command is unaffected by numeric arguments.

copy-region

Default binding: M-W

This command copies the contents of the current region into the kill buffer.

This command is unaffected by numeric arguments.

count-words

Default binding: M-^C

This command displays, on the message line, the number of words in the current region, along with the number of characters, lines and the average number of characters per word.

This command is unaffected by numeric arguments.

ctlx-prefix

Default binding: ^X

This command is rarely used for execution in the macro language. Its main purpose is to be mentioned in a bind-to-key command, to redefine the **^X** prefix. For instance, the line:

```
bind-to-key  ctlx-prefix  FN1
```

redefines function key **F1** as the prefix to be used in all keystrokes that begin by "**^X-**". After this, keystrokes such as ^X^C would be actually typed by pressing and releasing the **F1** key and then pressing the **Control** key and the **C** key together.

cut-region

Default binding: S-FND (Shift+Delete)

This command deletes the contents of the current region after copying them into the clipboard, overwriting any previous clipboard data.

This command is unaffected by numeric arguments.

cycle-ring

Default binding: ^XY

Syntax:

```
n  cycle-ring
```

This command causes the kill ring to rotate by *n* positions. For instance, if the contents of the kill ring were K_1, K_2 ... K_{14}, K_{15} and K_{16}, the kill buffer would be K_{16}. After a command:

```
2  cycle-ring
```

the kill buffer would be K_{14} and the kill ring would now be ordered: K_{15}, K_{16}, K_1, K_2 ... K_{14}.

If no numeric arguments is specified, this command does not have any effect.

cycle-screens

Default binding: A-C

This command takes the rearmost screen (actually, the last screen in the screen list) and moves it to the front.

This command is unaffected by numeric arguments.

delete-blank-lines

Default binding: ^X^O

If the point is on an empty line, this command deletes all the empty lines around (above and below) the current line. If the point is on a non empty line then this command deletes all of the empty lines immediately following that line.

This command is unaffected by numeric arguments.

delete-buffer

Default binding: ^XK

Syntax:

```
delete-buffer  buffer name
```

This command attempts to discard the named buffer, reclaiming the memory it occupied. It will not allow the destruction of a buffer which is currently visible through any window on any screen.

This command is unaffected by numeric arguments.

delete-global-mode

Default binding: M-^M

Syntax:

```
delete-global-mode  mode
```

or:

```
delete-global-mode  color
```

This command causes the specified mode to be removed from the ones inherited by future (not yet created) buffers (such global modes would have been set by the add-global-mode command). It can also be used to specify the foreground or background color for future windows.

This command does not modify the modes/colors of the current buffer/window. To do so, use the delete-mode command.

This command is unaffected by numeric arguments.

delete-kill-ring

Default binding: M-^Y

This command empties the kill ring (this includes the current contents of the kill buffer) and reclaims the memory space it occupied.

This command is unaffected by numeric arguments.

delete-mode

Default binding: ^X^M

Syntax:

```
delete-mode  mode
```

or:

```
delete-mode  color
```

This command removes the specified mode from the current buffer (these modes would have been set by the add-mode or add-global-mode commands). It can also be used to specify the foreground or background color for the current window.

To set the default modes/colors for all future buffers/windows, use the delete-global-mode command.

This command is unaffected by numeric arguments.

delete-next-character

Default binding: ^D

Syntax:

```
n  delete-next-character
```

or:

```
delete-next-character
```

If *n* is positive, this command deletes, and stores into the kill buffer, *n* characters after the point. If *n* is negative, the *-n* characters preceding the point are deleted and stored into the kill buffer.

If no numeric argument is specified, the character following the point is deleted, but it is **not stored** into the kill buffer.

If an attempt to delete past the end or beginning of the buffer is made, the command fails.

Note that end of lines are counted as one character each for the purpose of deletion.

delete-next-word

Default binding: M-D

Syntax:

```
n  delete-next-word
```

This command deletes the text from the point to the beginning of the next word, saving it into the kill buffer.

If a positive numeric argument is present, it specifies the number of words to be deleted. A null numeric argument is treaded as a 1. A negative numeric argument causes the command to fail.

delete-other-windows

Default binding: ^X1

This command deletes all other windows but the active one from the current screen. It does not discard or destroy any text, just stops looking at those buffers.

This command is unaffected by numeric arguments.

delete-previous-character

Default binding: ^H (Backspace key) and FND (Delete key)

Syntax:

```
n  delete-previous-character
```

or:

```
delete-previous-character
```

If *n* is positive, this command deletes, and stores into the kill buffer, the *n* characters preceding the point. If *n* is negative, the *-n* characters following the point are deleted and stored into the kill buffer.

If no numeric argument is specified, the character preceding the point is deleted, but it is **not stored** into the kill buffer.

If an attempt to delete past the end or beginning of the buffer is made, the command fails.

Note that end of lines are counted as one character each for the purpose of deletion.

delete-previous-word

Default binding: M-^H

Syntax:

```
n  delete-previous-word
```

This command deletes the text from the point to the beginning of the previous word, saving it into the kill buffer.

If a positive numeric argument is present, it specifies the number of words to be deleted. A negative or null numeric argument causes the command to fail.

delete-screen

Default binding: A-D

Syntax:

```
delete-screen  screen name
```

This command deletes the named screen, providing it is not the active one. Note that buffers being displayed on that screen are not discarded.

This command is unaffected by numeric arguments.

delete-window

Default binding: ^X0

This command removes the active window from the screen, giving its space to the window above (or, if there is none, the window below). It does not discard or destroy any text, just stops looking at that buffer.

If the window is alone on the screen, it cannot be removed and the command fails.

This command is unaffected by numeric arguments.

describe-bindings

No default binding

This command creates a list of all commands and macros, each with all the keys which are currently bound to it. Commands are listed first, followed by the macros (macro names are surrounded by square brackets "**[**" and "**]**").

This command is unaffected by numeric arguments.

Note: The list is actually built in a special buffer named "**Binding list**". It is displayed as a popup buffer or in a normal window, depending on the value of the $popflag variable.

describe-functions

No default binding.

This command creates a list of all the functions available in the MicroEMACS macro language..

This command is unaffected by numeric arguments.

Note: The list is actually built in a special buffer named "**Function list**". It is displayed as a popup buffer or in a normal window, depending on the value of the $popflag variable.

describe-key

Default binding: ^X?

Syntax:

```
describe-key  keystroke
```

This command displays the command or macro bound to the specified *keystroke* on the message line (macro names are surrounded by square brackets "**[**" and "**]**"). If the *keystroke* has no binding, the text "Not Bound" is displayed.

When this command is used within a macro, the *keystroke* is specified using the MicroEMACS keystroke syntax or the mouse syntax(a ^G, for instance, is typed as a hat character "**^**" followed by the letter "**G**").

When this command is used interactively mode, it displays a prompt: ": describe-binding" and the keystroke is expected to by typed as if the actual bound command or macro was being invoked (a ^G, for instance, is typed by holding down the Control key and pressing the G key).

This command is unaffected by numeric arguments.

describe-variables

Default binding:

No default binding.

This command creates a list of all the variables and their value. Environmental variables are listed first, followed by user variables.

This command is unaffected by numeric arguments.

Note: The list is actually built in a special buffer named "**Variable list**". It is displayed as a popup buffer or in a normal window, depending on the value of the $popflag variable.

detab-region

Default binding: ^X^D

Syntax:

```
n  detab-region
```

or:

```
detab-region
```

This command causes tab characters to be changed into the appropriate number of spaces in the affected lines (the spacing between tab stops is considered to be the value of the $hardtab variable).

If a numeric arguments is specified, *n* lines, starting from the one containing the point, are affected. If *n* is null, the command modifies no line.

If no numeric argument is specified, all the lines belonging to the current region are affected. If no region is defined, the command modifies no line.

After this command has executed, the point is left at the beginning of the last affected line. The buffer is marked as modified, even if no modification actually took place.

display

Default binding: ^XG

Syntax:

```
display  variable
```

This command displays the value of the specified variable on the message line. If *variable* is not an existing environmental variable or user variable, the command fails.

This command is unaffected by numeric arguments.

end-macro

Default binding: ^X)

This command stops the recording of keystrokes, commands or mouse clicks into the keyboard macro.

The command fails if MicroEMACS is not currently in recording mode.

This command is unaffected by numeric arguments.

See also: begin-macro and execute-macro.

end-of-file

Default bindings: M-> and FN> (End key)

This command places the point at the end of the buffer.

This command is unaffected by numeric arguments.

end-of-line

Default binding: ^E

This command places the point at the end of the current line.

This command is unaffected by numeric arguments.

end-of-word

No default binding.

Syntax:

```
n  end-of-word
```

This command moves the point to the end of the *n*th following word. If the point was located within a word before invoking the command, that word counts as the first one (thus, if *n* is 1, the point moves to the first character following the current word). If an attempt is made to move past the buffer's end, the command fails but the point is still moved to the end of the buffer.

If no numeric argument is specified, it is equivalent to *n* = 1.

If *n* is null, the command has no effect.

If *n* is negative, it causes the command to behave like previous-word (invoked with the numeric argument *-n*).

entab-region

Default binding: ^X^E

Syntax:

```
n  entab-region
```

or:

```
entab-region
```

This command causes space characters to be compressed into tab characters wherever possible in the affected lines (the spacing between tab stops is considered to be the value of the $hardtab variable).

If a numeric arguments is specified, *n* lines, starting from the one containing the point, are affected. If *n* is null, the command modifies no line.

If no numeric argument is specified, all the lines belonging to the current region are affected. If no region is defined, the command modifies no line.

After this command has executed, the point is left at the beginning of the last affected line. The buffer is marked as modified, even if no modification actually took place.

exchange-point-and-mark

Default binding: ^X^X

Syntax:

```
n  exchange-point-and-mark
```

This command swaps the point and the mark number *n*.

If no numeric argument is specified, it is equivalent to *n* = 0.

If mark*n* does not exist, the command fails.

execute-buffer

No default binding.

Syntax:

```
n  execute-buffer  buffer
```

This command executes the macro language statements from the specified buffer.

The command fails if the *buffer* does not exist or if an executed macro statement (within the *buffer*) fails.

If a positive numeric argument is specified, the buffer is executed *n* times. If *n* is negative or null, the command has no effect.

execute-command-line

Default binding: M-^X

Syntax:

```
execute-command-line  command line
```

This command executes the specified *command line* exactly as if it were part of a macro. This is mostly used interactively to invoke a command but prevent it from fetching its own arguments interactively.

This command is unaffected by numeric arguments (note that the *command line* itself may have its own numeric argument).

execute-file or **source**

Default binding: M-^S

Syntax:

```
n  execute-file  file
```

or:

```
n  source  file
```

This command executes the macro language statements from the specified *file*, after reading it into an invisible buffer.

The *file* does not need to be a fully qualified path name: if it is a simple filename, it is searched along the path.

The command fails if the *file* cannot be found or if an executed macro statement (within the *file*) fails.

If a positive numeric argument is specified, the *file* is executed *n* times. If *n* is negative or null, the command has no effect.

execute-macro

Default binding: ^XE

Syntax:

```
n  execute-macro
```

This command replays the last recorded keyboard macro.

If a negative or null numeric argument is specified, the command does nothing. If a positive numeric argument is given, the recorded keyboard macro is played *n* times. If no numeric argument is given, the recorded macro is played once.

The command fails if MicroEMACS is currently in recording mode.

See also: begin-macro and end-macro.

execute-macro-*n*

Default binding (*n* from 1 to 9): S-FN*n*, for *n* = 10: S-FN0
No default binding for *n* greater than 10.

Syntax:

```
arg  execute-macro-n
```

MicroEMACS has 40 such commands (i.e. *n* can be a number from 1 to 40). Each causes the execution of the corresponding numbered macro (created by the store-macro command).

If a strictly positive numeric argument is specified, the macro is executed repetitively *arg* times. If *arg* is negative or null, nothing happens.

See also: execute-procedure

execute-named-command

Default binding: M-X

Syntax:

```
n  execute-named-command  command
```

In interactive mode, this command causes a colon "**:**" to appear on the message line. You can then type the name of the command you want to execute and strike Enter. If you type a space or the meta key, MicroEMACS will attempt to complete the name for you. This interactive use provides access to commands that do not have a key binding.

When used within a macro, **execute-named-command** makes the named *command* behave as if it had been called interactively, thus causing it to prompt the user for any arguments it needs.

If a numeric argument is specified, it is simply transmitted to the named *command*.

execute-procedure or **run**

Default binding: M-^E

Syntax:

```
n  execute-procedure  macro
```

or:

```
n  run  macro
```

These two commands are synonyms. They both cause the execution of the named *macro* (created by the store-procedure command).

If a strictly positive numeric argument is specified, the *macro* is executed repetitively *n* times. If *n* is negative or null, nothing happens.

See also: execute-macro-*n*

execute-program

Default binding: ^X$

Syntax:

```
execute-program  program
```

or:

```
n  execute-program  program
```

This command spawns an external *program*, without an intervening shell.

The *program* argument is a string. Note that if it contains spaces (as would be necessary to specify command line options), the string should be quoted.

Under MS-Windows:

> This command allows you to launch a Windows application from MicroEMACS. The current working directory where the application executes is set to the directory of the file in the current window (or, if that window is not associated to a filename, to the last visited directory).
>
> If no numeric argument is specified, MicroEMACS and the launched application run independently. If a numeric argument is specified, MicroEMACS synchronizes with the application.

Note: Under MS-DOS, you cannot use this command to invoke built-in system commands (like DIR, for instance). Use shell-command instead.

exit-emacs

Default binding: ^X^C

Syntax:

```
n  exit-emacs
```

This command terminates MicroEMACS.

If no numeric argument is specified and some buffers contain text that has been changed but not yet saved, you will be asked for a confirmation. If a numeric argument is specified, the command terminates MicroEMACS unconditionally.

fill-paragraph

Default binding: M-Q

This command reformats the current paragraph, causing all of its text to be filled out to the current fill column (Which is 72 by default and is set with the set-fill-column command or the $fillcol variable).

This command is unaffected by numeric arguments.

filter-buffer

Default binding: ^X#

Syntax:

```
filter-buffer  program
```

This command spawns the external filter *program* (for instance: SORT or FIND) and feeds it the contents of the current buffer. The results replace the original text in the buffer.

Under Microsoft Windows, this command creates a DOS box and synchronizes with it.

This command is unaffected by numeric arguments.

find-file

Default binding: ^X^F

Syntax:

```
find-file  file name
```

If the named file is already loaded somewhere in the editor, this command brings its buffer up in the current window. Otherwise, the file is searched for on disk. If it is found, a new buffer is created and the contents of the file are read into it. If the file does not exist, a new empty buffer is created. In all cases, the buffer is brought up in the current window.

This command is unaffected by numeric arguments.

find-screen

Default binding: A-F

Syntax:

```
find-screen  screen name
```

This command brings up the named screen. If the *screen name* does not exist, a new screen is created. On text systems, this screen is displayed on top of the others. On graphic systems, the OS window containing this screen is brought to the foreground.

This command is unaffected by numeric arguments.

forward-character

Default binding: ^F and FNF (right arrow)

Syntax:

```
n  forward-character
```

This command moves the point forward by *n* characters. If *n* is a negative number, the point is moved backward. If no numeric arguments is specified, the point is moved forward by one character.

Note: end of lines count as one character.

If the move would take the point beyond the boundaries of the buffer, this command fails and the point is left at said boundary.

goto-line

Default binding: M-G

Syntax:

```
n  goto-line
or
goto-line  n
```

This command moves the point to the first character of line number *n* in the current buffer.

The command fails if *n* is lower than 1 or if the buffer is empty. If *n* is greater than the number of lines in the buffer, the point is simply positioned at the end of the buffer.

goto-mark

Default binding: M-^G

Syntax:

```
n  goto-mark
```

This command moves the point to the location of the mark number *n*.

If no numeric arguments is specified, the mark number 0 is used.

If *n* is greater than 9, it is treated as the remainder of the division of *n* by 10.

goto-matching-fence

Default binding: M-^F

When the point is located on a fence character (curly brace, bracket, or parenthesis), this command will make it jump to the matching fence character.

If the point is not located on a fence character or there is no matching fence, a beep sounds and the command fails.

This command is unaffected by numeric arguments.

grow-window

Default binding: ^X^ and ^XZ

Syntax:

```
n  grow-window
```

If *n* is a positive number, this command increases the height of the current window by *n* lines. The window located immediately below the current window (or, if the current window is at the bottom of the screen, the window located immediately above it) shrinks by *n* lines. If that would cause the shrinking window to become too small to display any text, the command fails.

If the current screen contains only one window, the command fails.

If *n* is a negative number, this command acts as if the shrink-window command had been invoked with the corresponding positive number (-*n*).

If no numeric arguments is specified, the height of the window is increased by one line.

To change the size of the current window by specifying an absolute value, use the resize-window command.

handle-tab

Default binding: ^I (Tab key)

Syntax:

```
n  handle-tab
```

or:

```
handle-tab
```

The behavior of this command depends on the numeric argument (*n*) that is supplied to it:

With no argument, it simply inserts a single tab character or enough space characters (depending on its configuration...) to get to the next tab stop.

With an non-zero argument *n*, tabs stops are reset to every *n*th column and **handle-tab** is reconfigured to insert space characters in sufficient number to get to the next tab stop. This also sets the $softtab variable to *n*.

With an argument *n* of zero, **handle-tab** is reconfigured so that it inserts true tab characters (its default behavior) and the tab stop interval is reset to its default value of 8.

The distance which a true tab character moves the cursor is reflected by the value of the $hardtab variable. Initially set to 8, this determines how far each tab stop is placed from the previous one.

help

Default binding: M-?

This command brings up a window to display the contents of a text file named EMACS.HLP located on the path. This file usually contains a summary of the MicroEMACS commands and default key bindings.

The command fails if the EMACS.HLP file cannot be found.

This command is unaffected by numeric arguments.

help-engine

No default binding.

Syntax:

```
help-engine  file  key
```

or:

```
help-engine  file
```

This command invokes the MS Windows WinHelp application to display the specified help *file*. If a *key* is specified, the WinHelp application is instructed to search and display the first topic that matches that *key*. Otherwise, the first topic displayed is the help file's table of content.

This command is unaffected by numeric arguments.

This command is available only under the MS Windows version of MicroEMACS.

hunt-backward

Default binding: A-R

Syntax:

```
n  hunt-backward
```

If *n* is a positive number, this command searches backwards for the *n*th occurrence of the search string. That search string is the one that was used the last time a search-forward or search-reverse command was issued. The interpretation of the search string is dependant on whether MAGIC mode is set or not in the current buffer.

If a matching text is found in the buffer, the point is moved to the first character of that text. Otherwise, the command fails. The command also fails if there is no search string.

If *n* is a negative number, this command acts as if the hunt-forward command had been invoked with the corresponding positive number (-*n*).

If no numeric arguments is specified, or if the numeric argument is null, it is equivalent to *n* = 1.

hunt-forward

Default binding: A-S

Syntax:

```
n  hunt-forward
```

If *n* is a positive number, this command searches forward for the *n*th occurrence of the search string. That search string is the one that was used the last time a search-forward or search-reverse command was issued. The interpretation of the search string is dependant on whether MAGIC mode is set or not in the current buffer.

If a matching text is found in the buffer, the point is moved to the first character following that text. Otherwise, the command fails. The command also fails if there is no search string.

If *n* is a negative number, this command acts as if the hunt-backward command had been invoked with the corresponding positive number (-*n*).

If no numeric arguments is specified, or if the numeric argument is null, it is equivalent to *n* = 1.

i-shell

Default binding: ^XC

This command spawns a command line shell.

Under MS Windows, this command launches a DOS box (a "shell box" under Windows NT). The current working directory where the shell starts is set to the directory of the file in the current window (or, if that window is not associated to a filename, to the last visited directory).

This command is unaffected by numeric arguments.

incremental-search

Default binding: ^XS

This command is always interactive. It prompts the user for a search string but, unlike what happens with the search-forward command, the search happens and the display is updated as each new search character is typed.

While searching towards the end of the buffer, each successive character leaves the point at the end of the entire matched string. Typing a ^S causes the next occurrence of the string to be searched for (where the next occurrence does not overlap the current occurrence). A ^R changes the direction to a backwards search (as performed by a reverse-incremental-search command), pressing the meta key terminates the search and ^G aborts the operation. Pressing the Backspace key (or using ^H) backs up to the previous match of the string or, if the starting point is reached, it deletes the last character from the search string.

The characters composing the search string are always interpreted literally. MAGIC mode has no effect on incremental searches.

If the search fails, a beep sounds and the search stalls until the search string is edited back into something that exists (or until the operation is aborted).

This command is unaffected by numeric arguments.

indent-region

Default binding: M-)

Syntax:

```
n  indent-region
```

This command inserts *n* tab characters in front of each line within the current region.

If the numeric argument *n* is not specified, one tab is inserted per line.

If CMODE is set in the current buffer, lines that begin by a pound sign "**#**" are not modified (this is to keep C preprocessor directives flush to the left).

Note: the undent-region command can be used to undo the effect of this command.

insert-clip

Default binding: S-FNC (Shift + Insert)

Syntax:

```
n  insert-clip
```

This command is only available under MS Windows. It inserts the contents of the Windows clipboard at the point.

If the numeric argument *n* is specified, *n* copies of the clipboard's contents are inserted.

insert-file

Default binding: ^X^I

Syntax:

```
insert-file  file
```

This command inserts the contents of the specified *file* into the current buffer, at the point. After the insertion, the point remains at its original place if the $yankflag variable is TRUE. Otherwise, the point is moved to the end of the inserted text.

This command is unaffected by numeric arguments.

insert-space

Default binding: ^C

Syntax:

```
n  insert-space
```

This command inserts *n* space characters at the point. After the insertion, the point remains at its original place.

If the numeric argument *n* is not specified, a single space character is inserted.

insert-string

No default binding.

Syntax:

```
n  insert-string  string
```

This command inserts the specified *string* at the point. After the insertion, the point is moved to the end of the inserted text.

If the numeric argument *n* is specified, *n* copies of the specified *string* are inserted (if *n* is negative, it is taken as -*n*). If *n* is 0, nothing happens.

kill-paragraph

Default binding: M-^W

Syntax:

```
n  kill-paragraph
```

This command deletes the current paragraph, leaving a copy of it in the kill buffer.

If a positive numeric argument *n* is specified, *n* paragraphs, starting with the current one, are deleted. If *n* is negative or null, nothing happens.

kill-region

Default binding: ^W

This command deletes the characters belonging to the current region, leaving a copy of the deleted text in the kill buffer.

This command is unaffected by numeric arguments.

kill-to-end-of-line

Default binding: ^K

Syntax:

```
n  kill-to-end-of-line
```

This command's deletes text, leaving a copy of it in the kill buffer. The text affected depends on the numeric arguments applied to the command:

If it is used without a numeric argument, kill-to-end-of-line truly behaves as its name indicates, deleting the text from the point to the end of the current line, but preserving the newline character, unless the point is located at the end of a line in which case the command just deletes the newline character.

If the numeric argument is 0, the command deletes the text from the start of the current line up to the point.

If the numeric argument *n* is positive, the command deletes text from the point forward until *n* newlines have been removed.

If the numeric argument *n* is negative, the command deletes text from the point backwards until *n* newlines have been removed and the beginning of a line has been reached.

list-buffers

Default binding: ^X^B

Syntax:

```
list-buffers
```

or:

```
n  list-buffers
```

This command creates a list of all the buffer with, for each buffer, the file it was read from, its size, and the active modes. The list is stored in a buffer named "**[Buffers]**" and is displayed in either a popup buffer or a regular window, depending on the value of the $popflag variable.

Within the list, an at sign "**@**" in column one shows that a file has already been read into a buffer. A star "*****" in column two means that the contents of the buffer have been modified since the last time they were written to disk. A pound sign "**#**" in column three indicates the file was to large to read into memory and was truncated. A lower than sign "**<**" in column four indicates that the buffer has been narrowed.

The modes are shown in columns 5 through 14, using a single letter code for each active mode:

Code	**Corresponding mode:**
W	WRAP
C	CMODE
E	EXACT
V	VIEW
O	OVER
M	MAGIC
Y	CRYPT
A	ASAVE
R	REP

Used without a numeric argument, list-buffers does not list invisible buffers. If a numeric argument is given, this command lists all buffers, including those hidden buffers used by MicroEMACS for internal data and macros storage.

list-screens

Default binding: A-B

This command creates a list of all the screens with, for each screen, the names of the buffers visible in windows on that screen. The list is stored in a buffer named "**[Screens]**" and is displayed in either a popup buffer
or a regular window, depending on the value of the $popflag variable.

This command is unaffected by numeric arguments.

macro-to-key

Default binding: ^X^K

Syntax:

```
macro-to-key  macro name  keystroke
```

This command associates a macro with a *keystroke*, thus creating a binding. A keystroke can be bound only to one command or macro at a time, so when you rebind it, the previous binding is forgotten. On the other hand, a macro can have more than one keystroke bound to it.

This command cannot be used to specify the key binding for a command. That is performed by the bind-to-key command.

The *keystroke* is specified using the keystroke syntax or the mouse syntax.

This command is unaffected by numeric arguments.

macro-to-menu

No default binding

Syntax:

```
macro-to-menu  macro name  menu name
```

This command is available only under Microsoft Windows. It creates a menu item associated with the specified macro. The *menu name* is specified using the menu name syntax.

If the *menu name* designates a menu item that already exists, the command fails.

If the *menu name* specifies menus that do not exist yet, they are created as part of the creation of the menu item.

This command cannot be used to bind a command to a menu. That is performed by the bind-to-menu command.

This command is unaffected by numeric arguments.

maximize-screen

No default binding.

This command is available only under Microsoft Windows. It causes the current screen to be enlarged so that it occupies all the available space on MicroEMACS's frame window. If the current screen is already maximized at the time this command is invoked, nothing happens.

This command is unaffected by numeric arguments.

To restore the current screen to the size and position it had before invoking this command, use the restore-screen command.

meta-prefix

Default binding: ^[(Escape key)

This is a dummy command meant to be used in combination with the bind-to-key command in order to redefine the meta key.

For example, to define the F1 function key as being the meta key:

```
unbind-key  ^[
bind-to-key  meta-prefix  FN1
```

minimize-screen

No default binding.

This command is available only under Microsoft Windows. It causes the current screen to be reduced to an icon. Unless there exists only one screen at the time this command is invoked another screen becomes the current one. If the screen being minimized was maximized (see maximize-screen), the screen becoming current is also maximized.

This command is unaffected by numeric arguments.

To restore the current screen to the size and position it had before invoking this command, use the restore-screen command.

mouse-move

Default binding: MSm, S-MSm and MS^m (mouse movement)

This command is meant to be associated with a mouse movement. It updates the $xpos and $ypos variables to contain the coordinates of the mouse pointer.

If the $hilight variable is set to a value *n* between 0 and 14, this command updates the mark number *n*+1 so that the highlighted region is automatically updated.

This command is unaffected by numeric arguments.

Note: the mouse actions MSm, S-MSm or MS^m may not be generated for all mouse movements, depending on the value of the $mmove variable.

mouse-move-down

Default binding: MSa (Press on left mouse button)

This command is meant to be associated with a mouse action. It depends on the $xpos and $ypos variables to contain the coordinates of the mouse pointer. It makes the screen and window where the mouse was clicked the current ones. If the mouse pointer is within the text part of a window (as opposed to the mode line) the point is placed at that position in the text (or at the end of the line if the mouse pointer lies beyond the end of a line).

This command is unaffected by numeric arguments.

Note: Under the MS-Windows version of MicroEMACS, the selection of the current screen is performed by the press on the left mouse button, regardless of the button's binding. Mouse commands themselves cannot select the current screen.

See also: mouse-move-up

mouse-move-up

Default binding: MSb (Release of left mouse button)

This command is meant to be associated with a mouse action. It depends on the $xpos and $ypos variables to contain the coordinates of the mouse pointer. The actions performed by this command depend of where the previous mouse-move-down command was invoked:

If the mouse pointer was in the mode line part of a window and still is within that mode line, or if it was in the text part of the window and still is, the text in the window is scrolled as if it had been dragged by the mouse. Note that diagonal dragging is possible only if the $diagflag variable is set to TRUE.

If the mouse pointer was on a mode line (except the bottom one), but has moved above or under it, the mode line is moved up or down as if it had been dragged by the mouse, thus resizing the affected windows.

Other cases produce no effect.

The command fails (putting FALSE in the $status variable) if the position of the mouse pointer is the same as that for the last mouse-move-down command. This allows easy detection of lack of mouse movement when the command is used in a macro.

This command is unaffected by numeric arguments.

Note: Under the MS-Windows version of MicroEMACS, the top left and bottom right corners of a screen have no special meaning. Under other versions, mouse-move-up will move the screen if the mouse-move-down was done in the top left corner and resize the screen if mouse-move-down was done in the bottom right corner.

mouse-region-down and **mouse-region-up**

Default binding: MSe (Press on right mouse button)
and: MSf (Release of right mouse button)

These commands are meant to be associated with the two parts of a mouse click. Their rather complex behavior is dependant on where the last mouse action took place and is best described by the following topics:

Copying a Region
Killing a Region
Pasting Text

These commands are unaffected by numeric arguments.

mouse-resize-screen

No default binding

This command is meant to be associated with a mouse action. It depends on the $xpos and $ypos variables to contain the coordinates of the mouse pointer. It modifies the size of the current screen, bringing its lower right corner to where the mouse was clicked.

This command is unaffected by numeric arguments.

move-window-down

Default binding: ^X^N

Syntax:

```
n  move-window-down
```

This command moves the window's view into it's buffer down by *n* lines, causing the text visible in the window to scroll up. If the point scrolls out of view, it is repositioned on the first character of the line located at the center of the window.

If no numeric argument is specified, the text is scrolled by one line.

move-window-up

Default binding: ^X^P

Syntax:

```
n  move-window-up
```

This command moves the window's view into it's buffer up by *n* lines, causing the text visible in the window to scroll down. If the point scrolls out of view, it is repositioned on the first character of the line located at the center of the window.

If no numeric argument is specified, the text is scrolled by one line.

name-buffer

Default binding: M-^N

Syntax:

```
name-buffer  name
```

This command renames the current buffer, giving it the specified *name*. Note that when a buffer is associated with a file, changing the buffer's name has no effect on the file's name.

If a buffer bearing the specified *name* already exists, another argument is required, and so on until a unique name is supplied.

This command is unaffected by numeric arguments.

narrow-to-region

Default binding: ^X<

This command causes the text that does not belong to the current region to become inaccessible until the widen-from-region command is invoked. The mode line displays the symbol "**<>**" to indicate that the current window is associated with a narrowed buffer.

This command is unaffected by numeric arguments.

newline

Default binding: ^M (Return key)

Syntax:

```
n  newline
```

This command inserts *n* newline characters at the point. If the numeric arguments is absent, it is taken as 1.

If *n* is equal to 1 and the buffer is in CMODE mode, C language indentation is performed:

If the new line is not empty (i.e. the point was not at the end of a line), no other action takes place.

The new line is indented at the same level as the closest preceding non blank line

If the newline was inserted right after an opening brace "**{**", the new line is further indented by one tab stop (as if the handle-tab command had been used).

If the buffer is in WRAP mode and the point is past the fill column, wrapping is performed on the last word of the current line before the newline character is inserted.

The command fails if *n* is negative.

newline-and-indent

Default binding: ^J

Syntax:

```
n  newline-and-indent
```

This command inserts *n* newline characters at the point. If the numeric arguments *n* is absent, it is taken as 1.

The new line is indented with enough tab and space characters to match the indentation of the preceding line (the one where the point was when newline-and-indent was invoked).

The command fails if *n* is negative.

next-buffer

Default binding: ^XX

Syntax:

```
n  next-buffer
```

This command causes the current window to display the *n*th next buffer in the circular list of buffers kept by MicroEMACS. If the numeric arguments *n* is absent, it is taken as 1.

The command fails if *n* is not positive.

next-line

Default binding: ^N

Syntax:

```
n  next-line
```

This command moves the point to the *n*th next line. If the numeric arguments *n* is absent, it is taken as 1.

If *n* is negative, the point is moved to the *n*th previous line. If *n* is 0, nothing happens.

When line move commands (**next-line** or previous-line) are used in a row, the point is kept at the same column it was at before the first of the line moves. If that column lies beyond the end of the current line the point is temporarily brought back to the end of that line.

The command fails if the point is already at the end of the buffer (or the beginning if *n* is negative).

next-page

Default bindings: ^V and FNV (Page Down key)

Syntax:

```
next-page
```

or:

```
n  next-page
```

This command has two different behaviors, depending on the presence or absence of a numeric arguments:

If no numeric argument is specified, the window's view into it's buffer is paged down. The new view overlaps the previous one by the number of lines specified by the $overlap variable. By default, $overlap is equal to 2, so the last two lines of text in the initial view are displayed at the top of the window.

If a positive numeric argument *n* is specified, the window's view into it's buffer is moved down by *n* lines, causing the text visible in the window to scroll up.

If a negative numeric argument *n* is specified, the window's view into it's buffer is moved up by *n* lines, causing the text visible in the window to scroll down, as if the previous-page command had been invoked, with a numeric argument of -*n*.

In all cases, even if a numeric argument of 0 is given, the point is moved to the first character at the top of the window.

next-paragraph

Default binding: M-N

Syntax:

```
n  next-paragraph
```

If used without a numeric arguments, this command moves the point just past the last character of the current paragraph or, if outside a paragraph, to the end of the next paragraph.

If this command is used with a positive numeric argument *n*, the point is moved to the *n*th next end of paragraph.

If *n* is negative, next-paragraph behaves as if the previous-paragraph command had been invoked with an argument of *-n*.

next-window

Default binding: ^XO

Syntax:

```
n  next-window
```

If used without a numeric arguments, this command makes the next window immediately below the current one the new current window. MicroEMACS updates the highlight of the mode line to indicate the new current window, and places the blinking cursor at the point within that window.

If this command is used with a positive numeric argument *n*, the *n*th window from the top of the screen is made the current one (window numbering starts at 1).

If *n* is negative, the *-n*th window from the bottom of the screen is made the current one.

The command fails if *n* (or *-n*) is greater than the number of windows in the screen.

next-word

Default bindings: M-F and FN^F (Ctrl + Right arrow)

Syntax:

```
n  next-word
```

This command moves the point to the first character of the *n*th next word. If an attempt is made to move past the buffer's end, the command fails but the point is still moved to the end of the buffer.

If no numeric argument is specified, it is equivalent to *n* = 1.

If *n* is null, the command has no effect.

If *n* is negative, it causes the command to behave like previous-word (invoked with the numeric argument -*n*).

nop

No default binding.

This command has no effect and is unaffected by numeric arguments. Its main purpose is to be the command pointed to by the $bufhook, $cmdhook, $exbhook, $readhook and $writehook variables.

open-line

Default binding: ^O

Syntax:

```
n  open-line
```

This command adds *n* newline characters after the point. If the numeric arguments is absent, it is taken as 1.

The command fails if *n* is negative.

overwrite-string

No default binding.

Syntax:

```
overwrite-string  string
```

This command replaces the characters from the point on with the characters from the specified *string*. If the overwriting would extend past the end of the line, the remaining characters from the *string* are simply added at the end of the line (the newline character is not overwritten).

This command is unaffected by numeric arguments.

pipe-command

Default binding: ^X@

Syntax:

```
pipe-command  program
```

This command uses the shell to execute a program, but rather than displaying what the program prints, it attempts to place it in a buffer named "command" to let you edit it and/or save it.

The *program* argument is a string. Note that if it contains spaces (as would be necessary to specify command line options), the string should be quoted.

The VIEW mode is set on the "command" buffer at completion of this command.

Under Microsoft Windows, this command launches the *program* within a DOS box and synchronizes with it. The current working directory where the *program* executes is set to the directory of the file in the current window (or, if that window is not associated to a filename, to the last visited directory).

This command is unaffected by numeric arguments.

pop-buffer

No default binding.

Syntax:

```
pop-buffer  buffer
```

or:

```
n  pop-buffer  buffer
```

This command causes the specified buffer to be displayed as a popup in the current screen.

If a numeric arguments is present, the buffer is marked as being invisible (hidden from the next-buffer command).

previous-line

Default binding: ^P

Syntax:

```
n  previous-line
```

This command moves the point to the *n*th previous line. If the numeric arguments *n* is absent, it is taken as 1.

If *n* is negative, the point is moved to the *n*th next line. If *n* is 0, nothing happens.

When line move commands (next-line or **previous-line**) are used in a row, the point is kept at the same column it was at before the first of the line moves. If that column lies beyond the end of the current line the point is temporarily brought back to the end of that line.

The command fails if the point is already at the beginning of the buffer (or the end if *n* is negative)

previous-page

Default bindings: M-V and FNZ (Page Up key)

Syntax:

```
previous-page
```

or:

```
n  previous-page
```

This command has two different behaviors, depending on the presence or absence of a numeric arguments:

If no numeric argument is specified, the window's view into it's buffer is paged up. The new view overlaps the previous one by the number of lines specified by the $overlap variable. By default, $overlap is equal to 2, so the top two lines of text in the initial view are displayed at the bottom of the window.

If a positive numeric argument *n* is specified, the window's view into it's buffer is moved up by *n* lines, causing the text visible in the window to scroll down.

If a negative numeric argument *n* is specified, the window's view into it's buffer is moved down by *n* lines, causing the text visible in the window to scroll up, as if the next-page command had been invoked, with a numeric argument of -*n*.

In all cases, even if a numeric argument of 0 is given, the point is moved to the first character at the top of the window.

previous-paragraph

Default binding: M-P

Syntax:

```
n  previous-paragraph
```

If used without a numeric arguments, this command moves the point to the first character of the current paragraph or, if outside a paragraph, to the beginning of the previous paragraph.

If this command is used with a positive numeric argument *n*, the point is moved back to the *n*th beginning of paragraph.

If *n* is negative, next-paragraph behaves as if the next-paragraph command had been invoked with an argument of -*n*.

previous-window

Default binding: ^XP

Syntax:

```
n  previous-window
```

If used without a numeric arguments, this command makes the window immediately above the current one the new current window. MicroEMACS updates the highlight of the mode line to indicate the new current window, and places the blinking cursor at the point within that window.

If this command is used with a positive numeric argument *n*, the *n*th window from the bottom of the screen is made the current one (window numbering starts at 1).

If *n* is negative, the -*n*th window from the top of the screen is made the current one.

The command fails if *n* (or -*n*) is greater than the number of windows in the screen.

previous-word

Default bindings: M-B and FN^B (Ctrl + Left arrow)

Syntax:

```
n  previous-word
```

This command moves the point to the beginning character of the *n*th preceding word. If the point was located within a word before invoking the command, that word counts as the first one (thus, if *n* is 1, the point moves to the first character of the current word). If an attempt is made to move beyond the buffer's beginning, the command fails but the point is still moved to the beginning of the buffer.

If no numeric argument is specified, it is equivalent to *n* = 1.

If *n* is null, the command has no effect.

If *n* is negative, it causes the command to behave like next-word (invoked with the numeric argument -*n*).

query-replace-string

Default binding: M-^R

Syntax:

```
n  query-replace-string  pattern  replacement
```

This command attempts to replace, from the point onward, each piece of text that matches the *pattern* string by the *replacement* string. The *pattern* string is interpreted literally, unless MAGIC mode is enabled in the current buffer.

Each time a match is found, you are queried and can answer by one of the following keystrokes:

Y	replaces the current matching text
N	skips the current match
!	replaces the current matching text and all following matches without anymore queries.
U	jumps back to the last performed replacement and undoes it
^G	aborts the command, leaving the point at its current position
. (dot)	aborts and moves the point back to where the command was originally issued
?	lists the above options

If no numeric arguments is specified, all the matching pieces of text are processed until the end of the buffer is reached. If a positive numeric argument is used, only the first *n* matches are taken into account. If *n* is negative, the command fails.

When this command is invoked interactively, the keystroke used to signal the end of the *pattern* or *replacement* string is specified by the $sterm variable (it is usually the Meta key). Also, both strings may have default values (which are stored in the $search and $replace variables). If you want to replace a string with nothing, and there is a non-empty default for the *replacement* string, striking ^K will override that default and enter an empty string instead.

Note: to perform global string replacements without interactive involvement, use the replace-string command.

quick-exit

Default binding: M-Z

This command causes MicroEMACS to terminate, but only after having written all the changed buffers into their respective files.

This command is unaffected by numeric arguments.

Note: to terminate MicroEMACS without saving the changed buffers, use the exit-emacs command.

quote-character

Default binding: ^Q

Syntax:

```
n  quote
```

This command inserts literally the next character typed by the user at the point. Even the newline character can be inserted this way, but this causes it to loose its line-splitting meaning.

If a positive numeric arguments is specified, the quoted character is inserted *n* times. If *n* is negative, the command fails. If *n* is null, nothing is inserted, but the typing of a character is still required.

read-file

Default binding: ^X^R

Syntax:

```
read-file  file name
```

This command reads the named file into the current buffer, replacing the buffer's contents with the text from the file. The file name associated to the buffer is not changed, so you must make sure that replacing the text in the original file with that from the read one is what you are intending when you use this command.

This command is unaffected by numeric arguments.

redraw-display

Default bindings: M-^L and M-!

Syntax:

```
n  redraw-display
```

If a non zero numeric argument is specified, this command scrolls the text in the current window so that the current line is displayed as the *n*th line from the top of the window if *n* is positive, or as the *-n*th line from the bottom of the window if *n* is negative.

If no numeric argument is specified, or if *n* is zero, the current line is displayed at the center of the window.

remove-mark

Default binding: ^X__ (Ctrl+X Spacebar)

Syntax:

```
n  remove-mark
```

This command eliminates the mark number *n*.

If no numeric argument is specified, it is equivalent to *n* = 0.

If mark*n* does not exist, nothing happens.

rename-screen

No default binding.

Syntax:

```
rename-screen  new name
```

This command changes the name of the current screen to the specified *new name*. If the *new name* is already in use, the command fails.

This command is unaffected by numeric arguments.

replace-string

Default binding: M-R

Syntax:

```
n  replace-string  pattern  replacement
```

This command replaces, from the point onward, each piece of text that matches the *pattern* string by the *replacement* string. The *pattern* string is interpreted literally, unless MAGIC mode is enabled in the current buffer.

If no numeric arguments is specified, all the matching pieces of text are processed until the end of the buffer is reached. If a positive numeric argument is used, only the first *n* matches are processed. If *n* is negative, the command fails.

When this command is used interactively, the keystroke used to signal the end of the *pattern* or *replacement* string is specified by the $sterm variable (it is usually the Meta key). Also, both strings may have default values (which are stored in the $search and $replace variables). If you want to replace a string with nothing, and there is a non-empty default for the *replacement* string, striking ^K will override that default and enter an empty string instead.

Note: to have more interactive control over the replacement process, use the query-replace-string command.

resize-window

Default binding: ^XW

Syntax:

```
n  resize-window
```

If *n* is a positive number, this command changes the height of the current window so that it displays *n* lines of text. The window located immediately below the current window (or, if the current window is at the bottom of the screen, the window located immediately above it) shrinks accordingly. If that would cause the shrinking window to become too small to display any text, the command fails.

If the current screen contains only one window, or if *n* is a negative number, the command fails.

If no numeric arguments is specified, nothing happens.

To change the size of the current window by specifying a relative value, use the grow-window or the shrink-window command.

restore-screen

No default binding.

This command is available only under Microsoft Windows. It causes the current screen to be restored to the size and position it had before it was maximized (see maximize-screen) or iconized.(see minimize-screen). If the current screen is neither maximized nor iconized this command has no effect.

This command is unaffected by numeric arguments.

restore-window

No default binding.

This command is only useful when there are multiple windows displayed on the current screen. It causes the window that was current the last time the save-window command was invoked to become the current window again.

If the window that was current the last time **save-window** was invoked no longer exists, or if the screen is not the same, this command fails.

This command is unaffected by numeric arguments.

reverse-incremental-search

Default binding: ^XR

This command is always interactive. It prompts the user for a search string but, unlike what happens with the search-reverse command, the search happens and the display is updated as each new search character is typed.

While searching towards the beginning of the buffer, each successive character leaves the point at the beginning of the matched string. Typing a ^R causes the next occurrence of the string to be searched for (where the next occurrence does not overlap the current occurrence). A ^S changes the direction to a forward search (as performed by an incremental-search command), pressing the meta key terminates the search and ^G aborts the operation. Pressing the Backspace key (or using ^H) returns to the previous match of the string or, if the starting point is reached, it deletes the last character from the search string.

The characters composing the search string are always interpreted literally. MAGIC mode has no effect on incremental searches.

If the search fails, a beep sounds and the search stalls until the search string is edited back into something that exists (or until the operation is aborted).

This command is unaffected by numeric arguments.

save-file

Default binding: ^X^S

This command writes the contents of the current buffer to disk, if the buffer's contents have been changed since the last read or write operation or the last invocation of the unmark-buffer command.

If the current buffer does not have a file name associated to it (for instance if the buffer has never been subjected to a find-file, read-file, write-file or change-file-name command), the save-file command fails.

If the current buffer is narrowed, a confirmation is requested before writing the text to the file.

This command is unaffected by numeric arguments.

save-window

No default binding.

This command saves a reference to the current window, so that the next time the restore-window command is invoked, that window becomes the current window again.

This command is unaffected by numeric arguments.

scroll-next-down

Default binding: M-^V

Syntax:

```
scroll-next-down
```

or:

```
n  scroll-next-down
```

This command causes the equivalent of a next-page command to be performed on the window located just below the current one (or the top window if the current one is at the bottom of the screen).

If there is only one window displayed in the current screen, this command is equivalent to the next-page command.

scroll-next-up

Default binding:

Syntax:

```
scroll-next-up
```

or:

```
n  scroll-next-up
```

This command causes the equivalent of a previous-page command to be performed on the window located just below the current one (or the top window if the current one is at the bottom of the screen).

If there is only one window displayed in the current screen, this command is equivalent to the previous-page command.

search-forward

Default binding: ^S

Syntax:

```
n  search-forward  search string
```

If *n* is a positive number, this command searches forward for the *n*th occurrence of the *search string*. The interpretation of the *search string* is dependant on whether MAGIC mode is set or not in the current buffer.

If a matching text is found in the buffer, the point is moved to the first character following that text. Otherwise, the command fails.

If *n* is a negative number, this command acts as if the search-reverse command had been invoked with the corresponding positive number (-*n*).

If no numeric arguments is specified, or if the numeric argument is null, it is equivalent to *n* = 1.

Note: the *search string* becomes the value of the $search variable

search-reverse

Default binding: ^R

Syntax:

```
n  search-reverse  search string
```

If *n* is a positive number, this command searches backwards for the *n*th occurrence of the *search string*. The interpretation of the *search string* is dependant on whether MAGIC mode is set or not in the current buffer.

If a matching text is found in the buffer, the point is moved to the first character of that text. Otherwise, the command fails.

If *n* is a negative number, this command acts as if the search-forward command had been invoked with the corresponding positive number (-*n*).

If no numeric arguments is specified, or if the numeric argument is null, it is equivalent to *n* = 1.

Note: the *search string* becomes the value of the $search variable

select-buffer

Default binding: ^XB

Syntax:

```
select-buffer  buffer
```

or:

```
n  select-buffer  buffer
```

This command displays the named *buffer* in the current window. If that buffer does not yet exist, it is created.

If a numeric arguments is present, the buffer is marked as being invisible (hidden from the next-buffer command).

set

Default binding: ^X^A

Syntax:

```
set  variable  value
```

or:

```
n  set  variable
```

This command sets the value of the specified variable to *n* if a numeric arguments is present and to *value* otherwise.

The *variable* must be a user variable or an environmental variable . In the latter case, if the environmental variable does not exist, the command fails.

If *value* is the string "ERROR", the command fails (this allows detection of error cases when *value* is actually a function.

set-encryption-key

Default binding: M-E

Syntax:

```
set-encryption-key  key
```

This command sets the current buffer's encryption key (used when the buffer is in CRYPT mode). The specified key can be up to 128 characters long. A length of at least 5 characters is recommended.

This command is unaffected by numeric arguments.

set-fill-column

Default binding: ^XF

Syntax:

```
n  set-fill-column
```

This command sets the fill column, (used by the fill-paragraph command) to *n*.

Note that this also sets the $fillcol variable to *n*.

set-mark

Default bindings: M- (Esc Spacebar) and M-.

Syntax:

```
n  set-mark
```

This command sets the mark number *n* at the point.

If no numeric argument is specified, it is equivalent to *n* = 0.

shell-command

Default binding: ^X!

Syntax:

```
shell-command  program
```

or:

```
n  shell-command  program
```

This command uses the shell to execute the named *program*.

The *program* argument is a string. Note that if it contains spaces (as would be necessary to specify command line options), the string should be quoted.

Under MS-Windows:

This command launches the *program* within a DOS box. The current working directory where the *program* executes is set to the directory of the file in the current window (or, if that window is not associated to a filename, to the last visited directory).

If no numeric argument is specified, MicroEMACS and the launched *program* run independently. If a numeric argument is specified, MicroEMACS synchronizes with the *program*.

Note: Under MS-Windows 3.x, you cannot use this command to launch a Windows application. Use execute-program instead.

show-files

No default binding

Syntax:

```
show-files  starname
```

This command creates a list of all the files matching the specified *starname*. The starname can contain a directory specification.

For instance, under MS-Windows, the command:

```
show-files  "C:\WINDOWS\*.INI"
```

will create a list of all the files ending by ".INI" in the directory "C:\WINDOWS".

MicroEMACS appends a star "*****" to the end of the specified starname, and appends a dot-star "**.***" if the starname does not contain a dot character. Thus:

```
show-files  "C:\WINDOWS\A"
```

is equivalent to specifying:

```
show-files  "C:\WINDOWS\A*.*"
```

This command is unaffected by numeric arguments.

Note: The list is actually built in a special buffer named "**File List**". It is displayed as a popup buffer or in a normal window, depending on the value of the $popflag variable.

shrink-window

Default binding: ^X^Z

Syntax:

```
n  shrink-window
```

If *n* is a positive number, this command decreases the height of the current window by *n* lines. The window located immediately below the current window (or, if the current window is at the bottom of the screen, the window located immediately above it) grows by *n* lines. If the decrease of height would cause the current window to become too small to display any text, the command fails.

If the current screen contains only one window, the command fails.

If *n* is a negative number, this command acts as if the grow-window command had been invoked with the corresponding positive number (-*n*).

If no numeric arguments is specified, the height of the window is decreased by one line.

To change the size of the current window by specifying an absolute value, use the resize-window command.

split-current-window

Default binding: ^X2

Syntax:

```
n  split-current-window
```

This command splits the current window into two windows. Both windows view the current buffer at the current point.

If a numeric arguments is present and not equal to 1, the lower of the two windows becomes current. If *n* = 1, the upper window becomes current.

If no numeric argument is present, the upper window is selected as current if the point was in the upper half of the split window, otherwise, the lower window is selected.

The command fails if it would result in a window too small to display any line of text.

To rid the screen of extraneous windows, use the delete-window or the delete-other-windows commands.

store-macro

No default binding

Syntax:

```
n  store-macro
   contents
     of
   macro
!endm
```

This command stores the commands and directives that follow it, up to the next !ENDM directive, into a "numbered macro". That macro can be invoked later by the execute-macro-*n* command.

A numeric arguments must be specified and it must be a number from 1 to 40. Otherwise, the command fails.

store-procedure

No default binding

Syntax:

```
store-procedure  name
   contents
     of
   macro
!endm
```

or:

```
n  store-procedure
   contents
     of
   macro
!endm
```

If no numeric arguments is specified, this command stores the commands and directives that follow it, up to the next !ENDM directive, into a "named macro" or "procedure". That procedure can be invoked later by the run or execute-procedure command, with the argument *name*.

If a numeric argument is specified, this command is equivalent to store-macro.

tile-screens

No default binding

Syntax:

```
n  tile-screens
```

This command is available only under Microsoft Windows. It causes all non-iconic screens to be rearranged in a tiled scheme. If the current screen is maximized (see maximize-screen) at the time this command is invoked, it is restored to its non-maximized size first.

If a numeric arguments is present and equals 1, the screens are tiled vertically (i.e. on top of each other). Otherwise, the screens are tiled horizontally (i.e. side by side). However, if there are too many screens to tile (more than 3), the argument is ignored and a mix of vertical and horizontal tiling is used.

transpose-characters

Default binding: ^T

This command swaps the character that is before the point and the character that is at the point, unless the point is at the end of a line, in which case the two last characters of the line are swapped around.

This command fails if the point is located at the beginning of a line.

This command is unaffected by numeric arguments.

trim-region

Default binding: ^X^T

Syntax:

```
trim-region
```

or:

```
n  trim-region
```

This command causes all the trailing space and tab characters between the column position of the point and the end of the processed lines to be deleted.

If a numeric arguments is present, *n* lines, starting from the current one, are processed.

If no numeric argument is present, the lines processed are the ones that belong to the current region.

unbind-key

Default binding: M-^K

Syntax:

```
unbind-key  keystroke
```

This command removes the association between a *keystroke* and a macro or a command, thus destroying a binding.

The *keystroke* is specified using the keystroke syntax or the mouse syntax.

This command is unaffected by numeric arguments.

unbind-menu

No default binding

Syntax:

```
unbind-menu  menu name
```

This command is available only under Microsoft Windows. It destroys a menu item. The *menu name* is specified using the menu name syntax.

If the *menu name* designates a menu item that does not exist, the command fails.

If the *menu name* specifies a menu (that itself contains menu items), all the menu hierarchy under it is destroyed.

This command is unaffected by numeric arguments.

undent-region

Default binding: M-(

Syntax:

```
n  undent-region
```

This command deletes the first *n* tab characters in front of each line within the current region.

If the numeric argument *n* is not specified, the first tab of each line is deleted.

Note: this command is often used to undo the effect of an indent-region command.

universal-argument

Default binding: ^U

This is a dummy command meant to be used in combination with the bind-to-key command in order to redefine the universal argument key.

To define the F1 function key as being the universal argument key:

```
bind-to-key  universal-argument  FN1
```

Pressing the universal argument key causes a numeric argument of 4 to be generated. If digits (and the minus sign) are entered following the universal argument, they are interpreted to compose a numeric argument, much as if the meta key had been pressed. Also, each further action on the universal argument key multiplies the existing numeric argument by 4.

unmark-buffer

Default binding: M-~

This command clears the change flag of the current buffer. This causes MicroEMACS to forget that the buffer's contents have changed since they were last made equivalent to the contents of a disk file (by append-file, find-file, read-file, save-file, view-file or write-file).

This command is unaffected by numeric arguments.

Note: the change flag of the current buffer can also be accessed via the $cbflags variable.

update-screen

No default binding

This command immediately updates all elements of the MicroEMACS display during the execution of a macro. It has no visible effect when used interactively.

This command is unaffected by numeric arguments.

view-file

Default binding:

Syntax:

```
find-file  file name
```

If the named file is already loaded somewhere in the editor, this command brings its buffer up in the current window. Otherwise, the file is searched for on disk. If it is found, a new buffer is created and the contents of the file are read into it. If the file does not exist, a new empty buffer is created. In all cases, the buffer is brought up in the current window, in VIEW mode.

This command is unaffected by numeric arguments.

widen-from-region

Default binding: ^X>

This command causes all the invisible text in the narrowed buffer becomes accessible and visible again.

This command is unaffected by numeric arguments.

wrap-word

No default binding

This command replaces by a newline the first group of space or tab characters preceding the point on the current line. The point is left where it was when the command was invoked.

If no space or tab character is found before the point, a new line is created after the current one and the point is moved to it.

This command is unaffected by numeric arguments.

Note: the $wraphook variable (which points to the command or macro use to perform line wrapping in WRAP mode) is set to wrap-word by default.

write-file

Default binding: ^X^W

Syntax:

```
write-file  file name
```

This command writes the contents of the current buffer to disk, using the specified *file name*. This file name becomes the one associated with the buffer (indicated by the $cfname variable).

This command is unaffected by numeric arguments.

write-message or **print**

No default binding

Syntax:

```
print  message
```

or:

```
write-message  message
```

This command causes the specified *message* to appear on the message line.

This command is unaffected by numeric arguments.

yank

Default binding: ^Y

Syntax:

```
n  yank
```

This command inserts the contents of the kill buffer at the point. If a numeric arguments is present, the command is repeated *n* times.

If *n* is negative, the command fails.

The placement of the point after the execution of this command is determined by the value of the $yankflag variable.

yank-pop

Default binding: M-Y

Syntax:

```
n  yank-pop
```

This command cycles the kill ring *n* times (as done by the cycle-ring command) and inserts the contents of the kill buffer at the point. If the previous command was yank or yank-pop, the text inserted by that command is deleted before the new text is inserted.

If no numeric argument is specified, it is equivalent to *n* = 1.

The placement of the point after the execution of this command is determined by the value of the $yankflag variable.