MicroEmacs '02

Table of Contents

Table of Contents

Table of Contents

Table of Contents

Table of Contents

Table of Contents

Table of Contents

Table of Contents

Table of Contents

Table of Contents

Table of Contents

Table of Contents

Table of Contents

Table of Contents

Table of Contents

Table of Contents

Table of Contents

Table of Contents

Table of Contents

Table of Contents

Table of Contents

MicroEmacs '02

MICROEMACS

MicroEmacs '02, JASSPA Distribution, is defined as follows, refer to me(1) for a description of the command line variables.

The following sections describe the topics that are available as part of the on−line **MicroEmacs '02** manual pages.

Acknowledgments, Copyright, Origins and Contact Information.
Frequently Asked Questions.

See Help! for some information on using the hypertext manual pages.

me(1)

NAME

me − MicroEmacs '02 text editor

SYNOPSIS

me [*options*] [*files ...*]

me [*@startupFile*] [**−b**] [**−c**] [**−d**] [**−h**] [**−i**] [**−l***lineNo*] [**−m***command*] [**−n**] [**−0***file*] [**−p**] [**−r**] [**−s***string*] [**−u***username*] [**−v***variable=string*] [**−x**] *files...*

DESCRIPTION

MicroEmacs '02 is a cut down version of the EMACS text editor, based on Danial Lawrences MicroEmacs. **MicroEmacs '02** 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 '02 can be extensively customized for the needs of the individual user.

MicroEmacs '02 allows multiple files to be edited at the same time. The screen may 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 '02** 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 '02** behaves. Also, any and all the commands can be used by any key stroke 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.

The command line options to **MicroEmacs '02** are defined as follows:−

@startFile

Initialize MicroEmacs '02 using *startFile*[**.emf**]. The default when omitted is **me.emf**. See start−up(3) and Command Line Filters for more information.

−b

Load next file as a binary file (binary editor mode, uses binary(2m) buffer mode).

−c

Continuation mode. Load **MicroEmacs '02** last edit session, restoring the buffers to their previous loaded state and position. Note that history mode must be enabled. The **−c** option is generally used with windowing interfaces (X−Windows/Microsoft Windows) as the shortcut icon invocation.

−d

Enable debug mode (for macro files).

−h

Show the help page (does not start the editor).

−i

MS−DOS versions of **MicroEmacs '02** only. Insert the contents of the current screen into the ***scratch*** buffer

−k[*key*]

Load next file as an encrypted file (uses crypt(2m) buffer mode). The optional adjoining argument can be used to specify the decrypting key, if this argument is not specify the user will be prompted for it on start−up.

−l*lineNo*

Go to line *lineNo* in the next given file. Typically used with utilities such a **more(1)** where an external editor may be invoked from other viewer.

−m*command*

Sends a client−server command to an existing MicroEmacs session. The command takes the form "**C:**<*client*>**:**<*command*>" i.e. to write "`Hello World`" on the message line then a client may issue the command:−

```
; launch server
me &
; send message
me −m "C:ME:ml−write \"Hello world\"
```

Note that the <*command*> is a MicroEmacs macro command, the escape sequences must be adhered to. The *client−server* interface is typically used to load a file, this may be performed as follows:−

```
me −m "C:myutility:find−file \"/path/foo.bar\""
```

The absolute path is specified in this type of transaction as the current working directory of the active MicroEmacs session is unknown. The **−m** option de−iconize's the existing editor session and bring it to the foreground.

−n

UNIX X−Windows environments only and MicroSoft Windows NT console versions. Execute **MicroEmacs '02** using termcap rather than X−Windows for UNIX; typically used within an **xterm** shell to fire up **MicroEmacs '02** for a quick edit. For Microsoft Windows, a console window is started as opposed to a GUI window.

−o<*file*>

Use already running version of MicroEmacs '02 to load the <*file*>, if it exists, otherwise start a new editor session. This uses the *client−server* interface to push the new file into the existing editor session. Refer to the Client−Server Interface for details.

−p

Pipe *stdin* into buffer ***stdin***, when saved output to *stdout*, following is a simple example which changes '`a`'s to '`b`'s:

```
define−macro start−up
    find−buffer "*stdin*"
    beginning−of−buffer
    replace−string "a" "b"
    save−buffer
    quick−exit
!emacro
```

This can be used in the following manner:

```
me "@testpipe.emf" < foo.a > foo.b
```

−r

Read−only, all buffers will be in view mode

−s*string*

Search for string "*string*" in the current buffer. e.g. `me −sfoo bar` starts **MicroEmacs '02**, loads file `bar` and initiates a search for *foo*. The cursor is left at the end of the string if located, otherwise at the top of the buffer.

−u*username*

Set the current user name to *username* before MicroEmacs is initialized. This is done by setting the environment variable MENAME(5) to the given value.

−v*variable=string*

Assign the MicroEmacs '02 *variable* with *string*. The assignment is performed before the buffers are loaded. Typically used to change the start−up characteristics of the startup file(s).

−x

UNIX environments. Disable the capture of signals. **MicroEmacs '02** by default captures an handles all illicit signal interrupts. The option is enabled when debugging the source code allowing exception conditions to be trapped within the debugger.

−y

Load next file as a reduced binary file (uses rbin(2m) buffer mode). **ENVIRONMENT**

The following environment variables are used by **MicroEmacs '02**.

DISPLAY

UNIX environments running X−Windows only. The identity of the X−Windows server. Typically set to **unix:0.0**, refer to the X−Windows documentation for details of this environment variable.

MENAME and **LOGNAME**

The identity of the user, **$MENAME** takes precedence over **$LOGNAME**. **$LOGNAME** variable is generally defined within UNIX as part of the login script. The variables are used to determine which start−up configuration to use in the initialization of **MicroEmacs '02** (*$MENAME*.erf).

Non−UNIX platforms usually need to explicitly set the **$MENAME** environment variable to identify the aforementioned files. for MS−DOS and Microsoft Windows this is typically performed in the `AUTOEXEC.BAT` file.

PATH

The **$PATH** environment variable is used on most operating systems as a search path for executable files. This **$PATH** environment variable must be defined with **MicroEmacs '02** on the search path. Under UNIX this is set in the `.login`, `.cshrc` or `.profile` file i.e.

```
export PATH $PATH:/usr/name/me
```

Within MS−DOS or Microsoft Windows environments it is defined in the `AUTOEXEC.BAT` file. e.g.

```
set PATH=%PATH%;c:\me
```

MicroEmacs '02 utilizes information in the **$PATH** environment variable to locate the start−up files, dictionaries etc.

TERM

The terminal identification sting. In UNIX environments the environment variable **$TERM** is set to "`vt...`", in this case it is assumed that the machine is a server, and the host cannot support X (see command line option **−n**).

In MS−DOS the environment variable is usually set to define the graphics adapter mode. **%TERM** is assigned a string, understood by the `me.emf` start−up file, to set the graphics mode. Predefined strings include:−

E80x50

Initiates an 80 column by 50 line screen.

E80x25

Initiates an 80 column by 25 line screen.

userDefined

A user defined string to set an explicit graphics card mode. The operation is dependent upon the support offered by the graphics adapter.

MEPATH

MicroEmacs '02 uses the environment variable `$MEPATH` as the directory(s) used to search for the macro files (see emf(8)). Within the UNIX `$MEPATH` is a semi−colon separated list of directories which are used to search for the MicroEmacs '02 macro files. The path is searched from left to right. The environment variable is typically defined in the in the `.login`, `.cshrc` or `.profile` file i.e.

export MEPATH /usr/name/me/macros:/usr/local/microemacs

The default when omitted is `/usr/local/microemacs`.

Within MS−DOS or Microsoft Windows environments it is defined in the `AUTOEXEC.BAT` file. e.g.

set MEPATH=c:\me\username;\me\macros

There is no default location in these environments. For Microsoft Windows environments refer to me32.ini(8) for a method of setting up the `$MEPATH` from the windows configuration file.

INFOPATH

MicroEmacs '02 uses the environment variable `$INFOPATH` as the directory(s) used to search for GNU **Info** files. Within the UNIX `$INFOPATH` is a semi−colon separated list of directories which are used to search for the MicroEmacs '02 macro files. The path is searched from left to right. The environment variable is typically defined in the in the `.login`, `.cshrc` or `.profile` file i.e.

export INFOPATH /usr/local/info:$HOME/info

The default when omitted is `/usr/local/info`.

Within MS−DOS or Microsoft Windows environments it is defined in the `AUTOEXEC.BAT` file. e.g.

set MEPATH=c:\usr\local\info

There is no default location in these environments. For Microsoft Windows environments refer to me32.ini(8) for a method of setting up the `$INFOPATH` from the windows configuration file.

FILES

All of the macro files and dictionaries are located in the **MicroEmacs** home directory. The standard file extensions that are utilized are:−

.eaf

MicroEmacs '02 abbreviation file, defines completion definitions for buffer dependent text expansion.

.edf

A **MicroEmacs '02** spelling dictionary. *<language>***.edf** provide language specific dictionaries; *$LOGNAME***.edf** is personal spelling dictionary.

.ehf

MicroEmacs '02 help file information. On−line help information for emacs, the main file is `me.ehf`.

.emf

A **MicroEmacs '02** macro file. The following classes of macro file exist:

me.emf

The default startup file.

<platform>.emf

A platform specify startup file, these include UNIX generic (`unixterm.emf`), UNIX specific (`irix.emf`, `hpux.emf`, `unixwr1.emf`, `linux.emf`, `sunos.emf` etc), Microsoft Windows (`win32.emf`), MS−DOS (`dos.emf`).

hk*xxxxxx***.emf**

Buffer context specific hook files to initialize a buffer with macros and highlighting appropriate to the contents of the file type. e.g. 'C' language editing (`hkc.emf`), N/Troff typesetting (`hknroff.emf`), UNIX Manual page display (`hkman.emf`), Makefiles (`hkmake.emf`), etc.

.erf

Registry files, used to retain personal information, users history in the file etc.

.etf

Template files used to seed new files. Typically contains standard header information, copyright notices etc. that are placed at the head of files. The 'C' programming language is called `c.etf` **MICROSOFT WINDOWS**

Microsoft Windows environments should refer to me32.ini(8) for a method of setting up the environment variables without editing the `AUTOEXEC.BAT` configuration file.

SEE ALSO

emf(8), erf(8), **emacs(1)** [GNU], **more(1)**, **vi(1)**.
Client−Server Interface.
Command Line Filters.

Acknowledgments

ACKNOWLEDGMENTS

The following persons contributed to this release of **MicroEmacs '02** over the last decade, roughly in the order of participation. This list represents the main developers:−

Danial M. Lawrence (Original Author)
Martin House
Jon Green − *Current Maintainer*
Callen McNally
Steven Phillips − *Current Maintainer*

Additional contributions have been made as follows, in chrononlogical order:−

Detlef Groth *[June 1999]*

Setting up and validating the German environment.
Latex features and excellent feedback.

Pedro Gomes *[May 1999]*

Portuguese Dictionary.
Cobol and Intel x86 language templates.
Metapost/Meta Font templates.

Matthew Robinson *[Feburary 1999]*

Developed the WinConsole version for Windows NT.

Thanks to everybody else that has used and abused it locally feeding back comments and preferences, wishes and desires.

Copyright

COPYRIGHT

GNU General Public License (GPL)

All source and macro code is covered by the GPL.

GNU Free Documentation License (GFDL)

All documentation is covered by the GFDL.

and the Document's license notice requires Cover Texts, you must enclose the copies in covers that carry, clearly and legibly, all these Cover Texts: Front-Cover Texts on the front cover, and Back-Cover Texts on the back cover. Both covers must also clearly and legibly identify you as the publisher of these copies. The front cover must present the full title with all words of the title equally prominent and visible. You may add other material on the covers in addition. Copying with changes limited to the covers, as long as they preserve the title of the Document and satisfy these conditions, can be treated as verbatim copying in other respects.

If the required texts for either cover are too voluminous to fit legibly, you should put the first ones listed (as many as fit reasonably) on the actual cover, and continue the rest onto adjacent pages.

If you publish or distribute Opaque copies of the Document numbering more than 100, you must either include a machine-readable Transparent copy along with each Opaque copy, or state in or with each Opaque copy a publicly-accessible computer-network location containing a complete Transparent copy of the Document, free of added material, which the general network-using public has access to download anonymously at no charge using public-standard network protocols. If you use the latter option, you must take reasonably prudent steps, when you begin distribution of Opaque copies in quantity, to ensure that this Transparent copy will remain thus accessible at the stated location until at least one year after the last time you distribute an Opaque copy (directly or through your agents or retailers) of that edition to the public.

It is requested, but not required, that you contact the authors of the Document well before redistributing any large number of copies, to give them a chance to provide you with an updated version of the Document.

4. MODIFICATIONS

You may copy and distribute a Modified Version of the Document under the conditions of sections 2 and 3 above, provided that you release the Modified Version under precisely this License, with the Modified Version filling the role of the Document, thus licensing distribution and modification of the Modified Version to whoever possesses a copy of it. In addition, you must do these things in the Modified Version:

A. Use in the Title Page (and on the covers, if any) a title distinct from that of the Document, and from those of previous versions (which should, if there were any, be listed in the History section of the Document). You may use the same title as a previous version if the original publisher of that version gives permission.
B. List on the Title Page, as authors, one or more persons or entities responsible for authorship of the modifications in the Modified Version, together with at least five of the principal authors of the Document (all of its principal authors, if it has less than five).
C. State on the Title page the name of the publisher of the Modified Version, as the publisher.
D. Preserve all the copyright notices of the Document.
E. Add an appropriate copyright notice for your modifications adjacent to the other copyright notices.
F. Include, immediately after the copyright notices, a license notice

5. COMBINING DOCUMENTS

You may combine the Document with other documents released under this License, under the terms defined in section 4 above for modified versions, provided that you include in the combination all of the Invariant Sections of all of the original documents, unmodified, and list them all as Invariant Sections of your combined work in its license notice.

The combined work need only contain one copy of this License, and multiple identical Invariant Sections may be replaced with a single copy. If there are multiple Invariant Sections with the same name but different contents, make the title of each such section unique by adding at the end of it, in parentheses, the name of the original author or publisher of that section if known, or else a unique number. Make the same adjustment to the section titles in the list of Invariant Sections in the license notice of the combined work.

In the combination, you must combine any sections entitled "History" in the various original documents, forming one section entitled "History"; likewise combine any sections entitled "Acknowledgements", and any sections entitled "Dedications". You must delete all sections entitled "Endorsements."

6. COLLECTIONS OF DOCUMENTS

You may make a collection consisting of the Document and other documents released under this License, and replace the individual copies of this License in the various documents with a single copy that is included in the collection, provided that you follow the rules of this License for verbatim copying of each of the documents in all other respects.

You may extract a single document from such a collection, and distribute it individually under this License, provided you insert a copy of this License into the extracted document, and follow this License in all other respects regarding verbatim copying of that document.

7. AGGREGATION WITH INDEPENDENT WORKS

A compilation of the Document or its derivatives with other separate and independent documents or works, in or on a volume of a storage or distribution medium, does not as a whole count as a Modified Version of the Document, provided no compilation copyright is claimed for the compilation. Such a compilation is called an "aggregate", and this License does not apply to the other self-contained works thus compiled with the Document, on account of their being thus compiled, if they are not themselves derivative works of the Document.

If the Cover Text requirement of section 3 is applicable to these copies of the Document, then if the Document is less than one quarter of the entire aggregate, the Document's Cover Texts may be placed on covers that surround only the Document within the aggregate. Otherwise they must appear on covers around the whole aggregate.

8. TRANSLATION

```
"Front−Cover Texts being LIST"; likewise for Back−Cover Texts.

If your document contains nontrivial examples of program code, we
recommend releasing these examples in parallel under your choice of
free software license, such as the GNU General Public License,
to permit their use in free software.
```

License History

JASSPA MicroEmacs is derived from the MicroEmacs 3.8 source base of 1998. As such, all of the software has been under a commercially restrictive license. JASSPA has upheld the original license terms laid down my the original author and copyright holder Danial M Lawrence.

JASSPA is the collective name given to the maintainers of JASSPA MicroEmacs. The current maintainers at the 1st January 2002 are Steven Phillips and Jon Green.

On Wednesday 19th January 2001 JASSPA applied for, and was granted, permission by Danial M Lawrence to move to the less restrictive licensing terms of GPL. As of 1st January 2002 the JASSPA distribution of MicroEmacs shall be supplied under this licensing arrangement. The licence is not transferable to earlier versions of the distribution or to the original program from which is was derived known as MicroEmacs.

License Terms to 1988

The following copyrights apply from the original source code of version 3.8. No explicit copyrights were found with the original distribution apart from the following found in the main source code,

> **(C)opyright 1987 by Daniel M. Lawrence**
> MicroEMACS can be copied and distributed freely for any non−commercial purposes. Commercial users may use MicroEMACS inhouse. Shareware distributors may redistribute MicroEMACS for media costs only. MicroEMACS can only be incorporated into commercial software or resold with the permission of the current author.

License Terms 1998−2001

The following notices apply after 1988 to 31st December 2001

> **Copyright (C) 1988 − 2001, JASSPA**
> JASSPA MicroEmacs can be copied and distributed freely for any non−commercial purposes. Commercial users may use JASSPA MicroEmacs inhouse. Shareware distributors may redistribute JASSPA MicroEmacs for media costs only. JASSPA MicroEmacs can only be incorporated into commercial software or resold with the permission of the current author.

License Terms 2002 and subsequent years

Origins

ORIGINS

This version of MicroEmacs is based on an early *MicroEmacs* release of 3.8 in 1988, the origins of which are unknown, except to say it was delivered on a unmarked 5 1/2" floppy disk.

The program was originally ported to a Motorola MVME147 UNIX box as an alternative to **vi**. Reliability of the program proved to be a problem as it constantly crashed. In an attempt to rectify the problems the development of MicroEmacs '02 commenced.

Development has continued from 1988 through to today, on the whole oblivious to further developments of the existing **MicroEmacs** program. This was due to no Internet access. It was not until 1996 the next version of **MicroEmacs** and **mewin** (Microsoft Windows (TM) port of the same program) was downloaded from the Internet and compared. By this time MicroEmacs '02 was radically different and we were not about to mesh the two together − that would be a step backwards.

Development of MicroEmacs '02 has been biased towards the UNIX platform, as most of the early development was performed in the UNIX domain. The first of the window servers was X−Windows, which in turn has shaped the implementation of the Microsoft Windows port. Latterly, we have seen the resurgence of the IBM−PC platform which is now commonplace. For the return port to the DOS environment, and subsequent development of the Microsoft Windows port, a UNIX like interface was required. Most existing users could not abide the primitive editors found on these machines; Microsoft Windows was an alien and hostile environment when compared with UNIX. Hence, the MicroEmacs '02 interface utilizes UNIX style cut and paste across all platforms.

For portability, MicroEmacs '02 utilizes character rendering on all platforms regardless of the window manager. Under X−Windows and Microsoft Windows, the display is still treated as a character based display, the subtle difference is that the display pane is re−sizable. This means that the scroll bars, fonts etc. are not as slick as they could be, certainly under Microsoft Windows MicroEmacs '02 looks positively primitive!! Regardless of the look, the goal of a common editor across all working platforms has been achieved!

Development History

1988−92

◊ Ported to MVME147, UNIX using curses.
◊ Fire fighting to get a stable version.
◊ Expanded regular expression syntax.

1992−1993

◊ Ported to IBM AIX
◊ Ported to Silicon Graphics

1994

◊ DOS built with djgpp, allowed large files to be edited.
◊ Color hi−lighting.
◊ Re−implemented the macro language. Allowed separately named macros.
◊ Get−next−line support.
◊ File hooks added.
◊ Implemented Electric C.
◊ RCS support.
◊ Re−implemented backups and auto−saves.
◊ Re−implemented isearch.
◊ Re−implemented of keyboard macros.
◊ Binary file reading support.

1995

◊ Integral speller.
◊ Ported to HP−UX.
◊ Multiple ipipes supported on Unix.
◊ Poke−screen support.
◊ Call−back macro support.
◊ Metris created.
◊ Abbreviation and completion.
◊ First implementation of mailing and View Mail.
◊ Isearch expanded to support magic mode.
◊ Session history support.
◊ First ported to X−Terminal.
◊ Mouse support.
◊ Initial printer support.

1996

◊ Ported to Slackware Linux.
◊ First menu system (implemented in macros).
◊ Undo support.
◊ First ported to Microsoft Windows 95.
◊ Re−implementation of get−next−line.
◊ Auto mode support.
◊ Magic file hooks added.
◊ Proper key name support.
◊ Key bindings support numeric arguments.

1997

◊ Re−implementation of ipipes to enable terminal support.
◊ Initial Directory−Tree support.
◊ First implementation of the On Screen Display (OSD) menus and dialogues.
◊ Horizontal split window support.
◊ Scroll bar support.

◊ Added menu bar.
◊ Cursor position correction for hilights with invisible character.
◊ Indentation scheme support.
◊ Ported to Microsoft NT
◊ Ipipes supported on NT.

1998

◊ Registry features for configuration.
◊ Re−implementation of OSD.
◊ Re−implementation of termcap extended key support.
◊ Narrow support.
◊ Re−implementation of the session history.
◊ Re−implemented the speller based on ispell dictionaries.
◊ Re−worked the printer interface for Windows.
◊ Rationalised mouse key bindings.
◊ Added $system variable to configure MicroEmacs.
◊ Added random hilight−token addition and removal support.
◊ **1st Release** − September 1998.
◊ Support True−type fonts under windows, font selection dialog.
◊ **Minor Patch** − October 1998.
◊ Enhanced the operation of the Window pipe's
◊ Undo past the last save operation.
◊ Added Tabbed entries to OSD.
◊ Enhanced user setup using OSD.

1999

◊ Introduction of the address and date organizer(3) (replacing the existing **cal** interface)
◊ Generic buffer folding using narrow−buffer(2)
◊ Rendered cursor support on all platforms.
◊ Smooth scrolling mode.
◊ Win32s port, for Microsoft 3.1/3.11 O/S.
◊ Re−worked **translate−tcap−key** to generic translate−key(2) to solve many of the foreign language problems.
◊ Re−worked the ALT key mapping to allow conventional Emacs meta key bindings.
◊ Implemented Auto−Spell utility.
◊ Introduction of private macro variables of the form *.name* and *.macro.name*.
◊ Port to Sun Solaris Intel platform (2.6)
◊ Merged the init−hilight and hilight−token into the single hilight(2) command. Similarly for indent(2).
◊ Enhanced the regexp support in hilight tokens, vastly improving it capability and usability. Similarly for indent tokens.
◊ Reorganized the hilighting files. Introduction of the scheme−editor(3) and *Hilight Search* OSD's.
◊ Implemented box character override support ($system(5) bit 0x10000) on Win32 and Xterm platforms.
◊ Microsoft Windows native console support.
◊ **2nd Release − Beta #1** − May 1999.

◊ Bug fixes with multi−language support and spelling dictionaries.
◊ Fixed fill−paragraph(2) such that it retains the cursor position in the paragraph when invoked without arguments.
◊ Moved to GNU regex for search/replace engine.
◊ Enhanced *isearch* such that it operates in **shell** buffers (again).
◊ gdiff(3) macro implementation of a graphical diff to allow color annotation of differences and difference selection. Uses the output of a standard **diff(1)** utility.
◊ 2nd port to IBM AIX.
◊ **2nd Release − Beta #2** − November 1999
◊ Enhanced the latex(9) support macros following various contributions from users.
◊ Added *Favorites* to the *File* pull down menu. This is a simple mechanism to allow the user to add a file to a favorites list.
◊ Ground up implementation of the regular expression pattern matcher, following licensing problems with the GNU regex. The resultant pattern matcher is now a little faster than GNU regex and is capable of all of the standard regular expression pattern matches. The pattern matcher has diverged from GNU in that double backslashes are required in the character classes `[..]` to allow for escape sequence short cuts such as '`\n`' (newline), '`\t`' (tab) etc. It was felt that this compromise was better than having to quote the more obscure search characters.
◊ Changed the hilighting and indenting syntax to be GNU regex compliant.
◊ Modified compare−windows(2) to ignore white space by default, compare−windows−exact(3) performs an exact character for character comparison.
◊ Enhanced the tags support to handle multiple tags and recursive directory tree searching.
◊ Implemented osd−help, a gui front end to the on−line help. Required changes to the help system which gives macros access to the on−line help buffer, needed for the index and search.
◊ Over−hauled the hilighting scheme files and editor to support the disabling of buffer hilighting.
◊ Many bugs fixed on all platforms, in particular unix cutting and pasting (crashes exceed) and focus problems and NT exit delay.
◊ Added tab and newline character printing in buffers.
◊ Introduced message line variables @mx and @mxa.
◊ Added ftp support with a new ftp 'file−browser' interface to give easy to use ftp capability.

2000

◊ Fixed several millennium bugs :)
◊ Revamped the printing interface to support colors
◊ Started using **CVS(1)** at last, much better history from now on.
◊ Got the emain #define's properly supported again so options so MicroEmacs can be successfully compiled without options like SPELL and OSD etc. This can reduce the binary size by up to 37%.
◊ Improved the macro based tag generators to support source code trees and many new items of information, added new generation GUI.
◊ Greatly improved the OSD based search and replace dialog, also added the line hilight into this dialog.
◊ Added new −k and −u command−line options and improved −s option.

◊ Greatly improved and increase the file system operations via the new file−op(2) command, created better menus in the file−browser(3).
◊ Added new set−position(2) which can be used to store all information about the current window and goto−position(2) which will then restore them. Allowed macros to use non−letter characters for alpha marks and positions so they no longer need to clobber user ones.
◊ Greatly improved ipipe−shell−command efficiency on windows by introducing a new thread approach to listen for activity. Implemented a shell command−line in an ipiped environment so ishell(3) command is now usable on Windows platforms.
◊ Revamped the main ftp GUI to make it work much more rationally.

2001

◊ Changed buffer variables to except the form :<buffer−name>:<variable>, like command variables. This allows macros to access variable not defined in the current buffer.
◊ Added new command−wait(2) command to enable macros to wait for user dependent completion, used in **gdiff**.
◊ Added support for new $buffer−backup(5) variable for setting the back−up file location.
◊ Major work to enable the complete rebinding of all keys so the user interface to MicroEmacs can be completely changed, this allows for example a Windows feel to be created. This required many internal ghosts to be exorcised and add new macro functions
◊ Added new generic buffer setup, menu and help macros, creating the new buffer−setup(3) command. Ported all existing file hooks to this new interface greatly reducing the buffer hook size while increasing its functionality and consistency.
◊ Added generic commands for the creation, deletion and reformatting of comments now used by most of the file hooks.
◊ **3rd Release** − July 2001. Fixed many bugs in the last 12 months as well as adding the above features. &cbind(4), &kbind(4) and &nbind(4).

Work In Progress or Planned

Development of MicroEmacs '02 is an on−going process, follows is a list of work items which is currently being undertaken or planned:−

♦ Support for multiple frames.
♦ Horizontal scroll bars.
♦ On−line tutorial to help beginners get up and running.
♦ Native printer support for the generation of postscript.
♦ GNU Emacs compatibility macro file.

Release History

May 1999

2nd Major Release − MicroEmacs '99.

October 1998

Minor patch to MicroEmacs '98 to correct a font problem on Windows platforms.

September 1998

1st Major Release − MicroEmacs '98. **Documentation**

The documentation is all written in UNIX **nroff(1)** and converted to HTML, Microsoft Windows Help File format and MicroEmacs Help file format.

Contact Information

Spelling Dictionary Copyrights

The spelling dictionaries are converted from *ispell* dictionaries, each spelling dictionary has it's own copyright which is reproduced within the appropriate language spelling macro file.

NO WARRANTY

CONTACT INFORMATION

The following contact point may be used to report all problems and request information:−

```
Email:support@jasspa.com
```

Also visit our web site's on the Internet:−

```
http://www.jasspa.com
http://www.geocities.com/jasspa
```

This page carries the latest information and patches on the distribution.

E−Mail Reflector

A E−Mail reflector is available onto which questions, suggestions and code fragments may be posted. This is an un−moderated E−Mail group. The mailing list archives may be found at:−

```
http://groups.yahoo.com/group/jasspa/
```

These should be referenced before posting any questions, as an answer may have already been given. This mailing list is likely to contain the most up−to−date information available, as JASSPA will use this for any notifications.

If you want to join the E−Mail reflector then mail a request, with an empty message body to:−

```
jasspa−subscribe@yahoogroups.com
```

This should add you to the mailing list. If you want to be subsequently removed from the mailing list then mail an empty message to:−

```
jasspa−unsubscribe@yahoogroups.com
```

and you will be removed from the lists. There are facilities on the site to allow you to receive digests, rather than multiple posts to the group once you have subscribed.

The mailing list is hosted by **YAHOO! Groups**. General help information on the mailgroup is available at their site:−

```
http://groups.yahoo.com
```

Help, FAQ and other queries

Please use **YAHOO! Groups** help information for any queries about the mailing lists, JASSPA have simply registered and E−Mail group at the site and have not investigated fully all of the facilities that may be available.

Please check the Frequently Asked Questions. list and the home page FAQ before you submit any information to JASSPA, your problem may have already been addressed in these pages.

Every effort will be made to deal with your problem as soon as possible. Please send mail with the titles indicated below so that it maybe filtered. Unless explicitly requested (or appropriate) JASSPA will only respond through the FAQ lists on the Internet site. It will not be possible to answer individual enquiries and questions.

Any sites that wish to mirror the JASSPA distribution should contact JASSPA first (use PORT), we will then include details of the mirror on the home page.

Reporting Problems

Mail as:−

Title: **BUG**
Message Body: Description of the problem ....

Problems should be reported such that they may be reproduced, this may not be that easy. The information that is required is:−

Platform

The host platform which is exhibiting a problem

Version

The version of MicroEmacs '02. Use the `esc x about` to retrieve the information e.g.

```
MicroEmacs 01 − Date 01/01/01 − win32
```

Description

A description of the problem. Try to include as much information as possible. Include any material necessary to reproduce the problem (i.e. macro files, text file that demonstrates problem etc). **Suggestions**

Mail as:−

Title: **SUGGEST**
Message Body: Your suggestion, macro code fragment etc.

We always appreciate suggestions, new macro code fragments etc. We do not have support for all languages, e.g. **Perl**, **Latex**... If you wish have developed new macro templates, or games (we get a bit bored with the ones that we have developed ourselves) then please mail them to `SUGGEST` and we will incorporate them into the release.

Feedback

Mail as:−

Title: **FEEDBACK**
Message Body: Your feedback.

Any general comments (which are not suggestions), your feelings about this version of MicroEmacs or any other non−technical dialogue.

Porting

Mail as:−

Title: **PORT**
Message Body: Details

If you wish to port MicroEmacs '02 to another platform and are willing to undertake responsibility for maintenance of that platform then we would like to here from you. Send us some details.

Help!

Help!

The on−line manual pages are defined as follows:−

[logo] **Title**
[link1][link2]..[linkn]

The components of the header are defined as follows:−

[logo]

The MicroEmacs '02 Logo in the top left hand corner of the screen exists on every page and hides a hypertext link. Selecting this item will take you back to the main MicroEmacs '02 page (Only present on HTML and Windows help pages).

Title

The title identifies the title of the topic that you are viewing.

[link]

The link line provides quick links to other related material in the on−line manual pages. Use these to take you to a chosen topic. The **[home]** link returns to the contents page of the currently viewed topic, this is equivalent to the **Contents** button when viewing Microsoft Windows help files. **Section Numbering**

The section numbering conventions used in these pages is defined as follows:−

(1) − Executable command line.
(2) − Editor built in commands.
(2m) − Editor built in modes.
(3) − Editor commands implemented as macros.
(4) − Editor macro language syntax
(5) − Editor variables
(8) − Editor specific file formats

Installation(1)

INSTALLATION

This page describes introductory notes for the installation and setup of MicroEmacs '02.

Quick Install

The quickest way to install MicroEmacs without reading the rest of this document is to:−

- Create a new directory i.e. `me` or `microemacs`.
- Unpack the macros archive into this directory.
- Unpack any spelling dictionaries into this directory.
- Unpack the executable into this directory.
- Run `me` from this directory.

On starting, use the mouse and configure the user from the menu bar:−

```
Help−>User Setup
```

This allows the user and screen settings to be altered. On becoming more accustomed to the editor then a fuller installation may be performed.

Getting Help

See Contact Information for full contact information. A mail archive exists at:−

```
http://groups.yahoo.com/group/jasspa/
```

If you wish to participate in the list then you must first register by sending an empty mail message body to:−

```
jasspa−subscribe@yahoogroups.com
```

You will then be able to mail any questions into the group. Registration is required in order to prevent *spam* mailings from entering into the lists.

Distribution

MicroEmacs is distributed in the following files:−

Complete Installations

The Microsoft '95/'98/NT platforms may be installed using the **Install Shield** installation utility and do not require the components specified in later sections.
`jasspame.exe` − '95/'98/NT Self Extracting Install Shield Installation

Executable Source Code

The source code release for MicroEmacs '02 contains makefiles (`*.mak`) for all supported platforms. Microsoft '95/'98/NT makefiles contain options at the top of the makefile to enable/disable console and URL support. `mesrc.zip` − Source code for all platforms
`mesrc.tar.gz` − Source code

Executable Images

`medos.zip` − DOS Executable
`mewin32.zip` − Windows 32' (95/98/NT) Executable
`mewin32s.zip` − Windows win32s (Win3.1/3.11) Executable
`meirix6.gz` − Silicon Graphics Irix 6 Executable
`meaix43.gz` − IBM's AIX 4.3 Executable
`mehpux10.gz` − Hewlett Packard HP−UX 10 Executable
`mehpux11.gz` − Hewlett Packard HP−UX 11 Executable
`mesunos55.gz` − Sun OS 5.5 Executable
`mesunos56.gz` − Sun OS 5.6 Executable
`mesolx86.gz` − Sun Solaris 2.6 Intel Platform Executable
`melinux20.gz` − Linux 2.0.0 Executable
`mefreebsd.gz` − Free BSD Executable

Help File Images (all platforms)

`mewinhlp.zip` − Windows Help file
`mehtm.zip` − HTML Help files for 8.3 file systems (.htm)
`mehtml.tar.gz` − HTML Help files (.html)

Macro File Images (all platforms)

`memacros.zip` − Macro files
`memacros.tar.gz` − Macro files

Spelling Dictionaries (all platforms)

One of the following base dictionaries is required for spelling. The extended dictionaries require the base dictionary and are recommended for a more comprehensive spelling list. Other languages are supported.

`lsdmenus.zip` − American rules and base dictionary.
`lsdxenus.zip` − American extended dictionary.
`lsdmengb.zip` − British rules and base dictionary.
`lsdxengb.zip` − British extended dictionary.
`lsdmfifi.zip` − Finnish rules and dictionary.
`lsdmfrfr.zip` − French rules and dictionary.
`lsdmdede.zip` − German rules and base dictionary.
`lsdxdede.zip` − German extended dictionary.
`lsdmitit.zip` − Italian rules and dictionary
`lsdmplpl.zip` − Polish rules and dictionary.

`lsdmptpt.zip` − Portuguese rules and dictionary.
`lsdmeses.zip` − Spanish rules and dictionary.

`lsdmenus.tar.gz` − American rules and base dictionary.
`lsdxenus.gz` − American extended dictionary.
`lsdmengb.tar.gz` − British rules and base dictionary.
`lsdxengb.gz` − British extended dictionary.
`lsdmfifi.tar.gz` − Finnish rules and dictionary.
`lsdmfrfr.tar.gz` − French rules and dictionary.
`lsdmdede.tar.gz` − German rules and base dictionary.
`lsdxdede.gz` − German extended dictionary.
`lsdmitit.tar.gz` − Italian rules and dictionary
`lsdmplpl.tar.gz` − Polish rules and dictionary.
`lsdmptpt.tar.gz` − Portuguese rules and dictionary.
`lsdmeses.tar.gz` − Spanish rules and dictionary.

NOTE: The binary versions of the executables held on the site include the platform name as part of the executable name i.e. **me** for `DOS` is called **medos.exe**. On installing the binaries onto the target machine, you should rename the executable to **me** or **me.exe**, whatever is appropriate. The ONLY exception to this rule is the Microsoft Windows executable where **mewin32.exe** should be renamed to **me32.exe**. Our reason for this naming is to allow the executables to be unpacked in the same directory and not be confused with each other.

Quick Start Guild For All Platforms

Simply create a directory, down−load the files required (see list for each platform below) and extract into this directory. From a shell or command prompt, change to the directory, making it the current one (i.e. **cd** to it), and run the executable. MicroEmacs '02 should open with the on−line help page visible.

On Windows based systems this can also be achieved by creating a short−cut and setting the Working Directory in Properties to this path.

To enable MicroEmacs to be run from any directory, simply include this directory in you **PATH** environment variable. Alternatively, copy the executable to somewhere in your PATH and set the environment variable MEPATH to point to this directory.

MicroEmacs '02 will function normally in this environment, but in multi−user environments and for up−dating purposes, it is strongly recommended that a proper installation is used, see below.

Installation

DOS

Executable:

Compiled with DJGPP V1.0

Distribution components required:

> `medos.zip`
> `memacros.zip`
> <*spelling*>`.zip`
>
> `mewinhlp.zip` if you are using windows 3.1/3.11

Recommended installed components:

`4dos` − Command shell (giving *stderr* redirection).
`grep` − Version of grep (djgpp recommended)
`make` − Version of make (djgpp recommended)
`diff` − Version of diff (djgpp recommended)

Installation:

Create the directory `c:\me` (or other location)

> Unzip the MicroEmacs components into `c:\me`
>
> Edit "`c:\autoexec.bat`" and add the following lines:−

```
SET MENAME=<name>
SET PATH=%PATH%;c:\me
SET MEPATH="c:\me"
```

> Reboot the system.
>
> > MicroEmacs may be run from the command line using

```
me
```

Graphics Cards:

MicroEmacs may be configured to the text modes of your graphics card. Refer to you graphics card DOS text modes to identify the text modes supported by your monitor. The text mode number may be entered into the user monitor configuration, defined in **Help−>User Setup**.

Running From Windows (3.x)

> The DOS version of MicroEmacs may be executed from a **.pif** file. Use the pif editor to create a new **.pif** file to launch MicroEmacs. The size of the DOS window may be configured from the command line, set the terminal size using one of the following command lines:−

```
me −c −v$TERM=E80x50        - 80 x 50 window
me −c −v$TERM=E80x25        - 80 x 25 window.
```

We usually add the `-c` option so that MicroEmacs is executed with history information. This may be omitted if required.

Windows 3.1/3.11

Executable:

Compiled with Microsoft Developer 2.0

Helper DLL:

Under **Win32s** a helper DLL **methnk16.dll** is required to perform the pipe−shell−command(2) in a synchronous manner. This should be installed into the `C:\WINDOWS\SHELL` directory. This (rather inelegantly) gets around the problems of spawning a process under **win32s** due to a number of Microsoft bugs in the operating system. Note: that on a spawn operation a `MS−DOS` window is visible, this is due to the nature of the command shell on this platform which has a tendency to prompt the user at every opportunity, hence a certain amount of interaction (which is out of our control) is necessary.

The helper DLL is compiled with a 16−bit Windows compiler − MSVC 1.5.

Distribution components required:

`mewin32s.zip`
`memacros.zip`
`mewinhlp.zip`
<*spelling*>`.zip`

Recommended installed components:

`4dos` − command shell (giving *stderr* redirection)
`grep` − Version of grep (GNU port of grep recommended)
`diff` − Version of diff (GNU port of grep recommended)
`make` − use nmake or GNU port of make.

win32s

win32s is a requirement on this platform, typically taken from **pw1118.exe** which freely available on the Internet.

Installation:

This version of Windows does not have a *install* directory as '95/'98 and it is expected that the `MS−DOS` version will coexist. No *Install Shield* installation is provided. Install in a directory structure similar to `MS−DOS`. Install the helper DLL **methnk16.dll** in the `C:\WINDOWS\SHELL` directory. Create a me32.ini(8) file in the `C:\WINDOWS` directory to identify the location of the MicroEmacs '02 components, this much the same as the '95/'98 file, change the directory paths to suite the install base.

Support Status:

The **win32s** release has not been used with vengeance, although no specific problems have been reported with this release.

Windows '95/'98/NT

Executable:

Compiled with Microsoft Developer 5.0

Install Shield

An **Install Shield** version of MicroEmacs is available which includes all of the distribution components.

Distribution components required:

`mewin32.zip`
`memacros.zip`
<spelling>`.zip`
`mewinhlp.zip` (optional)

Recommended installed components:

`4dos` or `4nt` − command shell
`grep` − Version of grep (GNU port of grep recommended)
`diff` − Version of diff (GNU port of grep recommended)
`make` − use nmake or GNU port of make.

Installation:

Create the directory "`C:\Program Files\Jasspa\MicroEmacs`" (or other location)

Unzip the MicroEmacs components into "`C:\Program Files\Jasspa\MicroEmacs`"

Create the file "`c:\windows\me32.ini`" and add the following lines:−

```
[Defaults]
mepath=C:\Program Files\Jasspa\MicroEmacs
userPath=C:\Program Files\Jasspa\MicroEmacs
fontfile=dosapp.fon
```

Create a short cut to MicroEmacs for the Desktop

Right click on the desk top

```
=> New
=> Short
=> Command Line: "c:\Program Files\Jasspa\MicroEmacs\me.exe -c"
=> Short Cut Name: "MicroEmacs"
```

MicroEmacs may be executed from the shortcut.

Open Actions

Microsoft Windows 95/98/NT provide short cut actions, assigning an open action to a file. The short cuts may be installed from the **Install Shieled** installation, but may alternativelly be explictly defined by editing the registry file with **regedit(1)**.

A file open action in the registry is bound to the file file extension, to bind a file extension *.foo* to the editor then the following registry entries should be defined:−

```
[HKEY_CLASSES_ROOT\.foo]
"MicroEmacs_foo"
[HKEY_CLASSES_ROOT\MicroEmacs_foo\DefaultIcon]
"C:\Program File\JASSPA\MicroEmacs\meicons,23"
[HKEY_CLASSES_ROOT\MicroEmacs_foo\Shell\open]
"&Open"
[HKEY_CLASSES_ROOT\MicroEmacs_foo\Shell\open\command]
"C:\Program File\JASSPA\MicroEmacs\me32.exe −o "%1""
```

In the previous exaple the *DefaultIcon* entry is the icon assigned to the file. This may be an icon taken from `meicons.exe` (in this case icon number 23), or may be some other icon. The open action in the example uses the **−o** option of the *client−server*, which loads the file into the current MicroEmacs '02 session, alternatively the **−c** option may be used to retain the previous context, or no option if a new session with no other files loaded is started.

A generic open for ALL files may be defined using a wildcard, this may be used to place a *MicroEmacs* edit entry in the right−click popup menu, as follows:−

```
[HKEY_CLASSES_ROOT\*\shell]
[HKEY_CLASSES_ROOT\*\shell\MicroEmacs]
"&MicroEmacs"
[HKEY_CLASSES_ROOT\*\shell\MicroEmacs\command]
"C:\Program File\JASSPA\MicroEmacs\me32.exe −o "%1""
```

UNIX

Executable:

Compiled with native compilers.

Distribution Components Required:

`me`*<unix>*`.gz`
`memacros.tar.gz`
<spelling>`.gz`
`html.tar.gz` (optional)

Installation:

It is recommended that all files are placed in `/usr/local`, although they may be installed locally.

> Unpack the executable and placed in "`/usr/local/bin`"
>
> Create the new directory "`/usr/local/microemacs`", unpack and install the `memacros.tar.gz` into this directory.
>
> For **csh(1)** users execute a "`rehash`" command and then me(1) can be executed from the command line.
>
> By default a X−Windows terminal is displayed, ensure that `$DISPLAY` and `$TERM` are correctly configured. To execute a terminal emulation then execute **me** with the `−n` option i.e. "`me   −n`". Note that this is not required if you are using a `vt100` emulation.

Organizing a local user profile

MicroEmacs uses local user configuration profiles to store user specific information. The user information may be stored in the MicroEmacs directory, or more typically in a users private directory. The environment variable `$MENAME` is typically used to determine the identity of the user.

The location of the user profile will depend upon your installation configuration.

Single Machine

For a single user machine it is typically easiest to use the installed MicroEmacs directory where user specific files are placed. This method, although not recommended, is simple as all files that are executed are in the same location. The `$MEPATH` is not changed.

UNIX

The UNIX environment is fairly easy and operates as most other UNIX applications. The user should create a MicroEmacs directory in their home directory for their own local configuration. Assigning a suitable name such as "`microemacs`", or if the file is to be hidden "`.microemacs`".

> The `$MEPATH` environment variable of the user should be modified to include the users MicroEmacs path BEFORE the default macros MicroEmacs path i.e.
>
> Ksh/Zsh:

```
export MEPATH=$HOME/microemacs:/usr/local/bin
```

> Csh/Bash:

```
setenv MEPATH $HOME/microemacs:/usr/local/bin
```

Where `$HOME` is defined as "`/usr/<name>`" (typically by default).

DOS/Windows

DOS and Windows are a little more tricky as numerous directories at the root level are more than a little annoying. It is suggested that the user directory is created as a sub−directory of the MicroEmacs directory. i.e.

"`c:\me\`*<user>*" for DOS

or

"`c:\Program Files\Jasspa\MicroEmacs\`*<user>*" for Windows

The `$MEPATH` environment variable (see me32.ini(8) for Windows) is modified to include the user component before the MicroEmacs component where `$MEPATH` is defined i.e.

```
SET MEPATH=c:\me\<user>;c:\me
```

where `<user>` is the user name (or `$MENAME`).

Alternative Directory Configurations

Numerous other configurations exist to organize the macro directories, to take the directory organization to the extreme then it is sometimes easiest to keep all of the macro components separate. An installation layout which encompasses different macro directories for:−

- User profiles − 1 per user.
- Shared company profiles − 1 per organization.
- MicroEmacs macros which are updated from time to time.

The configuration on different systems may be defined as follows:−

UNIX

The shared files are placed in `/usr/local`

```
/usr
   \
  local
     \
  microemacs   - Spelling + standard macros
       \
     company   - Company specific files
```

The user profile is stored in the users directory

```
/usr
  \
<name>
```

```
              \
           microemacs    - User specific files
```

The user should configure the `$MEPATH` as:

```
MEPATH=$(HOME)/microemacs:/usr/local/microemacs/company:/usr/local/microema
```

DOS/WINDOWS

For DOS and MS−Windows environments, bearing in mind the problem of the root directory, then it is easier to use the "`me`" directory as a place holder for a number of sub−directories, using a configuration such as:−

```
            c:
            |
            me      - Place holder directory
          / | \
        /   |   \
     <name> macros company
```

The user should configure the `$MEPATH` as:−

```
SET MEPATH=c:\me\<name>;c:\me\company;c:\me\macros
```

User Profile Files

Files contained in the user profiles typically include:−

<*name*>`.emf` − The users start up profile.
<*name*>`.edf` − The users spelling dictionary.
<*name*>`.erf` − The users registry configuration file.

These files are established from the menu "**Help−>User Setup**". The "**Setup Path**" item defines the location of the files, but must be MANUALLY included in the `$MEPATH` environment.

Company Profiles

Company profiles include standard files and extensions to the standard files which may be related to a company, this is typically <*company*>.emf where <*company*> is the name of the company.

The directory may also include template files etf(8) files which defines the standard header template used in the files. Files in the "`company`" directory would over−ride the standard template files.

The company directory should be added to the `$MEPATH` after the user profile and before the MicroEmacs standard macro directory.

SEE ALSO

$MENAME(5), $MEPATH(5), Company Profiles, File Hooks, File Language Templates, User Profiles.

User Profiles(2)

USER PROFILES

This section describes how a user profile should be incorporated into MicroEmacs '02. A user profile defines a set of extensions to MicroEmacs which encapsulates settings which are used by an individual user.

The user profile allows:−

- Saving of the last session (history), allowing the next invocation of MicroEmacs '02 to restore your previous session.
- Personalized spelling dictionaries.
- Redefinition of MicroEmacs '02, allowing the editor to be tailored to an individual's requirements. Including the re−binding of keys, modification of the screen colors. Definition of personal macros etc.

Identification

In order to identify a user MicroEmacs '02 uses information in the system to determine the name of the user, and in turn the configuration to use. On all systems the value of the environment variable $MENAME(5) takes priority over any other means of user identification. If this variable is not defined then the host system typically provides a mechanism to determine the current user. DOS and *Windows* systems present problems where a login prompt is not supplied.

Each of the supported platforms are now described.

UNIX

The environment variable `$LOGNAME` is defined. This is the user name used by the system.

DOS

MS−DOS typically has no concept of the user name. The user name should be defined in the `autoexec.bat` file, choose a name of 8 characters or less, i.e. to fix the user name to `fred` then add the following line:−

```
SET MENAME=fred
```

Remember to re−boot the system before the new command takes effect. (see the next step, there is another change to `autoexec.bat`).

Microsoft Windows

Microsoft windows environments may, or may not, have logging enabled. If you have to log into your system then a login identification has been supplied and will be recognized by MicroEmacs, setting the environment variable $MENAME(5) to this value.

If login is not enabled then the me32.ini(8) file may be modified to provide a default login name. To add the user **fred** then add the following lines to the *ini* file:−

```
[guest]
MENAME=fred
```

If login is subsequently enabled on the system then these lines should be removed. These lines force the user identification to be **fred**.

The above technique may be used within the windows environment to modify your login name. Assuming that the system administrator has assigned **fred** a user login name of **fwhite**, and *fred* requires all of his configuration files to be the same name as his UNIX login which is **fred**. Then *fred* may force his user name to *fred* from the `me32.ini` file as follows:−

```
[fwhite]
MENAME=fred
```

Once *fred* has entered MicroEmacs he will adopt his new login name which will be used to identify his own files etc. The action of this statement is to force the environment variable `$MENAME` to a new value. Any other environment variables may be forced in this way i.e. `$HOSTNAME` is a good candidate here as the `me32.ini` is local to the machine.

Shared Platforms

Platforms may share the same set of configuration files. Consider a system which may boot under MS−DOS, Windows '98, NT and Linux. Provided that the macro files are located on a file system that may be mounted by all of the other operating systems and the $MEPATH is set appropriately, then a single set of MicroEmacs macro files may be shared across all platforms. **Personal MicroEmacs Directory**

The private user profile is stored in a separate directory. The directory that MicroEmacs uses must be created by the user, create the directory in your local file system. In addition, the MicroEmacs search path $MEPATH(5) should be modified to include your new MicroEmacs personal directory.

UNIX

Create in your local directory, typically called `microemacs` or `.microemacs` (if it is to be hidden).

Add/modify the $MEPATH(5) environment variable to include your personal directory in your `.login`, `.chsrc` or `.profile` file, the file and exact syntax will depend upon your shell. For a Korn shell the following line would be added to the `.profile` file:−

```
export MEPATH=$HOME/.microemacs:/usr/local/microemacs
```

Where `$HOME` is assumed to be the users login home directory, or use the directory location of your new directory.

DOS

For MS−DOS environments, there is typically no user directory, it is suggested that the user directory is created in the MicroEmacs directory, use the `$MENAME` defined in the previous step i.e.

```
mkdir c:\me\fred
```

Change the $MEPATH(5) in the **autoexec.bat** to include the new directory i.e.

```
SET MEPATH=c:\me\fred;c:\me
```

Windows

Windows environments, the me32.ini(8) **userPath** entry defines the location of the user profile directories, within the **Install Shield** installation, the `me32.ini` is typically defined as:−

```
userPath=C:\Program Files\JASSPA\MicroEmacs
```

Create your MicroEmacs personal directory in this folder, the name of the folder should be your login name or `$MENAME`, depending upon how your name is identified.

Creating Your Profile

Once you have created a new directory to store your user profile, create a default profile for yourself from MicroEmacs using the user−setup(3) dialog:−

```
Help => User Setup
```

Fill in the entries in the dialog, and ensure that **Save** is depressed on exit to write the files.

The dictionaries often present difficulties the first time, a prompt to save the dictionary requires the full pathname and the name of the file, the pathname is the path to your personal folder, the filename is typically your *username*.edf. Once the file is created you will not have a problem in the future.

The User Profile

Files created in the user directory include:−

- Setup registry and previous session history *username*.erf, see erf(8)). This stores the **user−setup** settings and also the context from your previous edit session.
- Users start−up file *username*.emf, see emf(8) the user may make local changes to MicroEmacs in this file, this may include changing key bindings, defining new hook functions etc. You should over−ride the standard MicroEmacs settings from your start−up file rather than modifying the standard MicroEmacs files.
- Personal spelling dictionary *username.edf*, see edf(8). This file contains your personal spelling modifications, any words that are added to the spelling dictionary are added to this file.

In addition to the above, if new file hooks are defined then they should be added to this directory (if they are not global to the company).

EXAMPLE

The following are examples of some individuals start−up files:−

```
; Jon's special settings
;
; Last Modified <190698.2226>
;
; Macro to delete the whitespace, or if an a word all of the
; word until the next word is reached.
define-macro super-delete
    set-variable #l0 0
    !while &not &sin @wc " \t\n"
        forward-char
        set-variable #l0 &add #l0 1
    !done
    !repeat
        !force forward-char
        !if $status
            set-variable #l0 &add #l0 1
        !endif
    !until &or &seq @wc "" &not &sin @wc " \t\n"
    #l0 backward-delete-char
    !return
!emacro
; Make a previous-buffer command.
define-macro previous-buffer
    &neg @# next-buffer
!emacro
; spotless; Perform a clean and remove any multi-blank lines.
define-macro spotless
    -1 clean
!emacro
; comment-adjust; Used for comments in electric-c mode (and the other
; electic modes. Moves to the comment fill position, saves having to mess
; around with comments at the end of the line.
0 define-macro comment-adjust
    ; delete all spaces up until the next character
    !while &sin @wc " \t"
        forward-delete-char
    !done
    ; Fill the line to the current $c-margin. We use this as
    ; this is the only variable that tells us where the margin
    ; should be.
    !if &gre $window-acol 0
        backward-char
        !if &sin @wc " \t"
            forward-delete-char
            !jump -4
        !else
            forward-char
        !endif
    !endif
    ; Now fill to the $c-margin
    &sub $c-margin $window-acol insert-string " "
!emacro
; Macro to force buffer to compile buffer for C-x '
define-macro compile-error-buffer
    !force delete-buffer *compile*
    change-buffer-name "*compile*"
!emacro
;
; Set up the bindings.
;
global-bind-key super-delete            "C-delete"
global-bind-key beginning-of-line       "home"
global-bind-key end-of-line             "end"
global-bind-key undo                    "f4"
!if &seq %emulate "ERROR"
    global-bind-key comment-adjust      "esc tab"
    global-bind-key comment-adjust      "C-insert"
    ; Like a korn shell please.
    ml-bind-key tab "esc esc"
!endif
;
; Setup for windows and UNIX.
;
; Define my hilighting colour for Windows and UNIX.
!if &equ &band $system 0x001 0
    !if &not &seq $platform "win32"
        ; Small bold font is better for me.
        change-font "-*-clean-medium-r-*-*-*-130-*-*-*-*-*-*"
        ; Small non-bold font.
        ; change-font "-misc-fixed-medium-r-normal--13-*-*-*-c-70-iso8859-1"
        ; Change the size of the screen
        82 change-frame-width
        50 change-frame-depth
    !endif
!endif
; Change the default diff command-line for GNU diff utility all platforms
set-variable %diff-com "diff --context --minimal --ignore-space-change --report-id
set-variable %gdiff-com "diff --context --ignore-space-change -w"
; Setup for cygnus
!if &seq $platform "win32"
    set-variable %cygnus-bin-path "c:/cygwin/bin"
    set-variable %cygnus-hilight 1
    set-variable %cygnus-prompt "$"
!endif
; Set up the ftp flags. The letters have the following meaning:
; c   - Create a console (*ftp-console* for ftp, *http-console* for http)
; s   - Show the console
; p   - Show download progress ('#' every 2Kb downloaded)
set-variable %ftp-flags "csp"
; Info files
;To hilight the .info and also the dir file
add-file-hook ".info dir"                                fhook-info   ; Info-fi
;To hilight all info files without the extension .info
;but starting with the text "This is info file..
-2 add-file-hook "This is Info file"                     fhook-info

; Finished
ml-write "Configured to Jon's requirements"
```

SEE ALSO

$MEPATH(5), $MENAME(5), user−setup(3), Company Profiles, File Hooks, File Language Templates, Installation.

CompanyProfiles(2)

COMPANY PROFILES

This section describes how a company profile should be incorporated into MicroEmacs '02. A company profile defines a set of extensions to MicroEmacs which encapsulate settings which are used on a company wide basis. This type of configuration is typically used with a networked (shared) installation. The company profile would typically include:−

- Name of the company.
- Standard header files including company copyright statements.
- Standard file layouts
- Company defined language extensions.

Location Of The Company Information

It is suggested that all of the company extensions applied to MicroEmacs '02 are performed in a separate directory location which shadows the MicroEmacs standard macro file directory. This enables the original files to be sourced if a user does not want to include the company files. This method also allows MicroEmacs to be updated in the future, whilst retaining the company files. For our example, we shall use a company called **JASSPA**, you should replace references to *jasspa* with your own company name. The steps involved are laid out as follows:−

Create a new company directory

You may skip this step if you are going to modify the standard installation.

Create a new directory to hold the company information. i.e.

`/usr/local/microemacs/jasspa` − UNIX
`c:\Program Files\JASSPA\MicroEmacs\jasspa` − Microsoft

Modify the $MEPATH(5) of the (of all users) to include the company directory on the search path i.e.

UNIX

Users edit their local `$MEPATH` or a base $MEPATH is added to the system `.login` or `.profile` scripts.

```
MEPATH=/usr/local/microemacs
MEPATH=/usr/local/microemacs/jasspa:$MEPATH
```

Microsoft Windows Platforms

Edit the `me32.ini` file and modify the `mepath` entry to reflect the location of the company directory:−

```
mepath=C:\Prog....\Mic...\macros\jasspa;C:\Prog...\Mic...\
```

DOS Platforms

Edit the **autoexec.bat** file and modify `MEPATH` to include the company directory location.

```
SET MEPATH=c:\me\jasspa;c:\me
```

Content Of The Company Information

Company macro file

The company file is typically called by the company name (i.e. `jasspa.emf`) create a new company file. The file includes your company name and hook functions for any new file types that have been defined for the company, an example company file for **Jasspa** might be defined as:−

```
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;
;  Author        : Jasspa
;  Created       : Thu Jul 24 09:44:49 1997
;  Last Modified : <190698.2225>
;
;  Description     Extensions for Jasspa
;
;  Notes
;
;  History
;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
; Define the name of the company.
set-variable %company-name "Jasspa"
; Add Jasspa specific file hooks
; Make-up foo file hook
add-file-hook ".foo"    fhook-foo
1 add-file-hook "-!-[ \t]*foobar.*-!-" fhook-foo ; -!- foobar -!-
; Override the make with localised build command
set-variable %compile-com "build"
```

The file contains company specific file hooks and the name of the company.

Other Company Files

Files defined on behalf of the company are included in the company directory. These would include:−

- Template header files etf(8).
- Hook file definitions (**hk***XXX***.emf**) for company specific files, see

add−file−hook(2).

- Extensions to the standard hook definitions (**my*XXX*.emf**) for company specific language extensions to the standard hook files. See File Hooks and File Language Templates.

SEE ALSO

$MENAME(5), $MEPATH(5), File Hooks, File Language Templates, Installation, user−setup(3), User Profiles.

MainMenu(3)

NAME

Main Menu − The top main menu

SYNOPSIS

n osd

DESCRIPTION

The main menu is provided to give an easier access to parts of MicroEmacs functionality, the menu is not burnt into MicroEmacs but defined on start−up in `me.emf` and `osd.emf`. The user−setup(3) command can be used to set whether the menu is always visible and if the Alt−Hotkeys are enabled (i.e. '`A−f`' to open the **File** menu).

The main menu is osd(2) dialog number `0` so key bindings can be made which will open the main menu, an argument of `0` will simply open the main menu, an argument of `0x0n0000` will not only open the main menu but also the `n`th sub menu, e.g. to open the edit menu use:

```
          0x020000 osd
```

Following is a brief description of the main menu items:

File Menu

`New`

Changes the current buffer to a new buffer.

`Open`

Opens a dialog enabling the user to select files for opening into MicroEmacs. By default the dialog opens the selected file using command find−file(2), but if the view option is selected the view−file(2) command is used. The binary or encrypt options configure whether the files are to be loaded with binary(2m) or crypt(2m) modes enabled.

`Quick Open`

Opens a sub−menu list all user file types (defined in user−setup(3)). Selecting one will open another sub−dialog list all files of that type in the current directory, selecting a file will open it using command find−file(2).

`Favorites`

Opens a sub−menu enabling the user to add new favorite files, edit the existing list of favorite files, or select an existing favorite file in which case the file is opened using command find−file(2). The favorite file using to store the list is "**$MENAME**`.eff`" and is saved in the first path given in the $search−path(5). Each favorite file takes 2 lines in the file, the first is the text displayed in the dialog (note that characters '\' and '&' must be protected with a '\' and the '&' can be used to set the Hot key) and the second line is the file name. A line with a single '**−**' character creates a separater line in the dialog.

`Find Tag`

Only visible when a `tags` file is found in the current directory, the command jumps to the current tag or if not on a tag or the tag is not found, opens a dialog enabling the user to select a tag. See command find−tag(2) for more information.

`Find File`

Executes command file−browser(3).

`FTP`

Executes command ftp(3).

`Close`

Executes a dialog form of the command delete−buffer(2).

`Attributes`

Opens a dialog enabling the user to set the current buffers file attributes. See command file−attrib(3) for more information.

`Save`

Executes a dialog form of the command save−buffer(2).

`Save As`

Executes a dialog form of the command write−buffer(2).

`Save All`

Executes a dialog form of the command save−all(3).

`Printer Setup`

Opens a dialog which enables the user to configure the printer driver, output location and page layout (executes command print−setup(3)).

`Print`

Executes command print−buffer(2).

`Buffer`

Opens a sub−menu listing all created buffers, selecting one will change the current buffer to the selected one.

`Exit`

Executes command save−buffers−exit−emacs(2). **Edit Menu**

`Undo`

Undoes the last edit in the current buffer (executes command undo(2)).

`Redo`

Redo the last undo, only available immediately after an undo. This is also done via the undo(2) command.

`Undo All`

Undo all edits in the current buffer until the last save or no more undo history is available. Executes the command undo(2) with a 0 numerical argument.

`Set Mark`

Executes command set−mark(2).

`Cut`

Executes command kill−region(2).

`Copy`

Executes command copy−region(2).

`Paste`

Executes command yank(2).

`Narrow Out`

Executes command narrow−buffer(2) with a numeric argument of 4.

`Narrow To`

Executes command narrow−buffer(2) with a numeric argument of 3.

`Remove Single Narrow`

Executes command narrow−buffer(2) with a numeric argument of `2`.

`Remove All Narrows`

Executes command narrow−buffer(2) with a numeric argument of `1`. **Search Menu**

`Search`

Executes a dialog form of the command isearch−forward(2).

`Replace`

Executes a dialog form of the command query−replace−string(2).

`Hilight Search`

Opens another dialog which can be used to add and remove hilighting of individual lines in the current buffer. Note that setting a line hilight is a temporary change, it will not effect any files etc and will be lost when the buffer is deleted.

`Goto Line`

Executes a dialog form of the command goto−line(2).

`Goto Fence`

Executes command goto−matching−fence(2).

`Set Bookmark`

Executes command set−alpha−mark(2).

`Goto Bookmark`

Executes command goto−alpha−mark(2). **Insert Menu**

`Symbol`

Executes command symbol(3).

`Date & Time`

Opens a dialog with the current date and time in a selection of common formats; selecting one of these will insert the string into the current buffer at the current position. Note that the format text strings depend on the current language (Default and American languages use the order MM−DD−YY

etc whereas the rest use DD−MM−YY). The names used for the day and month names can be defined using the Setup page of Organizer(3).

`File`

Executes command insert−file(2).

`File Name`

Executes command insert−file−name(2).

`Macro...`

Executes command insert−macro(2). **Format Menu**

`Restyle Buffer`

Executes command restyle−buffer(3).

`Restyle Region`

Executes command restyle−region(3).

`Clean Buffer`

Executes command clean(3).

`Change Buffer Char Set`

Executes command charset−change(3).

`IQ Fill Paragraph`

Executes command ifill−paragraph(3).

`Fill Paragraph`

Executes command fill−paragraph(2).

`Fill All Paragraphs`

Executes command fill−paragraph(2) with a very large positive numerical argument. Note that this only effects paragraphs from the current position onwards.

`Paragraph to Line`

Executes command paragraph−to−line(3).

All Paragraphs to Line

Executes command paragraph−to−line(3) with a very large positive numerical argument. Note that this only effects paragraphs from the current position onwards.

Sort Lines

Executes command sort−lines(2).

Ignore Case Sort Lines

Executes command sort−lines−ignore−case(3).

Capitalize Word

Executes command capitalize−word(2).

Lower Case Word

Executes command lower−case−word(2).

Lower Case Region

Executes command lower−case−region(2).

Upper Case Word

Executes command upper−case−word(2).

Upper Case Region

Executes command upper−case−region(2). **Execute Menu**

Execute Command

Executes command execute−named−command(2).

Execute Buffer

Executes command execute−buffer(2).

Execute File

Executes command execute−file(2).

Start Kbd Macro

Executes command start−kbd−macro(2).

`Query Kbd Macro`

Executes command kbd−macro−query(2).

`End Kbd Macro`

Executes command end−kbd−macro(2).

`Execute Kbd Macro`

Executes command execute−kbd−macro(2).

`Name Kbd Macro`

Executes command name−kbd−macro(2).

`Ipipe command`

Executes command ipipe−shell−command(2).

`Shell`

Executes command shell(2). **Tools Menu**

`Current Buffer Tools`

For some file formats MicroEmacs provides a file format specific set of tools, see the file type help page for more specific information.

`Count Words`

Executes command count−words(2).

`Spell Word`

Executes command spell−word(3).

`Spell Buffer`

Executes command spell−buffer(3).

`Word Complete`

Takes the incomplete word to the left of the cursor and attempts to complete the word by using the users current language dictionary. Executes command expand−word(3).

`Compare Windows`

Executes command compare−windows(2).

`Compile`

Executes command compile(3).

`Grep`

Executes command grep(3).

`Graphical Diff`

Executes command gdiff(3).

`Diff`

Executes command diff(3).

`Diff Changes`

Executes command diff−changes(3).

`Organizer`

Executes command organizer(3).

`Mail`

Executes command mail(3).

`View Mail`

Executes command vm(3).

`More...`

Opens a sub−menu with a collection of other useful miscellaneous tools. **Window Menu**

`Split Window V`

Executes command split−window−vertically(2).

`Grow Window V`

Executes command change−window−depth(2) with an argument of 1.

`Shrink Window V`

Executes command change−window−depth(2) with an argument of −1.

`Split Window H`

Executes command split−window−horizontally(2).

`Grow Window H`

Executes command change−window−width(2) with an argument of 1.

`Shrink Window H`

Executes command change−window−width(2) with an argument of −1.

`One Window`

Executes command delete−other−windows(2).

`Delete Window`

Executes command delete−window(2).

`Previous Window`

Executes command previous−window(2).

`Next Window`

Executes command next−window(2).

`Create New Frame`

Create an new external frame, only available on version which support multiple−window frames. Executes command create−frame(2).

`Create New Frame`

Closes the current frame, only available on version which support multiple−window frames. The command will fail if this is the only frame, use File −> Exit to exit MicroEmacs, executes command delete−frame(2).
Help Menu

`Curr Buffer Help`

For some file formats MicroEmacs provides a file format specific help page giving details of key−bindings and tools specific to the current buffers file type.

`General Help`

Executes command osd−help(3).

```
Help on Command
```

Executes command help−command(2).

```
Help on Variable
```

Executes command help−variable(2).

```
Describe Bindings
```

Executes command describe−bindings(2).

```
Describe key
```

Executes command describe−key(2).

```
Describe Variable
```

Executes command describe−variable(2).

```
Describe Word
```

Executes command describe−word(3).

```
List Buffers
```

Executes command list−buffers(2).

```
List Commands
```

Executes command list−commands(2).

```
List Registry
```

Executes command list−registry(2).

```
List Variables
```

Executes command list−variables(2).

```
Command Apropos
```

Executes command command−apropos(2).

```
Buffer Setup
```

Executes command buffer−setup(3).

`User Setup`

Executes command user−setup(3).

`Scheme Editor`

Executes command scheme−editor(3).

`Games`

Opens a sub−menu listing all available games, see Games for more information.

`Product Support`

Opens on−line Contact information.

`About MicroEmacs`

Executes command about(2). **NOTES**

The main menu is defined using osd(2) in macro files me.emf and osd.emf.

General user extensions to the main menu can be added to the user file `myosd.emf` which is executed once when the main menu is first opened. The macro file can add new items to any of the main sub menus and can delete most existing items (some are dynamically added when appropriate, these should not be deleted). See `osd.emf` for examples of how to add items to the menu.

New sub−menus should be added in the company or user setup files as this must be done at start−up. The content on the menu is not required until the main menu is used so populating the new sub−menu can be done in `myosd.emf`.

SEE ALSO

user−setup(3).

Essential Commands

ESSENTIAL COMMANDS

The very essential commands which are the most important commands to know include:

abort−command(2) (**C−g**) Abort command
backward−char(2) (**C−b**) Move the cursor left
backward−delete−char(2) (**backspace**) Delete the previous character at the cursor position
backward−line(2) (**C−p**) Move the cursor to the previous line
file−browser(3) (**f10**) Browse the file system
file−browser−close(3) Close the file−browser
file−browser−swap−buffers(3) Swap between file−browser windows
forward−char(2) (**C−f**) Move the cursor right
forward−delete−char(2) (**C−d**) Delete the next character at the cursor position
forward−line(2) (**C−n**) Move the cursor to the next line
isearch−forward(2) (**C−s**) Search forward incrementally (interactive)
quick−exit(2) (**esc z**) Exit the editor writing changes
save−buffer(2) (**C−x C−s**) Save contents of changed buffer to file
save−buffers−exit−emacs(2) (**esc z**) Exit the editor prompt user to write changes
undo(2) (**C−x u**) Undo the last edit

Help Information

HELP INFORMATION

Commands to retrieve on−line help information and status.

about(2) Information About MicroEmacs
command−apropos(2) (**C−h a**) List commands involving a concept
describe−key(2) (**C−x ?**) Report keyboard key name and binding
describe−variable(2) (**C−h v**) Describe current setting of a variable
help(2) (**esc ?**) Help; high level introduction to help
help−command(2) (**C−h C−c**) Help; command information
help−item(2) (**C−h C−i**) Help; item information
help−variable(2) (**C−h C−v**) Help; variable information
info(3) Display a GNU Info database
info−goto−link(3) Display Info on a given link
info−on(3) Display Info on a given topic
list−buffers(2) (**C−x C−b**) List all buffers and show their status
osd−help(3) GUI based on−line help

Bindings(2)

DEFAULT KEY BINDINGS

The default key bindings are presented below in four alphabetical lists, one for single key bindings and one for each of the 4 bound prefixes (`esc`, `C-x`, `C-h` & `C-c`). See Key Names for a list of valid key names.

Single−Key Sequences

`backspace` backward−delete−char Delete the previous character.
`delete` forward−delete−char Delete character under the cursor.
`down` forward−line Move to next line.
`end` end−of−buffer Move to the end of the buffer.
`esc` prefix 1 Meta character prefix.
`f1` osd Open top main menu.
`home` beginning−of−buffer Move to the start of the buffer.
`insert` buffer−mode Toggle over−write mode.
`left` backward−char Move backward one character (left).
`page-down` scroll−down Move forward by one screen.
`page-up` scroll−up Move backward by one screen.
`return` newline Insert a new line.
`right` forward−char Move forward one character (right).
`tab` tab Insert a tab character.
`up` backward−line Move to previous line.

`S-backspace` backward−delete−char Delete the previous character.
`S-delete` forward−delete−char Delete character under the cursor.
`S-tab` backward−delete−tab Delete white space to previous tab−stop.

`C-a` beginning−of−line Move to beginning of line.
`C-b` backward−char Move backwards by one character
`C-c` prefix Control character prefix.
`C-d` forward−delete−char Delete character under the cursor.
`C-e` end−of−line Move to end of line.
`C-f` forward−char Move forward one character (right).
`C-g` abort−command Abort current command.
`C-h` prefix Control character prefix.
`C-i` insert−tab Insert tab character.
`C-k` kill−line Delete from cursor to the end of the line.
`C-l` recenter Redraw screen with current line in the center.
`C-m` newline Insert a new line.
`C-n` forward−line Move to next line (down).
`C-o` insert−newline Open up a blank line.
`C-p` backward−line Move to previous line (up).
`C-q` quote−char Insert literal character.

`C-r` isearch−backward Start incremental search backwards.
`C-s` isearch−forward Start incremental search forwards.
`C-t` transpose−chars Transpose two letters.
`C-u` universal−argument Repeat the next command *n* times (default is 4).
`C-v` scroll−down Move forward by one screen.
`C-w` kill−region Delete a marked region.
`C-x` prefix Control character prefix.
`C-y` yank Restore what was copied or deleted.
`C-z` scroll−up Move backward by one screen.
`C-_` undo Undo the previous edit.
`C-down` forward−line Move forward five lines.
`C-left` backward−word Move one word backward.
`C-page-down` scroll−next−window−down Scroll next window down a page.
`C-page-up` scroll−next−window−up Scroll the next window up a page.
`C-right` forward−word Move one word forward.
`C-up` backward−line Move backward 5 lines.

`A-e` file−browser Browse the file system.
`A-r` replace−all−string Replace string with new string in a list of files.
`A-down` scroll−down Scroll the current window down one line.
`A-left` scroll−left Scroll the current window left one character.
`A-right` scroll−right Scroll the current window right one character.
`A-up` scroll−up Scroll the current window up one line.

esc Prefix Sequences

`esc !` pipe−shell−command Pipe a shell command to a buffer.
`esc $` spell−word Spell a word.
`esc .` set−mark Set the start of a region.
`esc /` execute−file Execute script lines from a file.
`esc <` beginning−of−buffer Move to the start of the buffer.
`esc >` end−of−buffer Move to the end of the buffer.
`esc ?` help Help − high level introduction to MicroEmacs.
`esc @` pipe−shell−command Pipe a shell command to a buffer.
`esc [` backward−paragraph Goto the beginning of the paragraph.
`esc \` ipipe−shell−command Incrementally pipe a shell command to a buffer.
`esc ]` forward−paragraph Move forward one paragraph
`esc ^` delete−indentation Join 2 lines deleting white spaces.
`esc b` backward−word Move one word backwards
`esc c` capitalize−word Capitalize first letter of a word
`esc d` forward−kill−word Delete word the cursor is on.
`esc e` set−encryption−key Reset the encryption key.
`esc f` forward−word Move one word forward.
`esc g` goto−line Goto a line.
`esc i` tab Insert a tab character.
`esc k` global−bind−key Bind a key to a command or macro.
`esc l` lower−case−word Lowercase word.
`esc m` global−mode Toggle a global mode.

esc n forward−paragraph Move forward one paragraph
esc o fill−paragraph Reformat (fill) current paragraph.
esc p backward−paragraph Goto the beginning of the paragraph.
esc q ifill−paragraph Reformat (fill) current paragraph.
esc r replace−string Search and replace text (no query).
esc t find−tag Find a tag.
esc u upper−case−word Uppercase word.
esc v scroll−down Move to the previous screen.
esc w copy−region Copy region to the kill buffer.
esc x execute−named−command Execute the named command.
esc y reyank Kill current yank data and restore previous kill buffer data.
esc z quick−exit Save all buffers and exit.

esc ~ buffer−mode Remove edited status from current buffer.
esc backspace backward−kill−word Delete the word under the cursor.
esc esc expand−abbrev Expand an abbreviation.
esc space set−mark Set the start of a region.

esc C−c count−words Count words in a region.
esc C−f goto−matching−fence Reposition the cursor at an opposing bracket.
esc C−g abort−command Abort current command.
esc C−i tab Insert tab character.
esc C−k global−unbind−key Unbind a key from a command or macro
esc C−n change−buffer−name Rename current buffer.
esc C−r query−replace−string Search and replace with query.
esc C−v scroll−next−window−down Scroll next window down a page.
esc C−w kill−paragraph Delete current paragraph.
esc C−z scroll−next−window−up Scroll the next window up a page.

esc A−r query−replace−all−string Query replace string in a list of files.

C−x Prefix Sequences

C−x # filter−buffer Filter the buffer through a shell filter.
C−x (start−kbd−macro Start recording a keyboard macro.
C−x) end−kbd−macro Stop recording a keyboard macro.
C−x / isearch−forward Start incremental search forwards.
C−x 0 delete−window Delete the current window.
C−x 1 delete−other−windows Delete other windows.
C−x 2 split−window−vertically Split the current window into two.
C−x 3 next−window−find−buffer Find a buffer into the next window, split if necessary.
C−x 4 next−window−find−file Load a file into the next window, split if necessary.
C−x 5 split−window−horizontally Split the current window horizontally into two.
C−x 9 find−bfile Find and load a file for binary editing.
C−x < scroll−left Scroll the window left by one screen width.
C−x = buffer−info Show cursor position information
C−x > scroll−right Scroll the window right by one screen width.
C−x ? describe−key Describe binding of command to key.

`C-x @`	pipe−shell−command	Pipe a shell command to buffer.
`C-x [`	scroll−up	Move backward by one screen.
`C-x ]`	scroll−down	Move forward by one screen.
`C-x ^`	grow−window−vertically	Enlarge the current window by a line.
``C-x ` ``	get−next−line	Find the next command line.
`C-x a`	goto−alpha−mark	Move the cursor to an alphabetic mark.
`C-x b`	find−buffer	Switch window to a buffer.
`C-x c`	shell	Start a new command processor.
`C-x e`	execute−kbd−macro	Execute a macro.
`C-x h`	hunt−forward	Continue search in forward direction.
`C-x k`	delete−buffer	Delete buffer.
`C-x m`	buffer−mode	Toggle a local buffer mode.
`C-x n`	change−file−name	Rename current buffer file name.
`C-x o`	next−window	Move to the next window.
`C-x p`	previous−window	Move to the previous window.
`C-x q`	kbd−macro−query	Query keyboard macro.
`C-x r`	search−backward	Search in a reverse direction.
`C-x s`	search−forward	Search in a forward direction.
`C-x u`	undo	Undo the previous edit.
`C-x v`	set−variable	Assign a new value to a variable.
`C-x w`	resize−window−vertically	Resize the window.
`C-x x`	next−buffer	Switch to the next buffer.
`C-x z`	grow−window−vertically	Enlarge the current window.
`C-x {`	shrink−window−horizontally	Shrink current window horizontally.
`C-x }`	grow−window−horizontally	Enlarge current window horizontally.

`C-x C-a`	set−alpha−mark	Mark the current position with an alphabetic mark.
`C-x C-b`	list−buffers	Display buffer list.
`C-x C-c`	save−buffer−exit−emacs	Exit MicroEmacs '02.
`C-x C-d`	change−directory	Change the current working directory.
`C-x C-e`	execute−kbd−macro	Execute a macro.
`C-x C-f`	find−file	Find a file and load into buffer.
`C-x C-g`	abort−command	Abort current command.
`C-x C-h`	hunt−backward	Resume search in backwards direction.
`C-x C-i`	insert−file	Insert file into the current buffer.
`C-x C-l`	lower−case−region	Lowercase region.
`C-x C-n`	scroll−down	Scroll the current window down one line.
`C-x C-o`	delete−blank−lines	Delete blank lines about the cursor.
`C-x C-p`	scroll−up	Scroll the current window up one line.
`C-x C-q`	rcs−file	Interact with RCS to check in/out a file.
`C-x C-r`	read−file	Read a file from disk.
`C-x C-s`	save−buffer	Save current file to disk.
`C-x C-t`	transpose−lines	Swap adjacent lines.
`C-x C-u`	upper−case−region	Uppercase region.
`C-x C-v`	view−file	Read a file for viewing (read only).
`C-x C-w`	write−buffer	Write a file to disk witn new name.
`C-x C-x`	exchange−point−and−mark	Exchange cursor with mark position.
`C-x C-y`	insert−file−name	Insert filename into current buffer.
`C-x C-z`	shrink−window−vertically	Reduce size of current window.

C−h Prefix Sequences

`C−h a`	command−apropos	List commands involving a concept.
`C−h b`	describe−bindings	Show current command/key binding.
`C−h c`	list−commands	List available commands.
`C−h d`	describe−variable	Describe current setting of a variable.
`C−h k`	describe−key	Describe keyboard binding.
`C−h v`	list−variables	List defined variables.
`C−h C−c`	help−command	Display command help information.
`C−h C−i`	help−item	Display item help information.
`C−h C−v`	help−variable	Display variable help information.

File Handling Commands

FILE HANDLING COMMANDS

Commands to read, write and interact with files:

Commands

append−buffer(2) Write contents of buffer to end of named file
change−directory(2) [**C−x C−d**] Change the current working directory
change−file−name(2) (**C−x n**) Change the file name of the current buffer
directory−tree(2) Draw the file directory tree
file−attrib(3) Set the current buffers system file attributes
file−browser(3) (**f10**) Browse the file system
file−browser−close(3) Close the file−browser
file−browser−swap−buffers(3) Swap between file−browser windows
file−op(2) File system operations command
find−bfile(3) (**C−x 9**) Load a file as binary data
find−cfile(3) Load a crypted file
find−file(2) (**C−x C−f**) Load a file
find−zfile(3) Compressed file support
ftp(3) Initiate an FTP connection
insert−file(2) (**C−x C−i**) Insert file into current buffer
insert−file−name(2) (**C−x C−y**) Insert filename into current buffer
read−file(2) (**C−x C−r**) Find and load file replacing current buffer
reread−file(3) Reload the current buffer's file
save−all(3) Save all modified files (with query)
save−buffer(2) (**C−x C−s**) Save contents of changed buffer to file
save−some−buffers(2) Save contents of all changed buffers to file (with query)
set−encryption−key(2) (**esc e**) Define the encryption key
suspend−emacs(2) Suspend editor and place in background
view−file(2) (**C−x C−v**) Load a file read only
write−buffer(2) (**C−x C−w**) Write contents of buffer to named (new) file
zfile−setup(3) Compressed file support setup

Variables

$auto−time(5) Automatic buffer save time
$file−ignore(5) File extensions to ignore
$home(5) Users `home' directory location
$kept−versions(5) Number of backups to be kept
$timestamp(5) Time stamp string
%ftp−flags(5) Configure the FTP console
%http−flags(5) Configure the HTTP console
%http−proxy−addr(5) Set HTTP proxy server address

%http−proxy−port(5) Set HTTP proxy server port
Dialogs and Menus

Dialogs and Menus

DIALOGS AND MENUS

Menus and dialogs in the system:

Commands

buffer−help(3) Displays help page for current buffer
buffer−setup(3) Configures the current buffer settings
describe−word(3) Display a dictionary definition of a word
find−word(3) Find a using spelling dictionaries
generate−tags−file(3) Generate a tags file
line−scheme−search(3) Search and annotate the current buffer
MainMenu(3) The top main menu
organizer(3) Calendar and address organizer
osd(2) Manage the On−Screen Display
osd−dialog(3) OSD dialog box
osd−entry(3) OSD entry dialog box
osd−xdialog(3) OSD Extended dialog box
print−setup(3) Configure (*mS's printer interface
scheme−editor(3) Color Scheme Editor
spell−buffer(3) Spell check the current buffer
spell−edit−word(3) Edits a spell word entry
spell−word(3) (**esc $**) Spell check a single word
symbol(3) Insert an ASCII character
user−setup(3) Configure MicroEmacs for a specific user

Variables

$osd−scheme(5) OSD color scheme

Cursor Movement Commands

CURSOR MOVEMENT COMMANDS

The cursor movement commands control how the cursor is moved around the buffer.

Commands

backward−char(2) (**C−b**) Move the cursor left
backward−line(2) (**C−p**) Move the cursor to the previous line
backward−paragraph(2) (**esc p**) Move the cursor to the previous paragraph
backward−word(2) (**esc b**) Move the cursor to the previous word
beginning−of−buffer(2) (**esc** <) Move to beginning of buffer/file
beginning−of−line(2) (**C−a**) Move to beginning of line
display−matching−fence(3) Display the matching bracket
end−of−buffer(2) (**esc** >) Move to end of buffer/file
end−of−line(2) (**C−e**) Move to end of line
forward−char(2) (**C−f**) Move the cursor right
forward−line(2) (**C−n**) Move the cursor to the next line
forward−paragraph(2) (**esc n**) Move the cursor to the next paragraph
forward−word(2) (**esc f**) Move the cursor to the next word
goto−line(2) (**esc g**) Move the cursor to specified line
goto−matching−fence(2) (**esc C−f**) Move the cursor to matching fence
goto−position(2) Restore a stored position
goto−window(2) Restore a saved window to the current window (historic)
list−commands(2) (**C−h c**) List available commands
list−variables(2) (**C−h v**) List defined variables
recenter(2) (**C−l**) Recenter the window (refresh the screen)
set−position(2) Store the current position
set−window(2) Save the current window for restore (historic)
universal−argument(2) (**C−u**) Set the command argument count

Variables

$fmatchdelay(5) Fence matching delay time

Insertion and Deletion Commands

INSERTION AND DELETION COMMANDS

Commands that initiate insertion or deletion of text include:

Deletion

backward−delete−char(2) (**backspace**) Delete the previous character at the cursor position
backward−delete−tab(2) (**S−tab**) Delete white space to previous tab−stop
backward−kill−word(2) (**esc backspace**) Delete the previous word at the cursor position
clean(3) Remove redundant white spaces from the current buffer
delete−blank−lines(2) (**C−x C−o**) Delete blank lines about cursor
delete−indentation(3) Join 2 lines deleting white spaces
forward−delete−char(2) (**C−d**) Delete the next character at the cursor position
forward−kill−word(2) (**esc d**) Delete the next word at the cursor position
kill−line(2) (**C−k**) Delete all characters to the end of the line
kill−paragraph(2) Delete a paragraph
kill−rectangle(2) (**esc C−w**) Delete a column of text
kill−region(2) (**C−w**) Delete all characters in the marked region
yank−rectangle(2) (**esc C−y**) Insert a column of text

Insertion

insert−newline(2) (**C−o**) Insert new line at cursor position
insert−tab(2) (**C−i**) Insert tab(s) into current buffer
normal−tab(3) Insert a normal tab
quote−char(2) (**C−q**) Insert literal character
reyank(2) (**esc y**) Restore next yank buffer
tab(2) (**tab**) Handle the tab key
yank(2) (**C−y**) Paste (copy) kill buffer contents into buffer

Variables

$tabsize(5) Tab character width
$tabwidth(5) Tab character interval

Paragraph and Text Formatting Commands

PARAGRAPH AND TEXT FORMATTING COMMANDS

Commands that operate on paragraphs, and the layout of paragraphs:

Paragraph

Paragraphs are separated by blank lines. A single paragraph is defined as all of the text enclosed between two blank lines, with no intervening blank lines.

backward−paragraph(2) (**esc p**) Move the cursor to the previous paragraph
fill−paragraph(2) (**esc o**) Format a paragraph
forward−paragraph(2) (**esc n**) Move the cursor to the next paragraph
ifill−paragraph(3) (**esc q**) Format a paragraph
kill−paragraph(2) Delete a paragraph
paragraph−to−line(3) Convert a paragraph to a single line
wrap−word(2) Wrap word onto next line

Regions and Marks

A region is the text located between the **point** (the current cursor position) and the **mark** defined by set−mark.

copy−region(2) (**esc w**) Copy a region of the buffer
count−words(2) (**esc C−c**) Count the number of words in a region
exchange−point−and−mark(2) (**C−x C−x**) Exchange the cursor and marked position
goto−alpha−mark(2) (**C−x a**) Move the cursor to a alpha marked location
kill−rectangle(2) (**esc C−w**) Delete a column of text
kill−region(2) (**C−w**) Delete all characters in the marked region
set−alpha−mark(2) (**C−x C−a**) Place an alphabetic marker in the buffer
set−mark(2) (**esc space**) Set starting point of region
yank−rectangle(2) (**esc C−y**) Insert a column of text

Variables

$fill−bullet(5) Paragraph filling bullet character set
$fill−bullet−len(5) Paragraph filling bullet search depth
$fill−col(5) Paragraph Mode; right fill column
$fill−eos(5) Paragraph filling; end of sentence fill characters
$fill−eos−len(5) Paragraph filling; end of sentence padding length
$fill−ignore(5) Ignore paragraph filling character(s)
$fill−mode(5) Paragraph mode; justification method

Capitalization and Transposition Commands

CAPITALIZATION AND TRANSPOSITION COMMANDS

Commands to change the capitalization and transposition of text:

capitalize−word(2) (**esc c**) Capitalize word
lower−case−region(2) (**C−x C−l**) Lowercase a region (downcase)
lower−case−word(2) (**esc l**) Lowercase word (downcase)
sort−lines(2) Alphabetically sort lines
sort−lines−ignore−case(3) Alphabetically sort lines ignoring case
transpose−chars(2) (**C−t**) Exchange (swap) adjacent characters
transpose−lines(2) (**C−x C−t**) Exchange (swap) adjacent lines
uniq(3) Make lines in a sorted list unique
upper−case−region(2) (**C−x C−u**) Uppercase a region (upcase)
upper−case−word(2) (**esc u**) Uppercase word (upcase)

Searching and Replacing

SEARCHING AND REPLACING

Text searching and replacing commands:

hunt−backward(2) (**C−x C−h**) Resume previous search in backward direction
hunt−forward(2) (**C−x h**) Resume previous search in forward direction
isearch−backward(2) (**C−r**) Search backwards incrementally (interactive)
isearch−forward(2) (**C−s**) Search forward incrementally (interactive)
item−list(3) (**F7**) Abbreviated search and list buffer contents
item−list−close(3) (**esc F7**) Close the item list
item−list−find(3) Find the selected item in the item list
line−scheme−search(3) Search and annotate the current buffer
occur(3) Regular expression search for occurrences
query−replace−all−string(3) Query replace string in a list of files
query−replace−string(2) (**esc C−r**) Search and replace a string − with query
RegularExpressions(2) Regular Expressions
regex−backward(3) Search for a magic string in the backward direction
regex−forward(3) Search for a magic string in the forward direction
replace−all−pairs(3) Replace string pairs in a list of files
replace−all−string(3) Replace string with new string in a list of files
replace−string(2) (**esc r**) Replace string with new string
search−backward(2) (**C−x r**) Search for a string in the backward direction
search−forward(2) (**C−x s**) Search for a string in the forward direction

Macro Commands

MACRO COMMANDS

Everyday macro commands used by the user. See Macro Development Commands for commands related to macro development.

Commands

end−kbd−macro(2) (**C−x)**) Stop recording keyboard macro
execute−buffer(2) Execute script lines from a buffer
execute−file(2) (**esc /**) Execute script lines from a file
execute−kbd−macro(2) (**C−x e**) Execute a keyboard macro
execute−line(2) Execute a typed in script line
execute−named−command(2) [**esc x**] Execute a named command
insert−macro(2) Insert keyboard macro into buffer
kbd−macro−query(2) (**C−x q**) Query termination of keyboard macro
name−kbd−macro(2) Assign a name to the last keyboard macro
start−kbd−macro(2) (**C−x (**) Start recording keyboard macro

Variables

$debug(5) Macro debugging flag

Buffer Manipulation Commands

BUFFER MANIPULATION COMMANDS

A buffer is where MicroEmacs '02 stores text. Normally text is read from a file and is visible in an editing window. The name, associated file and operating modes of the buffer, are generally shown in the mode line.

Commands that deal with buffers include:

Commands

buffer−info(2) (**C−x =**) Status information on current buffer position
change−buffer−name(2) (**esc C−n**) Change name of current buffer
change−file−name(2) (**C−x n**) Change the file name of the current buffer
delete−buffer(2) (**C−x k**) Delete a buffer
delete−some−buffers(2) Delete buffers with query
execute−buffer(2) Execute script lines from a buffer
execute−line(2) Execute a typed in script line
find−buffer(2) (**C−x b**) Switch to a named buffer
insert−file−name(2) (**C−x C−y**) Insert filename into current buffer
list−buffers(2) (**C−x C−b**) List all buffers and show their status
narrow−buffer(2) Hide buffer lines
next−buffer(2) (**C−x x**) Switch to the next buffer
save−all(3) Save all modified files (with query)
save−some−buffers(2) Save contents of all changed buffers to file (with query)

Variables

$MEBACKUPPATH(5) Backup file location
$MEBACKUPSUB(5) Backup file name modifier
$buffer−backup(5) Buffer backup file name
$buffer−bname(5) Name of the current buffer
$buffer−fmod(5) Buffer file modes (or attributes)
$buffer−fname(5) Name of the current buffer's file name
$buffer−mask(5) Current buffer word class mask
$buffer−mode−line(5) Buffer mode line string
$buffer−names(5) Filtered buffer name list
$file−names(5) Filtered file name list
$global−fmod(5) Global file modes (or attributes)
$mode−line(5) Mode line format
$mode−line−scheme(5) Mode line color scheme
$show−modes(5) Select buffer modes to display

Window Commands

WINDOW COMMANDS

MicroEmacs '02 uses windows to display and allow you to edit the contents of buffers. Multiple windows may be present on the screen at once, each is 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 cursor commands. Note that if a line of text extends beyond the boundary of a window, a dollar "**$**" sign is displayed instead of the first/last visible character.

Commands that operate on windows are defined as follows:

Commands

change−window−depth(2) Change the depth of the current window
change−window−width(2) Change the width of the current window
compare−windows(2) Compare buffer windows, ignore whitespace
compare−windows−exact(3) Compare buffer windows, with whitespace
create−frame(2) Create a new frame
delete−frame(2) Delete the current frame
delete−other−windows(2) (**C−x 1**) Delete other windows
delete−window(2) (**C−x 0**) Delete current window
grow−window−horizontally(2) Enlarge current window horizontally (relative)
grow−window−vertically(2) Enlarge the current window (relative change)
next−frame(2) Change the focus to the next frame
next−window(2) (**C−x o**) Move the cursor to the next window
next−window−find−buffer(2) [] Split the current window and show new buffer
next−window−find−file(2) (**C−x 4**) Split the current window and find file
previous−window(2) (**C−x p**) Move the cursor to the previous window
resize−all−windows(2) Resize all windows (automatic change)
resize−window−horizontally(2) Resize current window horizontally (absolute)
resize−window−vertically(2) Resize the current window (absolute change)
scroll−down(2) (**C−n**) Move the window down (scrolling)
scroll−left(2) (**C−x** <) Move the window left (scrolling)
scroll−next−window−down(2) (**esc C−v**) Scroll next window down
scroll−next−window−up(2) (**esc C−z**) Scroll next window up
scroll−right(2) (**C−x** >) Move the window right (scrolling)
scroll−up(2) (**C−p**) Move the window up (scrolling)
shrink−window−horizontally(2) Shrink current window horizontally (relative)
shrink−window−vertically(2) Shrink the current window (relative change)
split−window−horizontally(2) (**C−x 5**) Split current window into two (horizontally)
split−window−vertically(2) (**C−x 2**) Split the current window into two

Variables

$scroll(5) Screen scroll control
$scroll−bar(5) Scroll bar configuration
$window−acol(5) Window cursor actual column
$window−aline(5) Window cursor actual line
$window−col(5) Window cursor column (no expansion)
$window−depth(5) Number of text lines in a window
$window−line(5) Window cursor line
$window−mode−line(5) Window mode line position
$window−scroll−bar(5) Window scroll bar (or separator) position
$window−wcol(5) Window cursor column (historic)
$window−width(5) Number of character columns in a window
$window−wline(5) Window cursor line (historic)
$window−x−scroll(5) Current window X scroll
$window−xcl−scroll(5) Current window current line X scroll
$window−y−scroll(5) Current window Y scroll

Keyboard Binding Commands

KEYBOARD BINDING COMMANDS

Keyboard binding allows key strokes to be associated with commands and macros such that when a bound key stroke sequence is recognized its associated (or bound) command is invoked, thereby controlling the editor. A set of Default Bindings exist for MicroEmacs '02 which may be altered using the binding commands. There are three types of key bindings:

Global

Associates a key−stroke with a command for all buffers. Used to establish the standard keyboard controls i.e. cursor movement, search, replace etc.

Local

Associates a key−stroke with a command for a specified buffer only, i.e. a binding local to the buffer. Local bindings allow macro accelerators to be bound to keys without affecting other buffers containing different types of data. Local bindings are used extensively in the buffer hook commands.

Message Line

Associates a key binding for use on the command line only, allowing command completion to be diverted etc.

To bind a command to a key, the command and key names must be known, see Command Glossary for a complete list of commands and Key Names for a complete list of key names.

The binding related commands are defined as follows:

Commands

buffer−bind−key(2) Create local key binding for current buffer
buffer−unbind−key(2) Remove local key binding for current buffer
command−apropos(2) (**C−h a**) List commands involving a concept
describe−bindings(2) (**C−h b**) Show current command/key binding
describe−key(2) (**C−x ?**) Report keyboard key name and binding
expand−iso−accents(3) Expand an ISO accent
global−bind−key(2) (**esc k**) Bind a key to a named command or macro
global−unbind−key(2) (**esc C−k**) Unbind a key from a named command or macro
iso−accents−mode(3) ISO accent expansion short−cut mode
ml−bind−key(2) Create key binding for message line
ml−unbind−key(2) Remove key binding from message line
osd−bind−key(2) Create key binding for OSD dialog
osd−unbind−key(2) Remove key binding from OSD dialog
set−char−mask(2) Set character word mask

translate−key(2) Translate key

Variables

Alt Key

The **Alt Key** has special binding priorities defined as follows:−

- Direct key binding (e.g. **A−b** executes file−browser)
- Main menu hot key (e.g. **A−f** opens the File menu)
- Meta key binding (e.g. **A−space** −> **esc space** −> set−mark)

If the `ALT` key is to be used strictly as the Emacs Meta key then the bindings for the menu should be over−ridden by *Direct Key Bindings* from the user configuration file i.e. to re−map the default MicroEmacs Alt key to equivalent `esc` keys then the following keys should be re−bound.

```
global-bind-key forward-word "A-f"      ; Over-ride File menu binding
:                                       ; For all of the other menu items.
:
global-bind-key backward-word "A-b"     ; Over-ride the file browser.
global-bind-key replace-string "A-r"    ; Over-ride tools binding.
```

This creates a higher priority binding which overrides the underlying default. The commands that are displaced would have to be re−bound to different keys if required.

Operating Modes

OPERATING MODES

Modes are switches (or states) that may be applied globally or on a per buffer basis whose settings determine how MicroEmacs '02 operates. Modes affect operations within a buffer, global modes determine the modes of newly created buffers.

Commands to alter the operating state:

add−global−mode(3) Set a global buffer mode
add−mode(3) Set a local buffer mode
buffer−mode(2) (**C−x m**) Change a local buffer mode
delete−global−mode(3) Remove a global buffer mode
delete−mode(3) Remove a local buffer mode
global−mode(2) (**esc m**) Change a global buffer mode
named−buffer−mode(2) Change a named buffer mode
unmark−buffer(3) Remove buffer edited flag

Modes

The operating modes are defined as follows:

auto(2m) Automatic source file line type detection
autosv(2m) Automatic file save
backup(2m) Automatic file backup of last edit
binary(2m) Binary editor mode
cmode(2m) C Programming language mode
crlf(2m) File's line feed style
crypt(2m) Encrypted file mode
ctrlz(2m) File's termination style
del(2m) Flag buffer to be deleted
dir(2m) Buffer is a directory listing
edit(2m) Buffer has be changed
exact(2m) Searching and sorting case sensitivity
fence(2m) Auto fence matching mode
hide(2m) Hide buffer
indent(2m) Automatic indentation
justify(2m) Justification Mode
letter(2m) Letter kill policy
line(2m) Line kill policy
lock(2m) Pipe cursor position lock
magic(2m) Regular expression search
nact(2m) Buffer not active
narrow(2m) Buffer contains a narrow
over(2m) Over−strike Mode

pipe(2m) Incremental Pipe running
quiet(2m) Quiet mode
rbin(2m) Reduced binary editor mode
save(2m) Flag buffer to be saved
tab(2m) Tabulation mode
time(2m) File time stamping
undo(2m) Retain edit modifications
usr(2m) User buffer modes
view(2m) Read only
wrap(2m) Line wrap entered text

Mode Line

The buffer modes may be shown on the mode line as single letter mnemonics as follows:−

Auto, **a**utosv, **B**ackup, **b**inary, **C**mode, **c**rlf, cr**Y**pt, ctrl**z**, **d**el, **D**ir, **e**dit, **E**xact, **H**ide, **I**ndent, **J**ustify, **l**etter, **L**ine, loc**k**, **M**agic, **n**act, **N**arrow, **O**ver, **P**ipe, **Q**uiet, **S**ave, **T**ab, **t**ime, **U**ndo, usr**1**, usr**2**, usr**3**, usr**4**, usr**5**, usr**6**, usr**7**, usr**8**, **V**iew, **W**rap.

Shell and Command Controls

SHELL AND COMMAND CONTROLS

Operating system and external system call invocations:

Commands

add−next−line(2) Define the searching behavior of command output
compile(3) Start a compilation process
cvs(3) MicroEmacs CVS interface
cvs−add(3) MicroEmacs CVS interface − add file
cvs−checkout(3) MicroEmacs CVS interface − checkout files
cvs−commit(3) MicroEmacs CVS interface − commit changes
cvs−diff(3) MicroEmacs CVS interface − diff changes
cvs−gdiff(3) MicroEmacs CVS interface − graphical diff changes
cvs−log(3) MicroEmacs CVS interface − log changes
cvs−remove(3) MicroEmacs CVS interface − remove file
cvs−resolve−conflicts(3) MicroEmacs CVS interface − resolve conflicts
cvs−state(3) MicroEmacs CVS interface − list state of directory files
cvs−update(3) MicroEmacs CVS interface − update directory files
cygnus(3) Open a Cygwin BASH window
dbx(3) UNIX Debugger
diff(3) Difference files or directories
diff−changes(3) Find the differences from a previous edit session
execute−tool(3) Execute a user defined shell tool
filter−buffer(2) (**C−x #**) Filter the current buffer through an O/S command
gdb(3) GNU Debugger
gdiff(3) Graphical file difference
generate−tags−file(3) Generate a tags file
get−next−line(2) (**C−x `**) Find the next command line
grep(3) Execute grep command
ipipe−kill(2) Kill a incremental pipe
ipipe−shell−command(2) (**esc backslash**) Incremental pipe (non−suspending system call)
ipipe−write(2) Write a string to an incremental pipe
ishell(3) Open a Cygwin BASH window
item−list(3) (**F7**) Abbreviated search and list buffer contents
item−list−close(3) (**esc F7**) Close the item list
item−list−find(3) Find the selected item in the item list
occur(3) Regular expression search for occurrences
perldb(3) Perl Debugger
pipe−shell−command(2) (**esc @**) Execute a single operating system command
rcs−file(2) (**C−x C−q**) Handle Revision Control System (RCS) files
rgrep(3) Execute recursive grep command
shell(2) [**C−x c**] Create a new command processor or shell
shell−command(2) Perform an operating system command

Variables

$ME_ISHELL(5)	Windows ishell command.com
$ME_PIPE_STDERR(5)	Command line diversion to stderr symbol
$buffer−ipipe(5)	Divert buffer incremental pipe input through macro
$file−template(5)	Regular expression file search string
$line−template(5)	Command line regular expression search string
$rcs−ci−com(5)	RCS (and SCCS) check in command
$rcs−cif−com(5)	RCS (and SCCS) check in first command
$rcs−co−com(5)	RCS (and SCCS) check out command
$rcs−cou−com(5)	RCS (and SCCS) check out unlock command
$rcs−file(5)	RCS (and SCCS) file name
$rcs−ue−com(5)	RCS (and SCCS) unedit file command
$result(5)	Various command return values
%compile−com(5)	Default system compile command line
%cygnus−bin−path(5)	Cygwin BASH directory
%cygnus−hilight(5)	Cygwin shell hilight enable flag
%cygnus−prompt(5)	Cygwin shell prompt
%diff−com(5)	Diff command line
%gdiff−com(5)	Gdiff command line
%grep−com(5)	Grep command line

Spelling Commands

SPELLING COMMANDS

Commands related to spelling:

Commands

add−dictionary(2) Declare existence of a spelling dictionary
add−spell−rule(2) Add a new spelling rule to the dictionary
auto−spell(3) Auto−spell support
auto−spell−buffer(3) Auto−spell whole buffer
auto−spell−ignore(3) Auto−spell ignore current word
auto−spell−reset(3) Auto−spell hilight reset
delete−dictionary(2) Remove a spelling dictionary from memory
describe−word(3) Display a dictionary definition of a word
edit−dictionary(3) Insert a dictionary in a buffer
expand−word(3) Complete a word by invocation of the speller
find−word(3) Find a using spelling dictionaries
restore−dictionary(3) Save dictionary user changes
save−dictionary(2) Save changed spelling dictionaries
spell(2) Spell checker service provider
spell−add−word(3) Add a word to the main dictionary
spell−buffer(3) Spell check the current buffer
spell−edit−word(3) Edits a spell word entry
spell−word(3) (**esc $**) Spell check a single word

Variables

$find−words(5) Filtered word list

Hilighting, Color and Screen Appearance

HILIGHTING, COLOR AND SCREEN APPEARANCE

Commands that change the hilighting, screen color and screen appearance:

Commands

add−color(2) Create a new color
add−color−scheme(2) Create a new color scheme
change−font(2) Change the screen font
change−frame−depth(2) Change the number of lines on the current frame
change−frame−width(2) Change the number of columns on the current frame
change−screen−depth(2) Change the number of lines on the screen
change−screen−width(2) Change the number of columns on the screen
hilight(2) Manage the buffer hilighting schemes
indent(2) Manage the auto−indentation methods
line−scheme−search(3) Search and annotate the current buffer
print−color(2) Create a new printer color
print−scheme(2) Create a new printer color and font scheme
restyle−buffer(3) Automatically reformat a buffer's indentation
restyle−region(3) Automatically reformat a regions indentation
scheme−editor(3) Color Scheme Editor
show−region(2) Show the current copy region

Variables

$box−chars(5) Characters used to draw lines
$buffer−hilight(5) Define current buffer hilighting scheme
$buffer−scheme(5) Buffer color scheme
$cursor−blink(5) Cursor blink rate
$cursor−color(5) Cursor foreground color
$frame−depth(5) Number of lines on the current frame canvas
$frame−width(5) Number of columns on the current frame canvas
$global−scheme(5) Global buffer color scheme
$line−scheme(5) Set the current line color scheme
$mode−line(5) Mode line format
$mode−line−scheme(5) Mode line color scheme
$mouse−pos(5) Mouse position information
$screen−depth(5) Number of character lines on the screen canvas
$screen−width(5) Number of character columns on the screen canvas
$scroll−bar(5) Scroll bar configuration
$scroll−bar−scheme(5) Scroll bar color scheme
$show−modes(5) Select buffer modes to display
$show−region(5) Enable the hilighting of regions

$system(5) System configuration variable
$trunc−scheme(5) Truncation color scheme
$window−chars(5) Character set used to render the windows

Comparison and Differencing

Comparison and Differencing

Commands that perform comparisons and differences:−

Commands

compare−windows(2) Compare buffer windows, ignore whitespace
compare−windows−exact(3) Compare buffer windows, with whitespace
diff(3) Difference files or directories
diff−changes(3) Find the differences from a previous edit session
gdiff(3) Graphical file difference

Variables

%diff−com(5) Diff command line
%gdiff−com(5) Gdiff command line

Short Cuts and Abbreviations

SHORT CUTS

Automatic commands, history and automatic formatting modes such as **C−mode** (see cmode(2m)).

Commands

buffer−abbrev−file(2) Set buffers' abbreviation file
comment−end(3) End the current comment
comment−line(3) Comment out the current line
comment−restyle(3) Reformat the current comment
comment−start(3) Start a new comment
comment−to−end−of−line(3) Extend comment to end of line
expand−abbrev(2) Expand an abbreviation
expand−abbrev−handle(3) (**esc esc**) Expand an abbreviation handler
expand−iso−accents(3) Expand an ISO accent
expand−look−back(3) Complete a word by looking back for a similar word
expand−word(3) Complete a word by invocation of the speller
find−tag(2) (**esc t**) Find tag, auto−load file and move to tag position
generate−tags−file(3) Generate a tags file
global−abbrev−file(2) Set global abbreviation file
indent(2) Manage the auto−indentation methods
iso−accents−mode(3) ISO accent expansion short−cut mode
read−history(2) Read in session history information
save−history(2) Write history information to history file
uncomment−line(3) Uncomment current line

Variables

$c−brace(5) C−mode; brace indentation
$c−case(5) C−mode; case indentation
$c−contcomm(5) C−mode; comment continuation string
$c−continue(5) C−mode; line continuation indent
$c−contmax(5) C−mode; line continuation maximum indent
$c−margin(5) C−mode; trailing comment margin
$c−statement(5) C−mode; statement indentation
$c−switch(5) C−mode; switch indentation
%tag−file(5) Tag file name
%tag−option(5) Tag file search option
%tag−template(5) Tag file search string

Message Line Commands

MESSAGE LINE COMMANDS

The message line appears at the bottom of the screen and is used for the input of commands and also to receive errors and information (see also Mode Line).

Commands and variables that interact with the message line include:

Commands

ml−bind−key(2) Create key binding for message line
ml−clear(2) Clear the message line
ml−unbind−key(2) Remove key binding from message line
ml−write(2) Write message on message line
osd−bind−key(2) Create key binding for OSD dialog
osd−unbind−key(2) Remove key binding from OSD dialog

Variables

$ml−scheme(5) Message line color scheme

Printing Commands

PRINTING COMMANDS

Printing within MicroEmacs '02 is fairly restrictive, the following commands are used in conjunction with the print facility.

print−buffer(2) Print buffer, with formatting
print−color(2) Create a new printer color
print−region(2) Print region, with formatting
print−scheme(2) Create a new printer color and font scheme
print−setup(3) Configure (*mS's printer interface

Macro Development Commands

MACRO DEVELOPMENT COMMANDS

Commands used in macro development, and more specialized commands which are only invoked from macros. Refer to Macro Commands for keyboard macros etc.

An additional set of commands for use with macros is outlined in the Introduction to Variable Functions section. The Macro Language Glossary contains a full list of macro related commands and special variables.

Commands

add−file−hook(2) Declare file name context dependent configuration
ascii−time(3) Return the current time as a string
command−wait(2) Conditional wait command
create−callback(2) Create a timer callback
create−frame(2) Create a new frame
define−macro−file(2) Define macro file location
delete−frame(2) Delete the current frame
directory−tree(2) Draw the file directory tree
etfinsrt(3) Insert template file into current buffer
execute−string(2) Execute a string as a command
file−op(2) File system operations command
fileHooks(2) File Hooks
goto−position(2) Restore a stored position
goto−window(2) Restore a saved window to the current window (historic)
hilight(2) Manage the buffer hilighting schemes
insert−space(2) Insert space(s) into current buffer
insert−string(2) Insert character string into current buffer
languageTemplates(2) File Language Templates
localeSupport(2) Locale Support
newline(2) (**return**) Insert a new line
next−frame(2) Change the focus to the next frame
osd−dialog(3) OSD dialog box
osd−entry(3) OSD entry dialog box
osd−xdialog(3) OSD Extended dialog box
popup−window(2) Pop−up a window on the screen
regex−backward(3) Search for a magic string in the backward direction
regex−forward(3) Search for a magic string in the forward direction
screen−poke(2) Immediate write string to the screen
screen−update(2) (**redraw**) Force screen update
set−cursor−to−mouse(2) Move the cursor to the current mouse position
set−position(2) Store the current position
set−scroll−with−mouse(2) Scroll the window with the mouse
set−variable(2) (**C−x v**) Assign a new value to a variable

set−window(2) Save the current window for restore (historic)
show−cursor(2) Change the visibility of the cursor
shut−down(3) Editor exit callback command
spell(2) Spell checker service provider
start−up(3) Editor startup callback command
unset−variable(2) Delete a variable
void(2) Null command

Variables

$MEBACKUPPATH(5) Backup file location
$MEBACKUPSUB(5) Backup file name modifier
$buffer−backup(5) Buffer backup file name
$buffer−bhook(5) Buffer macro hook command name (buffer current)
$buffer−dhook(5) Buffer macro hook command name (buffer deletion)
$buffer−ehook(5) Buffer macro hook command name (buffer swapped)
$buffer−fhook(5) Buffer macro hook command name (buffer creation)
$buffer−fmod(5) Buffer file modes (or attributes)
$buffer−indent(5) Current buffer indentation scheme
$buffer−input(5) Divert buffer input through macro
$buffer−ipipe(5) Divert buffer incremental pipe input through macro
$buffer−names(5) Filtered buffer name list
$command−names(5) Filtered command name list
$cursor−x(5) Mouse X (horizontal) position
$cursor−y(5) Mouse Y (vertical) position
$debug(5) Macro debugging flag
$file−names(5) Filtered file name list
$find−words(5) Filtered word list
$global−fmod(5) Global file modes (or attributes)
$mode−names(5) Filtered mode name list
$mouse(5) Mouse configuration variable
$mouse−x(5) Mouse X (horizontal) position
$mouse−y(5) Mouse Y (vertical) position
$platform(5) MicroEmacs host platform identifier
$progname(5) Program file name
$random(5) Generate a random number
$result(5) Various command return values
$status(5) Macro command execution status
$system(5) System configuration variable
$temp−name(5) Temporary file name
$variable−names(5) Filtered variable name list
$version(5) MicroEmacs version date−code
$window−flags(5) Current window setup flags
$window−mode−line(5) Window mode line position
$window−scroll−bar(5) Window scroll bar (or separator) position
%company−name(5) Name of company for template
.calc.result(5) Last calc calculation result

Registry

REGISTRY

The registry commands provide an interface to manage the registry files defined by erf(8). The registry is a mechanism which allows the binding of information to a hierarchical tree node, using a file system metaphor to access the data. **MicroEmacs** uses a reserved root node `history` to save session information (see save−history(2)).

Commands

delete−registry(2)	Delete a registry tree
find−registry(2)	Index search of a registry sub−tree
get−registry(2)	Retrieve a node value from the registry
list−registry(2)	Display the registry in a buffer
mark−registry(2)	Modify the operating mode of a registry node
read−registry(2)	Read in a registry definition file
save−registry(2)	Write a registry definition file
set−registry(2)	Modify a node value in the registry

Macro Functions

&reg(4)	Retrieve a registry value (with default)

Command Line Filters

COMMAND LINE FILTERS

MicroEmacs may be invoked from the command line to perform a specific set of filtering tasks, under control of a dedicated start up macro, see me(1) and start−up(3). A number of standard macros are provided, most of which are invoked automatically from the editor itself when requested by the user. Having said that, it has not been unknown for a colleague of mine to use the editor as a replacement for a more intelligent **sed(1)** filter, with 12 hours to go and a huge ugly 3−D geometric database to convert, what better way than run it through a set of MicroEmacs macros to turn it into another database format that can be handled − probably not for the uninitiated, but that person did pull it off and went home for tea !!

Macro Command Line Filters

benchmrk(3f) Benchmark MicroEmacs macro processor speed
ctags(3f) Generate a C tags file
dos2unix(3f) Convert DOS format files to UNIX format files
ehftools(3f) Generate a MicroEmacs help file
emftags(3f) Generate a MicroEmacs macro tags file
gdiff(3f) Command line graphical file difference
javatags(3f) Generate a C tags file from Java sources
ntags(3f) Generate a nroff tags file
printall(3f) Formatted print job
tcltags(3f) Generate a Tcl/Tk tags file
textags(3f) Generate a LaTeX/BibTeX tags file

Macro Functions

shut−down(3) Editor exit callback command
start−up(3) Editor startup callback command

Macro Variables

Games

GAMES

The following is a list of all of the games provided by **MicroEmacs '02**:

Mahjongg(3) MicroEmacs '02 version of the solitaire Mah Jongg game
Match−It(3) MicroEmacs '02 version of the Match−It game
Metris(3) MicroEmacs '02 version of the falling blocks game
Patience(3) MicroEmacs '02 version of Patience (or Solitaire)
Triangle(3) MicroEmacs '02 version of Triangle patience game

languageTemplates(2)

FILE LANGUAGE TEMPLATES

MicroEmacs '02 provides a large range of macros and templates to deal with the most commonly occurring types of ASCII file that may be edited. However, there is a requirement for users to extend this capability to include more obscure file types, in addition to bespoke files found internally within organizations, or devised by the user.

For each file type, MicroEmacs '02 may be tailored to recognize the file and modify it's hilighting, key binding configuration, osd display and indentation to accommodate the file. In addition, new shorthand macros may be introduced to help deal with the contents of the file.

This section outlines the steps to be taken to integrate a new file language template into MicroEmacs '02.

The scope of the File Type

The first step is to decide the scope of the file, this will determine where the file hook should be defined. The options are:−

A standard file type not supported

If this is a standard file type not supported by MicroEmacs '02 then it should be added to `me.emf`, in addition contact us and we will add it to the standard release. Any macro files associated with this file type should be available globally and are added to the MicroEmacs *macro* directory.

Local To your organization

If it is a file type local to your organization then it should be added to your *company*.emf file. Any macro files associated with the file type should be added to your local company MicroEmacs '02 directory.

Local to an individual

If this is a file type that is only used by a limited number of individuals then it should be added to the *user*.emf file. Any files associated with the file type are added to your local user MicroEmacs '02 directory.

Recognizing the File Type

The next step to adding a new file type is to get MicroEmacs '02 to recognize the file as the new type. Recognition is performed by the File Hooks which perform recognition on the file extension and/or the file content. The name of the file type must be determined, this is typically the name of the file prepended by `hk`. e.g. a file with extension *foo* uses the file `hkfoo.emf` for it's language specific definitions.

Using the add−file−hook(2) invocation the file recognition is bound to the file hook macro whenever the file type is loaded. The file hook is added to the appropriate global, company or user start up file as determined in step 1. The file hooks for file *foo* might be defined as follows, depending upon the recognition method:−

Recognizing the extension

To recognize the file extension, then a space separated list of extensions may be defined, including the dot `'.'` (or other) extension separator.

```
add−file−hook ".foo"       fhook−foo
```

Recognizing a magic editor string in the file

If the file type adopts multiple extensions (or does not use a file extension) then an editor specific string may be inserted into the file to enable the editor to recognize it, typically of the form `−!−` *type* `−!−`, if the string is GNU Emacs compatible then the `−*−` convention may be used. The binding is defined as:−

```
−1 add−file−hook "−!−[ \t]*foo.*−!−"          fhook−foo
```

Recognizing a magic string in the file

UNIX files use a "`#!<path>`" notation for executable ASCII files. If the file is this type of file (or uses any other type of common string in the as the first characters of a file) then the binding may be defined as follows, in this case we have assumed *foo* is the UNIX executable variety i.e. `#!/usr/local/bin/foo`:−

```
1 add−file−hook "^#!/.*foo" fhook−foo
```

Any, or all of the above recognition methods may be employed to invoke the language specific macro. Note that the methods are evaluated in a LIFO order, hence it is possible to over−ride an existing method.

Defining the Macro File

Once the hook has been defined, the language specific file must be created. Create the language specific file with the same name as defined in the hooks, removing the **fhook−** prefix and replacing it with **hk**, i.e. `fhook−foo` invokes the language specific file `hkfoo.emf`. Create, the file and add the file hook macro. for example hk*foo*.emf contents may be defined as:

```
define−macro fhook−foo
    ; Temporary comment to make sure that it works.
    ml−write "Loaded a foo file"
!emacro
ml−write "[MicroEmacs foo file hook loaded]"
```

The file hook may be tested by exiting and re−loading MicroEmacs '02, or simply by executing the file containing the `add−file−hook` function. Once the file bindings are installed a *foo* file may be loaded and the hook message should be displayed.

Modifying an Existing file hook

The standard file hooks supplied with MicroEmacs '02 should not be modified, typically a user will want to extend the repertoire of hi−lighting tokens to encompass locally defined programming libraries or syntactical extensions, in addition to extending support macros that are associated with the file type. In this case, an extension to the hook function is required. The hook file **my*XXX*.emf**, allows extensions to be made to the **hk*XXX*.emf**, without editing the original file. This may be considered to be an *include* file and is executed, if it exists, after the **hk** file has been executed. i.e. if the hook file **hkfoo.emf** is already defined and extensions are added to **myfoo.emf**.

Note that the **my*XXX*.emf** files do not typically include any **fhook−XXX** functions, the original *fhook* functions would be used. However, if a different buffer environment is required from the one created be the hook, such as a different setting of tab(2m) mode, the hook function should be copied to **my*XXX*.emf** and altered appropriately.

Adding Hilighting definitions

File specific hilighting is used to pick out key words and tokens used within the file type, it greatly improves readability; the hilighting is also used for printing. The hilighting is defined within the body of the file and is executed once when the hook file is loaded, this occurs when the hook function is executed. During development of the hilighting code, it is usually necessary to execute the hook buffer to view the effects of any changes to the hilighting.

The hilighting is defined using the command hilight(2) which requires a hilighting identifier, used to identify the hilighting scheme. This identifier is dynamically allocated when the hook file is loaded, again using *foo*, the identifier is allocated at the top of the file and is protected such that a value is assigned once only.

```
!if &sequal .hilight.foo "ERROR"
    set−variable .hilight.foo &pinc .hilight.next 1
!endif
```

The variable `.hilight.next` allocates unique hilighting numbers, typically a single hilighting number is consumed, incrementing the `.hilight.next` variable ready for the next allocation. The hilighting color scheme is defined in a macro variable **.hilight.***ext*, where *ext* is the name of the language scheme (i.e. *foo*).

Given a hilighting number, the hilighting scheme may be defined. Each of the tokens in the language is assigned a hilighting color, for our simple *foo* file type:−

```
0 hilight .hilight.foo 1                $global−scheme
hilight .hilight.foo 2 "#"              .scheme.comment
hilight .hilight.foo 4 "\"" "\"" "\\"   .scheme.string
hilight .hilight.foo 0 "'.'"            .scheme.quote
hilight .hilight.foo 0 "'\\\\.'"        .scheme.quote ; '\?' quoted char

hilight .hilight.foo 1 "if"             .scheme.keyword
hilight .hilight.foo 1 "then"           .scheme.keyword
hilight .hilight.foo 1 "else"           .scheme.keyword
hilight .hilight.foo 1 "endif"          .scheme.keyword
```

When the hilighting tokens have been defined, the hilighting scheme is bound to the buffer. This is performed by assigning $buffer−hilight(5) with the hilighting scheme within the *fhook* macro body, e.g.

```
define−macro fhook−foo
    ; Assign the hilighting
    set−variable $buffer−hilight .hilight.foo
    ; Temporary comment to make sure that it works.
    ml−write "Loaded a foo file"
!emacro
```

Putting it all together `hkfoo.emf` now comprises:−

```
!if &sequal .hilight.foo "ERROR"
    ; Allocate a hilighting scheme number
    set−variable .hilight.foo &pinc .hilight.next 1
!endif

; Define the hilighting scheme
0 hilight .hilight.foo 1                $global−scheme
hilight .hilight.foo 2 "#"              .scheme.comment
hilight .hilight.foo 4 "\"" "\"" "\\"   .scheme.string
hilight .hilight.foo 0 "'.'"            .scheme.quote
hilight .hilight.foo 0 "'\\\\.'"        .scheme.quote ; '\?' quoted char

hilight .hilight.foo 1 "if"             .scheme.keyword
hilight .hilight.foo 1 "then"           .scheme.keyword
hilight .hilight.foo 1 "else"           .scheme.keyword
hilight .hilight.foo 1 "endif"          .scheme.keyword

; File hook − called when new file is loaded.
define−macro fhook−foo
    ; Assign the hilighting
    set−variable $buffer−hilight .hilight.foo
    ; Temporary comment to make sure that it works.
    ml−write "Loaded a foo file"
!emacro

; Notification that hook is loaded.
ml−write "[MicroEmacs foo file hook loaded]"
```

Adding a Template

A template inserts initial text into a new file that is created. This mechanism is typically used to insert a standard header into the file on creation. The insertion text is defined within a template file, given the file extension etf(8), which is created in the corresponding global, company or user directory as determined in step 1. The template is named *ext*.etf, so for our example file *foo*, the template file is called `foo.etf`. We shall simply add a file header, our comment is # (as defined by the hilighting tokens). Our example *foo* template file `foo.etf` may be defined as follows:−

```
#-!- foo -!- #################################
#
#  Created By    : $USER_NAME$
#  Created       : $ASCII_TIME$
#  Last Modified : <160495.1521>
#
#  Description
#
#  Notes
#
#  History
#
#  Copyright (c) $YEAR$ $COMPANY_NAME$.
##############################################
```

The template file must be explicitly loaded by the hook file, within the **fhook** function. A new file condition may be tested within the fhook macro by checking the numerical argument, an argument of 0 indicates that this is a new file. The template file is inserted with an invocation of etfinsrt(3). The **fhook** macro checks the argument and inserts the template file as follows:−

```
; File hook - called when new file is loaded.
define-macro fhook-foo
    ; if arg is 0 this is a new file so add template
    !if &not @#
        etfinsrt "foo"
    !endif
    ; Assign the hilighting
    set-variable $buffer-hilight .hilight.foo
    ; Temporary comment to make sure that it works.
    ml-write "Loaded a foo file"
!emacro
```

Adding abbreviations

Abbreviations are short−cut expansions which may be defined for the language specific file. The abbreviations are defined in a eaf(8) file, *ext*.eaf, located in the appropriately defined MicroEmacs directory. The abbreviation file defines the key sequences which may be automatically inserted, under user intervention, using expand−abbrev(2). An abbreviation file for *foo*, `foo.eaf`, may be defined as:−

```
if "if \p\rthen\rendif\P"
el "else\r\p\P"
```

The binding to the hook is defined in the *fhook* macro using buffer−abbrev−file(2). For the example language file *foo* the *fhook* macro becomes:−

```
; File hook - called when new file is loaded.
define-macro fhook-foo
    ; if arg is 0 this is a new file so add template
    !if &not @#
        etfinsrt "foo"
    !endif
    ; Assign the hilighting
    set-variable $buffer-hilight .hilight.foo
```

```
        ; Set the abbreviation file
        buffer-abbrev-file "foo"
        ; Temporary comment to make sure that it works.
        ml-write "Loaded a foo file"
    !emacro
```

Automatic Indentation

Automatic indentation may be applied to the file, such that the indentation is automatically performed when new lines are entered into the file. Indentation also benefits from automatic re−styling operations using restyle−region(3) and restyle−buffer(3).

The indentation style is declared by defining language tokens that constitute positions in the syntax where the indentation is changed. The indentation requires a unique identifier to identify the indentation style, the hilighting identifier is used. If hilighting is not defined, then the language template may still obtain an identifier as described in the hilighting section.

The indention is create with an argument of `0` to the indent(2) command, the subsequent tokens are defined using **indent** with no argument. For our simple *foo* syntax then the indentation might be defined as follows:−

```
0 indent  .hilight.foo 2 10
indent .hilight.foo n "then" 4
indent .hilight.foo s "else" -4
indent .hilight.foo o "endif" -4
```

This provides an indentation of the form:−

```
if condition
then
    XXXX
else
    if condition
    then
        XXXX
    endif
endif
```

The indentation is bound to the buffer in the *fhook* macro by defining $buffer−indent(5). For the example file *foo* then the *fhook* is defined as:−

```
; File hook - called when new file is loaded.
define-macro fhook-foo
    ; if arg is 0 this is a new file so add template
    !if &not @#
        etfinsrt "foo"
    !endif
    ; Assign the hilighting
    set-variable $buffer-hilight .hilight.foo
    ; Assign the buffer indentation
    set-variable $buffer-indent .hilight.foo
    ; Set the abbreviation file
    buffer-abbrev-file "foo"
```

```
        ; Temporary comment to make sure that it works.
        ml-write "Loaded a foo file"
    !emacro
```

Setting Buffer Modes

Buffer modes which are to be adopted (or discarded) by the language specific file are defined in the *fhook* macro. Typical modes that are applied are:−

time

Enables time stamping on the file, modifying the time stamp field with the modification date and time.

indent

Automatic indentation, where the cursor is returned to the same column on entering a new line, rather than to the start of the line.

As an example, the *foo fhook* file becomes:−

```
    ; File hook - called when new file is loaded.
    define-macro fhook-foo
        ; if arg is 0 this is a new file so add template
        !if &not @#
            etfinsrt "foo"
        !endif
        ; Assign the hilighting
        set-variable $buffer-hilight .hilight.foo
        ; Assign the buffer indentation
        set-variable $buffer-indent .hilight.foo
        ; Set the abbreviation file
        buffer-abbrev-file "foo"
        ; Set up the buffer modes
        1 buffer-mode "time"
        1 buffer-mode "indent"
        ; Temporary comment to make sure that it works.
        ml-write "Loaded a foo file"
    !emacro
```

Assigning New Bindings

New bindings and language specific macros may be added to the language specific file. New macros, to extend the repertoire of commands specifically developed for the language file are defined within the macro body using define−macro(2) these are automatically loaded when the hook file is loaded, which in turn is loaded when the file type is identified and loaded.

New bindings, which may be associated with new macros or existing commands, are assigned within the *fhook* macro. As an example, we shall extend the *foo* language file to include a commenting and uncommenting macros, locally binding the macros to the keys "`C−c  C−c`" and "`C−c C−d`" respectively. The macro definitions are defined as follows:−

```
; Macro to comment a line
define−macro foo−comment−line
    !while &gre &pdec @# 1 0
        beginning−of−line
        insert−string "#"
        beginning−of−line
        forward−line
    !done
!emacro

; Macro to remove a comment from a line
define−macro foo−uncomment−line
    !while &gre &pdec @# 1 0
        beginning−of−line
        −1 search−forward "#"
        backward−delete−char
        forward−line
    !done
!emacro
```

The key bindings for the macros are defined for the local buffer ONLY, as such are added using buffer−bind−key(2). The bindings are declared in the *fhook* macro as follows:−

```
; File hook − called when new file is loaded.
define−macro fhook−foo
    ; if arg is 0 this is a new file so add template
    !if &not @#
        etfinsrt "foo"
    !endif
    ; Assign the hilighting
    set−variable $buffer−hilight .hilight.foo
    ; Assign the buffer indentation
    set−variable $buffer−indent .hilight.foo
    ; Set the abbreviation file
    buffer−abbrev−file "foo"
    ; Set up the buffer modes
    1 buffer−mode "time"
    1 buffer−mode "indent"
    ; Set up local bindings
    buffer−bind−key foo−comment−line "C−c C−c"
    buffer−bind−key foo−uncomment−line "C−c C−d"
    ; Temporary comment to make sure that it works.
    ml−write "Loaded a foo file"
!emacro
```

Allowing Other to Modify the Hook

Other users of the file hook may need to modify or extend the file hook, the most common form is the addition of user specific hilight tokens. MicroEmacs uses a simple mechanism of executing a user hook extension file if it exists. The extension file name must be of the form **my*XXX*.emf**, i.e. for our example it must be "`myfoo.emf`". This is performed at the end of the macro file so that anything within the file can be altered, it is executed as follows:−

```
; load in user extensions if found
!force execute-file "myfoo"
```

Note the !force(4) directive is used as the file may not exist.

Summing Up

The previous sections have presented the basic steps involved in setting up a new language file template. They cater for simple file types, for more complex examples then browse the **hk***xxx*.emf files.

The completed files that should have been generated by following the previous examples are now presented:−

file.foo

```
# This is a comment.
if condition
then
    do something
else
    if condition
    then
        do something
    endif
endif
```

hkfoo.emf

```
!if &sequal .hilight.foo "ERROR"
    ; Allocate a hilighting scheme number
    set-variable .hilight.foo &pinc .hilight.next 1
!endif

; Define the hilighting scheme
0 hilight .hilight.foo 1                $global-scheme
hilight .hilight.foo 2 "#"              .scheme.comment
hilight .hilight.foo 4 "\"" "\"" "\\"   .scheme.string
hilight .hilight.foo 0 "'.'"            .scheme.quote
hilight .hilight.foo 0 "'\\\\.'"        .scheme.quote ; '\?' quoted char

hilight .hilight.foo 1 "if"             .scheme.keyword
hilight .hilight.foo 1 "then"           .scheme.keyword
hilight .hilight.foo 1 "else"           .scheme.keyword
hilight .hilight.foo 1 "endif"          .scheme.keyword

; File hook − called when new file is loaded.
define−macro fhook−foo
    ; Assign the hilighting
    set−variable $buffer−hilight .hilight.foo
    ; Temporary comment to make sure that it works.
    ml−write "Loaded a foo file"
!emacro
```

```
; Define the indentation scheme
0 indent  .hilight.foo 2 10
indent .hilight.foo n "then" 4
indent .hilight.foo s "else" -4
indent .hilight.foo o "endif" -4

; Reset the hilighting printer format and define the color bindings.
0 hilight-print .hilight.foo
hilight-print .hilight.foo "i"  .scheme.comment
hilight-print .hilight.foo "b"  .scheme.keyword
hilight-print .hilight.foo "bi" .scheme.string .scheme.quote

; Macro to comment a line
define-macro foo-comment-line
    !while &gre &pdec @# 1 0
        beginning-of-line
        insert-string "#"
        beginning-of-line
        forward-line
    !done
!emacro

; Macro to remove a comment from a line
define-macro foo-uncomment-line
    !while &gre &pdec @# 1 0
        beginning-of-line
        -1 search-forward "#"
        backward-delete-char
        forward-line
    !done
!emacro

; File hook - called when new file is loaded.
define-macro fhook-foo
    ; if arg is 0 this is a new file so add template
    !if &not @#
        etfinsrt "foo"
    !endif
    ; Assign the hilighting
    set-variable $buffer-hilight .hilight.foo
    ; Assign the buffer indentation
    set-variable $buffer-indent .hilight.foo
    ; Set the abbreviation file
    buffer-abbrev-file "foo"
    ; Set up the buffer modes
    1 buffer-mode "time"
    1 buffer-mode "indent"
    ; Set up local bindings
    buffer-bind-key foo-comment-line "C-c C-c"
    buffer-bind-key foo-uncomment-line "C-c C-d"
    ; Temporary comment to make sure that it works.
    ml-write "Loaded a foo file"
!emacro

; Notification that hook is loaded.
ml-write "[MicroEmacs foo file hook loaded]"

; load in user extensions if found
!force execute-file "myfoo"
```

foo.eaf

```
if "if \p\rthen\rendif\P"
el "else\r\p\P"
```

foo.etf

```
#-!- foo -!- #################################
#
#  Created By    : $USER_NAME$
#  Created       : $ASCII_TIME$
#  Last Modified : <160495.1521>
#
#  Description
#
#  Notes
#
#  History
#
#  Copyright (c) $YEAR$ $COMPANY_NAME$.
##############################################
```

SEE ALSO

add−file−hook(2), buffer−abbrev−file(2), etfinsrt(3), execute−buffer(2), expand−abbrev(2), global−abbrev−file(2), hilight(2), scheme−editor(3), indent(2), indent(2m), restyle−buffer(3), restyle−region(3), time(2m), $buffer−hilight(5), $buffer−indent(5), etf(8), eaf(8), File Hooks.

fileHooks(2)

FILE HOOKS

File hooks provide a mechanism to automatically invoke a set of macros for a given buffer type when the following events occur:

- Loading of a file into a buffer
- Moving into a buffer (i.e. making a buffer current)
- Moving out of a buffer (i.e. making another buffer current)
- Deleting an active buffer

The file hook selection (see below) is performed on the file name / extension and on the textual content of the buffer using add−file−hook.

Refer to Language Templates for a description of how the file hooks are used to define a new template for a new text format.

The hook macros allow buffer modes and highlighting, applicable to the text type of the file, to be applied to the buffer. In addition, the associated hook macros may be located in a separate file and are loaded on demand when the file reading determines that a set of hook macros are required.

Consider a file hook definition of the form;

```
add-file-hook ".c .h" "fhook-c"
```

which binds the file hook **fhook−c** to any files that are loaded with the extension **.c** and **.h**. The operations undertaken by MicroEmacs '02 are defined as follows when a file `foo.c` is loaded:−

- Attempt to load file `foo.c`, if `foo.c` is not found then create a new buffer and assign file name `foo.c`.
- If `foo.c` is found then load file into buffer. Search the first line(s) of the buffer for magic hook text (*add−file−hook* with argument).
- If magic hook was not found then determine hook name from the file extension (*add−file−hook* information).
- If a hook command is located, assign the file hook **fhook−c** to the buffer, assign the buffer entry (begin) hook macro of **bhook−c**; assign a buffer exit hook of **ehook−c**.
- If the macro **fhook−c** is undefined then execute the macro file **hkc.emf** from the MicroEmacs home directory in an attempt to load the macro. If the file **myc.emf** is defined, then the modifications to the language template are applied after **hkc.emf** is loaded.
- If the macro **fhook−c** is (now) defined then `foo.c` is TEMPORARILY made the current buffer and the file hook macro **fhook−c** is executed to completion and the previous current buffer is restored. [*TEMPORARY* here implies that no buffer hooks are executed on the flip in/out of `foo.c`].
- The current buffer is officially swapped to `foo.c`. At this point the *ehook* of the old current buffer is executed (while its still current) and then `foo.c` is swapped in to become the current buffer; the begin buffer hook *bhook−cmode* is then executed for `foo.c` (if it exists).

- If the user moves to another buffer execute the end hook macro **ehook−cmode** (if it exists) and move to the new buffer, executing it's begin hook.
- If the user subsequently returns to buffer `foo.c` execute the previous buffers end hook macro, set the current buffer to *foo.c* and execute the begin hook macro **bhook−c** (if it exists).
- If the user kills buffer `foo.c`, if `foo.c` is the current buffer then an alternative buffer is made current, `ehook` and `bhook` executed as normal. If macro **dhook−c** is defined then `foo.c` is TEMPORARILY made the current buffer and the delete hook macro **dhook−c** is executed to completion and the previous current buffer is restored.

The name of the file hook macro name is important, hook commands must commence with the text **fhook−***mode* where *mode* is an identifier for the operating mode. The name space is decomposed as follows:−

- The initial **f** is removed and replaced with **b** for the begin hook macro and **e** for the end hook macro.
- When the **fhook** macro is undefined the *mode* component is removed and the file **hk***mode***.emf** is executed from the MicroEmacs home directory in an attempt to define the macro.

The **fhook−** nomenclature may be omitted provided that the name is less than 6 characters, however the file, begin and end hook macros MUST commence with **f**, **b** and **e** respectively. In addition the macros must be defined as no auto file loading is performed.

Buffer Hook Variables

The macros bound to a buffer may be interrogated, the variables $buffer−fhook(5), $buffer−bhook(5), $buffer−ehook(5) and $buffer−dhook(5) contain the names of any associated macro attached as a macro hooks, defining the *file*, *begin*, *end* and *delete* hooks respectively. If a macro is not bound then the empty string `""` is returned. Setting the variables has the effect of defining the hook and is a method by which the buffer hooks may be affected after the buffer has been loaded.

Determination of a new file

The *file* hook **fhook−XXX** numeric argument may be used to determine if the file associated with a buffer is a new file created by the user, or an existing file. Typically this distinction is used to determine whether a boiler template is added to the file or not. The macro argument **@#** is defined as zero (0) if this is a new file that has been created, or non−zero otherwise.

The macro argument status is typically tested on entry to the macro as follows:−

```
define−macro fhook−mode
    !if &not @#
        ; This is a new file. Do new file things
    !else
        ; This is an existing file
    !endif
    ; Set up bindings
!emacro
```

An example of a generic **hook** file is given at the end of this section which elaborates on the file hooks.

Begin and End hooks

The *begin* and *end* hooks are usually used to save and restore global states which require special settings for a particular buffer type. This typically involves saving and restoring global variables which are used by other buffers in a different configuration. For example the following is used to reformat the time stamp string; the time stamp is a global variable $timestamp(5) and if it is changed in one buffer, it must be restored ready for another. In this case the old time stamp is retained in a local buffer variable whenever the buffer is entered, the time stamp is then modified for the buffers requirements. On exit from the buffer the old time stamp format is restored to it's former state.

```
0 define-macro bhook-foo
    set-variable .timestamp $timestamp      ; Save old time stamp.
    set-variable $timestamp "19%Y/%M/%D %h:%m:%s"
!emacro

0 define-macro ehook-foo
    set-variable $timestamp .bhook-foo.timestamp
!emacro
```

Note that in both cases the define−macro(2) invocation is defined as zero, this merely hides the macro from the command line since both are private macros not normally invoked by the user.

FILE HOOK SELECTION

MicroEmacs '02 may be reconfigured to operate in different modes (referred to a *Major Modes* in GNU **emacs(1)**) using the macro file hooks. The file hooks allow the working environment to be customized for the editing of text of a particular sort, by importing text specific macros, key rebinding and highlighting.

MicroEmacs '02, by default, loads a file into a buffer with default global modes with no highlighting. There are no mode specific key bindings, variable settings, macros or highlights, buffer interaction behaves in it's default state. The state of the buffer interaction may be modified through the use of the buffer modes (see Operating Modes), for example the 'C' programming language cmode(2m) changes the characteristics of the `tab` character and performs language specific indentation of statements. When a text specific set of highlighting rules are applied to the buffer, the text becomes emphasized through the use of color applied selectively to the text i.e. comments, keywords, strings are shown in different colors, allowing them to be differentiated without studying the content.

Setting the operating mode of the buffer would be tedious to perform from the command line, instead MicroEmacs '02 uses three different prioritized criteria to endeavor to select the correct operating mode. The operating mode is applied to the buffer by execution of a set of file specific macros, referred to a hook commands. The selection criteria of the hook commands is performed as follows, ordered in lowest to highest priority:−

File Name

MicroEmacs '02 uses the filename and/or the file extension to select a start−up hook command. File names and extensions are bound to a set of macro hooks in a space separated list e.g.

> add−file−hook "**c cpp**" "**fhook−cmode**"
> add−file−hook "**doc txt README**" "**fhook−doc**"

The space separated list of names are interpreted as either file extensions or filenames. In this case any file with the extension **.c**, **.cpp** is bound to a file hook called **fhook−cmode** e.g. `foo.c`. Similarly files with the extension **.doc** or **.txt** are interpreted as plain text documents and are bound to **fhook−doc**. e.g. `foo.txt`. The entry **README** that exists in the documentation hook list may refer to a file `README` and also `foo.README`, both cases invoke the document hook.

The file selection is the lowest priority selection criteria but usually satisfies most mode selection requirements.

Magic Strings

There are cases when file extensions may be omitted from files, typically these files include an identifier, or magic string, on the first line of the file which is used to identify the file to the operating system or application e.g. shell scripts under UNIX. MicroEmacs '02 automatically interrogates the top of every file that is loaded to locate some form of identification string. The identification strings are defined in a similar way to the file name hooks, except instead of defining a file extension the location and text content of the identifier is defined:

> 1 add−file−hook "**#!/bin/sh**" "**fhook−shell**"
> 1 add−file−hook "**#!/usr/local/bin/wish**" "**fhook−tcl**"

In this case, any file that commences with "**#!/bin/sh**" is interpreted as a shell script and invokes the shell hook **fhook−shell**. Where the identifier does not appear on the first non−blank line, the argument may be increased to the number of lines to be searched. Also it the magic sting should be search for without exact(2m) mode then the argument should be negated, e.g.

> −4 **add−file−hook** "<html>" "**fhook−html**"

invokes **fhook−html** whenever "`<html>`", "`<HTML>`" etc. is found in the first 4 lines of a file header, e.g.:

```
<!-- Comment line -->
<HtMl>
```

A match on a string identifier is assigned a higher priority than the file extension. It is recommended that magic strings are only used where there are no predefined file extensions, or conflicts exist between files with the same extension containing data interpreted in a different context.

Explicit Strings

The last method allows an explicit identifier string to be embedded into the text of the file informing MicroEmacs '02 which mode it should adopt. GNU Emacs supports this (see **Major Mode** in the GNU Emacs documentation) type of operation by insertion of strings of the form:

−*− *mode* **−*−**

Where *mode* represents the major mode within GNU Emacs. The same format as used by **Magic Strings** can be used to find and extract the *mode*, e.g.:

−1 add−file−hook "**−[*!]−[\t]nroff.*−[*!]−**" "**fhook−nroff**"

The definition would detect the GNU Emacs mode defined in an Nroff file e.g.

.\" −*− nroff −*− "
.TH man 1
.SH NAME
...

It should be stressed that the **−*−** syntax belongs to GNU Emacs and NOT MicroEmacs '02, MicroEmacs '02 provides a mechanism to locate, extract and interpret the string. The **−*−** syntax should only be applied to files if it is known that the *mode* is a GNU mode.

A MicroEmacs '02 specific string is also provided, defined as:

−!− *mode* **−!−**

where *mode* is an arbitrary string defined by *add−file−hook*. User defined modes may be created and assigned to files with this syntax, this does not conflict with the GNU Emacs command. For example to assign a new mode *mymode* to a file we would define the following:−

−1 add−file−hook "**−!−[\t]mymode.*−!−**" "**fhook−mymode**"

Files containing a the following identifier would be loaded with *mymode* hook:

# −!− mymode −!−
#
# Last Modified: <120683.1014>

FILE HOOK SCRIPTS

The buffer hook files **hk***name***.emf** typically follow a standard layout, and are generally associated with hi−lighting as follows, **mode** in this case is the name of the file mode associated with the file:−

```
!if &seq .hilight.mode "ERROR"
```

```
    set-variable .hilight.mode &pinc .hilight.next 1
!endif
;
; Define the hilighting
;
0 hilight .hilight.mode 1                $global-scheme
hilight .hilight.mode 2 "\*\*"           .scheme.comment
hilight .hilight.mode 4 "\"" "\"" "\\"   .scheme.string
hilight .hilight.mode 0 "'.'"            .scheme.quote

hilight .hilight.mode 1 "if"             .scheme.keyword
hilight .hilight.mode 1 "elif"           .scheme.keyword
hilight .hilight.mode 1 "else"           .scheme.keyword
...

; Reset the hilighting printer format and define the color bindings.
0 hilight-print .hilight.mode
hilight-print .hilight.mode "i"  .scheme.comment
hilight-print .hilight.mode "b"  .scheme.keyword .scheme.variable
hilight-print .hilight.mode "bi" .scheme.string .scheme.quote
...

; Define the indentation tokens
0 indent  .hilight.mode 2 10
indent .hilight.mode n "if" 4
indent .hilight.mode s "elif" -4
indent .hilight.mode s "else" -4
indent .hilight.mode o "endif" -4
indent .hilight.mode n "while" 4
...

define-macro fhook-mode
    ; if arg is 0 this is a new file so add template
    !if &not @#
        etfinsrt "mode"
    !endif
    set-variable $buffer-hilight .hilight.mode
    set-variable $buffer-indent .hilight.mode
    1 buffer-mode "time"
    1 buffer-mode "indent"
    buffer-abbrev-file "mode"
!emacro
```

The previous example shows how the **fhook−mode** numeric argument is used to determine if this is a new file. If the argument **@#** is zero then this is interpreted as a new file, in this case a standard template is inserted (from file **mode.etf**) and the generic strings such as $YEAR$ replaced with construction information. The template is generally used for standard headers and skeleton text body.

In addition an abbreviation file **mode.eaf** (see eaf(8)) is bound to the buffer using the buffer−abbrev−file(2) command and the buffer hi−lighting enabled by assignment of the $buffer−hilight(5) variable.

MODIFYING FILE HOOKS

The standard hook files supplied with MicroEmacs '02 should not be modified, changes to the file hooks may be applied using a separate macro file called **my***XXX.emf*, this is automatically executed after the **hk***XXX.emf* file is executed.

The extended hook functions may be defined company wide, or by the user, to over−ride some of the standard hook functions, or to extend the syntax of the base files with locally defined extensions. As an example, consider the following file **myc.emf** which extends the basic **hkc.emf** file set of hi−lighting tokens for the 'C' Language.

```
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;
;  Created By    : Steven Phillips
;  Created       : Thu Jun 18 15:34:05 1998
;  Last Modified : <230798.0854>
;
;  Description   Extension hilighting for the 'C' language.
;
;  Notes         Define the locally defined 'C' library types and definitions
;                as extensions to the 'C' programming language.
;
;  History
;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

; MicroEmacs specific tokens
hilight .hilight.c 1 "LINE"           .scheme.type
hilight .hilight.c 1 "BUFFER"         .scheme.type
hilight .hilight.c 1 "WINDOW"         .scheme.type
hilight .hilight.c 1 "REGION"         .scheme.type
hilight .hilight.c 1 "KEYTAB"         .scheme.type
hilight .hilight.c 1 "KILL"           .scheme.type
hilight .hilight.c 1 "KLIST"          .scheme.type
hilight .hilight.c 1 "HILNODE"        .scheme.type
hilight .hilight.c 1 "HILNODEPTR"     .scheme.type
hilight .hilight.c 1 "HILCOLOR"       .scheme.type
hilight .hilight.c 1 "SELHILIGHT"     .scheme.type
hilight .hilight.c 1 "VIDEO"          .scheme.type
hilight .hilight.c 1 "VVIDEO"         .scheme.type
hilight .hilight.c 1 "FRAMELINE"      .scheme.type
hilight .hilight.c 1 "IPIPEBUF"       .scheme.type
hilight .hilight.c 1 "DIRNODE"        .scheme.type
hilight .hilight.c 1 "UNDOND"         .scheme.type
hilight .hilight.c 1 "meVARLIST"      .scheme.type
hilight .hilight.c 1 "meVARIABLE"     .scheme.type
hilight .hilight.c 1 "meCMD"          .scheme.type
hilight .hilight.c 1 "meAMARK"        .scheme.type
hilight .hilight.c 1 "meABREV"        .scheme.type
hilight .hilight.c 1 "meMACRO"        .scheme.type
hilight .hilight.c 1 "meNARROW"       .scheme.type
hilight .hilight.c 1 "meREGISTERS"    .scheme.type
hilight .hilight.c 1 "meSTAT"         .scheme.type
hilight .hilight.c 1 "osdITEM"        .scheme.type
hilight .hilight.c 1 "osdDIALOG"      .scheme.type
hilight .hilight.c 1 "osdCHILD"       .scheme.type
hilight .hilight.c 1 "meSCROLLBAR"    .scheme.type
hilight .hilight.c 1 "osdCONTEXT"     .scheme.type
hilight .hilight.c 1 "osdDISPLAY"     .scheme.type
hilight .hilight.c 1 "RNODE"          .scheme.type
hilight .hilight.c 1 "REGHANDLE"      .scheme.type
hilight .hilight.c 1 "meDIRLIST"      .scheme.type
hilight .hilight.c 1 "meNAMESVAR"     .scheme.type
hilight .hilight.c 1 "meDICTADDR"     .scheme.type
hilight .hilight.c 1 "meSPELLRULE"    .scheme.type
hilight .hilight.c 1 "meDICTWORD"     .scheme.type
hilight .hilight.c 1 "meDICTIONARY"   .scheme.type
hilight .hilight.c 1 "meMODE"         .scheme.type
```

SEE ALSO

Operating Modes, Language Templates, add−file−hook(2), cmode(2m).

Editor File Types

EDITOR FILE TYPES

Different file types used by MicroEmacs '02:

eaf(8) MicroEmacs abbreviation file format
edf(8) MicroEmacs spelling dictionary file
ehf(8) MicroEmacs help file
emf(8) MicroEmacs macro file
erf(8) MicroEmacs registry file
etf(8) MicroEmacs template file format

Compatibility(2)

COMPATIBILITY

JASSPA MicroEmacs is based on the original version of **MicroEMACS** produced by Danial Lawrence at revision 3.8, the source files were obtained in approximately 1990. The exact origin of the files is unknown. In that period of time the source files have undergone an awful lot of change, without reference to the subsequent releases made of MicroEMACS by Danial Lawrence (due to no network access). As a result the JASSPA version of **MicroEmacs** does not include any modifications or features that may have been implemented since. This version of **MicroEmacs** has been tailored to suite the requirements of a small group of individuals who have used the editor on a daily basis across a limited number of platforms, for a variety of very different tasks and operating requirements.

This version of MicroEmacs is biased towards UNIX environments, MS−DOS and Microsoft Windows ports have been performed however they are heavily influenced by UNIX and inherit UNIX characteristics wherever possible. The intention is that programmers, and alike, may move across platforms using a common editor environment without being frustrated by the idiosyncrasies of different platforms. The most noticeable platform is the Microsoft Windows platform which mimics the X−Windows cut and paste mechanism within the MicroEmacs environment. If you want a Windows style environment then use **Notepad(1)** or **Wordpad(1)**, this editor is not for you !!

The gross changes to **MicroEmacs '02** are summarized as follows:−

- Macro language interpreter re−written allowing an unlimited number of named macros to be supported. The macro implementation allows new commands to be created by the user, as opposed to continually extending the underlying command set. The named macros are transparent to the user, appearing as built in commands on the command line. Macro command set significantly increased. Support for global, buffer and register variables within the macro language.
- Display drivers re−written providing color hilighting support on most platforms. A macro interface allows information to be written directly to the display canvas allowing the screen to be annotated with additional transient information.
- Support for X−Window screen type in UNIX environments. Microsoft Window's environments (3.x, '95, NT) treated in the same was as X−Windows − this may be unorthodox for existing Window's users, UNIX users will find it more comfortable.
- Introduction of integrated spell checker. Support includes correction word guessing, word auto−correction and double word detection. Ignore and personal dictionaries supported.
- Horizontal window splitting.
- Introduction of scroll bars on all platforms that support a mouse. The scroll bar implementation is platform independent.
- Command and file completion available on all platforms. Most commands support a command history allowing previous command invocations to be recalled.
- Session history file kept, allowing the previous edit session to be reinstated.
- Undo capability, allows previous edits to be undone when mistakes are made.
- Backup capability, Includes a periodic timed backup while an editing session is in progress. The timed backup is automatically recovered by the next session in situations where the system (or editor) crashes.

- A regular expression incremental search becomes the default search forward mechanism.
- Support for abbreviation files allowing frequently used constructs to be automatically expanded.
- Automatic time stamping of files, allowing the edit time to be automatically maintained in the source file(s).
- Introduction of an electric 'C' mode. Editor intelligently handles the layout of 'C' files (under user control).
- Improved documentation text mode providing left/right/center and both justification methods with inclusion for bullet points. Automatic justification may be continually performed as text is entered, thereby maintaining the paragraph in the correct format.
- Integrated on−line help facilities. All commands are documented on−line. New macros may be documented within the macro files and become part of the help system.
- File type determination system, based on either the file name or embedded file text allows file type specific macros (hooks) to be applied, thereby configuring the editor into the correct mode for the file type.
- Introduction of special MicroEmacs search path allowing all of the standard configuration files to be utilized from a shared directory.

The name space of JASSPA MicroEmacs differs from the original MicroEMACS and has become more compliant with the GNU implementation of Emacs. A list of the original MicroEMACS verses the new command name set is as follows, executing the compatibility macro file `meme3_8.emf` will create macro versions of these commands:

add−global−mode => global−mode
add−mode => buffer−mode
apropos => command−apropos
backward−character => backward−char
begin−macro => start−kbd−macro
beginning−of−file => beginning−of−buffer
bind−to−key => global−bind−key
buffer−position => buffer−info
case−region−lower => lower−case−region
case−region−upper => upper−case−region
case−word−capitalize => capitalize−word
case−word−lower => lower−case−word
case−word−upper => upper−case−word
change−screen−depth => change−frame−depth
change−screen−width => change−frame−width
clear−message−line => ml−clear
ctlx−prefix => prefix 2
delete−global−mode => global−mode
delete−mode => buffer−mode
delete−next−character => forward−delete−char
delete−next−word => forward−kill−word
delete−previous−character => backward−delete−char
delete−previous−word => backward−kill−word
end−macro => end−kbd−macro
end−of−file => end−of−buffer
execute−command−line => execute−line

execute−macro => execute−kbd−macro
execute−macro−# => *Deleted*
file−name−insert => insert−file−name
forward−character => forward−char
grow−window => grow−window−horizontally
handle−tab => tab
i−shell => shell
incremental−search => isearch−forward
kill−to−end−of−line => kill−line
meta−prefix => prefix 1
move−window−down => scroll−down
move−window−up => scroll−up
name−buffer => change−buffer−name
next−line => forward−line
next−page => scroll−down
next−paragraph => forward−paragraph
next−word => forward−word
open−line => insert−newline
pipe−command => pipe−shell−command
previous−line => backward−line
previous−page => scroll−up
previous−paragraph => backward−paragraph
previous−word => backward−word
quote−character => quote−char
redraw−display => recenter
restore−window => goto−position
reverse−incremental−search => isearch−backward
save−file => save−buffer
save−window => set−position
scroll−next−down => scroll−next−window−down
scroll−next−up => scroll−next−window−up
search−reverse => search−backward
select−buffer => find−buffer
set => set−variable
shrink−window => shrink−window−vertically
split−current−window => split−window−vertically
top−bottom−switch => *Deleted*
transpose−characters => transpose−chars
unbind−key => global−unbind−key
update−screen => screen−update
write−message => ml−write

Interfacing(2)

INTERFACING

This sections describes how MicroEmacs '02 may be interfaced to external components.

Shells

A shell window may be opened within the context of the editor using the command ishell(3), whereby an interactive command shell is presented within a buffer.

In the Microsoft Windows environment a **cygnus** UNIX style BASH shell may be realised with the cygnus(3) command.

Debugger

Within the UNIX environment the GNU **gdb(1)** or native UNIX **dbx(1)** debuggers may be invoked from the editor using gdb(3) or dbx(3). respectively This invokes the debugger and follows the debugging process in the editor window, automatically opening the source files as the debugger calls for them.

Microsoft Developer Studio

In the Microsoft windows environment, the memsdev(1) DLL may be attached to the **Microsoft Developer Studio** to enable MicroEmacs '02 to be used in place of the in−built editor.

File Searching

File searching is performed using **grep(1)** using the grep(3) command. For Windows then the GNU grep utility is recommended, for MS−DOS then the DJGPP version of GNU grep is recommended.

File Differencing

Differencing files, or directories is performed using the **diff(1)** utility using the diff(3) command. For all platforms the GNU diff utility is recommended as this provides a comprehensive differencing that is not typically available with native UNIX diff utilities.

Tag Files

A **tag** capability exists (see find−tag(2)) such that source functions and alike may be located quickly using a **tags** file. The standard **ctags(1)** format is used by MicroEmacs. The **tags** file itself may be generated by MicroEmacs '02 from the menu (*Tools−>XX Tools−>Create Tags File*). Alternatively a **tags** file may be generated by the **ctags(1)** utility. This is typically standard on UNIX platforms. For Windows and DOS platforms then the **Exuberant Ctags** is recommended, this is available from:−

```
    http://darren.hiebert.com
```

A MicroEmacs '02 compatible tags file may be generated using the command line "`ctags −N −−format=1 .`" cataloging the current directory. To generate **tags** for a directory tree then use "`ctags −NR −−format=1 .`". Refer to the **Exuberant Ctags** documentation for a more detailed description of the utility.

Compilation

Compilation is performed using the compile(3) command. This invokes a command shell, typically using **make(1)** to initiate a build sequence.

Client−Server

The Client−Server interface allows other client applications to inject commands into an already existing MicroEmacs '02 session (the server), thereby controlling the editor remotely. This is typically used to inject new files into the editor to be presented to the user.

The *Client−Server* interface is available in both the UNIX and Microsoft Windows environments. This mechanism is used in the Microsoft windows environment by the memsdev(1) DLL to attach the **Microsoft Developer Studio** to MicroEmacs '02. This may be used with similar effects within the UNIX environments from the X−Window managers desktop in addition to other utilities such as **TkDesk(1)**.

Command Line Filer

MicroEmacs may be invoked as a command filter in it's own right, macro scripts have been developed to perform a **dos2unix(1)** conversion operation, generate tags files etc. See Command Line Filters.

SEE ALSO

ctags(1), compile(3), cygnus(3), dbx(3), diff(3), find−tag(2) gdb(3), grep(3), ishell(3), memsdev(1), Client−Server, Command Line Filters.

Supported File Types

SUPPORTED FILE TYPES

The file types currently supported by MicroEmacs '02 are defined in the following list. Other file types may be supported by definition of an appropriate hook function to handle the file, see fileHooks(2).

0−9(9) UNIX t/nroff file
asm(9) Assembler File
asn.1(9) ASN.1 file
awk(9) AWK File
bas(9) Visual Basic
bat(9) MS−DOS Batch File
bnf(9) Backus−Naur Form
btm(9) 4−DOS Batch File
c(9) C programming language
cbl(9) Cobol (85) File
cc(9) C++ programming language
cls(9) Visual Basic
cpp(9) C++ programming language
csh(9) C−Shell file
def(9) C or C++ definition file
doc(9) ASCII plain text document file
ehf(9) MicroEmacs '02 help file
emf(9) MicroEmacs '02 Macro File
erf(9) MicroEmacs '02 registry file
f(9) Fortran File
f77(9) Fortran 77 File
f90(9) Fortran 90 File
fvwm(9) FVWM configuration file
fvwmrc(9) FVWM configuration file
gawk(9) GNU AWK File
h(9) C programming language header
hpj(9) MS−Windows Help Project File
htm(9) HyperText Markup Language File
html(9) HyperText Markup Language File
i(9) C/C++ preprocessor output file
imakefile(9) Make file
info(9) GNU Info file
ini(9) MS−Windows Initialization File
jav(9) Java programming language
java(9) Java programming language
ksh(9) Korn shell file
l(9) LEX programming language
latex(9) TeX Documentation
login(9) Shell user login file

MetaFont(9) MetaFont/MetaPost File
m4(9) M4 Macro Processor
makefile(9) Make file
man(9) UNIX Manual Page
mf(9) MetaFont File
mp(9) MetaPost File
nawk(9) New AWK File
nroff(9) UNIX nroff file
p(9) Pascal File
pas(9) Pascal File
perl(9) Practical Extraction and Report Language File
pl(9) Practical Extraction and Report Language File
pm(9) Practical Extraction and Report Language File
profile(9) Shell user profile
py(9) Python Language File
python(9) Python Language File
rc(9) Microsoft Developer resource file
reg(9) Registry file
rgy(9) Registry file
rul(9) Install Shield Rules
s(9) Assembler File
sch(9) Scheme File
scheme(9) Scheme File
scm(9) Scheme File
sh(9) Bourne shell file
so(9) UNIX t/nroff include file
sql(9) SQL File
tcl(9) TCL programming language
tcshrc(9) T−Shell start up file
tex(9) TeX Documentation
texi(9) GNU Texinfo documentation file
texinfo(9) GNU Texinfo documentation file
tk(9) TK programming language
tni(9) UNIX t/nroff include file
troff(9) UNIX troff file
txt(9) ASCII plain text file
vb(9) Visual Basic
vhdl(9) VHDL hardware simulation File
vrml(9) VRML File
wish(9) TCL shell file
x86(9) Intel .x86 Assembler File
y(9) YACC programming language
zsh(9) Z−Shell file

Client−Server(2)

CLIENT−SERVER

This sections describes how MicroEmacs '02 may be interfaced to external components through the **Client−Server** interface.

The **Client−Server** interface of MicroEmacs '02 provides a capability for other applications to inject commands into a running version of the editor, which are interpreted and executed. The interface is only available on multi−tasking operating systems such as UNIX and Microsoft Windows; it is not available on MS−DOS systems.

Within the following discussions, the **Server** is a running version of the MicroEmacs '02 editor; the **client** is the application (or shell script) that communicates a new command to the *server*.

The **Client−Server** interface may provide a bidirectional interface such that a *client* may submit a command to the *server* and may also retrieve a response to that command.

DESCRIPTION

The **Client−Server** interface operates by making an external interface available which is continually monitored by the *server*. The external interface may be provided by a file, named pipe or socket (depending upon the platform) with a well know location in the file system. Typically two files are provided, an input file into which the *client* writes commands (*$TEMP*/**me***$MENAME***.cmd**); and an output file where responses to those commands my be read (*$TEMP*/**me***$MENAME***.rsp**).

Within MicroEmacs, the client server interface appears as a hidden ipipe−shell−command(2) buffer, with the name `*server*`. Commands are received through this buffer and responses are written back to the buffer.

Client Commands

Clients may write directly to the *command* through the use of explicit embedded code, or may use a me(1) invocation with the **−m** option. Commands to the client interface take the form "**C:**<*client*>**:**<*command*>".

<*client*>

<*client*> is an identification string that may be used to identify the client, this information may be used when the command is handled to interpret the command if some special client specific action is required.

<*command*>

The <*command*> is an editor command (or macro) of the given name with any arguments.

Standard command escape sequences must be adhered to. i.e. to write "`Hello World`" on the message line then a client may issue the command:−

```
me −m "C:<client>:ml−write \"Hello world\"
```

The *client−server* interface is typically used to load a file, this may be performed as follows:−

```
me −m "C:<client>:find−file \"/path/foo.bar\""
```

The absolute path is specified in this type of transaction as the current working directory of the active MicroEmacs session is unknown. The **−m** option de−iconize's the existing editor session and bring it to the foreground.

Client Responses

Responses from *client* commands are written to the response file, responses take a similar form to *client* commands except they are prefixed by an **R**, i.e. "**R:**<*client*>**:**<*data*>".

As multiple *clients* may be utilizing the *client−server* mechanism then the <*client*> sting passed in the command is typically returned in the response to allow the *client* to identify it's own response (rather than any other *clients*. It is the *clients* responsibility that this string is unique in order that it may be differentiated.

The returned <*data*> format is undefined and would be generated by a macro command used to handle the *client* command; sufficient to say that the data should exist on a single line.

Server Side

On the *server* side, the **Client−Server** interface is managed like an ipipe−shell−command(2) using the hidden buffer `*server*` (as previously mentioned).

The *Client−Server* interface is enabled from the user−setup(3) interface, the user setting of the interface is confirmed by checking bit `0x20000` of the $system(5) variable.

The client server interface is typically initialized within the `me.emf` initialization file, whereby the *ipipe* input handler is bound to the client pipe buffer and the buffer is hidden, so it is not available when the buffers are swapped. (Note that the client buffer may be explicitly interrogated using find−buffer `*server*`). The client handler is installed as follows:−

```
; Setup the Client Server
!if &band $system 0x20000
    define−macro−file meserver server−input
    find−buffer "*server*"
    set−variable :last−line 2
    set−variable :client−list ":"
    set−variable $buffer−ipipe server−input
    beginning−of−buffer
    goto−alpha−mark  "I"
    −1 find−buffer "*server*"
```

```
        !endif
```

This binds a MicroEmacs macro called *server−input* to handle the client commands as they arrive on the input, an alpha−mark is used to record the processed position at the end of the buffer. The pipe handler itself decodes the client request and executes it. The default handler supplied with MicroEmacs '02 is defined within the macro file `meserver.emf`

Responses to the client are inserted into the response file by writing directly into the ipipe buffer (`*server*`) using the ipipe−write(2) command. It is the calling macros responsibility to ensure that the response string adheres to the format outlined above in the previous sections.

NOTES

It is not possible to kill the `*server*` buffer, and ipipe−kill(2) is ignored within the context of the buffer.

FILES

`meserver.emf` − Default Client−Server ipipe handler.
$TEMP*/**me***$MENAME.**cmd** − Command file.
$TEMP*/**me***$MENAME.**rsp** − Response file.

BUGS

The first MicroEmacs '02 session that executes becomes the editor server, additional editor sessions that are executed do not become server processes. In the event that the *server* editor is terminated, any other sessions do not take over the role of server. Subsequently issuing a client command may fail, or invoke a new editor session which adopts the role of server.

SEE ALSO

me(1), ipipe−shell−command(2)

RegularExpressions(2)

REGULAR EXPRESSIONS

Regular Expressions are used in the search (and replace) operations. The following notes are applicable when magic(2m) mode is enabled.

Overview

A "*regular expression*" (or "*regex*", or "*pattern*") is a text string that describes some (mathematical) set of strings. A regex **R** "*matches*" a string **S** if **S** is in the set of strings described by **R**.

MicroEmacs '02 includes the GNU **reg**ular **exp**ression pattern matcher library, **regex** which provides a powerful search engine, using the search engine you can:

- see if a string matches a specified pattern as a whole, and
- search within a string for a substring matching a specified pattern.

Some regular expressions match only one string, i.e., the set they describe has only one member. For example, the regular expression '`foo`' matches the string '`foo`' and no others. Other regular expressions match more than one string, i.e., the set they describe has more than one member. For example, the regular expression '`f*`' matches the set of strings made up of any number (including zero) of '`f`'s. As you can see, some characters in regular expressions match themselves (such as '`f`') and some don't (such as '`*`'); the ones that do not match themselves instead let you specify patterns that describe many different strings.

Syntax of Regular Expressions

Regular expressions have a syntax in which a few characters are special constructs and the rest are "*ordinary*". An ordinary character is a simple regular expression which matches that same character and nothing else. The special characters are '`$`', '`^`', '`.`', '`*`', '`+`', '`?`', '`[`', '`]`' and '`\`'. Any other character appearing in a regular expression is ordinary, unless a '`\`' precedes it.

For example, '`f`' is not a special character, so it is ordinary, and therefore '`f`' is a regular expression that matches the string '`f`' and no other string. (It does **not** match the string '`ff`'.) Likewise, '`o`' is a regular expression that matches only '`o`'. (When case distinctions are being ignored, these regexs also match '`F`' and '`O`', but we consider this a generalization of "*the same string*", rather than an exception.)

Any two regular expressions `A` and `B` can be concatenated. The result is a regular expression which matches a string if `A` matches some amount of the beginning of that string and `B` matches the rest of the string.

As a simple example, we can concatenate the regular expressions '`f`' and '`o`' to get the regular expression '`fo`', which matches only the string '`fo`'. Still trivial. To do something nontrivial, you need to use one of the special characters. Here is a list of them.

. (Period)

is a special character that matches any single character except a newline. Using concatenation, we can make regular expressions like '`a.b`', which matches any three−character string that begins with '`a`' and ends with '`b`'.

***** (asterisk)

is not a construct by itself; it is a postfix operator that means to match the preceding regular expression repetitively as many times as possible. Thus, '`o*`' matches any number of '`o`'s (including no '`o`'s).

> '`*`' always applies to the **smallest** possible preceding expression. Thus, '`fo*`' has a repeating '`o`', not a repeating '`fo`'. It matches '`f`', '`fo`', '`foo`', and so on.
>
> The matcher processes a '`*`' construct by matching, immediately, as many repetitions as can be found. Then it continues with the rest of the pattern. If that fails, backtracking occurs, discarding some of the matches of the '`*`'−modified construct in case that makes it possible to match the rest of the pattern. For example, in matching '`ca*ar`' against the string '`caaar`', the '`a*`' first tries to match all three '`a`'s; but the rest of the pattern is '`ar`' and there is only '`r`' left to match, so this try fails. The next alternative is for '`a*`' to match only two '`a`'s. With this choice, the rest of the regex matches successfully.
>
> `+` (plus) is a postfix operator, similar to '`*`' except that it must match the preceding expression at least once. So, for example, '`ca+r`' matches the strings '`car`' and '`caaaar`' but not the string '`cr`', whereas '`ca*r`' matches all three strings.

'**?**' (question mark)

is a postfix operator, similar to '`*`' except that it can match the preceding expression either once or not at all. For example, '`ca?r`' matches '`car`' or '`cr`'; nothing else.

[...]

is a "character set", which begins with '`[`' and is terminated by '`]`'. In the simplest case, the characters between the two brackets are what this set can match.

> Thus, '`[ad]`' matches either one '`a`' or one '`d`', and '`[ad]*`' matches any string composed of just '`a`'s and '`d`'s (including the empty string), from which it follows that '`c[ad]*r`' matches '`cr`', '`car`', '`cdr`', '`caddaar`', etc.
>
> You can also include character ranges in a character set, by writing the starting and ending characters with a '`−`' between them. Thus, '`[a−z]`' matches any lower−case ASCII letter. Ranges may be intermixed freely with individual characters, as in '`[a−z$%.]`', which matches any lower−case ASCII letter or '`$`', '`%`' or period.
>
> Note that the usual regex special characters are not special inside a character set. A completely different set of special characters exists inside character sets: '`]`', '`−`' and '`^`'.

To include a '`]`' in a character set, you must make it the first character. For example, '`[]a]`' matches '`]`' or '`a`'. To include a '`−`', write '`−`' as the first or last character of the set, or put it after a range. Thus, '`[]−]`' matches both '`]`' and '`−`'.

To include '`^`' in a set, put it anywhere but at the beginning of the set.

When you use a range in case−insensitive search, you should write both ends of the range in upper case, or both in lower case, or both should be non−letters. The behavior of a mixed−case range such as '`A−z`' is somewhat ill−defined, and it may change in future Emacs versions.

[^ ...]

'`[^`' begins a "*complemented character set*", which matches any character except the ones specified. Thus, '`[^a−z0−9A−Z]`' matches all characters ***except*** letters and digits.

'`^`' is not special in a character set unless it is the first character. The character following the '`^`' is treated as if it were first (in other words, '`−`' and '`]`' are not special there).

A complemented character set can match a newline, unless newline is mentioned as one of the characters not to match. This is in contrast to the handling of regexs in programs such as **grep(1)**.

^ (caret)

is a special character that matches the empty string, but only at the beginning of a line in the text being matched. Otherwise it fails to match anything. Thus, '`^foo`' matches a '`foo`' that occurs at the beginning of a line.

$ (dollar)

is similar to '`^`' but matches only at the end of a line. Thus, '`x+$`' matches a string of one '`x`' or more at the end of a line.

\ (backslash)

has two functions: it quotes the special characters (including '`\`'), and it introduces additional special constructs.

Because '`\`' quotes special characters, '`\$`' is a regular expression that matches only '`$`', and '`\[`' is a regular expression that matches only '`[`', and so on.

Note: for historical compatibility, special characters are treated as ordinary ones if they are in contexts where their special meanings make no sense. For example, '`*foo`' treats '`*`' as ordinary since there is no preceding expression on which the '`*`' can act. It is poor practice to depend on this behavior; it is better to quote the special character anyway, regardless of where it appears.

For the most part, '\' followed by any character matches only that character. However, there are several exceptions: two−character sequences starting with '\' that have special meanings. The second character in the sequence is always an ordinary character when used on its own. Here is a table of '\' constructs.

\| (bar)

specifies an alternative. Two regular expressions `A` and `B` with '`\|`' in between form an expression that matches some text if either `A` matches it or `B` matches it. It works by trying to match `A`, and if that fails, by trying to match `B`.

Thus, '`foo\|bar`' matches either 'foo' or 'bar' but no other string.

'`\|`' applies to the largest possible surrounding expressions. Only a surrounding '`\( ... \)`' grouping can limit the grouping power of '`\|`'.

Full backtracking capability exists to handle multiple uses of '`\|`'.

\(... **\)**

is a grouping construct that serves three purposes:

- To enclose a set of '`\|`' alternatives for other operations. Thus, '`\(foo\|bar\)x`' matches either '`foox`' or '`barx`'.
- To enclose a complicated expression for the postfix operators '`*`', '`+`' and '`?`' to operate on. Thus, '`ba\(na\)*`' matches '`bananana`', etc., with any (zero or more) number of '`na`' strings.
- To record a matched substring for future reference. This last application is not a consequence of the idea of a parenthetical grouping; it is a separate feature that is assigned as a second meaning to the same '`\( ... \)`' construct. In practice there is no conflict between the two meanings.

'\D'

matches the same text that matched the Dth occurrence of a `` `\( `` ... `\)` ' construct.

After the end of a '`\( ... \)`' construct, the matcher remembers the beginning and end of the text matched by that construct. Then, later on in the regular expression, you can use '\' followed by the digit `D` to mean "match the same text matched the `D`th time by the '`\(` ... `\)`' construct."

The strings matching the first nine '`\(` ... `\)`' constructs appearing in a regular expression are assigned numbers 1 through 9 in the order that the open−parentheses appear in the regular expression. So you can use '`\1`' through '`\9`' to refer to the text matched by the corresponding '`\(` ... `\)`' constructs.

For example, '`\(.*\)\1`' matches any newline−free string that is composed of two identical halves. The '`\(.*\)`' matches the first half, which may be anything, but the

'`\1`' that follows must match the same exact text.

If a particular '`\( ... \)`' construct matches more than once (which can easily happen if it is followed by '`*`'), only the last match is recorded.

`` \` ``

matches the empty string, but only at the beginning of the buffer or string being matched against.

NOTE: This currently only matches the start of the current line − it does not match the start of the buffer.

`\'`

matches the empty string, but only at the end of the buffer or string being matched against.

NOTE: This currently only matches the end of the current line − it does not match the end of the buffer.

`\=`

matches the empty string, but only at point.

`\b`

matches the empty string, but only at the beginning or end of a word. Thus, '`\bfoo\b`' matches any occurrence of '`foo`' as a separate word. '`\bballs?\b`' matches '`ball`' or '`balls`' as a separate word.

'`\b`' matches at the beginning or end of the buffer regardless of what text appears next to it.

`\B` matches the empty string, but *not* at the beginning or end of a word.

`\<`

matches the empty string, but only at the beginning of a word. '`\<`' matches at the beginning of the buffer only if a word−constituent character follows.

`\>`

matches the empty string, but only at the end of a word. '`\>`' matches at the end of the buffer only if the contents end with a word−constituent character.

`\w`

matches any word−constituent character. The syntax table determines which characters these are.

`\W`

matches any character that is not a word−constituent.

`\sC`

matches any character whose syntax is `C`. Here `C` is a character that represents a syntax code: thus, '`w`' for word constituent, '`−`' for whitespace, '`(`' for open parenthesis, etc. Represent a character of whitespace (which can be a newline) by either '`−`' or a space character.

`\SC`

matches any character whose syntax is not `C`.

`\{N,M\}`

Matches an integer number of the previous item, where `N` and `M` are integer constants interpreted as follows:−

`\{N\}`

The preceeding item is matched exactly `N` times.

`\{N,\}`

The preceeding item is matched `N` or more times.

`\{N,M\}`

The preceeding item is matched at least `N` times, but no more than `M` times.

`\{,M\}`

The preceeding item is optional and is matched at most `M` times.

The constructs that pertain to words and syntax are controlled by the setting of the syntax table.

Syntax of Replacement Expressions

A regular expression replacement, query−replace−string(2) command (with magic(2m) mode enabled), replaces exact matches for a single string or pattern. The replacement pattern may be a constant; it may also refer to all or part of what is matched by the regular expression search string.

\&

In the replacement pattern, **\&** stands for the entire match being replaced. (as does `\0`).

\D

In the replacement pattern, where **D** is a digit 1−9, stands for whatever matched the Dth parenthesized grouping (`\( .. \)`) in search pattern. To include a '\' in the text to replace with, you must enter '`\\`'. For example,

```
M−x query−replace−string<RET> c[ad]+r <RET> \&−safe <RET>
```

replaces (for example) "`cadr`" with "`cadr−safe`" and "`cddr`" with "`cddr−safe`".

```
M−x query−replace−string<RET> \(c[ad]+r\)−safe <RET> \1 <RET>
```

performs the inverse transformation.

\0 is a special case, this represents the whole of the search pattern, it is equivalent to **\&**.

Searching and Case

Searching may be either case sensitive or case insensitive, and is controlled by the exact(2m) mode. When *exact* mode is enabled (default) the then searches are case sensitive; disabled then case is ignored. The exact(2m) mode is set on a per−buffer basis.

NOTES

The search engine searches for the longest string that matches a given pattern, the longest pattern is sometimes the pattern that is not actually required. For instance, consider searching for an HTML bracket set. The simplest search is:−

```
M−x search−forward "<.*>"
```

Unfortunately, this pattern is not specific enough, given an HTML line:−

```
<a href="www.jasspa.com">Jasspa Site</a>
```

Then the pattern matched is actually the whole line as the `.*` matches everything to the last `>`, this is the longest string. To rectify the pattern then we must be more specific, the correct search pattern to use in this instance is:−

```
M−x search−forward "<[^>]*>"
```

In this case we match any character excluding the closing character, this guarantees that we always find the shortest string match. A search of our HTML line locates two separate instances of the regular expression `<a href="www.jasspa.com">` and `</a>`.

SEE ALSO

search−forward(2), search−backward(2), buffer−mode(2), exact(2m), hunt−backward(2), hunt−forward(2), isearch−forward(2), magic(2m), replace−string(2).

Build(2)

BUILD

MicroEmacs '02 may be compiled from the source files using the command shell build scripts *build* (UNIX Bourne Shell) or *build.bat* (DOS/Windows). A default compile sequence may be achieved with a simple:

```
build
```

from the command line. The build script attempts to detect the host system and available compiler and build the editor.

The build script recognizes the following options:−

−C

Build clean. Delete all of the object files.

−d

Build a debugging version, the output is `med` (or `med32` for 32−bit Windows versions).

−h

Display a simple help page

−l *logfile*

Redirect all compilation output to the *logfile*, this may not work on DOS or Windows systems.

−la *logfile*

Append all compilation output to the end of *logfile*, this may not work on DOS or Windows systems.

−m *makefile*

Build using the specified makefile. over−riding the auto system detect. The supplied makefiles include:−

· `aix43.mak` IBM AIX 4.3 native
· `cygwin.gmk` Cygwin using GNU tools under Windows.
· `dosdj1.mak` Microsoft DOS build using djgpp version 1.
· `dosdj2.mak` Microsoft DOS build using djgpp version 2.
· `freebsd.gmk` Free BSD using GNU tools.
· `hpux9.gmk` HP−UX 9.x using GNU tools.
· `hpux9.mak` HP−UX 9.x native
· `hpux10.gmk` HP−UX 10.x using GNU tools.

· `hpux10.mak` HP−UX 10.x native
· `hpux11.gmk` HP−UX 11.x using GNU tools.
· `hpux11.mak` HP−UX 11.x native
· `irix5.gmk` Silicon Graphics IRIX 5.x using GNU tools
· `irix5.mak` Silicon Graphics IRIX 5.x native
· `irix6.gmk` Silicon Graphics IRIX 6.x using GNU tools
· `irix6.mak` Silicon Graphics IRIX 6.x native
· `linux2.gmk` Linux 2.x using GNU tools
· `openstep.mak` Openstep 4.2 on NeXTstep (BSD 4.3).
· `sunos55.gmak` Sun Solaris 5.5 using GNU tools
· `sunos55.mak` Sun Solaris 5.5 native
· `sunos56.gmak` Sun Solaris 5.6 using GNU tools
· `sunos56.mak` Sun Solaris 5.6 native
· `sunosx86.gmk` Sun Solarais 2.6 (Intel) using GNU tools.
· `win32bc.mak` Borland C, 32−bit Windows version.
· `win32b55.mak` Borland C 5.5, 32−bit Windows version (Free compiler).
· `win32sv2.mak` Microsoft Developer v2.x, Win32s (for Win 3.xx)
· `win32sv4.mak` Microsoft Developer v4.2, Win32s (for Win 3.xx)
· `win32v2.mak` Microsoft Developer v2.x, 32−bit Windows.
· `win32v5.mak` Microsoft Developer v5.x, 32−bit Windows.
· `win32v6.mak` Microsoft Developer v6.x, 32−bit Windows.

−ne

Build NanoEmacs (a cut down version aimed as a vi replacement), the output is `ne` (or `ned32` for 32−bit Windows versions).

−S

Build spotless. Deletes all of the object files and any backup files, tag files etc.

−t *type*

Set the build type, where *type* can be one of the following:

· `c` Build a console only version (i.e. no window support), the output is `mec` (or `mec32` on Windows).
· `w` Build a windows only version (i.e. no console support), the output is `mew` (or `mew32` on Windows).
· `cw` Build a version which supports both console and windows, the output is `mecw` (or `mecw32` on Windows).

−u

Build a URL version (Windows '95/'98/NT only), constructs the executable `meu32.exe`. **Makefiles**

The supplied makefiles are provided in two forms:−

- **.gmk** − GNU make, using gcc.
- **.mak** − Native make, consistent with the compiler and platform.

The makefiles are supplied with the following targets:−

- **all** − Default build.
- **clean** − Removes intermediate files.
- **spotless** − Removes intermediate files and any backup files.
- **med** − Builds a debugging version.
- **men** − Builds console version (Windows only).
- **men** − Builds a URL version (Windows only).
- **menu** − Builds console and URL version (Windows only).

NOTES

Other UNIX ports should be fairly easy from the base set of ported platforms. If any new platform ports are performed by individuals then please submit the makefiles and any source changes back to JASSPA − see Contact Information.

Command Glossary

COMMAND GLOSSARY

The following is a list of all of the commands (built−in and macro) provided by **MicroEmacs '02** [See split listing]:

abort−command(2) (**C−g**) Abort command
about(2) Information About MicroEmacs
add−color(2) Create a new color
add−color−scheme(2) Create a new color scheme
add−dictionary(2) Declare existence of a spelling dictionary
add−file−hook(2) Declare file name context dependent configuration
add−global−mode(3) Set a global buffer mode
add−mode(3) Set a local buffer mode
add−next−line(2) Define the searching behavior of command output
add−spell−rule(2) Add a new spelling rule to the dictionary
alarm(3) Set an alarm
aman(3) Compile an nroff file into a buffer (UNIX)
append−buffer(2) Write contents of buffer to end of named file
ascii−time(3) Return the current time as a string
auto−spell(3) Auto−spell support
auto−spell−buffer(3) Auto−spell whole buffer
auto−spell−ignore(3) Auto−spell ignore current word
auto−spell−reset(3) Auto−spell hilight reset
backward−char(2) (**C−b**) Move the cursor left
backward−delete−char(2) (**backspace**) Delete the previous character at the cursor position
backward−delete−tab(2) (**S−tab**) Delete white space to previous tab−stop
backward−kill−word(2) (**esc backspace**) Delete the previous word at the cursor position
backward−line(2) (**C−p**) Move the cursor to the previous line
backward−paragraph(2) (**esc p**) Move the cursor to the previous paragraph
backward−word(2) (**esc b**) Move the cursor to the previous word
beginning−of−buffer(2) (**esc <**) Move to beginning of buffer/file
beginning−of−line(2) (**C−a**) Move to beginning of line
buffer−abbrev−file(2) Set buffers' abbreviation file
buffer−bind−key(2) Create local key binding for current buffer
buffer−help(3) Displays help page for current buffer
buffer−info(2) (**C−x =**) Status information on current buffer position
buffer−mode(2) (**C−x m**) Change a local buffer mode
buffer−setup(3) Configures the current buffer settings
buffer−unbind−key(2) Remove local key binding for current buffer
c−hash−del(3) Remove C/C++ #define evaluation
c−hash−eval(3) Evaluate C/C++ #defines
c−hash−set−define(3) Set a C/C++ #define
c−hash−unset−define(3) Unset a C/C++ #define
calc(3) Integer calculator
capitalize−word(2) (**esc c**) Capitalize word

change−buffer−name(2) (**esc C−n**) Change name of current buffer
change−directory(2) [**C−x C−d**] Change the current working directory
change−file−name(2) (**C−x n**) Change the file name of the current buffer
change−font(2) Change the screen font
change−frame−depth(2) Change the number of lines on the current frame
change−frame−width(2) Change the number of columns on the current frame
change−screen−depth(2) Change the number of lines on the screen
change−screen−width(2) Change the number of columns on the screen
change−window−depth(2) Change the depth of the current window
change−window−width(2) Change the width of the current window
charset−change(3) Convert buffer between two character sets
charset−iso−to−user(3) Convert buffer from ISO standard to user character set
charset−user−to−iso(3) Convert buffer from user to ISO standard character set
check−line−length(3) Check the length of text lines are valid
clean(3) Remove redundant white spaces from the current buffer
command−apropos(2) (**C−h a**) List commands involving a concept
command−wait(2) Conditional wait command
compare−windows(2) Compare buffer windows, ignore whitespace
compare−windows−exact(3) Compare buffer windows, with whitespace
compile(3) Start a compilation process
copy−region(2) (**esc w**) Copy a region of the buffer
count−words(2) (**esc C−c**) Count the number of words in a region
create−callback(2) Create a timer callback
create−frame(2) Create a new frame
cvs(3) MicroEmacs CVS interface
cvs−add(3) MicroEmacs CVS interface − add file
cvs−checkout(3) MicroEmacs CVS interface − checkout files
cvs−commit(3) MicroEmacs CVS interface − commit changes
cvs−diff(3) MicroEmacs CVS interface − diff changes
cvs−gdiff(3) MicroEmacs CVS interface − graphical diff changes
cvs−log(3) MicroEmacs CVS interface − log changes
cvs−remove(3) MicroEmacs CVS interface − remove file
cvs−resolve−conflicts(3) MicroEmacs CVS interface − resolve conflicts
cvs−state(3) MicroEmacs CVS interface − list state of directory files
cvs−update(3) MicroEmacs CVS interface − update directory files
cygnus(3) Open a Cygwin BASH window
define−help(2) Define help information
define−macro(2) Define a new macro
define−macro−file(2) Define macro file location
delete−blank−lines(2) (**C−x C−o**) Delete blank lines about cursor
delete−buffer(2) (**C−x k**) Delete a buffer
delete−dictionary(2) Remove a spelling dictionary from memory
delete−frame(2) Delete the current frame
delete−global−mode(3) Remove a global buffer mode
delete−indentation(3) Join 2 lines deleting white spaces
delete−mode(3) Remove a local buffer mode
delete−other−windows(2) (**C−x 1**) Delete other windows
delete−registry(2) Delete a registry tree
delete−some−buffers(2) Delete buffers with query

delete−window(2) (**C−x 0**) Delete current window
describe−bindings(2) (**C−h b**) Show current command/key binding
describe−key(2) (**C−x ?**) Report keyboard key name and binding
describe−variable(2) (**C−h v**) Describe current setting of a variable
describe−word(3) Display a dictionary definition of a word
diff(3) Difference files or directories
diff−changes(3) Find the differences from a previous edit session
directory−tree(2) Draw the file directory tree
display−white−chars(3) Toggle the displaying of white characters
draw(3) Simple line drawing utility
edit−dictionary(3) Insert a dictionary in a buffer
end−kbd−macro(2) (**C−x)**) Stop recording keyboard macro
end−of−buffer(2) (**esc** >) Move to end of buffer/file
end−of−line(2) (**C−e**) Move to end of line
etfinsrt(3) Insert template file into current buffer
exchange−point−and−mark(2) (**C−x C−x**) Exchange the cursor and marked position
execute−buffer(2) Execute script lines from a buffer
execute−file(2) (**esc /**) Execute script lines from a file
execute−kbd−macro(2) (**C−x e**) Execute a keyboard macro
execute−line(2) Execute a typed in script line
execute−named−command(2) [**esc x**] Execute a named command
execute−string(2) Execute a string as a command
execute−tool(3) Execute a user defined shell tool
exit−emacs(2) Exit MicroEmacs
expand−abbrev(2) Expand an abbreviation
expand−abbrev−handle(3) (**esc esc**) Expand an abbreviation handler
expand−look−back(3) Complete a word by looking back for a similar word
expand−word(3) Complete a word by invocation of the speller
file−attrib(3) Set the current buffers system file attributes
file−browser(3) (**f10**) Browse the file system
file−browser−close(3) Close the file−browser
file−browser−swap−buffers(3) Swap between file−browser windows
file−op(2) File system operations command
fill−paragraph(2) (**esc o**) Format a paragraph
filter−buffer(2) (**C−x #**) Filter the current buffer through an O/S command
find−bfile(3) (**C−x 9**) Load a file as binary data
find−buffer(2) (**C−x b**) Switch to a named buffer
find−cfile(3) Load a crypted file
find−file(2) (**C−x C−f**) Load a file
find−registry(2) Index search of a registry sub−tree
find−tag(2) (**esc t**) Find tag, auto−load file and move to tag position
find−word(3) Find a using spelling dictionaries
find−zfile(3) Compressed file support
fold−all(3) (**f3**) (Un)Fold all regions in the current buffer
fold−current(3) (**f2**) (un)Fold a region in the current buffer
forward−char(2) (**C−f**) Move the cursor right
forward−delete−char(2) (**C−d**) Delete the next character at the cursor position
forward−kill−word(2) (**esc d**) Delete the next word at the cursor position
forward−line(2) (**C−n**) Move the cursor to the next line

forward−paragraph(2) (**esc n**) Move the cursor to the next paragraph
forward−word(2) (**esc f**) Move the cursor to the next word
ftp(3) Initiate an FTP connection
gdiff(3) Graphical file difference
generate−tags−file(3) Generate a tags file
get−next−line(2) (**C−x `**) Find the next command line
get−registry(2) Retrieve a node value from the registry
global−abbrev−file(2) Set global abbreviation file
global−bind−key(2) (**esc k**) Bind a key to a named command or macro
global−mode(2) (**esc m**) Change a global buffer mode
global−unbind−key(2) (**esc C−k**) Unbind a key from a named command or macro
goto−alpha−mark(2) (**C−x a**) Move the cursor to a alpha marked location
goto−line(2) (**esc g**) Move the cursor to specified line
goto−matching−fence(2) (**esc C−f**) Move the cursor to matching fence
goto−position(2) Restore a stored position
goto−window(2) Restore a saved window to the current window (historic)
grep(3) Execute grep command
grow−window−horizontally(2) Enlarge current window horizontally (relative)
grow−window−vertically(2) Enlarge the current window (relative change)
help(2) (**esc ?**) Help; high level introduction to help
help−command(2) (**C−h C−c**) Help; command information
help−item(2) (**C−h C−i**) Help; item information
help−variable(2) (**C−h C−v**) Help; variable information
hilight(2) Manage the buffer hilighting schemes
hunt−backward(2) (**C−x C−h**) Resume previous search in backward direction
hunt−forward(2) (**C−x h**) Resume previous search in forward direction
ifill−paragraph(3) (**esc q**) Format a paragraph
indent(2) Manage the auto−indentation methods
info(3) Display a GNU Info database
info−goto−link(3) Display Info on a given link
info−on(3) Display Info on a given topic
insert−file(2) (**C−x C−i**) Insert file into current buffer
insert−file−name(2) (**C−x C−y**) Insert filename into current buffer
insert−macro(2) Insert keyboard macro into buffer
insert−newline(2) (**C−o**) Insert new line at cursor position
insert−space(2) Insert space(s) into current buffer
insert−string(2) Insert character string into current buffer
insert−tab(2) (**C−i**) Insert tab(s) into current buffer
ipipe−kill(2) Kill a incremental pipe
ipipe−shell−command(2) (**esc backslash**) Incremental pipe (non−suspending system call)
ipipe−write(2) Write a string to an incremental pipe
isearch−backward(2) (**C−r**) Search backwards incrementally (interactive)
isearch−forward(2) (**C−s**) Search forward incrementally (interactive)
ishell(3) Open a Cygwin BASH window
kbd−macro−query(2) (**C−x q**) Query termination of keyboard macro
kill−line(2) (**C−k**) Delete all characters to the end of the line
kill−paragraph(2) Delete a paragraph
kill−rectangle(2) (**esc C−w**) Delete a column of text
kill−region(2) (**C−w**) Delete all characters in the marked region

line−scheme−search(3) Search and annotate the current buffer
list−buffers(2) (**C−x C−b**) List all buffers and show their status
list−commands(2) (**C−h c**) List available commands
list−registry(2) Display the registry in a buffer
list−variables(2) (**C−h v**) List defined variables
lower−case−region(2) (**C−x C−l**) Lowercase a region (downcase)
lower−case−word(2) (**esc l**) Lowercase word (downcase)
Mahjongg(3) MicroEmacs '02 version of the solitaire Mah Jongg game
MainMenu(3) The top main menu
Match−It(3) MicroEmacs '02 version of the Match−It game
Metris(3) MicroEmacs '02 version of the falling blocks game
mail(3) Compose and send an email
mail−check(3) Check for new email
man(3) UNIX manual page viewer
man−clean(3) Clean UNIX manual page
mark−registry(2) Modify the operating mode of a registry node
ml−bind−key(2) Create key binding for message line
ml−clear(2) Clear the message line
ml−unbind−key(2) Remove key binding from message line
ml−write(2) Write message on message line
name−kbd−macro(2) Assign a name to the last keyboard macro
named−buffer−mode(2) Change a named buffer mode
narrow−buffer(2) Hide buffer lines
newline(2) (**return**) Insert a new line
next−buffer(2) (**C−x x**) Switch to the next buffer
next−frame(2) Change the focus to the next frame
next−window(2) (**C−x o**) Move the cursor to the next window
next−window−find−buffer(2) [] Split the current window and show new buffer
next−window−find−file(2) (**C−x 4**) Split the current window and find file
normal−tab(3) Insert a normal tab
organizer(3) Calendar and address organizer
osd(2) Manage the On−Screen Display
osd−bind−key(2) Create key binding for OSD dialog
osd−dialog(3) OSD dialog box
osd−entry(3) OSD entry dialog box
osd−help(3) GUI based on−line help
osd−unbind−key(2) Remove key binding from OSD dialog
osd−xdialog(3) OSD Extended dialog box
Patience(3) MicroEmacs '02 version of Patience (or Solitaire)
paragraph−to−line(3) Convert a paragraph to a single line
pipe−shell−command(2) (**esc @**) Execute a single operating system command
popup−window(2) Pop−up a window on the screen
prefix(2) Key prefix command
previous−window(2) (**C−x p**) Move the cursor to the previous window
print−buffer(2) Print buffer, with formatting
print−color(2) Create a new printer color
print−region(2) Print region, with formatting
print−scheme(2) Create a new printer color and font scheme
print−setup(3) Configure (*mS's printer interface

query−replace−all−string(3) Query replace string in a list of files
query−replace−string(2) (**esc C−r**) Search and replace a string − with query
quick−exit(2) (**esc z**) Exit the editor writing changes
quote−char(2) (**C−q**) Insert literal character
rcs−file(2) (**C−x C−q**) Handle Revision Control System (RCS) files
read−file(2) (**C−x C−r**) Find and load file replacing current buffer
read−history(2) Read in session history information
read−registry(2) Read in a registry definition file
recenter(2) (**C−l**) Recenter the window (refresh the screen)
regex−backward(3) Search for a magic string in the backward direction
regex−forward(3) Search for a magic string in the forward direction
replace−all−pairs(3) Replace string pairs in a list of files
replace−all−string(3) Replace string with new string in a list of files
replace−string(2) (**esc r**) Replace string with new string
reread−file(3) Reload the current buffer's file
resize−all−windows(2) Resize all windows (automatic change)
resize−window−horizontally(2) Resize current window horizontally (absolute)
resize−window−vertically(2) Resize the current window (absolute change)
restore−dictionary(3) Save dictionary user changes
restyle−buffer(3) Automatically reformat a buffer's indentation
restyle−region(3) Automatically reformat a regions indentation
reyank(2) (**esc y**) Restore next yank buffer
rgrep(3) Execute recursive grep command
save−all(3) Save all modified files (with query)
save−buffer(2) (**C−x C−s**) Save contents of changed buffer to file
save−buffers−exit−emacs(2) (**esc z**) Exit the editor prompt user to write changes
save−dictionary(2) Save changed spelling dictionaries
save−history(2) Write history information to history file
save−registry(2) Write a registry definition file
save−some−buffers(2) Save contents of all changed buffers to file (with query)
scheme−editor(3) Color Scheme Editor
screen−poke(2) Immediate write string to the screen
screen−update(2) (**redraw**) Force screen update
scroll−down(2) (**C−n**) Move the window down (scrolling)
scroll−left(2) (**C−x <**) Move the window left (scrolling)
scroll−next−window−down(2) (**esc C−v**) Scroll next window down
scroll−next−window−up(2) (**esc C−z**) Scroll next window up
scroll−right(2) (**C−x >**) Move the window right (scrolling)
scroll−up(2) (**C−p**) Move the window up (scrolling)
search−backward(2) (**C−x r**) Search for a string in the backward direction
search−forward(2) (**C−x s**) Search for a string in the forward direction
set−alpha−mark(2) (**C−x C−a**) Place an alphabetic marker in the buffer
set−char−mask(2) Set character word mask
set−cursor−to−mouse(2) Move the cursor to the current mouse position
set−encryption−key(2) (**esc e**) Define the encryption key
set−mark(2) (**esc space**) Set starting point of region
set−position(2) Store the current position
set−registry(2) Modify a node value in the registry
set−scroll−with−mouse(2) Scroll the window with the mouse

set−variable(2) (**C−x v**) Assign a new value to a variable
set−window(2) Save the current window for restore (historic)
shell(2) [**C−x c**] Create a new command processor or shell
shell−command(2) Perform an operating system command
show−cursor(2) Change the visibility of the cursor
show−region(2) Show the current copy region
shrink−window−horizontally(2) Shrink current window horizontally (relative)
shrink−window−vertically(2) Shrink the current window (relative change)
shut−down(3) Editor exit callback command
sort−lines(2) Alphabetically sort lines
sort−lines−ignore−case(3) Alphabetically sort lines ignoring case
spell(2) Spell checker service provider
spell−add−word(3) Add a word to the main dictionary
spell−buffer(3) Spell check the current buffer
spell−edit−word(3) Edits a spell word entry
spell−word(3) (**esc $**) Spell check a single word
split−window−horizontally(2) (**C−x 5**) Split current window into two (horizontally)
split−window−vertically(2) (**C−x 2**) Split the current window into two
start−kbd−macro(2) (**C−x (**) Start recording keyboard macro
start−up(3) Editor startup callback command
stop−mail−check(3) Disable the check for new email
suspend−emacs(2) Suspend editor and place in background
symbol(3) Insert an ASCII character
Triangle(3) MicroEmacs '02 version of Triangle patience game
tab(2) (**tab**) Handle the tab key
tabs−to−spaces(3) Converts all tabs to spaces
tex2nr(3) Convert a Latex file into nroff
time(3) Command time evaluator
translate−key(2) Translate key
transpose−chars(2) (**C−t**) Exchange (swap) adjacent characters
transpose−lines(2) (**C−x C−t**) Exchange (swap) adjacent lines
undo(2) (**C−x u**) Undo the last edit
uniq(3) Make lines in a sorted list unique
universal−argument(2) (**C−u**) Set the command argument count
unmark−buffer(3) Remove buffer edited flag
unset−variable(2) Delete a variable
upper−case−region(2) (**C−x C−u**) Uppercase a region (upcase)
upper−case−word(2) (**esc u**) Uppercase word (upcase)
user−setup(3) Configure MicroEmacs for a specific user
view−file(2) (**C−x C−v**) Load a file read only
vm(3) Email viewer
void(2) Null command
which(3) Program finder
wrap−word(2) Wrap word onto next line
write−buffer(2) (**C−x C−w**) Write contents of buffer to named (new) file
yank(2) (**C−y**) Paste (copy) kill buffer contents into buffer
yank−rectangle(2) (**esc C−y**) Insert a column of text
zfile−setup(3) Compressed file support setup

Split Command Glossary

SPLIT COMMAND GLOSSARY

The following is a list of all of the built in commands provided by **MicroEmacs '02** [See mixed listing]:

abort−command(2) (**C−g**) Abort command
about(2) Information About MicroEmacs
add−color(2) Create a new color
add−color−scheme(2) Create a new color scheme
add−dictionary(2) Declare existence of a spelling dictionary
add−file−hook(2) Declare file name context dependent configuration
add−next−line(2) Define the searching behavior of command output
add−spell−rule(2) Add a new spelling rule to the dictionary
append−buffer(2) Write contents of buffer to end of named file
backward−char(2) (**C−b**) Move the cursor left
backward−delete−char(2) (**backspace**) Delete the previous character at the cursor position
backward−delete−tab(2) (**S−tab**) Delete white space to previous tab−stop
backward−kill−word(2) (**esc backspace**) Delete the previous word at the cursor position
backward−line(2) (**C−p**) Move the cursor to the previous line
backward−paragraph(2) (**esc p**) Move the cursor to the previous paragraph
backward−word(2) (**esc b**) Move the cursor to the previous word
beginning−of−buffer(2) (**esc <**) Move to beginning of buffer/file
beginning−of−line(2) (**C−a**) Move to beginning of line
buffer−abbrev−file(2) Set buffers' abbreviation file
buffer−bind−key(2) Create local key binding for current buffer
buffer−info(2) (**C−x =**) Status information on current buffer position
buffer−mode(2) (**C−x m**) Change a local buffer mode
buffer−unbind−key(2) Remove local key binding for current buffer
capitalize−word(2) (**esc c**) Capitalize word
change−buffer−name(2) (**esc C−n**) Change name of current buffer
change−directory(2) [**C−x C−d**] Change the current working directory
change−file−name(2) (**C−x n**) Change the file name of the current buffer
change−font(2) Change the screen font
change−frame−depth(2) Change the number of lines on the current frame
change−frame−width(2) Change the number of columns on the current frame
change−screen−depth(2) Change the number of lines on the screen
change−screen−width(2) Change the number of columns on the screen
change−window−depth(2) Change the depth of the current window
change−window−width(2) Change the width of the current window
command−apropos(2) (**C−h a**) List commands involving a concept
command−wait(2) Conditional wait command
compare−windows(2) Compare buffer windows, ignore whitespace
copy−region(2) (**esc w**) Copy a region of the buffer
count−words(2) (**esc C−c**) Count the number of words in a region
create−callback(2) Create a timer callback

create−frame(2) Create a new frame
define−help(2) Define help information
define−macro(2) Define a new macro
define−macro−file(2) Define macro file location
delete−blank−lines(2) (**C−x C−o**) Delete blank lines about cursor
delete−buffer(2) (**C−x k**) Delete a buffer
delete−dictionary(2) Remove a spelling dictionary from memory
delete−frame(2) Delete the current frame
delete−other−windows(2) (**C−x 1**) Delete other windows
delete−registry(2) Delete a registry tree
delete−some−buffers(2) Delete buffers with query
delete−window(2) (**C−x 0**) Delete current window
describe−bindings(2) (**C−h b**) Show current command/key binding
describe−key(2) (**C−x ?**) Report keyboard key name and binding
describe−variable(2) (**C−h v**) Describe current setting of a variable
directory−tree(2) Draw the file directory tree
end−kbd−macro(2) (**C−x)**) Stop recording keyboard macro
end−of−buffer(2) (**esc** >) Move to end of buffer/file
end−of−line(2) (**C−e**) Move to end of line
exchange−point−and−mark(2) (**C−x C−x**) Exchange the cursor and marked position
execute−buffer(2) Execute script lines from a buffer
execute−file(2) (**esc** /) Execute script lines from a file
execute−kbd−macro(2) (**C−x e**) Execute a keyboard macro
execute−line(2) Execute a typed in script line
execute−named−command(2) [**esc x**] Execute a named command
execute−string(2) Execute a string as a command
exit−emacs(2) Exit MicroEmacs
expand−abbrev(2) Expand an abbreviation
file−op(2) File system operations command
fill−paragraph(2) (**esc o**) Format a paragraph
filter−buffer(2) (**C−x #**) Filter the current buffer through an O/S command
find−buffer(2) (**C−x b**) Switch to a named buffer
find−file(2) (**C−x C−f**) Load a file
find−registry(2) Index search of a registry sub−tree
find−tag(2) (**esc t**) Find tag, auto−load file and move to tag position
forward−char(2) (**C−f**) Move the cursor right
forward−delete−char(2) (**C−d**) Delete the next character at the cursor position
forward−kill−word(2) (**esc d**) Delete the next word at the cursor position
forward−line(2) (**C−n**) Move the cursor to the next line
forward−paragraph(2) (**esc n**) Move the cursor to the next paragraph
forward−word(2) (**esc f**) Move the cursor to the next word
get−next−line(2) (**C−x `**) Find the next command line
get−registry(2) Retrieve a node value from the registry
global−abbrev−file(2) Set global abbreviation file
global−bind−key(2) (**esc k**) Bind a key to a named command or macro
global−mode(2) (**esc m**) Change a global buffer mode
global−unbind−key(2) (**esc C−k**) Unbind a key from a named command or macro
goto−alpha−mark(2) (**C−x a**) Move the cursor to a alpha marked location
goto−line(2) (**esc g**) Move the cursor to specified line

goto−matching−fence(2) (**esc C−f**) Move the cursor to matching fence
goto−position(2) Restore a stored position
goto−window(2) Restore a saved window to the current window (historic)
grow−window−horizontally(2) Enlarge current window horizontally (relative)
grow−window−vertically(2) Enlarge the current window (relative change)
help(2) (**esc ?**) Help; high level introduction to help
help−command(2) (**C−h C−c**) Help; command information
help−item(2) (**C−h C−i**) Help; item information
help−variable(2) (**C−h C−v**) Help; variable information
hilight(2) Manage the buffer hilighting schemes
hunt−backward(2) (**C−x C−h**) Resume previous search in backward direction
hunt−forward(2) (**C−x h**) Resume previous search in forward direction
indent(2) Manage the auto−indentation methods
insert−file(2) (**C−x C−i**) Insert file into current buffer
insert−file−name(2) (**C−x C−y**) Insert filename into current buffer
insert−macro(2) Insert keyboard macro into buffer
insert−newline(2) (**C−o**) Insert new line at cursor position
insert−space(2) Insert space(s) into current buffer
insert−string(2) Insert character string into current buffer
insert−tab(2) (**C−i**) Insert tab(s) into current buffer
ipipe−kill(2) Kill a incremental pipe
ipipe−shell−command(2) (**esc backslash**) Incremental pipe (non−suspending system call)
ipipe−write(2) Write a string to an incremental pipe
isearch−backward(2) (**C−r**) Search backwards incrementally (interactive)
isearch−forward(2) (**C−s**) Search forward incrementally (interactive)
kbd−macro−query(2) (**C−x q**) Query termination of keyboard macro
kill−line(2) (**C−k**) Delete all characters to the end of the line
kill−paragraph(2) Delete a paragraph
kill−rectangle(2) (**esc C−w**) Delete a column of text
kill−region(2) (**C−w**) Delete all characters in the marked region
list−buffers(2) (**C−x C−b**) List all buffers and show their status
list−commands(2) (**C−h c**) List available commands
list−registry(2) Display the registry in a buffer
list−variables(2) (**C−h v**) List defined variables
lower−case−region(2) (**C−x C−l**) Lowercase a region (downcase)
lower−case−word(2) (**esc l**) Lowercase word (downcase)
mark−registry(2) Modify the operating mode of a registry node
ml−bind−key(2) Create key binding for message line
ml−clear(2) Clear the message line
ml−unbind−key(2) Remove key binding from message line
ml−write(2) Write message on message line
name−kbd−macro(2) Assign a name to the last keyboard macro
named−buffer−mode(2) Change a named buffer mode
narrow−buffer(2) Hide buffer lines
newline(2) (**return**) Insert a new line
next−buffer(2) (**C−x x**) Switch to the next buffer
next−frame(2) Change the focus to the next frame
next−window(2) (**C−x o**) Move the cursor to the next window
next−window−find−buffer(2) [] Split the current window and show new buffer

next−window−find−file(2) (**C−x 4**) Split the current window and find file
osd(2) Manage the On−Screen Display
osd−bind−key(2) Create key binding for OSD dialog
osd−unbind−key(2) Remove key binding from OSD dialog
pipe−shell−command(2) (**esc @**) Execute a single operating system command
popup−window(2) Pop−up a window on the screen
prefix(2) Key prefix command
previous−window(2) (**C−x p**) Move the cursor to the previous window
print−buffer(2) Print buffer, with formatting
print−color(2) Create a new printer color
print−region(2) Print region, with formatting
print−scheme(2) Create a new printer color and font scheme
query−replace−string(2) (**esc C−r**) Search and replace a string − with query
quick−exit(2) (**esc z**) Exit the editor writing changes
quote−char(2) (**C−q**) Insert literal character
rcs−file(2) (**C−x C−q**) Handle Revision Control System (RCS) files
read−file(2) (**C−x C−r**) Find and load file replacing current buffer
read−history(2) Read in session history information
read−registry(2) Read in a registry definition file
recenter(2) (**C−l**) Recenter the window (refresh the screen)
replace−string(2) (**esc r**) Replace string with new string
resize−all−windows(2) Resize all windows (automatic change)
resize−window−horizontally(2) Resize current window horizontally (absolute)
resize−window−vertically(2) Resize the current window (absolute change)
reyank(2) (**esc y**) Restore next yank buffer
save−buffer(2) (**C−x C−s**) Save contents of changed buffer to file
save−buffers−exit−emacs(2) (**esc z**) Exit the editor prompt user to write changes
save−dictionary(2) Save changed spelling dictionaries
save−history(2) Write history information to history file
save−registry(2) Write a registry definition file
save−some−buffers(2) Save contents of all changed buffers to file (with query)
screen−poke(2) Immediate write string to the screen
screen−update(2) (**redraw**) Force screen update
scroll−down(2) (**C−n**) Move the window down (scrolling)
scroll−left(2) (**C−x <**) Move the window left (scrolling)
scroll−next−window−down(2) (**esc C−v**) Scroll next window down
scroll−next−window−up(2) (**esc C−z**) Scroll next window up
scroll−right(2) (**C−x >**) Move the window right (scrolling)
scroll−up(2) (**C−p**) Move the window up (scrolling)
search−backward(2) (**C−x r**) Search for a string in the backward direction
search−forward(2) (**C−x s**) Search for a string in the forward direction
set−alpha−mark(2) (**C−x C−a**) Place an alphabetic marker in the buffer
set−char−mask(2) Set character word mask
set−cursor−to−mouse(2) Move the cursor to the current mouse position
set−encryption−key(2) (**esc e**) Define the encryption key
set−mark(2) (**esc space**) Set starting point of region
set−position(2) Store the current position
set−registry(2) Modify a node value in the registry
set−scroll−with−mouse(2) Scroll the window with the mouse

set−variable(2) (**C−x v**) Assign a new value to a variable
set−window(2) Save the current window for restore (historic)
shell(2) [**C−x c**] Create a new command processor or shell
shell−command(2) Perform an operating system command
show−cursor(2) Change the visibility of the cursor
show−region(2) Show the current copy region
shrink−window−horizontally(2) Shrink current window horizontally (relative)
shrink−window−vertically(2) Shrink the current window (relative change)
sort−lines(2) Alphabetically sort lines
spell(2) Spell checker service provider
split−window−horizontally(2) (**C−x 5**) Split current window into two (horizontally)
split−window−vertically(2) (**C−x 2**) Split the current window into two
start−kbd−macro(2) (**C−x (**) Start recording keyboard macro
suspend−emacs(2) Suspend editor and place in background
tab(2) (**tab**) Handle the tab key
translate−key(2) Translate key
transpose−chars(2) (**C−t**) Exchange (swap) adjacent characters
transpose−lines(2) (**C−x C−t**) Exchange (swap) adjacent lines
undo(2) (**C−x u**) Undo the last edit
universal−argument(2) (**C−u**) Set the command argument count
unset−variable(2) Delete a variable
upper−case−region(2) (**C−x C−u**) Uppercase a region (upcase)
upper−case−word(2) (**esc u**) Uppercase word (upcase)
view−file(2) (**C−x C−v**) Load a file read only
void(2) Null command
wrap−word(2) Wrap word onto next line
write−buffer(2) (**C−x C−w**) Write contents of buffer to named (new) file
yank(2) (**C−y**) Paste (copy) kill buffer contents into buffer
yank−rectangle(2) (**esc C−y**) Insert a column of text

The following is a list of documented macro commands provided by **MicroEmacs '02**:

add−global−mode(3) Set a global buffer mode
add−mode(3) Set a local buffer mode
alarm(3) Set an alarm
aman(3) Compile an nroff file into a buffer (UNIX)
ascii−time(3) Return the current time as a string
auto−spell(3) Auto−spell support
auto−spell−buffer(3) Auto−spell whole buffer
auto−spell−ignore(3) Auto−spell ignore current word
auto−spell−reset(3) Auto−spell hilight reset
buffer−help(3) Displays help page for current buffer
buffer−setup(3) Configures the current buffer settings
c−hash−del(3) Remove C/C++ #define evaluation
c−hash−eval(3) Evaluate C/C++ #defines
c−hash−set−define(3) Set a C/C++ #define
c−hash−unset−define(3) Unset a C/C++ #define
calc(3) Integer calculator
charset−change(3) Convert buffer between two character sets

charset−iso−to−user(3) Convert buffer from ISO standard to user character set
charset−user−to−iso(3) Convert buffer from user to ISO standard character set
check−line−length(3) Check the length of text lines are valid
clean(3) Remove redundant white spaces from the current buffer
compare−windows−exact(3) Compare buffer windows, with whitespace
compile(3) Start a compilation process
cvs(3) MicroEmacs CVS interface
cvs−add(3) MicroEmacs CVS interface − add file
cvs−checkout(3) MicroEmacs CVS interface − checkout files
cvs−commit(3) MicroEmacs CVS interface − commit changes
cvs−diff(3) MicroEmacs CVS interface − diff changes
cvs−gdiff(3) MicroEmacs CVS interface − graphical diff changes
cvs−log(3) MicroEmacs CVS interface − log changes
cvs−remove(3) MicroEmacs CVS interface − remove file
cvs−resolve−conflicts(3) MicroEmacs CVS interface − resolve conflicts
cvs−state(3) MicroEmacs CVS interface − list state of directory files
cvs−update(3) MicroEmacs CVS interface − update directory files
cygnus(3) Open a Cygwin BASH window
delete−global−mode(3) Remove a global buffer mode
delete−indentation(3) Join 2 lines deleting white spaces
delete−mode(3) Remove a local buffer mode
describe−word(3) Display a dictionary definition of a word
diff(3) Difference files or directories
diff−changes(3) Find the differences from a previous edit session
display−white−chars(3) Toggle the displaying of white characters
draw(3) Simple line drawing utility
edit−dictionary(3) Insert a dictionary in a buffer
etfinsrt(3) Insert template file into current buffer
execute−tool(3) Execute a user defined shell tool
expand−abbrev−handle(3) (**esc esc**) Expand an abbreviation handler
expand−look−back(3) Complete a word by looking back for a similar word
expand−word(3) Complete a word by invocation of the speller
file−attrib(3) Set the current buffers system file attributes
file−browser(3) (**f10**) Browse the file system
file−browser−close(3) Close the file−browser
file−browser−swap−buffers(3) Swap between file−browser windows
find−bfile(3) (**C−x 9**) Load a file as binary data
find−cfile(3) Load a crypted file
find−word(3) Find a using spelling dictionaries
find−zfile(3) Compressed file support
fold−all(3) (**f3**) (Un)Fold all regions in the current buffer
fold−current(3) (**f2**) (un)Fold a region in the current buffer
ftp(3) Initiate an FTP connection
gdiff(3) Graphical file difference
generate−tags−file(3) Generate a tags file
grep(3) Execute grep command
ifill−paragraph(3) (**esc q**) Format a paragraph
info(3) Display a GNU Info database
info−goto−link(3) Display Info on a given link

info−on(3) Display Info on a given topic
ishell(3) Open a Cygwin BASH window
line−scheme−search(3) Search and annotate the current buffer
Mahjongg(3) MicroEmacs '02 version of the solitaire Mah Jongg game
MainMenu(3) The top main menu
Match−It(3) MicroEmacs '02 version of the Match−It game
Metris(3) MicroEmacs '02 version of the falling blocks game
mail(3) Compose and send an email
mail−check(3) Check for new email
man(3) UNIX manual page viewer
man−clean(3) Clean UNIX manual page
normal−tab(3) Insert a normal tab
organizer(3) Calendar and address organizer
osd−dialog(3) OSD dialog box
osd−entry(3) OSD entry dialog box
osd−help(3) GUI based on−line help
osd−xdialog(3) OSD Extended dialog box
Patience(3) MicroEmacs '02 version of Patience (or Solitaire)
paragraph−to−line(3) Convert a paragraph to a single line
print−setup(3) Configure (*mS's printer interface
query−replace−all−string(3) Query replace string in a list of files
regex−backward(3) Search for a magic string in the backward direction
regex−forward(3) Search for a magic string in the forward direction
replace−all−pairs(3) Replace string pairs in a list of files
replace−all−string(3) Replace string with new string in a list of files
reread−file(3) Reload the current buffer's file
restore−dictionary(3) Save dictionary user changes
restyle−buffer(3) Automatically reformat a buffer's indentation
restyle−region(3) Automatically reformat a regions indentation
rgrep(3) Execute recursive grep command
save−all(3) Save all modified files (with query)
scheme−editor(3) Color Scheme Editor
shut−down(3) Editor exit callback command
sort−lines−ignore−case(3) Alphabetically sort lines ignoring case
spell−add−word(3) Add a word to the main dictionary
spell−buffer(3) Spell check the current buffer
spell−edit−word(3) Edits a spell word entry
spell−word(3) (**esc $**) Spell check a single word
start−up(3) Editor startup callback command
stop−mail−check(3) Disable the check for new email
symbol(3) Insert an ASCII character
Triangle(3) MicroEmacs '02 version of Triangle patience game
tabs−to−spaces(3) Converts all tabs to spaces
tex2nr(3) Convert a Latex file into nroff
time(3) Command time evaluator
uniq(3) Make lines in a sorted list unique
unmark−buffer(3) Remove buffer edited flag
user−setup(3) Configure MicroEmacs for a specific user
vm(3) Email viewer

which(3) Program finder
zfile−setup(3) Compressed file support setup

abort−command(2)

NAME

abort−command − Abort command

SYNOPSIS

abort−command (**C−g**)

DESCRIPTION

Aborts the current command, when in trouble, this command will usually limit the damage. If you find yourself in a position where you do not want to be then this command will usually take you back to a sane state. This command rings the bell and stops keyboard macros.

Avoid re−binding this key where possible as it is used in other places.

When **abort−command** is invoked a warning is automatically given alerting the user, this may be an audible or a visual warning depending on the global state of the quiet(2m) mode.

SEE ALSO

buffer−mode(2), quiet(2m).

about(2)

NAME

about − Information About MicroEmacs '02

SYNOPSIS

about

DESCRIPTION

about displays information about the current MicroEmacs '02 editing session and includes the following information:−

- Version number and date information for MicroEmacs '02.
- Global status information including the number of active buffers and global mode status information.
- Current buffer status information; buffer modes and buffer size information.

EXAMPLE

The following is an example output from **about**.

```
MicroEmacs '98 - Date 1/1/98

Global Status:
  # buffers : 21

  Modes on  : auto backup crlf exact magic quiet tab undo
  Modes off : binary cmode crypt ctrlz del dir edit hide indent
              justify letter line lock nact narrow over pipe rbin
              save time usr1 usr2 usr3 usr4 usr5 usr6 usr7 usr8
              view wrap

Current Buffer Status:
  Buffer    : m2cmd148.2
  File name : c:/emacsdoc/m2cmd148.2

  Lines     : Total     34, Current     27
  Characters: Total    759, Current    683

  Modes on  : auto backup edit exact indent justify magic quiet
              tab time undo wrap
  Modes off : binary cmode crlf crypt ctrlz del dir hide letter
              line lock nact narrow over pipe save rbin usr1 usr2
              usr3 usr4 usr5 usr6 usr7 usr8 view
```

SEE ALSO

describe−bindings(2), list−buffers(2).

add−color(2)

NAME

add−color − Create a new color
add−color−scheme − Create a new color scheme

SYNOPSIS

add−color "*col−no*" "*red*" "*green*" "*blue*"
n **add−color−scheme** "*schemeNum*" "*fore*" "*back*" "*current−fore*" "*current−back*"

"*selected−fore*" "*selected−back*"
"*current−selected−fore*" "*current−selected−back*"
["*fm−fore*" "*fm−back*" "*fm−cur−fore*" "*fm−cur−back*"
"*fm−sel−fore*" "*fm−sel−back*"
"*fm−cur−sel−fore*" "*fm−cur−sel−back*"] **DESCRIPTION**

add−color creates a new color and inserts it into MicroEmacs '02 colors table, where *red*, *green* and *blue* are the color components and *col−no* is the MicroEmacs '02 color table number. The color table contains 256 entries indexed by *col−no* in the range 0−255.

On some platforms (DOS and UNIX termcap) the number of colors is physically limited by the hardware to less than 256 (typically 16), in this case all 256 colors can be defined and for each created color the closest system color is used.

By default, only color 0 (white) and 1 (black) are defined. Once created, the colors may be used to create color schemes, this is the sole use of colors.

add−color may be used to modify an existing *col−no* index by re−assignment, the existing color definition is over−written with the new color definition. **add−color−scheme** creates a color scheme entry used by hilight(2), screen−poke(2), osd(2) and variables such as $global−scheme(5), $buffer−scheme(5), $ml−scheme(5).

The command takes an index number "*schemeNum*" and eight color values (defined by **add−color**) alternating between foreground and background colors. The 8 colors represent the 4 color paired states of foreground and background that may appear in a text buffer. The paired states correspond to current and selected lines (or permutations thereof). If an argument *n* is given to the command then *schemeNum* is set to a duplicate of the *n*th scheme, no other arguments are required.

schemeNum is the identifying index that is used to recognize the scheme. By default only two color schemes are defined at initialization, they are a monochrome scheme and inverse scheme with indices 0 and 1 using white as foreground and black as background, selected text is inverted. When defining a color scheme, if an existing *schemeNum* index is used then that scheme is modified.

The next eight arguments must be given, they specify foreground and background color pairs for the four different situations, as follows:−

Default

Color combination used when none of the following three are applicable.

Current

Color combination used when the text is on the same line as the cursor. It is also used by the $mode−line−scheme(5) for the current window's mode line and for the current selection on an osd(2) dialog.

Selected

Color combination used when the text is in the current selected region, but is not on the current line. Also used by **osd** for non−current item Hot keys.

Current−selected

Color combination used when the text is on the current line and in the current selected region. Also used by **osd** for the current item's Hot key.

The following 8 arguments set up fonts and are optional, any missing arguments are defaulted to 0. Each argument is a bitmask indicating which font should be enabled, where each bit is as follows:

`0x01`	Enable bold font.
`0x02`	Enable italic font.
`0x04`	Enable light font.
`0x08`	Enable reverse font.
`0x10`	Enable underlining.

Normally only the foreground value is used, i.e. the first, third, fifth and seventh values. But screen−poke(2) can be used to draw reversed color scheme in which case the background values are used.

EXAMPLE

The color palette is typically created at start−up via the configuration file **scheme*X*.emf**. These files are not easily read as they are automatically generated via the scheme−editor(3) dialog. A more readable form of "`schemed.emf`" would be as follows:−

```
; Standard colors
add−color &set .white     0 200 200 200
add−color &set .black     1 0   0   0
add−color &set .red       2 200 0   0
add−color &set .green     3 0   200 0
add−color &set .yellow    4 200 200 0
add−color &set .blue      5 0   0   200
```

```
add-color &set .magenta   6 200 0   200
add-color &set .cyan      7 0   200 200
; Light colors
add-color &set .lwhite    8 255 255 255
add-color &set .lblack    9 75  75  75
add-color &set .lred     10 255 0   0
add-color &set .lgreen   11 0   255 0
add-color &set .lyellow  12 255 255 0
add-color &set .lblue    13 0   0   255
add-color &set .lmagenta 14 255 0   255
add-color &set .lcyan    15 0   255 255
; Selection color
add-color &set .sel-col  16 91  78  131
; Set the required cursor-color
set-variable $cursor-color .col12
; Set up the standard schemes for the text, mode line message line, scroll bar and
add-color-scheme $global-scheme  .white .black .lwhite .black ...
    ... .white .sel-col .lwhite .sel-col 0 8 1 9 8 0 9 1
add-color-scheme $ml-scheme  .white .black .lwhite .black ...
    ... .white .sel-col .lwhite .sel-col 0 8 1 9 8 0 9 1
add-color-scheme $mode-line-scheme  .white .red .lwhite .lred ...
    ... .white .red .lwhite .red 8 0 9 1 0 8 1 9
add-color-scheme $scroll-bar-scheme .white .lblack .lwhite .lblack ...
    ...  .lblack .white .lblack .lwhite 8 0 9 1 0 8 1 9
    .
    .
```

NOTES

Color schemes can be created and altered using the scheme−editor(3) dialog, the created color scheme can then the used from start−up by using the user−setup(3) dialog. Therefore direct use of these commands is largely redundant.

The existence of a color or scheme index is checked as each entry is submitted, therefore any color or scheme used must have been previously been created, otherwise a default value is substituted.

Changing any existing color definitions causes all references to the color from a scheme to adopt the new color.

Changing any existing color−scheme definitions changes the rendered color of any hilight(2) etc., that was using that color−scheme.

A −ve color scheme value (i.e. −*n*) uses the previous '*n*'th entry that is defined in the color block. i.e. if *current−fore* was specified as −2 then it would inherit the *fore* field color.

Not all UNIX terminals support all the above fonts.

On some telnet packages color is not directly supported and some of the termcap display attributes such as bold and italic are represented by a color (e.g. italic text is shown in green). Using this translation it is possible to achieve reasonable color support on a VT100 terminal − it is a little awkward but is worth while if you have to use this type of connection frequently.

SEE ALSO

scheme−editor(3), user−setup(3), change−font(2), hilight(2), screen−poke(2), $buffer−hilight(5), $cursor−color(5), $global−scheme(5), $trunc−scheme(5), $ml−scheme(5), $osd−scheme(5), $mode−line−scheme(5), $scroll−bar−scheme(5), $system(5).

add−dictionary(2)

NAME

add−dictionary − Declare existence of a spelling dictionary

SYNOPSIS

n **add−dictionary** "*file*"

DESCRIPTION

add−dictionary adds the given dictionary (specified by the given *file*) to the dictionary list. Note that the *file* may omit the **.edf** extension, this is automatically added.

The command accepts a numeric argument '*n*' which determines the actions to be undertaken. When *n* is omitted then the dictionary is marked for loading (on demand) − this is the standard invocation used in the start up files.

If an argument of **0** is given the dictionary is created but it is not marked for loading, this can be used to create an empty dictionary.

If an argument of **−1** is given the contents of the dictionary are dumped into the current buffer, used for dictionary maintenance. The two main uses of this command are discussed below.

Dictionary Loading

A call to **add−dictionary** with no numeric argument does not perform an immediate load of the dictionary, instead the dictionary is only loaded on demand, i.e. when a call to spell(2) (usually via spell−word(3) or spell−buffer(3)) is made, this ensures that the start up time for MicroEmacs does not become too long. When the dictionary is loaded it is checked for efficiency, if found to be inefficient it is automatically optimized and flagged as changed. On exiting MicroEmacs, the user is prompted to save any dictionary that has be altered or optimized.

The spelling search order is made from the last dictionary added to the first, as soon as a word is found in a dictionary the search is halted. This implies that if a word has been defined incorrectly in one dictionary, but correct in another, the order in which the dictionaries are added determines the result.

The number of dictionaries allowed is unlimited but note that any words added are always added to the LAST dictionary. The size of the dictionary is restricted to about 16Mb, the size is NOT tested when words are added and if this size is exceeded the results are undefined. However, it is unlikely that this limit will be reached, the largest dictionary created to date is 0.8Mb.

A new main dictionary may be created as follows:−

1)

Find a file containing an **ispell(1)** compatible list of words.

2)

execute−file(2) spellutl.emf to define macro spell−add−word(3).

3)

Start up MicroEmacs '02 and execute the command **add−dictionary** giving an appropriate new dictionary name.

4)

Load up the file containing the words and execute the command spell−add−word(3) with a very large argument so all the words are added.

5)

Save the dictionary by either executing the command save−dictionary(2) or exiting. **Dictionary Dump**

A call to **add−dictionary** with a numeric argument *n* of −1 causes the contents of the given dictionary to be dumped into the current buffer (make sure you are in an empty buffer or ***scratch***) where:

`xxxx` − Good word `xxxx` with no spell rules allowed
`xxxx/abc` − Good word `xxxx` with spell rules `abc` allowed
`xxxx>yyyy` − Erroneous word with an auto−replace to `yyyy`

The dump of the dictionary may be edited, allowing erroneous entries to be removed. The macro file `spellutl.emf` contains macros edit−dictionary(3) and restore−dictionary(3) which enable the user to edit a dictionary.

NOTES

MicroEmacs '02 is supplied with a dictionaries for American and British English, it is strongly suggested that these dictionaries are **NOT** modified in anyway. Ensure that the dictionary is protected by loading the base dictionaries first, followed by a personal dictionary. New words added during spelling will then be added to the personal dictionary rather than the main dictionary.

EXAMPLE

The MicroEmacs '02 start−up file **me.emf** executes **language.emf** which in turn executes the user language setup file, for example **american.emf**, which adds the main language dictionaries and rules.

language.emf then adds the user's dictionary, this process can be simplified to:−

```
; add the main American dictionary
add−dictionary "lsdmenus"

; reset the spell rules
0 add−spell−rule
; Now add the American spell rules
−2 add−spell−rule "A" "" "" "re" ; As in enter > reenter
−2 add−spell−rule "I" "" "" "in" ; As in disposed > indisposed
    .
    .
; Now add the user dictionary
add−dictionary $MENAME
```

SEE ALSO

add−spell−rule(2), save−dictionary(2), spell−add−word(3), edit−dictionary(3), spell−buffer(3).

add−file−hook(2)

NAME

add−file−hook − Declare file name context dependent configuration

SYNOPSIS

n **add−file−hook** "*extensions*" "*fhook−name*"

DESCRIPTION

add−file−hook defines a macro binding between a file name or file type and a set of macros. This binding enables file type dependent screen highlighting and key bindings to be performed. For a higher level introduction refer to File Hooks.

add−file−hook operates in two different modes to establish the type of file:−

- Content recognition, by examination of the contents of the file.
- File extension recognition.

Content recognition has the highest priority and is used in preference to the file extension.

add−file−hook is called multiple times to add new recognition rules. The rules are interrogated in last−in−first−out (LIFO) order, hence the extension added last has a greater precedence than those added first. This ordering allows default rules to be over−ridden.

Initialization

add−file−hook must be initialized prior to the first call, using an invocation of the form:−

```
0 add-file-hook
```

with a numeric argument *n* of 0, and no arguments. This invocation resets the file hooks by deleting all of the installed hooks.

File Extension Recognition

add−file−hook with no numerical argument *n* allows the extension of a file (or the base file name if there is no extension) to be used to determine which user defined setup macro is to be executed. The *extensions* argument is a space separated list of *file endings* (as opposed to true extensions) and is usually specified with the extension separator. For example, the extension "`.doc`" may indicate that the file is a document and therefore the indent, wrap and justify buffer modes are required. This may be performed automatically by defining a macro which adds these modes and adding a file hook to automatically execute this macro whenever a file "`*.doc`" is loaded.

The command arguments are defined as follows:−

extensions

A space separated list of file extensions, which are to be checked, this list includes the extension separator (typically dot ('`.`'). It should be noted that the extension search is actually a comparison of the tail of the string, as such files such as *makefile*, which do not have an extension, are specified literally.

fhook−name

The name of the file hook to execute. This is the name of the macro to execute that initializes the buffer.

As an example:−

```
define−macro fhook−doc
    1 buffer−mode "indent"
    1 buffer−mode "wrap"
    1 buffer−mode "justify"
!emacro

add−file−hook ".doc" "fhook−doc"
```

It is quite possible that the same macro should be executed for a text file, i.e. "`*.txt`" this is achieved by a single **add−file−hook** as the space (' ') character is used as an extension separator, e.g.

```
add−file−hook ".doc .txt" "fhook−doc"
```

There are three special file hooks, which are **fhook−binary**, **fhook−rbin** and **fhook−default**, these are not predefined, but if the user defines them then they are executed whenever a file is loaded in binary or reduced binary mode (see buffer−mode(2)) or the extension does not match any of those defined.

Considering the `fhook−XXX` prefix, the initial '**f**' character must be present as this is changed to a '**b**' and an '**e**' when looking for the enter (begin) buffer and exit buffer hooks. These hooks are executed whenever the user swaps to or from a buffer (including creating and deleting). So for the given example, if the tab size of 8 is required in a document (but 4 elsewhere) then this operation this is performed by defining the `bhook−XXX` and `ehook−XXX` macros, e.g.:−

```
define−macro bhook−doc
    set−variable $tabsize 8
!emacro

define−macro ehook−doc
    set−variable $tabsize 4
!emacro
```

File hooks are often used to setup the desired *buffer modes*, *hilighting*, *local key bindings*, *abbreviation file*, etc.

Buffer hooks are usually used to set and restore conflicting global variables.

File Content Recognition

add−file−hook with a non−zero numerical argument *n* defines a macro binding between the content in a file and a set of macros. This binding enables file type dependent screen hi−lighting and key binding to be performed. For a full description of file hooks refer to File Hooks, for file extension dependent hooking refer to add−file−hook(2).

The content defined file hooks interrogate the contents of a file on loading and search for a *magic* string identifier embedded in the text which uniquely identifies the file type.

The recognition process performs a search of the first *n* (numerical argument) non−blank lines of the file, searching for the regular expression specified by the *extensions* argument. The sign of the numerical argument *n* is interpreted as follows:−

- **−ve** − Case insensitive search
- **+ve** − Case sensitive search

The command arguments are defined as follows:−

extensions

A regular expression string defining the text to be searched for.

fhook−name

The name of the file hook to execute. This is the name of the macro to execute that will initialize the buffer.

The search commences from the first non−blank line in the file, if the regular expression, defined by *extensions* is located then the file hook *fhook−name* is invoked. This is typically used to identify files which do not have file extensions i.e. UNIX shell script files. To identify a shell script file which commences with:−

```
#!/bin/sh
```

The following file hook is used:−

1 add−file−hook "**#!/.*sh**" "**fhook−shell**"

Note that "`.*sh`" also matches `/bin/csh`, `/usr/local/bin/zsh` etc, so care should be taken to ensure that the regular expression string is sufficiently well specified to recognize the file type.

The second class of embedded text are explicit identifiers embedded into the text. The embedded strings take the form:

−*− *mode* −*
−*− **Mode:** *mode*; ... −*−
−!− *mode* −!−

The −*− notation belongs to GNU Emacs, but MicroEmacs '02 recognizes the construct and extracts the string correctly. The −!− notation is MicroEmacs '02 specific and is provided so as not to cause conflict with GNU Emacs. MicroEmacs '02 searches for either construct on the first non−blank line of the file.

The explicit strings are defined with a negative numerical argument *n*, which identifies them as **explicit** rather than **magic** text strings. The *string* should be defined in lower case and matches a case insensitive string take from the file. e.g. to define a file hook for a make file:

```
#______________________________−!−Makefile−!−________________________________
#
# Make file for MicroEmacs using the Microsoft MSCV 2.0/4.0 development kit.
#
# Author       : Jon Green
# Created      :  020197.1002
# Last Edited : <150297.1942>
# File         : makefile.w32
....
```

might be defined as:

−1 **add−file−hook** "−!−[\t]*makefile.*−!−" fhook−make

NOTES

Automatic Macro File Loading

add−file−hook performs an automatic load of a macro file if the **fhook** macro is not present in memory. The file name of the command file containing the macro is automatically derived from the *name* component of the **fhook** macro name. The **fhook−** part of the name is stripped off and prepended with **hk** and suffixed with **.emf**. Hence, macro **fhook−doc** would be searched for in file `hkdoc.emf` within the MicroEmacs '02 directory. The command file is automatically loaded and executed.

In cases where the fhook macro is not located in an equivalent hook file, the file location of the macro may be explicitly defined for auto loading via a define−macro−file(2) invocation.

As an example, consider the C−mode file hook, used to load `.c` files. The loading of a C header file (`.h`) utilizes the same highlighting modes, but it's startup sequence is slightly different when handling new files. In this case the **fhook−cmode** for `.c` and **fhook−hmode** for `.h` files are located in the same hook file namely `hkcmode.emf`.

```
define−macro−file hkcmode fhook−hmode

add−file−hook ".c .cc .cpp .def .l .y .i .ac"   "fhook−cmode"
add−file−hook ".h .hpp"                          "fhook−hmode"
```

In this case the define−macro−file has been used to inform MicroEmacs '02 of the location of the **fhook−hmode** macro thereby overriding the automatic load of a file called **hkhmode.emf**. The **fhook−cmode** macro requires no such definition as it is located in a hook file that matches the mode name, `hkcmode.emf`.

Extending a standard hook definition

The standard file hook files **hk*XXX*.emf** should not be modified. The standard file hooks may be extended with local definitions by defining a file **my*XXX*.emf**, which is an extension to the hook file **hk*XXX*.emf**. This is automatically executed after **hk*XXX*.emf**. Refer to sections Language Templates and File Hooks for details.

File Extensions

The file extensions are specified as a space separated list of file name endings. Back−up file endings such as tilde (~) are not classed as correct file endings and are skipped by the file hook search, hence a file ending "`.c~`" invokes the same hook function as a "`.c`" file. It is therefore not necessary to add the backup and auto−save endings to the file hook definition.

The extension separator, usually dot (`.`), is typically added to the *extensions* list, they may be omitted with effect where a file always ends in the same set of characters. A notable example is "`makefile`" which includes no extension, as such, MicroEmacs '02 applies the same hook function to a file called `Imakefile` as the endings are the same.

Binary Files

It is sometimes useful to associate file types as binary files, so that they are immediately loaded in binary. In this case, both file extension and content recognition methods (i.e. of a magic string) are applicable. In both cases the file is bound to the well known hook `fhook−binary` which automatically loads the file in a binary mode.

Note, that for the content recognition process for a binary hook, the load time is doubled as the file is initially loaded in the default text mode, the binary hook function forces a second load operation in binary.

SUMMARY

add−file−hook is summarized as follows:−

- Binds one or more extensions to a macro called fhook−*xxxx*.
- Extensions are typically specified with the dot (`.`) separator.
- Multiple extensions are specified as a space separated list.
- Binds a regular expression search string to a macro called fhook−*xxxx*.
- The absolute value of the numerical argument determines the number of lines in the file over which the regular expression search is made.

- The sign of the numerical argument determines if the regular expression search is case (in)sensitive.
- When one of the files with a known file extension, or recognized content, is loaded macro **fhook−xxxx** is executed.
- **fhook−xxxx**, if undefined, is automatically searched for in file **hk***xxxx***.emf**.
- When the buffer containing the known file is entered (i.e. gains focus), then entry macro **bhook−xxxx** is executed.
- When the buffer containing the known file is exited (i.e. looses focus), then the exit macro **ehook−xxxx** is executed.

EXAMPLE

The standard set of supported file types by MicroEmacs '02, at the time of writing, is defined as:−

```
; reset the file hook list
0 add-file-hook
; Add file extension hooks.
; Files loaded in binary mode do not need hook as fixed
add-file-hook "*help* *info* .ehf"                            fhook-ehf
add-file-hook "*bindings* *commands* *variables*"             fhook-lists
add-file-hook "*buffers*"                                     fhook-blist
add-file-hook "/ *directory* *files*"                         fhook-dir
add-file-hook "*registry*"                                    fhook-reg
add-file-hook "*icommand* *shell* *gdb* *dbx*"                fhook-ipipe
add-file-hook ".emf"                                          fhook-emf
add-file-hook ".doc .txt"                                     fhook-doc
add-file-hook ".1 .2 .3 .4 .5 .6 .7 .8 .9 .so .tni .sm"       fhook-nroff
add-file-hook ".c .h .def .l .y .i"                           fhook-c
add-file-hook ".cc .cpp .hpp .rc"                             fhook-cpp
add-file-hook "Makefile makefile .mak"                        fhook-make
add-file-hook "Imakefile imakefile"                           fhook-imake
add-file-hook ".sh .ksh .csh .login .cshrc .profile .tcshrc"  fhook-shell
add-file-hook ".bat .btm"                                     fhook-dos
add-file-hook ".man"                                          fhook-man
add-file-hook ".dmn"                                          fhook-dman
add-file-hook ".ini .hpj .reg .rgy"                           fhook-ini
add-file-hook ".htm .html"                                    fhook-html
add-file-hook ".htp .hts"                                     fhook-hts
add-file-hook ".tcl"                                          fhook-tcl
add-file-hook ".rul"                                          fhook-rul
add-file-hook ".awk .nawk .gawk"                              fhook-awk
add-file-hook ".p .pas"                                       fhook-pascal
add-file-hook ".vhdl .vhd"                                    fhook-vhdl
add-file-hook ".fvwm .fvwm2rc"                                fhook-fvwm
add-file-hook ".java .jav"                                    fhook-java
add-file-hook ".nsr"                                          fhook-nsr
add-file-hook ".erf"                                          fhook-erf
; Add magic hooks
 1 add-file-hook "^#!/.*sh"               fhook-shell ; UNIX shell files
 1 add-file-hook "^#!/.*wish"             fhook-tcl
 1 add-file-hook "^#!/.*awk"              fhook-awk
 1 add-file-hook "^#VRML"                 fhook-vrml
-4 add-file-hook "<html>"                 fhook-html
-1 add-file-hook "-[*!]-[ \t]*c.*-[*!]-"         fhook-c     ; -*- C -*-
-1 add-file-hook "-[*!]-[ \t]*c\\+\\+.*-[*!]-"   fhook-cpp   ; -*- C++ -*-
```

```
-1 add-file-hook "-[*!]-[ \t]nroff.*-[*!]-"      fhook-nroff ; -*- nroff -*-
-1 add-file-hook "-!-[ \t]*shell.*-!-"           fhook-shell ; -!- shell -!-
-1 add-file-hook "-!-[ \t]*msdos.*-!-"           fhook-dos   ; -!- msdos -!-
-1 add-file-hook "-!-[ \t]*makefile.*-!-"        fhook-make  ; -!- makefile -!-
-1 add-file-hook "-!-[ \t]*document.*-!-"        fhook-doc   ; -!- document -!-
-1 add-file-hook "-!-[ \t]*fvwm.*-!-"            fhook-fvwm  ; -!- fvwm -!-
-1 add-file-hook "-!-[ \t]*erf.*-!-"             fhook-erf   ; -!- erf -!-
-1 add-file-hook "-!-[ \t]*fold:.*-!-"           fhook-fold  ; -!- fold:... -!-
```

OBSCURE INFORMATION

This section includes some low−level information which is so obscure it is not relevant to the typical user.

Resolving Loading Order Problems

There is a potential loading order problem involving auto−loading of file libraries and the setting up of **bhook** and **ehook**. E.g. if the main fhook function has been defined as a define−macro−file(2), but the bhook or ehooks have not the when a buffer is created as only the fhook is define, only the fhook is set, the rest remain disabled even though the execution of the macro file will define these extra hooks.

To solve this problem simply define the bhook/ehooks as well. Note that automatically loaded hooks do not suffer from this problem as the macro file is executed before the hooks are assigned, thereby ensuring the all the hooks are defined.

SEE ALSO

File Hooks, Language Templates, $buffer−bhook(5), $buffer−ehook(5), $buffer−fhook(5).

global−mode(2)

NAME

global−mode − Change a global buffer mode
add−global−mode − Set a global buffer mode
delete−global−mode − Remove a global buffer mode

SYNOPSIS

n **global−mode** "*mode*" (**esc m**)
add−global−mode "*mode*"
delete−global−mode "*mode*"

DESCRIPTION

global−mode changes the state of one of the hereditary global modes. A buffer's modes are initialized to the global modes when first created. This command is very useful in changing some of the default behavior such as case sensitive searching (see the example below). See Operating Modes for a full list and description of modes. Also see buffer−mode(2) for a full description of the use of the argument *n*.

The about(2) command gives a list of the current global and buffer modes.

add−global−mode and **delete−global−mode** are macros defined in meme3_8.emf which use global−mode to add or remove a global mode. They are defined for backward compatibility with MicroEMACS v3.8 and for ease of use; they are simple macros, add−global−mode is defined as follows:

```
define−macro add−global−mode
    ; Has the require mode been given as an argument, if so add it
    !force 1 global−mode @1
    !if &not $status
        ; No − use 1 global−mode to add a mode
        !nma 1 global−mode
    !endif
!emacro
```

EXAMPLE

The following example globally disables exact(2m) and magic(2m) modes, if these lines are copied to the user setup file then are searches will be simple and case insensitive by default:

```
−1 global−mode "exact"
−1 global−mode "magic"
```

NOTES

Globally adding binary(2m), crypt(2m) and rbin(2m) modes is strongly discouraged as any file loaded would be assigned these modes. Instead use the numeric argument of command find−file(2) or commands find−bfile(3) and find−cfile(3).

auto(2m), autosv(2m), backup(2m), exact(2m), magic(2m), quiet(2m), tab(2m) and undo(2m) modes are present on all platforms by default. On Windows and DOS platforms crlf(2m) is also present and on DOS ctrlz(2m) is also present.

SEE ALSO

Operating Modes, buffer−mode(2), find−bfile(3), find−cfile(3), about(2).

buffer−mode(2)

NAME

buffer−mode − Change a local buffer mode
named−buffer−mode − Change a named buffer mode
add−mode − Set a local buffer mode
delete−mode − Remove a local buffer mode
unmark−buffer − Remove buffer change flag

SYNOPSIS

n **buffer−mode** "*mode*" (**C−x m**)
n **named−buffer−mode** "*buffer−name*" "*mode*"
add−mode "*mode*"
delete−mode "*mode*"
unmark−buffer

DESCRIPTION

buffer−mode changes the state of a given buffer mode, affecting only the current buffer. A buffer's mode affects the behavior of MicroEmacs '02. The about(2) command gives a list of the current global and buffer modes. Refer to Operating Modes for a description of the buffer modes.

The argument *n* when given, has the following meaning:

Delete	Add	toggle	Mode
−1	1	0	Use "mode"
−2	2	130	auto
−3	3	131	autosv
−4	4	132	backup
−5	5	133	binary
−6	6	134	cmode
−7	7	135	crlf
−8	8	136	crypt
−9	9	137	ctrlz
−10	10	138	del
−11	11	139	dir
−12	12	140	edit
−13	13	141	exact
−14	14	142	hide
−15	15	143	indent
−16	16	144	justify
−17	17	145	letter
−18	18	146	line
−19	19	147	lock
−20	10	148	magic
−21	21	149	nact
−22	22	150	narrow

−23	23	151	over
−24	24	152	pipe
−25	25	153	quiet
−26	26	154	rbin
−27	27	155	save
−28	28	156	tab
−29	29	157	time
−30	30	158	undo
−31	31	159	usr1
−32	32	160	usr2
−33	33	161	usr3
−34	34	162	usr4
−35	35	163	usr5
−36	36	164	usr6
−37	37	165	usr7
−38	38	166	usr8
−39	39	167	view
−40	40	168	wrap

Note that when omitted the default argument is 0, i.e. prompt for and toggle a mode.

named−buffer−mode changes the state of a given buffer mode for a given buffer which may not be the current buffer.

add−mode and **delete−mode** are macros which use buffer−mode to add and remove a buffer mode. **unmark−buffer** is also a macro which removes the edit flag from the current buffer. They are defined for backward compatibility with MicroEMACS v3.8 and can be found in meme3_8.emf; add−mode is defined as follows:

```
define−macro add−mode
    ; Has the require mode been given as an argument, if so add it
    !force 1 buffer−mode @1
    !if &not $status
        ; No − use 1 buffer−mode to add a mode
        !nma 1 buffer−mode
    !endif
!emacro
```

NOTES

When a buffer is created it inherits the current global mode state.

SEE ALSO

Operating Modes, global−mode(2), about(2), &bmode(4).

add−next−line(2)

NAME

add−next−line − Define the searching behavior of command output

SYNOPSIS

n **add−next−line** "*buffer−name*" ["*string*"]

DESCRIPTION

add−next−line is used to set up the *next−line* functionality which is used by the get−next−line(2) command. The *next−line* feature is aimed at giving the user easy access to file locations which are stored in another buffer. This buffer may typically be the output from the **grep(1)** command or a compiler (e.g. **cc(1)**) and needs to contain the file name and line number of the required location.

As long as the format of the buffer is consistent and there is a maximum of one location per line, the *next−line* feature can be successfully configured.

The first argument, "*buffer−name*", gives the name the aforementioned buffer, this is "***grep***" for the grep(3) command etc. There is no limit on the number of next−line formats, nor on the number of **add−next−line** strings which are given. While there is no real need to initialize each new type, it is advised that the first **add−next−line** is called with a numerical argument of zero, e.g.:

```
0 add-next-line "*grep*"
add-next-line "*grep*" "....."
```

This tells MicroEmacs to reinitialize the type by freeing off any strings currently stored, note that the "*string*" argument is not used in this case. Resetting the next−line type safe guards against duplicate strings being added to it, a common problem if MicroEmacs is reinitialized.

Following is a typical output from grep:

```
foo.c: 45:      printf("hello world\n") ;
foo.c: 46:      printf("hello again\n") ;
```

If we replace the file name with "`%f`" and the line number with "`%l`", this becomes:

```
%f: %l:      printf("hello world\n") ;
```

get−next−line works on a left to right basis, as soon as it has enough information from the line it does not need to continue. Therefore the previous example can be reduced to just "`%f: %l:`". This is the string argument that should be given for the above example, i.e.:

```
add-next-line "*grep*" "%f: %l:"
```

get−next−line takes the given string and replaces the "`%f`" with $file−template(5) and the "`%l`" with the $line−template(5) and then uses the resultant string as a regular expression search string to find the next location. Crudely these could be set to "`foo.c`" and "`45`" respectively to find the first example, but this would fail to find any other. As a result the templates are usually magic search strings which will match any file and line number.

Similarly, following is an example output of the **gcc(1)** compiler:

```
basic.c:522: warning: `jj' might be used uninitialized in this command
display.c:833: warning: implicit declaration of function `ScreenPutChar'
```

In this case the **add−next−line** given needs to be:

```
add−next−line "*compile*" "%f:%l:"
```

If a −ve numerical argument is given to **add−next−line** the given 'next−line' is ignored, this can be useful when some warnings are to be ignored. For example a common warning from gcc is given when a variable might be used uninitialized, given as follows:

```
bind.c:578: warning: `ssc' might be used uninitialized in this function
```

These warnings can be ignored using the following:

```
−1 add−next−line "*compile*" ...
    ... "%f:%l: warning: `.*' might be used uninitialized in this function"
```

Some versions of **grep(1)** give the file name first and then the lines on the following lines. This is not a major problem as **get−next−line** remembers the last file name. The only problem occurs when skipping some parts of the list at which point the last file name parsed may not be the current file. Following is an example output of such a **grep** and the setup required:

```
File foo.c:
Line 45:      printf("hello world\n") ;
Line 46:      printf("hello again\n") ;
```

The configuration to locate the lines is defined as:

```
0 add−next−line "*grep*"
add−next−line "*grep*" "File %f:"
add−next−line "*grep*" "Line %l:"
```

NOTES

The reinitialize command format of this command changed in January 2001, the format changed from:

```
add−next−line "*grep*" ""
```

SEE ALSO

$file−template(5), $line−template(5), **cc(1)**, compile(3), get−next−line(2), **grep(1)**, grep(3).

add−spell−rule(2)

NAME

add−spell−rule − Add a new spelling rule to the dictionary

SYNOPSIS

n **add−spell−rule** ["*rule−letter*" "*base−ending*" "*remove*" "*derive−ending*"]

DESCRIPTION

add−spell−rule adds a new spelling rule to the speller. The rules effectively define the prefix and suffix character replacements of words, which is given an alphabetical identifier used within the speller , in conjunction with the language dictionary. The letter conventions are defined by the **Free Software Foundation** GNU **ispell(1)** package.

add−spell−rule is used in the MicroEmacs '02 dictionary initialization files called <*language*>r.emf, e.g. `americar.erf`, `britishr.erf` supplied in the MicroEmacs macros directory.

The command takes a single numeric argument *n* to control the addition of a rule to the speller, as follows:−

`0` **add−spell−rule**

Removes all existing rules and re−initializes. This is, by convention, explicitly called before instantiating a new set of rules.

`−1` **add−spell−rule** "*rule−letter*" "*base−ending*" "" "*deriv−ending*"

`−2` **add−spell−rule** "*rule−letter*" "*base−ending*" "" "*deriv−ending*"

Adds a prefix rule, an argument of −1 indicates that this prefix rule cannot be used with a suffix rule. An argument of −2 indicates it can be matched with any suffix rule which can be used with a prefix rule (e.g. argument of 2).

> "*rule−letter*" is any character in the range `A−z` except '_', all rules of the given letter must be a prefix rule of the same type (i.e. same argument). The start of a base word must match the given "*base−ending*" regular expression string for the rule to be applied, the "*remove*" string must be empty for a prefix and the "*deriv−ending*" is the prefix string. Example, for the American language;−
>
> `−2 add−spell−rule "I" "" "" "in"` ; As in disposed > indisposed
>
> A prefix rule of type '`I`' can be applied to any base word which has rule '`I`' enabled, and it

prefixes "`in`" to the word.

`1` **add−spell−rule** "*rule−letter*" "*base−ending*" "*remove*" "*deriv−ending*"

`2` **add−spell−rule** "*rule−letter*" "*base−ending*" "*remove*" "*deriv−ending*"

Add suffix rules. An argument of 1 indicates that this prefix rule cannot be used with a prefix rule. An argument of 2 indicates it can be matched with any prefix rule which can be used with a suffix rule (i.e. argument of −2).

"*rule−letter*" is any character in the range `A−z`, all rules of the given letter must be a suffix rule of the same type (i.e. same argument). The end of a base word must match the given "*base−ending*" regular expression string for the rule to be applied, the "*remove*" string must be a fixed string and the "*deriv−ending*" must also be a fixed string which is appended to the base−word after "*remove*" has been removed. Example, for the American language;−

```
2 add-spell-rule "N" "e" "e" "ion"     ; As in create > creation
2 add-spell-rule "N" "y" "y" "ication" ; As in multiply > multiplication
2 add-spell-rule "N" "[^ey]" "" "en"   ; As in fall > fallen
```

A suffix rule of type '`N`' can be applied to any base word which has rule '`N`' enabled, and it can be used with prefixes, e.g. with rule;−

```
-2 add-spell-rule "A" "" "" "re"       ; As in enter > reenter
```

to derive "*recreation*" from "*create*". A rule which cannot be used with prefixes, i.e.:

```
1 add-spell-rule "V" "e" "e" "ive"     ; As in create > creative
1 add-spell-rule "V" "[^e]" "" "ive"   ; As in prevent > preventive
```

While some prefix words are legal, such as "*recreative*" but some are not, such as "*collect*" where "*recollect*" is correct, so is "*collective*" but "*recollective*" is not.

SPECIAL RULES

Following are special forms of add−spell−rule used for tuning the spell support, note that an argument can not be given:−

add−spell−rule "−" "*<y|n>*"

Enables and disables the acceptance of hyphens joining correct words. By default the phrase "`out-of-date`" would be accepted in American even though the phrase does not exist in the American dictionary. This is because the three words making up the phrase are correct and by default hyphens joining words are allowed. Some Latin language such as Spanish do not use this concept so this feature can be disable.

add−spell−rule "*#*" "*score*"

Sets the maximum allowed error score when creating a spelling guess list. When comparing a dictionary word with the user supplied word, **spell** checks for differences, each difference or error is scored in the range of 20 to 27 points, once the maximum allowed score has been exceeded the word is ignored. The default guess error score is 60, allowing for 2 errors.

add−spell−rule "*" "*regex*"

Adds a correct word in the form of a regex if a word being spell checked is completely matched by the **regex** the word is deemed to be correct. For example, the following rule can be used to make the spell−checker allow all hex numbers:

```
add-spell-rule "*" "0[xX][[:xdigit:]]+"
```

This will completely match the words "`0x0`", "`0xff`" etc but not "`0x00z`" as the whole word is not matched, only the first 4 letters.

NOTES

The format of the dictionary is a list of base words with each word having a list of rules which can be applied to that word. Therefore the list of words and the rules used for them are linked e.g.

```
aback
abaft
abandon/DGRS
abandonment/S
abase/DGRS
abasement/S
abash/DGS
abashed/U
abate/DGRS
```

where the "`/...`" is the valid list of rules for that word.

The '_' character is used to separate different rule lists for a single word.

The format of the dictionary word list and the rule list is compatible with **ispell(1)**.

SEE ALSO

add−dictionary(2), spell(2) spell−buffer(3), spell−word(3), **ispell(1)**.

alarm(3)

NAME

alarm − Set an alarm

SYNOPSIS

alarm "*message*" "*hours*" "*minutes*"

DESCRIPTION

alarm creates an alarm for the user which will print the given "*message*" in the given number of "*hours*" and "*minutes*" time from the moment of creation.

The message is printed on the screen using osd(2).

NOTES

alarm is a macro defined in `misc.emf`.

SEE ALSO

osd(2).

nroff(9)

SYNOPSIS

0−9, tni, so − UNIX t/nroff file.

FILES

hknroff.emf − UNIX t/nroff file.
nroff.etf − UNIX t/nroff template file
ntags.emf − t/nroff tags generator macro definition.

EXTENSIONS

1, **2**, **3**, **4**, **5**, **6**, **7**, **8**, **9** − UNIX t/nroff files.
tni, **so** − UNIX t/nroff include files.
sm − *[Special]* Superman t/nroff file.

MAGIC STRINGS

−*− nroff −*−

Recognized by GNU and MicroEmacs. Denotes a t/nroff type file, may be used in **.1**/**.9**, **.tni** and **.so** files.

DESCRIPTION

The **nroff** file type templates handle UNIX n/troff type files.

General Editing

On creating a new file, a new header is automatically included into the file. time(2m) is by default enabled, allowing the modification time−stamp to be maintained in the header.

Hilighting

The hilighting features allow commands, variables, logical, preprocessor definitions, comments, strings and characters of the language to be differentiated and rendered in different colors.

Tags

A C−tags file may be generated within the editor using the **Tools** −> **Nroff−Tools** −> **Create Tag File**. find−tag(2) takes the user to the file using the tag information. The tags are generated using the **.XI** keyword, this may not be standard for all nroff pages.

Folding and Information Hiding

Generic folding is enabled within the C and C++ files. The folds occur about sections **.S**[**HS**]....**S**[**HS**] located on the left−hand margin. fold−all(3) (un)folds all regions in the file, fold−current(3) (un)folds the current region. Note that folding does not operate on K&R style code.

Tools

The nroff buffer provides a facility to toggle the hilighting of the buffer on and off. If font change inserts are used (**fB**, **fR**, etc), then the enclosed **bold** and *italic* regions are hilighted, hiding the escape sequences. This allows the nroff text to be viewed in a more representative rendered format.

The local buffer command **aman** invokes, the following command sequence (defined in `hkman`) to render a nroff **man** file into a buffer window;−

```
soelim <file> | tbl −TX | neqn | nroff −man | col −x
```

The command **tex2nr** attempts to convert a latex(9) file into an nroff file. The *latex* escape sequences are converted into their nroff equivalents. The command is only made available when an Nroff file is loaded (as the command is defined in the `hknroff.emf` file).

Short Cuts

The short cut keys used within the buffer are:−

C−c C−s − Insert a font size escape character **S0**.
C−c C−r − Insert a roman font escape character **fR**.
C−c C−b − Insert a bold font escape character **fB**.
C−c C−i − Insert a italic font escape character **fI**.
C−c C−c − Insert a courier font escape character **fC**.
C−c C−p − Insert a previous font escape character **fP**.
esc o, **esc q** − fill−paragraph(2) fills paragraph to next `.XX` command.
C−c b − Bold region by inserting **fB** .. **fR**.
C−c c − Courier region by inserting **fC** .. **fR**.
C−c c − Italic region by inserting **fI** .. **fR**.
C−c C−h − Toggle hilighting on/off.
C−c C−& − Adds nroff padding **f&** about words.
C−x C−& − Removes nroff padding **f&** about words.
esc h − Nroff help.

f2 − (un)fold the current region
f3 − (un)fold all regions

BUGS

The nroff language template is heavily biased towards the **man** macros only and includes all of the extension macros used for generating the JASSPA hypertext documentation.

The template in the current form has been used entirely by JASSPA in generating all of the documentation (**HTML**, **Winhelp**, **ehf**, **PostScript**) used by MicroEmacs '02. It does not include all of the troff/nroff keywords, or keywords for any of the standard macro packages.

The JASSPA documentation preparation tools are proprietary and have not been made publicly available.

SEE ALSO

fill−paragraph(2), find−tag(2), fold−all(3), fold−current(3), ntags(3f), time(2m).

Supported File Types

append−buffer(2)

NAME

append−buffer − Write contents of buffer to end of named file

SYNOPSIS

n **append−buffer** "*file−name*"

DESCRIPTION

append−buffer is used to write the contents of the current buffer into an EXISTING file. Use save−buffer(2) if the buffer is to over−write the existing file already associated with the buffer. Use write−buffer(2) if the buffer is to be written out to a new file, or to replace an existing file.

append−buffer writes the contents of the current buffer to the named file *file−name*. But unlike write−buffer(2) the action of the write does not change the attributes associated with the file (if it exists), it also does not effect the stats of the current buffer.

On writing the file, append−buffer ignores the time(2m) and backup(2m) mode settings. The current buffer will not be time stamped and a backup will not be created for "*file−name*". If the buffer contains a narrow(2m) it will automatically be removed before saving so that the whole buffer is saved and restored when saving is complete

The argument *n* is a bit based flag, where:−

0x01

Enables validity checks (default). These include a check that the given file already exist, if not confirmation of writing is requested from the user. Without this flag the command will always succeed wherever possible. If "*file−name*" does not exist the buffer is written out in a similar fashion to using the command write−buffer(2).

0x02

Disables the expansion of any narrows (see narrow−buffer(2)) before appending the buffer.

0x04

Truncate the existing file before writing out the contents of the buffer. This means that the file will consist solely of the contents of the buffer, but it will still have the file attributes of the original file.

If *n* is not specified then the default argument of 1 is used.

EXAMPLE

The following example appends the current buffer onto the end of a file, creating the file if it does not exists

```
append−buffer "things_to_do.txt"
```

The following example truncates the users email file while maintaining the file attributes. This is taken from vm(3) where it is used to remove the current mail from the system mail box.

```
find−buffer "*vm−empty−buffer"
−1 buffer−mode "ctrlz"
5 append−buffer %vm−mail−src
delete−buffer $buffer−bname
```

Note that the macro ensures that ctrlz(2m) mode is removed. If it was enabled then the file written would not be empty.

SEE ALSO

write−buffer(2), save−buffer(2).

ascii−time(3)

NAME

ascii−time − Return the current time as a string

SYNOPSIS

ascii−time

DESCRIPTION

ascii−time returns the current time as a formatted string in `#p9` which is equivalent to `#l9` for the calling macro. The format of the time string is:

"*WWW MMM DD hh:mm:ss yyyy*"

Where: `WWW`	− Week day, `Sun` − `Sat`
`MMM`	− Month, `Jan` − `Dec`
`DD`	− Day, `1` − `31`
`hh`	− Hour, `00` − `23`
`mm`	− Minute, `00` − `59`
`ss`	− Second, `00` − `59`
`yyyy`	− Year, `1998`...

EXAMPLE

The following is taken from etfinsrt.emf, it uses **ascii−time** in replacing "`$ASCII_TIME$`" with the current.

```
0 define−macro etfinsrt
    .
    .
    ; Change the create date $ASCII_TIME$.
    beginning−of−buffer
    ; Get ASCII time in #l9
    ascii−time
    !force replace−string "\\$ASCII_TIME\\$" #l9
    .
    .
!emacro
```

NOTES

ascii−time is a macro defined in `utils.emf`.

SEE ALSO

$buffer−fhook(5), &find(4), ascii−time(3).

auto−spell(3)

NAME

auto−spell − Auto−spell support
auto−spell−buffer − Auto−spell whole buffer
auto−spell−reset − Auto−spell hilight reset
auto−spell−ignore − Auto−spell ignore current word

SYNOPSIS

n **auto−spell**
auto−spell−buffer
auto−spell−reset
n **auto−spell−ignore**

DESCRIPTION

auto−spell enables and disables the auto spell checking of the current buffer. Auto spell detects word breaks as you type and checks the spelling of every completed word hilighting any erroneous words in the error color scheme (usually red).

The argument *n* determines whether auto−spell is enabled or disabled, a +ve argument enables and a −ve argument disables. If no argument or *0* is supplied the auto−spell state is toggled.

auto−spell−buffer checks all words within the current buffer for spell, hilighting any unknown or miss−spelled words found.

auto−spell−reset resets the buffer hilighting scheme, removing any added erroneous words.

auto−spell−ignore gets the current word and deletes the erroneous hilighting and adds the word to the current temporary ignore dictionary, auto−spell and the spelling−checker will now ignore the word. If an argument **n** of 2 is given to the command the word is added to the users personal dictionary instead of the temporary ignore dictionary so the word is 'ignored' in all future sessions of MicroEmacs as well.

NOTES

auto−spell, **auto−spell−buffer**, **auto−spell−reset** and **auto−spell−ignore** are macros defined in `spellaut.emf`.

SEE ALSO

user−setup(3), spell−buffer(3), spell(2).

forward−char(2)

NAME

forward−char − Move the cursor right backward−char − Move the cursor left

SYNOPSIS

n **forward−char** (**C−f**)
n **backward−char** (**C−b**)

DESCRIPTION

backward−char moves the cursor *n* characters to the left. Move to the end of the previous line if the cursor was at the beginning of the current line.

forward−char moves the cursor *n* characters to the right. Move to the beginning of the next line if the cursor was already at the end of the current line.

NOTES

backward−char is also bound to **left**.
forward−char is also bound to **right**.

SEE ALSO

forward−line(2), backward−line(2).

forward−delete−char(2)

NAME

forward−delete−char − Delete next character at the cursor position
backward−delete−char − Delete previous character at the cursor position

SYNOPSIS

n **forward−delete−char** (**C−d**)
n **backward−delete−char** (**backspace**)

DESCRIPTION

forward−delete−char deletes the next *n* characters from the current cursor position. If the cursor is at the end of a line, the next line is joined on the end of the current line. If an argument is given or letter(2m) mode is enabled then the character is added to the kill buffer, otherwise the kill buffer is unaltered.

backward−delete−char deletes the next *n* characters immediately to the left of the cursor (e.g. more conventionally backspace). If the cursor is at the beginning of a line, this will join the current line on the end of the previous one. If an argument is given or letter mode is enabled then the character is added to the kill buffer, otherwise the kill buffer is unaltered.

NOTES

forward−delete−char is also bound to **delete** and **S−delete**.

backward−delete−char is also bound to **S−backspace**.

SEE ALSO

backward−kill−word(2), forward−kill−word(2), letter(2m).

backward−delete−tab(2)

NAME

backward−delete−tab − Delete white space to previous tab−stop

SYNOPSIS

backward−delete−tab (**S−tab**)

DESCRIPTION

backward−delete−tab deletes all white characters left of the cursor back to the previous tab stop or non−white space, the deleted text is not added to the kill buffer.

SEE ALSO

tab(2), $tabsize(5), $tabwidth(5).

forward−kill−word(2)

NAME

forward−kill−word − Delete next word at the cursor position
backward−kill−word − Delete previous word at the cursor position

SYNOPSIS

n **forward−kill−word** (**esc d**)
n **backward−kill−word** (**esc backspace**)

DESCRIPTION

forward−kill−word deletes the next *n* words starting at the current cursor position, the deleted text is added to the kill buffer. See forward−word(2) for a description of word boundaries. If the argument *n* is `0` the command has no effect. If a −ve argument is specified, +*n* words are deleted and the text is not added to the kill buffer.

backward−kill−word deletes the previous *n* words before the cursor, the deleted text is added to the kill buffer. The numeric argument has the same effect as with **forward−kill−word**.

NOTES

backward−kill−word is also bound to **esc backspace**.

The −ve argument is typically used from macro scripts where the kill buffer is more precisely controlled.

SEE ALSO

backward−delete−char(2), forward−delete−char(2), forward−word(2), yank(2).

forward−line(2)

NAME

forward−line − Move the cursor to the next line
backward−line − Move the cursor to the previous line

SYNOPSIS

n **forward−line** (**C−n**)
n **backward−line** (**C−p**)

DESCRIPTION

forward−line moves the cursor down *n* lines, default 1. If the line is not on the current screen then display the next page and move to the line.

backward−line moves the cursor up *n* lines, if the line is not on the current screen then display the previous page and move to the line.

For both invocations a negative value reverses the sense of movement as expected.

SEE ALSO

backward−word(2), forward−word(2), scroll−down(2), scroll−up(2).

forward−paragraph(2)

NAME

forward−paragraph − Move the cursor to the next paragraph
backward−paragraph − Move the cursor to the previous paragraph

SYNOPSIS

n **forward−paragraph** (**esc n**)
n **backward−paragraph** (**esc p**)

DESCRIPTION

forward−paragraph puts the cursor at the end of the *n*th paragraph after the cursor, default is 1.

backward−paragraph puts the cursor at the beginning of the *n*th paragraph before the cursor, default is 1.

DIAGNOSTICS

The following errors can be generated, in each case the command returns a FALSE status:

[end of buffer]

When moving forwards, the given argument *n* was greater that the number of remaining paragraphs, the cursor is left at the end of the buffer.

[top of buffer]

When moving backwards, the given argument *n* was greater than the number of paragraphs before the cursor, the cursor is left at the beginning of the buffer. **NOTES**

- For both invocations a negative value reverses the sense of movement as expected.
- A paragraph break is defined as a blank line.

SEE ALSO

backward−line(2), forward−line(2), scroll−down(2), scroll−up(2).

forward−word(2)

NAME

forward−word − Move the cursor to the next word
backward−word − Move the cursor to the previous word

SYNOPSIS

n **forward−word** (**esc f**)
n **backward−word** (**esc b**)

DESCRIPTION

forward−word places the cursor at the end of the *n*th word from the current position; the default is 1.

backward−word places the cursor at the beginning of the *n*th previous word, default 1.

NOTES

Words are distinguished by non−alphanumeric characters and need not be white space such as spaces and tabs.

A character is considered to be part of a word if it is in the $buffer−mask(5) character set. The default setting for **$buffer−mask** is "`luh`" which gives a word character set of the alphanumeric characters, i.e. `0−9`, `A−Z`, `a−z`, this may be changed by setting the **$buffer−mask** variable. The character sets (including 4 user character sets `1−4`) may be altered by using the command set−char−mask(2).

SEE ALSO

backward−line(2), backward−paragraph(2), forward−line(2), forward−paragraph(2), Locale Support, $buffer−mask(5), set−char−mask(2).

beginning−of−buffer(2)

NAME

beginning−of−buffer − Move to beginning of buffer/file end−of−buffer − Move to beginning/end of buffer/file

SYNOPSIS

beginning−of−buffer (**esc** <)
end−of−buffer (**esc** >)

DESCRIPTION

beginning−of−buffer places the cursor at the beginning of the buffer/file.

end−of−buffer places the cursor at the end of the buffer/file.

NOTES

beginning−of−buffer is typically bound to **home**.
end−of−buffer is typically bound to **end**.

SEE ALSO

beginning−of−line(2), end−of−line(2).

beginning−of−line(2)

NAME

beginning−of−line − Move to beginning of line
end−of−line − Move to end of line

SYNOPSIS

beginning−of−line (**C−a**)
end−of−line (**C−e**)

DESCRIPTION

beginning−of−line places the cursor at the beginning of the line.

end−of−line places the cursor at the end of the line.

SEE ALSO

beginning−of−buffer(2), end−of−buffer(2).

global−abbrev−file(2)

NAME

global−abbrev−file, buffer−abbrev−file − Set abbreviation file(s).

SYNOPSIS

n **global−abbrev−file** "*abbrev−file*"
n **buffer−abbrev−file** "*abbrev−file*"

DESCRIPTION

The abbreviation files allow the user to define a set of short−cut expansion text, whereby a short sequence of chararacters are associated with a longer text segment. When the short sequence is entered, the user may elect to maually expand the sequnce with the associated replacement text. Provision for cursor positioning may be made in the replacement text.

buffer−abbrev−file sets the current buffer's abbreviation file (limit of one abbreviation file per buffer). **buffer−abbrev−file** does the minimal amount of work to increase speed at load−up. The first use of expand−abbrev(2) attempts to load the abbreviation file at which point errors may be reported.

An argument *n* of zero, forces the buffer abbreviation file to be uncached, such that the next abbreviation that is expanded forces a re−load of the abbreviation file. This is typically only used when an abbreviation file is being constructed and tested.

global−abbrev−file assigns a global set of abbreviations accross ALL buffers, such that the abbreviation is available regardless of the current buffer type. The global abbreviation file has a lower presidence than the **buffer−abbrev−file**, hence the currently assigned **buffer−abbrev−file** is searched before the **global−abbrev−file**.

Similarly for **global−abbrev−file**, an argument of zero forces the global abbreviation file to be uncached and re−loaded on the next use.

An abbreviation is a string which is expanded to an alternate form, e.g.

e.g. −> **for example**

or

PI −> **3.1415926536**
etc.

An abbreviation file is an ordinary text file with a strict format, it is loaded only once at the first call to expand−abbrev(2), from then on it reminds buffered. An abbreviation file has an abbreviation per line, they cannot use multiple lines. This is not a draw back as the expansion string is executed using execute−string(2) so any MicroEmacs '02 command may also be called.

For example the following expansion string inserts the string "`!continue`" and a newline:−

```
"!abort\r"
```

Note that '`\r`' is used instead of '`\n`' as **C−m** is bound to newline(2) and not **C−j**. The expansion string can also make use of a few useful abbreviations:−

\p

Mark the current position (expanded to "`C−x C−a P`")

\P

Move cursor to the marked position (expanded to "`C−x a P`")

See help on execute−string(2) for more useful abbreviations.

EXAMPLE

The abbreviation must be on the left hand side followed by at least 1 space, the expansion string must then be on the same line in quotes. So for the given examples, the abbreviation file would be:

```
|
|e.g. "for example"
|PI   "3.1415926536"
|
```

The following abbreviation could be used for a C *if−else* statement.

```
|
|if "if(\p)\r{\r\r}\relse\r{\r\r}\r\P"
|
```

This is particularly useful for email address, e.g.

```
|
|JA "\"JASSPA\" <support@jasspa.com>"
|
```

The following example is MicroEmacs '02 C−Mode abbreviation file for constructing C files. Remember **\p** is where the cursor is positioned following the expansion.

```
#i "#include <\p>\r\P"
#d "#define "
if "if(\p)\r{\r\r}\r\P"
ef "else if(\p)\r{\r\r}\r\P"
el "else\r{\r\p\r}\r\P"
wh "while(\p)\r{\r\r}\r\P"
```

```
        sw "switch(\p)\r{\rcase :\rdefault:\r}\r\P"
```

NOTES

Abbreviation files are given the extension **.eaf** in the MicroEmacs '02 home directory.

One of the easiest ways to create more complex abbreviations is to record a keyboard macro, name it and then insert the resultant macro. See notes on commands start−kbd−macro(2), name−kbd−macro(2) and insert−macro(2).

Try to avoid using named key, such as "`up`" and "`return`", as the keyboard macro equivalent is not readable and is likely to change in future releases.

FILES

c.eaf − C−Mode abbreviation file. **emf.eaf** − Macro code abbreviation file.

SEE ALSO

execute−string(2), expand−abbrev(2), insert−macro(2), iso−accents−mode(3), name−kbd−macro(2), start−kbd−macro(2), eaf(8).

buffer−bind−key(2)

NAME

buffer−bind−key − Create local key binding for current buffer
buffer−unbind−key − Remove local key binding for current buffer

SYNOPSIS

n **buffer−bind−key** "*command*" "*key*"
n **buffer−unbind−key** "*key*"

DESCRIPTION

buffer−bind−key creates a key binding local to the current buffer, binding the command *command* to the keyboard input *key*. This command is particularly useful in conjunction with file loading hooks (see add−file−hook(2)) allowing local key bindings dependent upon the context of the buffer.

The message line input is not effected by the current buffers local bindings.

buffer−unbind−key unbinds a user created local key binding, this command effects only the current buffer. If a −ve argument is given to **buffer−unbind−key** then all the current buffer's bindings are removed.

NOTES

The prefix commands cannot be rebound with this command.

Key response time linearly increases with each local binding added.

SEE ALSO

global−bind−key(2), ml−bind−key(2), osd−bind−key(2), global−unbind−key(2).

buffer−help(3)

NAME

buffer−help − Displays help page for current buffer

SYNOPSIS

buffer−help

DESCRIPTION

buffer−help opens a dialog giving the user a brief help page on tools available for the current buffer. The help page changes depending on the type of the current buffer.

SEE ALSO

buffer−setup(3).

buffer−info(2)

NAME

buffer−info − Status information on current buffer position

SYNOPSIS

buffer−info (**C−x =**)

DESCRIPTION

buffer−info reports on the current and total lines and characters of the current buffer. It also gives the hexadecimal code of the character currently under the cursor.

The output of the command is displayed on the message line e.g.

```
Line 1845/3955 Col 0.0 Char 78267/167172 (46%) Win Line 99/48 Col/0/0 char = 0xA
```

$result(5) is set to the same output string.

SEE ALSO

$result(5), $mode−line(5), about(2).

buffer−setup(3)

NAME

buffer−setup − Configures the current buffer settings

SYNOPSIS

buffer−setup

DESCRIPTION

buffer−setup provides a dialog interface to configuring the setup of the current buffer's file type within MicroEmacs. **user−setup** may be invoked from the main *help* menu or directly from the command line using execute−named−command(2).

The changes made to a configuration in **buffer−setup** are maintained in future MicroEmacs sessions by storing them within the user's setup registry file, "*<logname>*`.erf`". Note that not all file types may be supported by **buffer−setup**, if not the help menu item will not be available.

The contents of the dialog change, depending on the features the current buffer's file type supports. These features are implemented and installed within the buffer's file hook. The following buttons are always present at the bottom of the dialog:

`Save`

Saves the changes made to the configuration back to the users registry file, i.e. "*<Log−Name>*`.erf`" but does not re−initialize the current buffer. No changes made will effect the current buffer unless the **Current** button is pressed. Buffers of the same type created after the save may inherrit some of the changes.

`Current`

Makes the current buffer reflect the changes made, dismissing the **buffer−setup** dialog. This also performs the above '**Save**' operation. Some changes such as dialog creation changes, will only take effect when MicroEmacs is restarted.

`Exit`

Quits buffer−setup, if changes where not **Save**d or made **Current** they will be lost.

Following is a list of configurable features which may be available:

Create Help Page

Enables/disables the creation of a help page dialog for the tools available for the current file type.

Create Tools Menu

Enables/disables the creation of a file type specific sub menu located within the main menu's **Tools** sub−menu.

Use Author Mode

For file types which have an automatic formatter/viewer (currently only html) enabling this will simply load the file enabling the source code to be viewed and edited. When disabled files of this type will be automatically processed giving a more readable 'formatted' representation.

Insert New Template

When creating a new buffer/file of this type, a default template will be inserted if this is enabled. When disabled the buffer will remain empty.

Fence Display

Enables or disables the displaying of matching fences for this file type. Note that the way in which the matching fence is display is determined by the **Fence Display** option on the Platform page of user−setup(3); the **buffer−setup** option is ignored if this option is set to "`Never Display`".

Setup Hilighting

Creates and enables the token hilighting for the current file type.

Setup Auto Indent

Enables automatic formating (indenting) for the current file type. The indentation rules are either the built in 'C' indentation cmode(2m) or created using the indent(2) command. When enabled the tab(2m) is still adhered to, but the indent(2m) mode is ignored; when disabled the indent mode can be used.

Setup Auto Spell

Enables the setting up of auto−spell(3). When enabled the auto−spell key bindings are created and auto−spell is enabled if enabled within the user−setup dialog.

Setup Folding

Enables the setting up of section folding, when enabled the folding key bindings are created.

Add Abbreviations

Adds the file type's abbreviation file to the buffer using buffer−abbrev−file(2)

Search Modes: Exact

Enables/disables the exact(2m) mode over−riding the setting within the user−setup(3) dialog. If this setting is changed the setting within user−setup will be ignored for the current file type.

Search Modes: Magic

Enables/disables the magic(2m) mode over−riding the setting within the user−setup(3) dialog. If this setting is changed the setting within user−setup will be ignored for the current file type.

Buffer Modes: Auto

Enables/disables the auto(2m) mode.

Buffer Modes: Backup

Enables/disables the backup(2m) mode.

Buffer Modes: Indent

Enables/disables the indent(2m) mode.

Buffer Modes: Justify

Enables/disables the justify(2m) mode.

Buffer Modes: Tab

Enables/disables the tab(2m) mode over−riding the setting within the user−setup(3) dialog. If this setting is changed the setting within user−setup will be ignored for the current file type.

Buffer Modes: Time

Enables/disables the time(2m) mode.

Buffer Modes: Undo

Enables/disables the undo(2m) mode over−riding the setting within the user−setup(3) dialog. If this setting is changed the setting within user−setup will be ignored for the current file type.

Buffer Modes: Wrap

Enables/disables the wrap(2m) mode. **NOTES**

buffer−setup is a macro using osd(2), defined in `buffstp.emf`.

SEE ALSO

buffer−help(3), user−setup(3), File Hooks.

c−hash−eval(3)

NAME

c−hash−eval − Evaluate C/C++ #defines
c−hash−del − Remove C/C++ #define evaluation
c−hash−set−define − Set a C/C++ #define
c−hash−unset−define − Unset a C/C++ #define

SYNOPSIS

n **c−hash−eval**
c−hash−del
c−hash−set−define "*variable*" "*value*"
c−hash−unset−define "*variable*"

DESCRIPTION

c−hash−eval evaluates C/C++ '#' lines, hiding sections of code which have been 'hashed' out. **c−hash−eval** evaluates the following '#' lines:−

```
#define <variable> ....
#ifdef <variable>
#if ...
#else
#endif
```

For `#define`s **c−hash−eval** creates a user variable "`%cd<variable>`", setting it to the value found. For `#ifdef` a simple check for the existence of variable "`%cd<variable>`" is made. If defined then code between the `#ifdef` and either its matching `#else` or `#endif` is displayed and code between the `#else` and `#endif` is hidden. If it is not defined then the reverse happens.

The state of `#if`'s are evaluated using calc(3), the following code is then displayed as for `#ifdef`.

Code is hidden by setting the $line−scheme(5) to a color similar to the back−ground. If an argument is given to the command the code is also narrowed out using narrow−buffer(2).

c−hash−del undoes the effect of **c−hash−eval** by restores hidden code.

c−hash−set−define and **c−hash−unset−define** can be used to manually set and unset `#define` variables.

NOTES

c−hash−eval, **c−hash−del**, **c−hash−set−define** and **c−hash−unset−define** are macros defined in `cmacros.emf`.

Executing **c−hash−eval** in a project header file (h file) which contains all used `#define` definitions will set up all `#define` variables ready for the main C files.

SEE ALSO

calc(3), $line−scheme(5), narrow−buffer(2).

calc(3)

NAME

calc − Integer calculator

SYNOPSIS

n **calc** "*string*"

DESCRIPTION

calc can perform simple integer based calculations given by "*string*", where the "*string*" takes the following form:−

```
"[b]<s>"
```

Where '*b*' is an optional letter setting the required output base which can be one of the following:

`b`	− Binary
`o`	− Octal
`d`	− Decimal
`x`	− Hexadecimal

Default when omitted is '*d*' (decimal). "*s*" is the sum to be calculated, which should be bodmas in form. Following is a list of valid symbols.

`(..)`	− Parentheses (contents calculated first)
`!`	− Logical not
`&&`	− Logical and
`||`	− Logical or
`==`	− Logical equals
`!=`	− Logical not equals
`~`	− Bitwise not
`&`	− Bitwise and
`|`	− Bitwise or
`^`	− Bitwise xor
`/`	− Divide
`*`	− Multiply
`%`	− Modulus
`+`	− Addition
`-`	− Subtraction
`0xNN`	− Hexadecimal number
`0NN`	− Octal number
`LR`	− Last calculation recall

Any MicroEmacs variables can be used in the calculation. The result of the calculation is stored in .calc.result(5). The argument *n* is a bitwise flag where:

0x01

Print out the result on the message−line.

0x02

Use string comparisons for == and != comparisons. This has the advantage of being able to calc "Foo" == "Bar" etc.

When omitted the default argument is 1.

EXAMPLE

To calculate the number of hours in a year:

```
calc "365*24"
```

To then calculate the number of seconds in the year:

```
calc "LR*60*60"
```

NOTES

calc is a macro defined in `calc.emf`.

SEE ALSO

.calc.result(5).

capitalize−word(2)

NAME

capitalize−word − Capitalize word
lower−case−word − Lowercase word (downcase)
upper−case−word − Uppercase word (upcase)
lower−case−region − Lowercase a region (downcase)
upper−case−region − Uppercase a region (upcase)

SYNOPSIS

n **capitalize−word** (**esc c**)
n **lower−case−word** (**esc l**)
n **upper−case−word** (**esc u**)

lower−case−region (**C−x C−l**)
upper−case−region (**C−x C−u**)

DESCRIPTION

capitalize−word capitalizes the next *n* words.

lower−case−word changes the next *n* words to lower case.

upper−case−word changes the next *n* words to upper case.

lower−case−region changes all alphabetic characters in the marked region to lower case (see set−mark(2)).

upper−case−region changes all alphabetic characters in the marked region to upper case

SEE ALSO

set−mark(2).

change−buffer−name(2)

NAME

change−buffer−name − Change name of current buffer

SYNOPSIS

n **change−buffer−name** "*buffer−name*" (**esc C−n**)

DESCRIPTION

change−buffer−name changes the name of the current buffer to *buffer−name*. Buffer names must be unique as they act as the identity handle. By default the buffer name is derived from the buffer's file name excluding the path. This can lead to conflicts, when editing files with the same name and different paths, in which case a counter is appended to the end of the buffer name to make the name unique. For example:

```
File Name               Buffer Name

/etc/file.c             file.c
/tmp/file.c             file.c<1>
```

By default, or an argument is given with bit 1 set, **change−buffer−name** will fail if a buffer with the given name already exists. This behavior can be changed by giving an argument with the first bit cleared, e.g. 0, in which case if a buffer with that name already exists then a counter as appended.

SEE ALSO

$buffer−fname(5), change−file−name(2), delete−buffer(2).

change−directory(2)

NAME

change−directory − Change the current working directory

SYNOPSIS

change−directory "*dir−name*" (**C−x C−d**)

DESCRIPTION

change−directory changes the current working directory to *dir−name*, on certain platforms (MS−DOS) this can also change the current drive. This command is largely redundant as any shell command automatically inherits the directory of the current buffer's file.

SEE ALSO

change−file−name(2).

change−file−name(2)

NAME

change−file−name − Change the file name of the current buffer

SYNOPSIS

change−file−name "*file−name*" (**C−x n**)

DESCRIPTION

change−file−name changes the file name of the current buffer to *file−name*. A validity check is made on the given file name and if found to be invalid (e.g. its a directory) the name is rejected.

SEE ALSO

change−buffer−name(2), change−directory(2), write−buffer(2).

change−font(2)

NAME

change−font − Change the screen font

SYNOPSIS

[X−Windows]
change−font "*fontName*"

[IBM−PC / MS−DOS]
change−font "*mode−no*" "*spec*"

[Microsoft Windows]
n **change−font** "*name*" *charSet weight width height*

DESCRIPTION

change−font is a platform specific command which allows the displayable font to be modified. The selection of font is determined by the monitor resolution and the capabilities of the graphics adapter.

This command is available on all systems except termcap. While MS−DOS does not support the concept of different fonts, it does (or at least the graphics card does) support the concept of changing screen resolution, which has the effect of changing the font. Each platform takes different arguments and are considered independently, as follows:

X−Windows

The X−Windows UNIX environments accept a single argument which is a fully qualified font name. Simply give the font X name and the font will change if it is available. The window size changes to attempt to retain the same number of rows and columns so ensure that when changing to a larger font then there is enough room (or a way) to resize a window which is larger than the actual screen.

The X font string describes the attributes of the font in terms of it's size name etc. as follows:−

−foundry−family−weight−slant−width−−pixels−point−hres−vres−space−av−set

Where

foundry

The type of foundry that digitized and supplied the font.

family

Font Family.

weight

Modifies the appearance of the font, the *weight* is usually **medium** or **bold**.

slant

Determines the orientation of the font. *slant* is usually **r**oman (upright), **i**talic or **o**blique.

width

Describes the proportionate width of the font. Typical widths include **normal**, **condensed**, **narrow**, **double**.

pixels

Pixel size of the font

point

The resolution of the font in tenths of a **dpi** (i.e. dpi*10)

hres

Horizontal resolution of the font in dpi.

vres

Vertical resolution of the font in dpi.

space

The spacing of the font. Typical spacing values include **m**onospaced (i.e. fixed width), **p**roportional and **c**haracter cell.

av

Mean width of all font characters, measured in tenths of a pixel.

set

Character set − character set standards e.g. **iso8859−1**.

The default font used by MicroEmacs '02 is

```
        −*−fixed−medium−r−normal−−15−*−*−*−c−90−iso8859−1
```

A good font to try is:

```
change-font "-misc-fixed-medium-r-normal--13-*-*-*-c-80-iso8859-1"
```

The font may also be changed in your **.Xdefaults** file by inserting the line:−

```
MicroEmacs.font "-misc-fixed-medium-r-normal--13-*-*-*-c-80-iso8859-1"
```

IBM−PC / MS−DOS

MS−DOS may only change the screen resolution, the standard screen resolution is either 80 columns by 25 rows or 80 by 50. A more advanced graphics card can typically support up to 132 by 60, MicroEmacs in theory has no limit but it has only been tested up to this size.

The main problem with MS−DOS machines is that there is no standard and this is no exception. The graphics mode needed to get a 132 by 60 screen (if available) varies from one card to the next so MicroEmacs '02 needs to know the graphic mode number your card uses to get your required screen resolution.

MicroEmacs '02 can also attempt a little bit of magic to double the number of rows on the screen for a given screen resolution. This is how 50 lines are obtained from the standard 25 line mode 3. If the value of "*spec*" is non−zero then this is attempted, to the authors knowledge this will either work or not depending on the direction of the wind and no harm will befall the users equipment. However the author also quickly disclaims anything and everything, the user uses this at their own peril, like everything else.

MicroEmacs '02 attempts to determine the new screen width and depth itself, in case this fails the commands change−frame−width(2) and change−frame−depth(2) may be used to correct the problem.

Following are the standard MS−DOS text modes:

```
change-font "2" "0"       ; Simple monochrome or EGA monitor, 80 by 25.
change-font "3" "0"       ; Simple EGA/VGA monitor, again 80 by 25.
change-font "3" "1"       ; Simple EGA/VGA monitor using spec, 80 by 50.
```

Most Trident cards support the following text mode:

```
change-font "86" "0"      ; Sweet 132 by 60
```

A Diamond Stealth supports the following mode:

```
change-font "85" "1"      ; Nice 132 by 50
```

Cirrus video cards (1MB) seem to support:

```
change-font "84" "1"      ; PT-526 (132x50)
```

Time to start digging out your graphics card manual!

Microsoft Windows

The Microsoft Windows environments utilize font files to drive the display. When **change−font** is invoked with no arguments, or a −ve argument then a font dialog is presented to the user to allow the font to be selected. The current font is not changed if a −ve argument is given, in both cases the variable $result(5) is set the the user selected font. The format of the returned string is "`OWwwwwhhhhhFontName`", where:−

O

The type of character set (0 for OEM and 1 for ANSI).

W

The font weight (0 − 9).

wwww

The font width.

hhhh

The font height.

FontName

The font name.

If a +ve argument is specified with **change−font** then the arguments are explicitly entered, arguments are defined as follows:−

font

The name of the font − maximum of 32 characters. Select Fixed mono fonts only. Proportional fonts may be specified but the cursor will not align with the characters on the screen.

An empty name ("") may be specified resulting in the selection of the default system OEM font. No other arguments are required when specified.

Note that **Courier New** is not actually a fixed mono font as might be expected.

charSet

The type of character set required, this is an integer value of:−

```
  0 − ANSI or Western (True Type etc)
161 − Greek
```

`162` − Turkish
`204` − Russian
`255` − OEM (or bitmapped)

weight

The weight of the font. The values are defined as:−

`0` − Don't care (Automatically selected).
`1` − Thin
`2` − Extra Light
`3` − Light
`4` − Normal
`5` − Medium
`6` − Semi−Bold
`7` − Bold
`8` − Extra−Bold
`9` − Heavy

Note that you may request a weight and it is not honored. Typically 4 and 7 are honored by most font definitions. 4 is typically used.

width

The width of the font. Specifies the average width, in logical units, of characters in the requested font. If this value is zero, the font mapper chooses a "closest match" value. The "closest match" value is determined by comparing the absolute values of the difference between the current device's aspect ratio and the digitized aspect ratio of available fonts.

Note that if the width is specified as zero then the height should be specified and the width will be automatically selected.

height

The height of the font. Specifies the desired height, in logical units, of the requested font's character cell or character. (The character height value is the character cell height value minus the internal−leading value.) If this value is greater than zero, the font mapper matches it against available character cell height values; if this value is zero, the font mapper uses a default height value when it searches for a match; if this value is less than zero, the font mapper matches it against available character height values.

Note: as with the weight the width and height may not be honored if the font cannot support the specified width/height in which case the closest matching height is automatically selected

Notes on the Standard Windows Configuration

For releases prior to '99, the **Terminal** font is the standard MS−DOS font used for the MS−DOS window. This is an OEM fixed width character set which contains all of the conventional symbols found in the DOS shell.

Releases of MicroEmacs post '99 may utilise any of the windows fonts, typically `Courier New` or `Lucida Console` are used, these provide the best screen rendering of characters. `Lucida Console` is slightly better with a smaller font size as this allows a '`1`' (one) and '`l`' (lower case L) to be distinguished.

The **Terminal** fonts are the same as shown in the DOS window the last 2 arguments are the width x height, the terminal equivalents (Bit Mapped) are commented here.

7x12

Regular weight seems to offer the best resolution for 14/15" monitors.

6x8

Regular weight is more suitable for 17−21" monitors which offer better resolutions.

The best options for the fonts are defined as follows:−

```
;Standard Terminal Fonts - standard weight
;change-font "Terminal" 0 4  4  6
change-font "Terminal" 0 4  6  8
;change-font "Terminal" 0 4  8  8
;change-font "Terminal" 0 4  5 12
;change-font "Terminal" 0 4  7 12
;change-font "Terminal" 0 4  8 12
;change-font "Terminal" 0 4 12 16
;change-font "Terminal" 0 4 10 18

;Standard Terminal Fonts - heavy weight
;change-font "Terminal" 0 7  4  6
;change-font "Terminal" 0 7  6  8
;change-font "Terminal" 0 7  8  8
;change-font "Terminal" 0 7  5 12
;change-font "Terminal" 0 7  7 12
;change-font "Terminal" 0 7  8 12
;change-font "Terminal" 0 7 12 16
;change-font "Terminal" 0 7 10 18
```

The "**Courier New**" font is not actually a fixed mono font as might be expected.

SEE ALSO

change−frame−width(2), change−frame−depth(2), $result(5), user−setup(3).

change−frame−depth(2)

NAME

change−frame−depth − Change the number of lines on the current frame
change−frame−width − Change the number of columns on the current frame

SYNOPSIS

n **change−frame−depth** ["*depth*"]
n **change−frame−width** ["*width*"]

DESCRIPTION

change−frame−depth changes the depth of the current frame, if the numeric argument *n* is given then the frame depth is changed by *n* lines. If *n* is not specified the user is prompted for the new *depth* and the frame depth will be changed to this value. It is assumed that the screen can draw the requested *n* lines and MicroEmacs draws the lines at the users peril.

A change in depth causes all of the internal windows currently displayed in the frame to be re−sized, the vertical position of the windows are modified to match the new screen dimension, the horizontal position of the windows remains unaltered. If the window is down−sized and the currently displayed windows are not able to fit into the new screen space then all windows are deleted with the exception of the current window.

change−frame−width changes the width of the current frame, if the numeric argument *n* is given then the frame width is changed by *n* characters. If *n* is not specified the user is prompted for the new *width* and the frame width will be changed to this value. It is assumed that the screen can draw the requested *n* columns and MicroEmacs draws them at the users peril. The windows are reorganized as **change−frame−depth** working horizontally rather than vertically.

NOTES

Within windowing environments such as **X−Windows** and **Microsoft Windows** these commands cause the canvas window to be re−sized to accommodate the change in screen size.

In MS−DOS and UNIX Termcap environments the physical size of the screen is determined by the characteristics of the display adapter. **change−frame−depth** may be used to correct anomalies (usually on portables) in the displayable screen area and the graphics mode. e.g. In DOS the graphics mode utilizes 50 lines, and only 47 lines are viewable. In this case change the screen depth to 47 and MicroEmacs will not utilize the remaining lines which are not viewable.

SEE ALSO

$frame−depth(5), $frame−width(5).

change−window−depth(2)

NAME

change−window−depth − Change the depth of the current window
grow−window−vertically − Enlarge the current window (relative change)
shrink−window−vertically − Shrink the current window (relative change)
resize−window−vertically − Resize the current window (absolute change)

SYNOPSIS

n **change−window−depth** ["*depth*"]

n **grow−window−vertically**
n **shrink−window−vertically**
n **resize−window−vertically**

DESCRIPTION

change−window−depth changes the depth of the current window, if the numeric argument *n* is given then the window depth is changed by *n* lines. If *n* is not specified the user is prompted for the new *depth* and the window depth will be changed to this value. The command aborts if the requested size cannot be achieved (the window becomes too small or a neighbouring one does).

NOTES

Commands **grow−window−vertically**, **shrink−window−vertically** and **resize−window−vertically** were replaced by the new **change−window−depth** command in April 2002. Following are macro implementations of the old commands:

```
define−macro grow−window−vertically
    @# change−window−depth
!emacro

define−macro shrink−window−vertically
    &neg @# change−window−depth
!emacro

define−macro resize−window−vertically
    !if &not @?
        !abort
    !endif
    change−window−depth @#
!emacro
```

SEE ALSO

change−window−width(2), resize−all−windows(2), split−window−vertically(2).

change−window−width(2)

NAME

change−window−width − Change the width of the current window
grow−window−horizontally − Enlarge current window horizontally (relative)
shrink−window−horizontally − Shrink current window horizontally (relative)
resize−window−horizontally − Resize current window horizontally (absolute)

SYNOPSIS

n **change−window−width** ["*width*"]

n **grow−window−horizontally**
n **shrink−window−horizontally**
n **resize−window−horizontally**

DESCRIPTION

change−window−width changes the width of the current window, if the numeric argument *n* is given then the window width is changed by *n* characters. If *n* is not specified the user is prompted for the new *width* and the window width will be changed to this value. The command aborts if the requested size cannot be achieved (the window becomes too small or a neighbouring does).

EXAMPLE

Refer to `mouse.emf` for an example of window growth using the mouse to manipulate the size of the windows.

NOTES

Commands **grow−window−horizontally**, **shrink−window−horizontally** and **resize−window−horizontally** were replaced by the new **change−window−width** command in April 2002. Following are macro implementations of the old commands:

```
define−macro grow−window−horizontally
    @# change−window−width
!emacro

define−macro shrink−window−horizontally
    &neg @# change−window−width
!emacro

define−macro resize−window−horizontally
    !if &not @?
```

```
            !abort
        !endif
        change−window−width @#
    !emacro
```

SEE ALSO

change−window−depth(2), resize−all−windows(2), split−window−horizontally(2).

charset−change(3)

NAME

charset−change − Convert buffer; between two character sets
charset−iso−to−user − Convert buffer; ISO standard to user character set
charset−user−to−iso − Convert buffer; user to ISO standard character set

SYNOPSIS

charset−change
charset−iso−to−user
charset−user−to−iso

DESCRIPTION

charset−change opens a dialog allowing the user to select a **From** and **To** character set. If the *Convert* button is selected the current buffer is converted to the destination character set. The command assumes that the current buffer is written in the **From** character set, no attempt is made to verify this.

charset−iso−to−user converts the current buffer, assumed to be in ISO−8859−1 (Latin 1) font format, to the current user's character set (defined by user−setup(3)). This process typically corrects any foreign language display problems.

Conversely, **charset−user−to−iso** converts the current buffer from the user's character set to ISO−8859−1 (Latin 1), this is typically used for the transfer of text files between different systems.

The current character set is configured using the user−setup(3) dialog (see Display Font Set). This in turn uses the command set−char−mask(2) to create the low level character conversion tables.

NOTES

charset−change, **charset−iso−to−user** and **charset−user−to−iso** are macros defined in `langutl.emf`.

SEE ALSO

user−setup(3), set−char−mask(2), Locale Support.

check−line−length(3)

NAME

check−line−length − Check the length of text lines are valid

SYNOPSIS

check−line−length

DESCRIPTION

check−line−length checks that the length of each line of the current buffer, starting with the current line, is less than or equal to fill−col(5). The command aborts if a line too long is found, leaving the cursor on the offending line. If no invalid lines are found the command succeeds leaving the cursor at the end of the buffer.

NOTES

check−line−length is a macro implemented in `misc.emf`.

SEE ALSO

$fill−col(5).

clean(3)

NAME

clean − Remove redundant white spaces from the current buffer

SYNOPSIS

n **clean**

DESCRIPTION

clean removes redundant white spaces from the current buffer, there are three types this command remove:

1)

Any space or tab character at the end of the line. All are removed until the last character is not a space or a tab, or the line is empty. Note that an empty line is not removed unless at the end of the buffer.

2)

Space characters are removed when the next character is a tab, making the space redundant, e.g. the strings "`Hello World`" and "` Hello World`" will look identical because the tab character (' ') will indent the text to the 8th column with or without the space so the space can be removed.

3)

Superfluous empty lines at the end of the buffer are removed, leaving only one empty line.

4)

If argument *n* is given (value is not used) multiple blank lines are reduced to a single blank line.
DIAGNOSTICS

```
[Command illegal in view mode]
```

Caused by a redundant white space being found and the buffer being in view mode. Note that if clean completes while the buffer is in view mode then no superfluous white spaces where found. **NOTES**

clean is a macro defined in `format.emf`.

Most of this command's operation is performed by simple regex search and replace strings:

a)

Search for: "[\t]+$" Replace with: "\\0"

b)

Search for: "[ ]+\t" Replace with: "\t"

c)

Search for: "\n\n\n" Replace with: "\n\n" **SEE ALSO**

replace−string(2), tab(2m), delete−blank−lines(2), tabs−to−spaces(3).

command−apropos(2)

NAME

command−apropos − List commands involving a concept

SYNOPSIS

command−apropos "*string*" (**C−h a**)

DESCRIPTION

command−apropos compiles a list of all commands with *string* in their name, also giving their current key bindings.

EXAMPLE

To find all of the commands with "`command`" in their name space then issue the command "`C−h a command`" which generates a list of commands such as:−

```
abort−command ................. "C−g"
                                "esc C−g"
                                "C−x C−g"
command−apropos ............... "C−h a"
command−complete
execute−named−command ......... "esc x"
help−command .................. "C−h C−c"
ipipe−shell−command ........... "esc \\"
list−commands ................. "C−h c"
pipe−shell−command ............ "esc !"
                                "esc @"
                                "C−x @"
shell−command
```

SEE ALSO

describe−bindings(2).

command−wait(2)

NAME

command−wait − Conditional wait command

SYNOPSIS

n **command−wait**

DESCRIPTION

When a +ve argument *n* is given **command−wait** waits for *n* milliseconds before returning, this wait cannot be interrupted. If a −ve argument is given, **command−wait** waits for −*n* milliseconds but the command will return if the user interrupts with any input activity (i.e. presses a key).

When no argument is given **command−wait** loops getting and processing events (user input, screen updates etc) until either the calling commands **.wait** command variable is undefined or set to false (`0`). This more complex use of the command is used when a main macro must wait and process input until an exit criteria has been met, the input is best processed by setting the $buffer−input(5) variable to a second macro. The macro gdiff(3) uses this command in this way.

EXAMPLE

The following macro code will display a message on the screen for a fixed 5 seconds:

```
16 screen−poke 10 10 0 "Hello World!"
5000 command−wait
```

Similarly the following macro code will display a message for up to 5 seconds or till the user presses a key:

```
16 screen−poke 10 10 0 "Hello World!"
−5000 command−wait
```

SEE ALSO

ml−write(2), $buffer−input(5).

compare−windows(2)

NAME

compare−windows − Compare buffer windows, ignore whitespace.
compare−windows−exact − Compare buffer windows, with whitespace.

SYNOPSIS

n **compare−windows**
compare−windows−exact

DESCRIPTION

compare−windows compares the textural content of ALL the current windows from their current cursor position. These commands are generally used to locate the next difference in the windows displayed. Returns TRUE if the buffers of the windows do not differ from the current position to the end of the file (inclusive), else returns FALSE setting the cursor of each buffer to the first point of difference.

The default mode of operation ignores white−space, a numeric argument *n* of zero (0) then an exact white−space match is performed.

compare−windows−exact is a macro short cut for *0 compare−windows*, forcing a white space comparison.

SEE ALSO

diff(3), diff−changes(3), gdiff(3).

compile(3)

NAME

compile − Start a compilation process

SYNOPSIS

n **compile** "*compile−command*"

DESCRIPTION

compile gets and executes the compile command using a pipe execution (incremental pipe on UNIX platforms), loading the output into a buffer called "***compile***", with go to error parsing using the command get−next−line(2). The default compile execution is set by variable %compile−com(5), the error parsing is setup using the command add−next−line(2).

Before the compile command is executed save−some−buffers(2) is executed to allow the user to ensure that all relevant buffers are saved. If an argument is given to compile then it is passed on to this command, so if an argument of 0 is given, all buffers are automatically saved.

NOTES

compile is a macro defined in `tools.emf`.

SEE ALSO

add−next−line(2), %compile−com(5), get−next−line(2), save−some−buffers(2), grep(3).

copy−region(2)

NAME

copy−region − Copy a region of the buffer

SYNOPSIS

copy−region (**esc w**)

DESCRIPTION

copy−region copies all the characters between the cursor and the mark set with the set−mark(2) command into the kill buffer (so they can later be yanked elsewhere).

If the last command also entered text into the kill buffer (or the @cl(4) variable is set to one of these commands) the **copy−region** text is appended to the last kill.

USAGE

To copy text from one place to another, using the **copy−region** command, the following operations are performed:

- Move to the beginning of the text you want to copy.
- Set the mark there with the set−mark (**esc−space**) command.
- Move the point (cursor) to the end of the text.
- Use **copy−region** to copy the region you just defined. The text will be saved in the kill buffer. (If you accidentally delete the text use yank (**C−y**) immediately or undo (**C−x u**) to restore the text).
- Move the point to the place you want the text to appear.
- Use the yank (**C−y**) command to copy the text from the kill buffer to the current point.

Repeat the last two steps to insert further copies of the same text.

NOTES

Windowing systems such as X−Windows and Microsoft Windows utilize a global windowing kill buffer allowing data to be moved between windowing applications (*cut buffer* and *clipboard*, respectively). Within these environments MicroEmacs '02 automatically interacts with the windowing systems kill buffer, the last MicroEmacs '02 **copy−region** entry is immediately available for a paste operation into another windowing application.

SEE ALSO

exchange−point−and−mark(2), kill−region(2), set−mark(2), yank(2).

count−words(2)

NAME

count−words − Count the number of words in a region

SYNOPSIS

count−words (**esc C−c**)

DESCRIPTION

count−words Counts the number of words between the set−mark(2) position and the current cursor position. The command also gives statistics on the number of characters and the average characters per word. The output appears on the message line in a format such as:−

```
54 Words, 345 Chars, 8 Lines
```

$result(5) is set to the same output string.

SEE ALSO

$result(5), buffer−info(2), set−mark(2).

create−callback(2)

NAME

create−callback − Create a timer callback

SYNOPSIS

n **create−callback** "*command*"

DESCRIPTION

create−callback creates a timer based callback command. The given *command* is called back in *n* milliseconds time. This can be used by the user to monitor system events (such as incoming mail). The command is called only once, but if the command creates a callback of itself a loop is created.

If a −ve argument *n* is given any pending callback for *command* is cancelled.

EXAMPLE

The following example creates a callback that is invoked every 10 minutes.

```
define−macro Example−callback
    ml−write "It was 10 minutes since you last saw me!"
    600000 create−callback Example−callback
!emacro
Example−callback
```

NOTES

A call−back cannot interrupt while MicroEmacs is active, instead the call−back is delayed until MicroEmacs becomes inactive. MicroEmacs is considered to be inactive when it is waiting for user input, this could be during the execution of another macro. If a command or macro requires no user input then once execution has started, it cannot be interrupted by a call−back macro.

The resolution of the clock is platform dependent, some platforms limit the minimum timer period to 10 milliseconds.

MicroEmacs does not guarantee to service the callbacks within any set time constraints, the resultant callback intervals may be of a slightly different duration than requested.

When a callback macro is executed, the key given by @cck(4) is "`callback`. If the current buffer has a $buffer−input(5) command set, this command will be called instead of the callback command with **@cc** and **@cck** set appropriately. It is the responsibility of the input macro to deal with the callback.

SEE ALSO

$auto−time(5), define−macro(2).

create−frame(2)

NAME

create−frame − Create a new frame

SYNOPSIS

n **create−frame**

DESCRIPTION

create−frame creates a new frame for the current MicroEmacs session. MicroEmacs support the creation of 'internal' multiple frames on all platforms and 'external' frames on windowing platforms (such as Windows and XTerm). An external frame creates a new OS window so both the existing frame and the new frame are visible, whereas an internal frame uses the same OS window or console which means that the existing frame is hidden and the new frame takes its place.

The numeric argument *n* can be used to define which type of frame is to be created. If an argument of `1` is given (the default argument) an external frame will be created, whereas an internal frame will be created if an argument of `0` is given.

NOTES

Internal frames can only be accessed via the next−frame(2) command, external frames can usually be accessed via the OS as well.

MicroEmacs is not multi−threaded in that only one frame can be active at any one time (the complexity of being able to run a command in one frame while editing in another would rapidly lead it away from the 'Micro' status). This means that if a command is left active (such as a search) in one frame and the focus is changed to another the input is 'sent' to the frame with the active command and the message '[NOT FOCUS]' will appear in the message−line of the frame with the OS focus.

create−frame may be useful in macros that rely on a window layout, this is because they can preserve the users current window layout by creating and new internal frame in which to run.

SEE ALSO

delete−frame(2), next−frame(2).

cvs(3)

NAME

cvs − MicroEmacs CVS interface
cvs−add − MicroEmacs CVS interface − add file
cvs−checkout − MicroEmacs CVS interface − checkout files and directories
cvs−commit − MicroEmacs CVS interface − commit changes
cvs−diff − MicroEmacs CVS interface − diff changes
cvs−gdiff − MicroEmacs CVS interface − graphical diff changes
cvs−log − MicroEmacs CVS interface − log changes
cvs−remove − MicroEmacs CVS interface − remove file
cvs−resolve−conflicts − MicroEmacs CVS interface − resolve conflicts
cvs−state − MicroEmacs CVS interface − list state of directory files
cvs−update − MicroEmacs CVS interface − update directory files

SYNOPSIS

cvs

cvs−add
cvs−checkout
cvs−commit
cvs−diff
cvs−gdiff
cvs−log
cvs−remove
cvs−resolve−conflicts
cvs−state
cvs−update

DESCRIPTION

The cvs and sub−commands provide MicroEmacs with an interface to **cvs(1)**. **CVS** is a version control system; using it, you can record the history of your source file modifications. CVS is licensed under the GNU General Public License and is freely available on the Internet, see the documentation provided with CVS for more information on its features and use.

The MicroEmacs **cvs** command opens up a modified file−browser(3) with an additional "`*cvs-console*`" window. The "`*files*`" window includes additional columns showing the CVS state, revision and repository date. The functionality of the file−browser is the same as a non−CVS folder with the exception that additional CVS item controls are located in the mouse context menu (opened by clicking the right mouse button in the `*files*` buffer). This menu item opens another sub−menu providing access to the following items:

Checkout files

Checks out a file or directory from the repository into the current directory. The file or directory is specified by typing the name into a dialog which is opened when this option is selected. This runs the command "`cvs checkout <file>`".

Update files

Updates the currently selected files, files are selected by clicking the left button to the left of the required file name. Multiple files may be selected by 'dragging' a hilight region over the required files. This runs the command "`cvs update <files>`".

Commit files

Commits any changes made to the selected files back to the CVS repository. This runs the command "`cvs commit <files>`".

Diff files

Displays any differences between the selected files and the CVS repository version in the **cvs−console** window. This runs the command "`cvs diff <files>`".

Log files

Displays the CVS logs for the selected files in the **cvs−console** window. This runs the command "`cvs log <files>`".

Status files

Displays the CVS status for each of the selected files in the **cvs−console** window. This runs the command "`cvs status -v <files>`".

Add files

Adds the selected files to the CVS repository. Note this command only performs the local add, a **CVS commit** is required to make the addition permanent. This runs the command "`cvs add <files>`".

Remove files

This command is deliberately not implemented as its far to dangerous! Instead it opens a dialog informing the user to use the **cvs−remove** command instead.

Graphical diff

This command opens a gdiff(3) window showing the differences between the currently selected file and the CVS repository version. Note this command only works with a single file.

Resolve conflicts

This command may be used to resolve merge conflicts created by a *CVS update* operation. The command opens a gdiff(3) window showing the areas of conflict allowing the user to select the correct version and saving the resultant version back to the local file. Note this command only works with a single file.

Clear cvs console

Clears the **cvs−console** buffer.

The **cvs−add** command adds the current buffers file to the repository. Note that this command only performs the local addition, a *CVS commit* is required to make the addition permanent.

The **cvs−checkout** command checks out a file or directory from the repository into the current directory. The user specifies the file on the message line.

The **cvs−commit** command commits any changes made to the currently buffer's file (including additions) to the repository. The user is prompted for a commit log message.

The **cvs−diff** command opens a **cvs−diff** window displaying the differences between the current buffer's local file and repository version. If the current buffer is a directory list it will list all the differences found in all files within the directory.

The **cvs−gdiff** command opens a gdiff(3) window displaying the differences between the current buffer's local file and repository version.

The **cvs−log** command opens a **cvs−log** window displaying the CVS log of the current buffer's file.

The **cvs−remove** command removes the current buffer's file from the repository − PLEASE NOTE THIS CAN LEAD TO LOST DATA!!! This command only performs the local removal; as it deletes the buffer and file the **cvs−commit** command cannot be used to commit the removal to the CVS repository. Instead the main **cvs** file−browser menu or **cvs(1)** itself must be used.

The **cvs−resolve−conflicts** command may be used to resolve any conflicts created by CVS when the current buffer's file is updated. The command opens a gdiff window displaying the areas of conflict, the user may then select the correct version in each case and save the resultant new version over the local file.

The **cvs−state** command opens a **cvs−state** window listing the state of any file in the current directory which is not up−to−date. Note that unlike most cvs sub commands this command executes over all files in the current buffer's file directory.

The **cvs−update** command updates all files in the current directory, the output being reported to a new **cvs−update** window. Note that unlike most cvs sub commands this command executes over all files in the current buffer's file directory.

NOTES

cvs and sub−commands are macros defined in file `cvs.emf`.

By default MicroEmacs's **cvs** commands skip all files ignored by **cvs(1)**. This is configured by the variable **.cvs.filter**, defining this variable to `0` disables this special filtering.

SEE ALSO

file−browser(3).

cygnus(3)

NAME

cygnus − Open a Cygwin BASH window
%cygnus−bin−path − Cygwin BASH directory
%cygnus−hilight − Cygwin shell hilight enable flag
%cygnus−prompt − Cygwin shell prompt

PLATFORM

Windows '95/'98/NT − win32 ONLY

SYNOPSIS

cygnus

%cygnus−bin−path "*path*"
%cygnus−hilight [`0`|`1`]
%cygnus−prompt "*hilightString*"

DESCRIPTION

cygnus creates an interactive BASH shell window within a MicroEmacs buffer window, providing a UNIX command line facility within the Microsoft Windows environment. This is a preferable environment to the MS−DOS shell and is certainly far more comfortable for those people familiar with UNIX.

Within the window BASH commands may be entered and executed, the results are shown in the window. Within the context of the BASH shell window then directory naming conforms to the **cygwin** standard conventions (as opposed to the Microsoft directory naming).

On running **cygnus** a new buffer is created called `*cygnus*` which contains the shell. Executing the command again creates a new shell window called `*cygnus1*`, and so on. If a cygwin window is killed off then the available window is used next time the command is run.

Additional controls are available within the shell window to control the editors interaction with the window. The operating mode is shown as a digit on the buffer mode line, this should typically show "3", which corresponds to *F3*. The operating modes are mapped to keys as follows:−

F2

Locks the window and allows local editing to be performed. All commands entered into the window are interpreted by the editors. **F2** mode is typically entered to cut and paste from the window, search

for text strings etc. In mode 2, a **2** is shown on the mode line.

F3

The normal operating mode, text typed into the window is presented to the shell window. Translation of MicroEmacs commands (i.e. beginning−of−word) are translated and passed to the shell. For interactive use this is the default mode. In mode 3, a **3** is shown on the mode line.

F4

All input is passed to the shell, no MicroEmacs commands are interpreted and keys are passed straight to the shell window. This mode is used where none of the keys to be entered are to be interpreted by MicroEmacs. Note that you have to un−toggle the F4 mode before you can swap buffers as this effectively locks the editor into the window.

F5

Clears the buffer contents. This simply erases all of the historical information in the buffer. The operation of the shell is unaffected.

To exit the shell then end the shell session using "`exit`" or "`C−d`" as normal and then close the buffer. A short cut "`C−c C−k`" is available to kill off the pipe. However, it is not recommended that this method is used as it effectively performs a hard kill of the buffer and attached process

%cygnus−bin−path is a user defined variable that defines the file system location of the *cygwin* directory. This variable MUST be defined within the user start up script in order for the **cygnus** command to start the shell. With a default installation of *cygwin* then the settings are typically defined as:−

Release B19

```
set−variable %cygnus−bin−path "C:/Cygnus/B19/h−i386~1/bin"
```

Release B20

```
set−variable %cygnus−bin−path "c:/cygnus/cygwin−b20/H−i586−cygwin32/bin"
```

%cygnus−hilight is a boolean flag which controls how the cygnus command shell window is hilighted. This value MUST be defined within the user start up script prior to executing cygnus if hilighting is to be enabled; by default hilighting is disabled. A value of 1 enables shell hilighting i.e.

```
set−variable %cygnus−hilight 1
```

%cygnus−prompt is an optional variable that is used in conjunction with **%cygnus−hilight**, it defines the hilighting string identifying the prompt. This allows the prompt to be rendered with a different color. The default prompt is `bash−2.01$` and may be hilighted using a definition:−

```
set−variable %cygnus−prompt "bash−2.01$"
```

The user typically overrides the prompt definition within the BASH startup file, a more appropriate definition of the prompt may be:−

```
        set-variable %cygnus-prompt "^[a-z]*@[^>]*>"
```

NOTES

The **cygnus** command uses the ipipe−shell−command(2) to manage the pipe between the editor and the **bash** shell. The window is controlled by the macro file `hkcygnus.emf` which controls the interaction with the shell.

The macro **cygnus** in `hkcygnus.emf` defines the parameter setup to connect to the cygwin bash shell (Version 19), installed in the default location `c:/cygnus`. If your installation of cygnus is in a different location then correct the macro to match your install location, preferably correct by creating a *mycygnus.emf* file in your user directory simply containing a re−defined **cygnus** macro.

If you have exported some of the cygwin environment variables in your `autoexec.bat` then you will have to figure out for yourself what variables macro *cygnus* needs to export − the current configuration is for a vanilla install.

The **bash** shell is executed with options *i*, for interactive shell and *m* to enable job control.

TESTED CONFIGURATIONS

This configuration has only been tested on a Windows '98 installation, whether this works on NT and Windows '95 (OEM SR2) is unknown.

We have only been running "make" operations in the shell and do not know how the likes of "more", "man" or anything other terminal interaction works.

Tested Configurations

Windows '98 (Pentium 120MHz/Pentium Pro 200MHz/Cyrix 300MHz/Pentium II 450MHz)

> cygwin version B19.3 − this is the original "cygwin" distribution + the latest "coolview.tar.gz" patch.
> cygwin version B20 − the latest cygwin distribution.

BUGS

Break Key

A break in a bash shell is `C−c`, the macros define the key `C−c C−c` to perform the break. This sequence is sent to the process but is not enacted by the shell. This is a property of the Bash shell rather than MicroEmacs.

Slow Response

If you are getting a very slow response from the bash shell then check the directory where *bash* was started. Sometimes there are problems if the shell is started in "`c:/`" (which is typically "`/`") then the *bash* shell is very unresponsive and tends to '*ignore me*' for periods of time. If it is started in another location, i.e. "*c:/temp*" directory, then this problem does not occur.

You can see the start−up location in the top of the buffer when the shell is started.

Prompt at top of buffer

Very, very occasionally the ishell sticks at the top of the buffer with only a couple of lines showing. A swap of the buffers or a quick window resize sorts out the problem. A fix for this problem has been applied but still may occasionally occur.

WinOldAp

Winoldap is created by the Microsoft environment whenever a BASH shell is created. On occasions where processes have terminated badly the user may be prompted to kill these off; this is the normal behaviour of windows. It is strongly advised that all of the BASH processes are killed from within the Bash shell itself and the shell is always exited correctly (i.e. `exit`) before leaving the editor. The Windows operating system for '95/'98 is not particularly resilient to erroneous processes (for those of us familiar with UNIX) and can bring the whole system down. I believe that NT does not suffer from these problems (much).

Locked Input

There are occasions after killing a process the editor appears to lock up. This is typically a case that the old application has not shut down correctly. Kill off the erroneous task (`Alt−Ctrl−Del` − *End Task*) then bring the editor under control using a few `C−g` abort−command(2) sequences. **SEE ALSO**

ipipe−shell−command(2), ishell(3).
Cygnus Win32 home sites **www.cygnus.com** and **www.cygnus.co.uk**

define−help(2)

NAME

define−help − Define help information

SYNOPSIS

define−help "*string*" ["*section*"]

Free form text

!ehelp

DESCRIPTION

define−help provides a mechanism to define help information for commands and variables within macro files. The command allows user defined macros to be documented with help information that is accessible from the command line via the normal help commands such as help−item(2).

The help information is typically embedded in the macro file with the macro command that it is documenting. When the macro file is loaded then the help information is loaded and integrated into the existing help database.

string is the name of the item that is being defined, *section* defines what section the item belongs to. Following is a table of standard MicroEmacs '02 sections:

`1`	MicroEmacs command line arguments.
`2`	Built−in commands.
`2m`	MicroEmacs buffer modes.
`3`	Macro commands.
`4`	Macro language commands.
`5`	MicroEmacs variables.
`8`	MicroEmacs file formats.

When *section* is omitted is defaults to the general section which is usually used for the higher level help pages.

Text following the **define−help** line contains the help information, this is a free form text area that is reproduced when the help information is requested. The end of the text area is delimited by a **!ehelp** construct. The help text is usually displayed using a special hilighting scheme to control the colors and hyper−text links to other help pages. As a result the text may contain escape ('`^[`') key sequences, see ehf(8) for more information on the format.

EXAMPLE

The following example is a define−help representation for the paragraph−to−line(3) macro.

```
define-help "paragraph-to-line" "3"

^[cENAME^[cA

     paragraph-to-line - Convert a paragraph to a single line
$a


^[cESYNOPSIS^[cA

     n paragraph-to-line


^[cEDESCRIPTION^[cA

     paragraph-to-line  reduces  each of the  next n  paragraphs  of text to a
     single  line.  This  is  used to  prepare  a  document  to go into a word
     processor environment where end of line marks represent paragraph marks.


^[cENOTES^[cA

     This command is a macro defined in format.emf.


^[cESEE ALSO^[cA

     ^[ls^[lm^[cGfill-paragraph(2)^[cA^[le.

!emacro
```

SEE ALSO

ehf(8), help−item(2), define−macro(2), help−command(2), help−variable(2).

define−macro(2)

NAME

define−macro − Define a new macro

SYNOPSIS

n **define−macro** *macro−name*

Macro body
!emacro DESCRIPTION

define−macro starts the definition of an macro named *macro−name*, only used within macro files or buffers. After the above header line, the body of the macro is added, one command or expression on a line. The macro definition is completed by the !emacro directive.

The numeric argument *n*, specified as zero, defines the macro as private such that it does not appear on a command completion list. A zero argument is generally used on helper macro's that form part of a larger macro. If the argument is omitted, or non−zero, then the macro appears in the command completion list.

See execute−file(2) for a complete definition and examples of the MicroEmacs '02 macro language.

Once the macro has been defined, it becomes indistinguishable from a standard MicroEmacs '02 command, i.e. execute−named−command(2) (`esc x`) can be used to execute the macro and global−bind−key(2) can be used to globally bind the command to a key combination.

There are no restrictions on the number of macros that may be defined, provided that the name space is managed properly. Consideration must be given as to when any additional macros that are created are loaded into MicroEmacs '02. We usually like start−up to be rapid and macros are loaded as and when requested by the user, or by the buffer hooks as new files are loaded (see add−file−hook(2) and define−macro−file(2)).

User defined macros may be documented with on−line help by including a define−help(2) construct within the macro file.

EXAMPLE

The following are two standard macros provided with MicroEmacs '02. The first is a macro called **clean**, this strips trailing white space from the ends of lines in a file and removes blank lines from the end of the file.

```
define−macro clean
    ;
    ; Prepare to clean up file.
    ; Remember line & magic mode
    set-variable #l0 $window-line
    set-variable #l1 &not &bmod magic
    !if #l1
        1 buffer-mode "magic"
    !endif
    ;
    ; Get rid of trailing white space on EOL
    beginning-of-buffer
    replace-string "[\t ]+$" "\\0"
    beginning-of-buffer
    replace-string "[ ]+\t" "\t"
    ;
    ; Strip trailing blank lines.
    end-of-buffer
    backward-line
    !while &and &gre $window-line 1 &sequal @wc "\n"
        kill-line
        backward-line
    !done
    ;
    ; Clean up - restore buffer modes etc.
    ; Move back to starting line & restore original magic mode
    !force goto-line #l0
    !if #l1
        -1 buffer-mode "magic"
    !endif
    screen-update
    ml-write "Cleaned up file."
!emacro
```

The second example converts all of the `<tab>` characters in the file to their `<SPACE>` character equivalent.

```
;
; tabs-to-spaces.
; Convert all of the tabs to spaces.
define-macro tabs-to-spaces
    ; Remember line
    set-variable #l0 $window-line
    beginning-of-buffer
    !force search-forward "\t"
    !while $status
        set-variable #l1 $window-acol
        backward-delete-char
        &sub #l1 $window-acol insert-space
        !force search-forward "\t"
    !done
    goto-line #l0
    screen-update
    ml-write "[Converted tabs]"
!emacro
```

Both of these commands are available from the command line, they are indistinguishable from the built in commands.

Macros may also be nested, as shown in the next example, this macro contains a **define−macro** within itself, when executed the macro creates another macro dynamically − dynamic macros are generally given a prefix of **%** and are highlighted differently in describe−bindings(2).

The following example is taken from the alarm(3) macro, executing **alarm** the user is prompted for a message, and the time interval before the alarm expires in hours and minutes. It then creates a new macro with a callback so that the new macro will be called at the correct time.

```
!if &seq %alarm-numb "ERROR"
    set-variable %alarm-numb 0
    set-variable %osd-alarm &pinc %osd 1
!endif

define-macro alarm
    set-variable %alarm-numb &add %alarm-numb 1
    set-variable #l0 &cat "%alarm-" %alarm-numb
    !force set-variable #l2 @3
    !if &not $status
        set-variable &ind #l0 @ml "Message"
        set-variable #l1 @ml "Hours"
        set-variable #l2 @ml "Minutes"
    !else
        set-variable &ind #l0 @1
        set-variable #l1 @2
    !endif
    set-variable #l1 &mul 60000 &add &mul 60 #l1 #l2
    define-macro #l0
        !bell
        set-variable #l0 &add &len &ind @0 10
        osd %osd-alarm 0 "bat" 9 3
        osd %osd-alarm 1 ""
        osd %osd-alarm 2 "c" "ALARM"
        osd %osd-alarm 3 ""
        osd %osd-alarm 4 "" &ind @0
        osd %osd-alarm 5 ""
        osd %osd-alarm 6 "Bcf" " OK " f void
        %osd-alarm osd
    !emacro
    #l1 create-callback #l0
!emacro
```

SEE ALSO

Refer to !return(4) and !abort(4) for details macro termination.

!emacro(4), add−file−hook(2), define−macro−file(2), define−help(2), describe−bindings(2), execute−file(2), execute−named−command(2), global−bind−key(2), insert−macro(2), start−kbd−macro(2).

define−macro−file(2)

NAME

define−macro−file − Define macro file location

SYNOPSIS

define−macro−file "*file−name*" ["*macro−name*" "*macro2−name*" ...]

DESCRIPTION

Macros are loaded as late as possible using an on−demand mechanism, this speeds up the load time of MicroEmacs '02, it also keeps the startup file clean since macros are not defined within the start−up file. Only when the user first executes a macro defined via **define−macro−file** is the file loaded, the macro becomes defined and is executed. Subsequent calls to the macro will not reload the file as the macro will now be fully defined.

define−macro−file binds macros (*macro−name* ...) to a file name (*file−name*). This operation informs MicroEmacs '02 which file should be loaded when *macro−name* is first executed. The *macro−name* arguments may be omitted if the file contains only one exported macro which has the same name as *file−name*.

Alternatively the macro file may contain many macros all of which can be defined by a single call to **define−macro−file**, listing all macros on the same line after the *file−name*. If a *macro−name* is given then the default macro *file−name* is not created, if a macro of that name does exist it must be added to the *macro−name* list.

EXAMPLE

The following definitions are found in the `me.emf` start−up file:−

```
0 define−macro−file utils ascii−time regex−forward regex−backward
define−macro−file format clean sort−lines−ignore−case tabs−to−spaces ...
define−macro−file cvs cvs cvs−state cvs−update cvs−commit cvs−log ...
define−macro−file abbrev expand−abbrev−handle expand−iso−accents ...
define−macro−file misc symbol check−line−length alarm time
define−macro−file search replace−all−string query−replace−all−string
define−macro−file tools compile grep rgrep which diff diff−changes
define−macro−file hkdirlst file−browser file−browser−close
define−macro−file comment comment−line uncomment−line comment−to−end−of−line
define−macro−file spell spell−word spell−buffer spell−edit−word find−word
define−macro−file games Metris Patience Triangle Mahjongg Match−It
define−macro−file buffstp buffer−setup buffer−help buffer−tool
define−macro−file fattrib file−attrib
define−macro−file osd osd−main
define−macro−file gdiff
```

```
    define-macro-file calc
    define-macro-file draw
```

Hilighting a number of entries as examples; macro file **calc** is defined with no macro definition, the macro is assumed to be **calc**. The file **tools.emf** contains multiple macros **compile**, **grep**, **diff** and **diff−changes**; all can be defined by a single **define−macro−file** entry.

NOTES

- Macro files are searched for in the current directory and along the $search−path(5).
- The macro file is not loaded unless a binding has been defined using **define−macro−file**.
- Any other macros that exist in the *file−name* macro file become defined when the entry point macro is loaded and are available for use. This is potentially useful as a single *entry* macro may be defined using **define−macro−file**, when invoked other helper macros become available.

SEE ALSO

add−file−hook(2), define−macro(2), $search−path(5), start−up(3).

delete−blank−lines(2)

NAME

delete−blank−lines − Delete blank lines about cursor

SYNOPSIS

delete−blank−lines (**C−x C−o**)

DESCRIPTION

delete−blank−lines deletes all the blank lines before and after the current cursor position. Note that the deleted lines are not added to a kill buffer.

SEE ALSO

delete−indentation(3), clean(3), kill−line(2).

delete−buffer(2)

NAME

delete−buffer − Delete a buffer

SYNOPSIS

n **delete−buffer** "*buffer−name*" (**C−x k**)

DESCRIPTION

delete−buffer disposes of buffer *buffer−name* in the editor and reclaim the memory. This does not delete the file that the buffer was read from.

If the buffer has been edited and its name does not start with a '*****' then the user is prompted as to whether the changes should be discarded. Also if the buffer has an active process running in it then confirmation is sort from the user before the process is killed.

The argument *n* can be used to change the default behavior of delete−buffer described above, *n* is a bit based flag where:−

0x01

Enables loss of work checks (default). These include a check that the buffer has not been modified, if so the user is prompted. Also if a process is running then user must confirm that the process can be killed. If this flag is not supplied then the buffer is killed without any user prompts (useful in macros). **SEE ALSO**

next−buffer(2).

delete−dictionary(2)

NAME

delete−dictionary − Remove a spelling dictionary from memory

SYNOPSIS

n **delete−dictionary** ["*dictionary*"]

DESCRIPTION

delete−dictionary removes the given *dictionary* from memory, where *n* is a bitwise flag determining the removal mode, defined as follows:−

0x01

Prompt the user before loosing any changes (except to the ignore dictionary).

0x02

Delete all the dictionaries other than the ignore dictionary.

0x04

Delete the ignore dictionary.

If the argument does not have bit 0x02 or 0x04 set, which specify the dictionaries to be deleted, the user is prompted for the "*dictionary*". The default argument is 1.

NOTES

The ignore dictionary is a temporary dictionary that exists in memory for duration of the MicroEmacs session; the dictionary holds words that have been ignored during any previous spell checks (see spell(2)). All of the words that have been ignored may be discarded with:−

```
4 delete−dictionary
```

i.e. **esc 4 esc x delete−dictionary**.

SEE ALSO

spell−buffer(3), add−dictionary(2), save−dictionary(2), spell(2).

delete−frame(2)

NAME

delete−frame − Delete the current frame

SYNOPSIS

n **delete−frame**

DESCRIPTION

delete−frame deletes the current frame.

SEE ALSO

create−frame(2), next−frame(2).

delete−indentation(3)

NAME

delete−indentation − Join 2 lines deleting white spaces

SYNOPSIS

n **delete−indentation**

DESCRIPTION

delete−indentation deletes all white characters between the beginning of the current line and the end of the previous line, including the line−feed. If the current line is not empty then a space is inserted to divide the two lines now joined.

If a positive argument *n* is given then the process is repeated *n* times. Note that the deleted characters are not added to a kill buffer.

NOTES

delete−indentation is a macro defined in `format.emf`.

SEE ALSO

delete−blank−lines(2), clean(3), kill−line(2).

delete−window(2)

NAME

delete−window − Delete the current window
delete−other−windows − Delete other windows

SYNOPSIS

n **delete−window** (**C−x 0**)
n **delete−other−windows** (**C−x 1**)

DESCRIPTION

delete−window attempts to delete the current window (remove window from the screen), retrieving the lines for use in the window adjacent to it. The command fails if there is no other window or if the current window is protected from deletion (see $window−flags(5)). The deletion protection can be overridden by giving the command a numerical argument *n* of 2.

The window deletion policy is determined by the formation of the windows displayed on the screen. The bias is for the *previous* window (above) the current window to be merged when split vertically, and for the left window to be merged when split horizontally.

delete−other−windows deletes all of the other windows, the current window becomes the only window, using the entire available screen area. Windows can be protected from deletion by using the $window−flags variable, giving the command a numerical argument *n* of 2 overrides this protection.

SEE ALSO

set−position(2), grow−window−vertically(2), resize−window−vertically(2), split−window−horizontally(2), split−window−vertically(2), $window−flags(5).

delete−registry(2)

NAME

delete−registry − Delete a registry tree

SYNOPSIS

delete−registry "*root*"

DESCRIPTION

delete−registry deletes a registry node *root* from the registry, any children belonging to the node are also deleted.

DIAGNOSTICS

delete−registry fails if *root* does not exist.

SEE ALSO

get−registry(2), list−registry(2), read−registry(2), set−registry(2), erf(8).

delete−some−buffers(2)

NAME

delete−some−buffers − Delete buffers with query

SYNOPSIS

n **delete−some−buffers**

DESCRIPTION

delete−some−buffers cycles through all visible buffers (buffers without mode hide(2m) set) and prompts the user [**y**/**n**] as to whether the buffer should be deleted. A **y** response deletes the buffer, a **n** response retains the buffer.

If a **y** response is given, the buffer has been edited, and its name does not start with a '*****' then the user is prompted as to whether the changes should be discarded. Also if the buffer has an active process running in it then confirmation is sort from the user before the process is killed.

The argument *n* can be used to change the default behavior of delete−some−buffers described above, *n* is a bit based flag where:−

0x01

Enables all checks (default). These include the initial y/n prompt on each buffer, the buffer has not been modified check, if so the user is prompted. Also if a process is running then user must confirm that the process can be killed. If this flag is not supplied then all visible buffers are killed without any user prompts (useful in macros). **SEE ALSO**

delete−buffer(2), next−buffer(2), hide(2m).

describe−bindings(2)

NAME

describe−bindings − Show current command/key binding

SYNOPSIS

describe−bindings (**C−h b**)

DESCRIPTION

describe−bindings pops up a window with a list of all the named commands, and the keys currently bound to them. Each entry is formatted as:

keyCode . . . . command

describe−bindings is buffer context sensitive and shows the bindings for the currently active buffer (i.e. the buffer that is active when the command is invoked). The resultant command list is divided into three sections as follows:

Buffer Bindings

The bindings for the active buffer when **describe−bindings** was invoked. These are the buffer bindings set by buffer−bind−key(2).

Ml Bindings

The message line bindings as set by ml−bind−key(2).

Global Bindings

Global binding of keys as set by global−bind−key(2). **EXAMPLE**

The following is an example of the displayed output from **describe−bindings**. This was invoked while editing buffer **m2fun038.2** which is the **Nroff** file for this manual page; the local bindings for the buffer are all Nroff related.

```
Buffer [m2cmd038.2] bindings:

    "C−c C−s" ..................... nroff−size
    "C−c C−r" ..................... nroff−roman
    "C−c C−b" ..................... nroff−bold
    "C−c C−i" ..................... nroff−italic
    "C−c C−c" ..................... nroff−mono
    "C−c C−o" ..................... nroff−para
    "esc o" ....................... nroff-para
    "esc q" ....................... nroff-para
    "C-c b" ....................... nroff-bold-block
    "C-c i" ....................... nroff-italic-block
    "C-c C-h" ..................... nroff-swap-hilight
    "C-c &" ....................... nroff-add-padding
    "C-x &" ....................... nroff-remove-padding
    "C-c C-p" ..................... nroff-prev
    "C-mouse-drop-1" .............. nroff-tag

Ml bindings:

    "esc esc" ..................... tab

Global bindings:

    "C-a" ......................... beginning-of-line
    "C-b" ......................... backward-char
    "C-c" ......................... 4 prefix
    "C-d" ......................... forward-delete-char
    "C-e" ......................... end-of-line
    "C-f" ......................... forward-char
    "C-g" ......................... abort-command
    "C-h" ......................... 3 prefix
    "C-i" ......................... insert-tab
    "C-k" ......................... kill-line
    "C-l" ......................... recenter
    "C-m" ......................... newline
    "C-n" ......................... forward-line
    "C-o" ......................... insert-newline
    "C-p" ......................... backward-line
    "C-q" ......................... quote-char
    "C-r" ......................... isearch-backward
    "C-s" ......................... isearch-forward
    "C-t" ......................... transpose-chars
    "C-u" ......................... universal-argument
    "C-v" ......................... scroll-down
    "C-w" ......................... kill-region
    "C-x" ......................... 2 prefix
    "C-y" ......................... yank
    "C-z" ......................... scroll-up
    "C-_" ......................... undo
    "A-e" ......................... file-browser
    "A-r" ......................... replace-all-string
    "esc C-c" ..................... count-words
    "esc C-f" ..................... goto-matching-fence
    "esc C-g" ..................... abort-command
    "esc C-i" ..................... goto-matching-fence
    "esc C-k" ..................... global-unbind-key
    "esc C-n" ..................... change-buffer-name
    "esc C-r" ..................... query-replace-string
    "esc C-v" ..................... scroll-next-window-down
    "esc C-w" ..................... kill-paragraph
    "esc C-z" ..................... scroll-next-window-up
    "esc space" ................... set-mark
    "esc !" ....................... pipe-shell-command
    "esc $" ....................... spell-word
    "esc ." ....................... set-mark
    "esc /" ....................... execute-file
```

```
"esc <" ....................... beginning-of-buffer
"esc >" ....................... end-of-buffer
"esc ?" ....................... help
"esc @" ....................... pipe-shell-command
"esc [" ....................... backward-paragraph
"esc \\" ...................... ipipe-shell-command
"esc ]" ....................... forward-paragraph
"esc ^" ....................... delete-indentation
"esc b" ....................... backward-word
"esc c" ....................... compile
"esc d" ....................... forward-kill-word
"esc e" ....................... set-encryption-key
"esc f" ....................... forward-word
"esc g" ....................... goto-line
"esc i" ....................... tab
"esc k" ....................... global-bind-key
"esc l" ....................... lower-case-word
"esc m" ....................... global-mode
"esc n" ....................... forward-paragraph
"esc o" ....................... fill-paragraph
"esc p" ....................... backward-paragraph
"esc q" ....................... fill-paragraph
"esc r" ....................... replace-string
"esc t" ....................... find-tag
"esc u" ....................... upper-case-word
"esc v" ....................... scroll-up
"esc w" ....................... copy-region
"esc x" ....................... execute-named-command
"esc y" ....................... reyank
"esc z" ....................... quick-exit
"esc ~" ....................... -30 buffer-mode
"esc A-r" ..................... query-replace-all-string
"C-x C-a" ..................... set-alpha-mark
"C-x C-b" ..................... list-buffers
"C-x C-c" ..................... save-buffers-exit-emacs
"C-x C-d" ..................... change-directory
"C-x C-e" ..................... execute-kbd-macro
"C-x C-f" ..................... find-file
"C-x C-g" ..................... abort-command
"C-x C-h" ..................... hunt-backward
"C-x C-i" ..................... insert-file
"C-x C-l" ..................... lower-case-region
"C-x C-o" ..................... delete-blank-lines
"C-x C-q" ..................... rcs-file
"C-x C-r" ..................... read-file
"C-x C-s" ..................... save-buffer
"C-x C-t" ..................... transpose-lines
"C-x C-u" ..................... upper-case-region
"C-x C-v" ..................... view-file
"C-x C-w" ..................... write-buffer
"C-x C-x" ..................... exchange-point-and-mark
"C-x C-y" ..................... insert-file-name
"C-x C-z" ..................... shrink-window-vertically
"C-x #" ....................... filter-buffer
"C-x (" ....................... start-kbd-macro
"C-x )" ....................... end-kbd-macro
"C-x /" ....................... isearch-forward
"C-x 0" ....................... delete-window
"C-x 1" ....................... delete-other-windows
```

```
"C-x 2" ....................... split-window-vertically
"C-x 3" ....................... next-window-find-buffer
"C-x 4" ....................... next-window-find-file
"C-x 5" ....................... split-window-horizontally
"C-x 9" ....................... find-bfile
"C-x <" ....................... scroll-left
"C-x =" ....................... buffer-info
"C-x >" ....................... scroll-right
"C-x ?" ....................... describe-key
"C-x @" ....................... pipe-shell-command
"C-x [" ....................... scroll-up
"C-x ]" ....................... scroll-down
"C-x ^" ....................... grow-window-vertically
"C-x `" ....................... get-next-line
"C-x a" ....................... goto-alpha-mark
"C-x b" ....................... find-buffer
"C-x c" ....................... shell
"C-x e" ....................... execute-kbd-macro
"C-x h" ....................... hunt-forward
"C-x k" ....................... delete-buffer
"C-x m" ....................... buffer-mode
"C-x n" ....................... change-file-name
"C-x o" ....................... next-window
"C-x p" ....................... previous-window
"C-x q" ....................... kbd-macro-query
"C-x r" ....................... search-backward
"C-x s" ....................... search-forward
"C-x u" ....................... undo
"C-x v" ....................... set-variable
"C-x w" ....................... resize-window-vertically
"C-x x" ....................... next-buffer
"C-x z" ....................... grow-window-vertically
"C-x {" ....................... shrink-window-horizontally
"C-x }" ....................... grow-window-horizontally
"C-h C-c" ..................... help-command
"C-h C-i" ..................... help-item
"C-h C-v" ..................... help-variable
"C-h a" ....................... command-apropos
"C-h b" ....................... describe-bindings
"C-h c" ....................... list-commands
"C-h d" ....................... describe-variable
"C-h k" ....................... describe-key
"C-h v" ....................... list-variables
"backspace" ................... backward-delete-char
"delete" ...................... forward-delete-char
"down" ........................ forward-line
"end" ......................... end-of-buffer
"esc" ......................... 1 prefix
"f1" .......................... menu
"home" ........................ beginning-of-buffer
"insert" ...................... 141 buffer-mode
"left" ........................ backward-char
"mouse-drop-1" ................ mouse-drop-left
"mouse-drop-2" ................ yank
"mouse-drop-3" ................ menu
"mouse-pick-1" ................ mouse-pick-left
"mouse-pick-2" ................ void
"mouse-pick-3" ................ void
"page-down" ................... scroll-down
```

```
"page-up" ..................... scroll-up
"redraw" ...................... screen-update
"return" ...................... newline
"right" ....................... forward-char
"tab" ......................... tab
"up" .......................... backward-line
"S-backspace" ................. backward-delete-char
"S-delete" .................... forward-delete-char
"S-tab" ....................... backward-delete-tab
"C-down" ...................... 5 forward-line
"C-left" ...................... backward-word
"C-mouse-drop-1" .............. mouse-control-drop-left
"C-mouse-pick-1" .............. set-cursor-to-mouse
"C-page-down" ................. scroll-next-window-down
"C-page-up" ................... scroll-next-window-up
"C-right" ..................... forward-word
"C-up" ........................ 5 backward-line
"A-down" ...................... 1 scroll-down
"A-left" ...................... 1 scroll-left
"A-right" ..................... 1 scroll-right
"A-up" ........................ 1 scroll-up
"esc backspace" ............... backward-kill-word
"esc esc" ..................... expand-abbrev
"C-c g" ....................... grep
```

Note that both internal commands and macro commands are shown in the list.

SEE ALSO

buffer−bind−key(2), command−apropos(2), describe−key(2), describe−variable(2), global−bind−key(2), list−commands(2), ml−bind−key(2).

describe−key(2)

NAME

describe−key − Report keyboard key name and binding

SYNOPSIS

describe−key (**C−x ?**)

DESCRIPTION

describe−key allows a key to be typed and it will report the name of the command bound to that key (if any) and the internal key−code. This command is useful when trying to locate the identity of keyboard keys for binding.

NOTES

describe−key is also bound to **C−h k**.

SEE ALSO

command−apropos(2), global−bind−key(2), describe−bindings(2), describe−variable(2).

describe−variable(2)

NAME

describe−variable − Describe current setting of a variable

SYNOPSIS

describe−variable (**C−h v**)

DESCRIPTION

describe−variable describes the current setting of the given variable (**%**, **:** and **$** variables), returning `ERROR` if the variable is undefined. If a `$` variable is not found then it is tested for an environment variable, i.e.

```
describe−variable $PATH
```

returns your environment `$PATH` setting. This is the easiest and best way of determining the current platform from within a Macro file.

The returned value of any undefined variable is the string `ERROR`.

NOTES

Completion is enabled on the command line for variable names.

SEE ALSO

describe−key(2), help−variable(2), set−variable(2).

describe−word(3)

NAME

describe−word − Display a dictionary definition of a word

SYNOPSIS

describe−word "*word*"

DESCRIPTION

describe−word can be used to interface to an external dictionary to get a definition of a given word. The interface has two modes of interface, the first simply launches an external program which provides the definition in its own user interface, e.g. MS Bookshelf. The second interface launches an external program which prints out the definition to `stdout`, MicroEmacs can then pull out the definition and display it in **describe−word**'s own GUI.

When executed **describe−word** will use the current word under the cursor as the initial *word* or will prompt the user if the cursor is not in a word.

When **describe−word**'s dialog is used the information presented is defined as follows:

Word

The word being defined, the entry can be edited and the new word will be automatically looked−up when the edit is completed.

Insert

The effect of this button is dependent on where describe−word was executed. If executed from the **Meaning** button within the spell checker the Word entry is changed to the current word. When executed outside the spell checker the definition of the current word is inserted into the current buffer.

Exit

Closes the dialog.

Main definition box

Displays the definition of the current word. The user can select a new word to describe by clicking the left mouse button on any word within the current definition. **NOTES**

describe−word is a macro implemented in `word.emf`.

Due to the size and availability of dictionaries etc. MicroEmacs is released without describe−word set up, the user must setup it up.

describe−word must be setup for each required language as follows:

1)

A command−line interface to a dictionary of the required language must be found. This could simply be a text file containing one word definition per line and using **grep(1)** as the command−line interface. In this example the text file could take the following form:

```
A () The first letter of the English...
Aam (n.) A Dutch and German measure of liquids...
Aardvark (n.) An edentate mammal...
.
.
```

The **grep** command−line interface required to look−up the word "`aardvark`" would be:

```
grep −i "^aardvark (" words.txt
```

The output produced from this will be the single line giving the required definition. A second common interface would be executing an external dictionary program typically using a command−line option to specify the word to define, e.g.:

```
mydict −d "aardvark"
```

2)

The MicroEmacs language name must be found, this can be done by first using user−setup(3) or spell−buffer(3) to ensure that the current language is set the the require one and then running **describe−word**. The command will probably fail, but before it does it will set the variable `.describe-word.lang`, use the command describe−variable(2) to get the value of this variable, this value is the internal language name. For example, when the current language is **American** or **American (Ext)** the language name is `american`.

3)

To execute the command−line interface the variable `.describe-word.`<*language*>`−command` must be set to the command−line required to obtain a word definition with the string "`%s`" used in place of the word and "`%%`" using in place of a single "`%`". For the first example in **(1)** above the following would be required:

```
set−variable .describe−word.american−command ...
        ... "grep −i \"^%s (\" /tmp/words.txt"
```

For the second example:

```
set−variable .describe−word.american−command "mydict −d \"%s (\""
```

4)

Only required for the second mode, for use with **describe−word**'s own GUI, the setting of another variable is required, the presence of this variable determines which mode is to be used.

> The variable `.describe−word.`<*language*>`−search` must be set to a regex search pattern which will match the required definition(s) in the command out put, the first group ("`\(...\)`") must enclose the required definition, again "`%s`" can be used in place of the word and "`%%`" for a single "`%`". **describe−word** simply uses regex−forward(3) repeatedly to find all definitions of the current word, it then uses the value of the variable @s1(4) to get the individual definitions. For example for the first example the following is required:
>
> ```
> set−variable .describe−word.american−search  "^\(%s (.*\)\n"
> ```
>
> Note that the word being defined should be kept in the definition if possible as the spell rules are used to look−up base words when a derivitive of a word is not found, therefore the word being defined may not be clear (e.g. *deactivate* can be derived from *activate* but their meanings are very different). Also long text lines are automatically wrapped by the GUI.

The required variables should be added to the user setup file.

SEE ALSO

spell−buffer(3).

diff(3)

NAME

diff − Difference files or directories
diff−changes − Find the differences from a previous edit session
%diff−com − Diff command line

SYNOPSIS

diff "*oldFile*" "*newFile*"
diff−changes
`%diff-com` "*string*"; Default is "`diff`"

DESCRIPTION

diff executes the **diff(1)** command with the command line set by the %diff−com(5) variable and the user supplied *oldFile* and *newFile*. The output of the command is piped into the ***diff*** buffer and is hilighted to show the changes (GNU diff only).

Your version of **diff(1)** will determine whether it is possible to difference directories.

diff−changes is a simple macro that differences the current buffer and the last backup of the associated file. It is a quick way to determine what has been modified recently. This macro only works if a backup file exists.

%diff−com is the command line that is used to execute a **diff(1)** system command.

For GNU diff then the following command line setting is recommended:−

```
diff --context --minimal --ignore-space-change \
    --report-identical-files --recursive
```

which should be defined in your personal user configuration. This is the default for Linux.

NOTES

diff and **dif−changes** are macros defined in `tools.emf`.

diff(1) must be executable on the system before diff or diff−changes can function.

diff(1) is a standard utility on UNIX systems. For Windows 95/NT a version of GNU **diff** may be found at:

<ftp.winsite.com/ftp/pub/pc/winnt/misc/gnudiff.zip>

For MS−DOS users, a DJGPP port of **diff** is also available on the net. A commercial version of **diff** is also available from MKS.

SEE ALSO

compare−windows(2), compile(3), gdiff(3), grep(3), %grep−com(5).

directory−tree(2)

NAME

directory−tree − Draw the file directory tree

SYNOPSIS

n **directory−tree** ["*directory*"]

DESCRIPTION

directory−tree creates or manipulates a view of the file systems directory structure. The command is quite complex to use directly so is largely used but macros such as file−browser(3).

The argument *n* is a bit based flag which is used to control the command, where the bits have the following meaning:−

0x01

If set, the focal directory of the command is set by the given "*directory*" argument. Otherwise the argument is not required and the command must be executed within the "**directory**" buffer; the current line sets the focal directory.

0x02

Specifies that the current line in resultant "**directory**" window should be set to the focal directory. If this bit is not set then the current line will be the last selected directory, or if none have been selected, the first line in the buffer.

0x04

Specifies that any evaluations required during the commands operation should be performed. Without this flag an open operation on a directory which has not previously been evaluated will not be perform an evaluation and the results will likely be incomplete.

0x08

Specifies that the current focal directory should be opened. This means that sub−directories within the current focal directory will also be drawn in the directory tree.

0x10

Specifies that the current focal directory should be closed. This means that sub−directories within the current focal directory will not be drawn in the directory tree.

0x20

Specifies that the current focal directory's open state should be toggled. This means that if the sub−directories are currently hidden they will now be drawn and vice−versa.

0x40

When specified any directory opened will be re−evaluated, ensuring the accuracy of the information.

0x80

Enables a recursive behavior, for example if this flag was specified with the open then not only will the focal directory be opened, but all of it's children, and their children etc. Note that if the Evaluation flag is not specified then only the already evaluated directories can be opened.

directory−tree creates a new buffer "**directory**" and draws the known directory tree. Every drawn directory is preceded by a character flag giving the user an indication of the directory state, where:

?

Directory has not been evaluated.

−

Directory has been evaluated and is visible.

+

Directory has been evaluated but is currently hidden.

Directories which have been evaluated and found to have no children use the '−' $box−chars(5) instead of a '−' character.

On UNIX platforms, if a directory is a symbolic link to another directory, the link name is given after the directory name.

EXAMPLE

The best example of the use of directory−tree is file−browser(3) which can be found in hkdirlst.emf.

SEE ALSO

file−browser(3), $box−chars(5).

display−white−chars(3)

NAME

display−white−chars − Toggle the displaying of white characters

SYNOPSIS

display−white−chars

DESCRIPTION

display−white−chars toggles the displaying of white characters in the main display. By default white characters, space tab and new−lines, are represented with invisible characters such as one or more ' 's for spaces and tabs and text moving to the next line for new−lines. The user can make this characters become 'visible' using this function.

When this function is first called it toggle enables the displaying of these characters, other characters are drawn in their place to make them visible. A subsequent call will disable the displaying of them.

NOTES

display−white−chars is a macro implemented in `misc.emf` and uses bit `0x80000` of the $system(5) variable.

The displaying of white characters can be enabled or disabled at start−up using user−setup(3).

This feature may be more confusing on some terminals due to the lack of characters available for displaying the white characters. The characters used when displaying white characters are defined in the variable $window−chars(5).

SEE ALSO

$system(5), user−setup(3), $window−chars(5).

draw(3)

NAME

draw − Simple line drawing utility

SYNOPSIS

draw

DESCRIPTION

draw provides a simple way of drawing lines into the current buffer, this has a variety of uses such as drawing tables. **draw** copies the current buffer into a temporary buffer and then allows the user to draw using simple commands until the user either aborts, discarding any changes, or exits insert the changes back into the buffer.

The keys for **draw** are defined as follows:−

esc h

Display a help dialog.

up, **down**, **left**, **right**

The cursor keys (or any other keys bound the the same commands) will move the cursor, drawing in the current mode.

d

Change the current mode to **d**raw (default), cursor movement will result in drawing in the current style.

e

Change the current mode to **e**rase, cursor movement will result in erasing to spaces.

m

Change the current mode to **m**ove, no drawing is performed with cursor movement.

u

Change the current mode to **u**ndo, cursor movement will result in undoing the character to the original or a space.

−

Sets the current horizontal line drawing style to use '`−`'s (default).

=

Sets the current horizontal line drawing style to use '`=`'s.

C−g

Abort − changes are lost.

return

Exit, inserting any changes into the current buffer. **NOTES**

draw is a macro defined in `draw.emf`.

edit−dictionary(3)

NAME

edit−dictionary − Insert a dictionary in a buffer
restore−dictionary − Save dictionary user changes

SYNOPSIS

edit−dictionary "*dictionary*"
restore−dictionary

DESCRIPTION

edit−dictionary dumps the contents of "*dictionary*" into the temporary buffer "**dictionary**", if this buffer already exists then **edit−dictionary** simply swaps to this buffer. This enables the user to correct and prune the words in any dictionary. The given dictionary must have already been added as a main dictionary using add−dictionary(2).

The format of the created buffer is one word on each line, each word takes one of the following 3 forms:

`xxxx` − Good word `xxxx` with no spell rules allowed
`xxxx/abc` − Good word `xxxx` with spell rules `abc` allowed
`xxxx>yyyy` − Erroneous word with an auto−replace to `yyyy`

Executing **restore−dictionary** in a buffer created by **edit−dictionary** will first call delete−dictionary(2) to remove the original dictionary from memory. It then uses add−dictionary(2) to create a new dictionary with the same name and then uses spell−add−word(3) to add all the words in the current buffer into the new dictionary.

restore−dictionary does not save the new dictionary.

NOTES

edit−dictionary and **restore−dictionary** are macros defined in file `spellutl.emf`. They are not defined by default so *spellutl.emf* must be executed first using execute−file(2).

SEE ALSO

spell−add−word(3), add−dictionary(2), save−dictionary(2), delete−dictionary(2).

start−kbd−macro(2)

NAME

start−kbd−macro − Start/stop recording keyboard macro end−kbd−macro − Stop recording keyboard macro

SYNOPSIS

start−kbd−macro (**C−x (**)
end−kbd−macro (**C−x)**)

DESCRIPTION

A keyboard macro is a short hand way to repeat a series of characters. In effect, a *recording* is made of the sequence of keys that you hit while defining a keyboard macro. The recording is started with **start−kbd−macro** and ended with **end−kbd−macro**. The recording is then repeated whenever you execute the keyboard macro using execute−kbd−macro(2).

Since it is key−strokes that are being saved, you can freely intermix commands and text to be inserted into the buffer.

You can save a keyboard macro for later using the name−kbd−macro(2) command, which saves the keyboard macro as a named macro. Otherwise if you start another keyboard macro recording session, the previously defined macro is lost. So make sure that you are done with the current keyboard macro before defining another one. If you have a series of commands that you would like to *record* for later use, insert−macro(2) can be used to insert the macro into a text file and can be reloaded using the execute−file(2) or execute−buffer(2) commands.

Recording commences with **start−kbd−macro** (`C−x (`) and terminates when an **end−kbd−macro** (`C−x )`) is encountered.

NOTES

Once **start−kbd−macro** has been executed, the mouse is disabled until **end−kbd−macro** is executed. This is because the mouse events cannot be successfully recorded in macros. The main menu can still be used, but only via the keyboard bindings and hot−keys (note that the layout of the menu may change).

SEE ALSO

execute−kbd−macro(2), insert−macro(2), kbd−macro−query(2), name−kbd−macro(2).

etfinsrt(3)

NAME

etfinsrt − Insert template file into current buffer

SYNOPSIS

etfinsrt "*template*"

DESCRIPTION

etfinsrt is generally called by file hooks when the new buffer has been created as opposed to loaded from a file (see $buffer−fhook(5)).

etfinsrt uses &find(4) to locate and insert the required "*template*.etf" file. If successful, **etfinsrt** then replaces the following strings in the template:

`$ASCII_TIME$`

To the current time. Inserts the output of ascii−time(3).

`$BUFFER_NAME$`

To the buffer name. The name is capitalized, '.'s are replaced with '_' and any trailing "<##>" digits (used to make the buffer name unique) are removed.

`$COMPANY_NAME$`

To the value of **%company−name**, or if not defined to the value used for `$USER_NAME$`. **%company−name** is usually set up in the company setup file defined in User setup.

`$USER_NAME$`

To the value of the registry entry "`/history/user-name`", or if not defined to the value "`<unknown>`". The user name is usually set up in the User setup dialog.

`$YEAR$`

To the current year (4 digit number).

`$CURSOR$`

To leave the cursor at this point, only one of these tokens should be used in the template and the token is removed. **EXAMPLE**

The following is taken from hkmake.emf and inserts the "*makefile.etf*" template if the buffer has been created.

```
define-macro fhook-make
    ; if arg is 0 this is a new file so add template
    !if &not @#
        etfinsrt "makefile"
    !endif
    set-variable $buffer-hilight .hilight.make
    -1 buffer-mode "tab"                ; Normal tabs please !!!
    1 buffer-mode "indent"
    1 buffer-mode "time"
!emacro
```

NOTES

etfinsrt is a macro defined in `etfinsrt.emf`.

magic(2m) mode is always used to perform the the search/replace so the replace strings should be appropriate for **magic**.

SEE ALSO

$buffer−fhook(5), &find(4), ascii−time(3).

exchange−point−and−mark(2)

NAME

exchange−point−and−mark − Exchange the cursor and marked position

SYNOPSIS

exchange−point−and−mark (**C−x C−x**)

DESCRIPTION

exchange−point−and−mark moves the cursor to the current marked position (see set−mark(2)) in the current window and moves the mark to where the cursor was. This is very useful in finding where a mark was, or in returning to a position previously marked.

SEE ALSO

set−mark(2), copy−region(2).

execute−buffer(2)

NAME

execute−buffer − Execute script lines from a buffer
execute−line − Execute a script line from the command line

SYNOPSIS

execute−buffer "*buffer−name*"
execute−line ["*command−line*"]

DESCRIPTION

execute−buffer executes script lines in the named buffer *buffer−name*. If the buffer is off screen and an error occurs during execution, the cursor is left on the line causing the error.

execute−line executes a in script line entered from the command line. Typically this is used in macros.

SEE ALSO

execute−file(2), execute−string(2), execute−named−command(2).

execute−file(2)

NAME

execute−file − Execute script lines from a file

SYNOPSIS

n **execute−file** "*file*" (**esc** /)

DESCRIPTION

execute−file executes script lines from the given *file n* times in succession, this is the normal way to execute a MicroEmacs '02 script. The command prompts for a file name, and will then search for <*file*>[.emf] in the search path. If the file is found then the file is loaded and the buffer is executed *n* times.

SEE ALSO

execute−buffer(2), execute−line(2), execute−named−command(2), execute−string(2).

execute−kbd−macro(2)

NAME

execute−kbd−macro − Execute a keyboard macro

SYNOPSIS

n **execute−kbd−macro** (**C−x e**)

DESCRIPTION

execute−kbd−macro executes a keyboard macro. The entire sequence of recorded key−strokes is repeated starting at the current point. The result is exactly as if you were retyping the sequence all over again. A numeric argument *n* prefixing the **execute−kbd−macro** command repeats the stored key−strokes *n* times.

Keyboard macros are recored with start−kbd−macro(2); recording is terminated with end−kbd−macro(2).

SEE ALSO

end−kbd−macro(2), kbd−macro−query(2), name−kbd−macro(2), start−kbd−macro(2).

execute−named−command(2)

NAME

execute−named−command − Execute a named command

SYNOPSIS

n **execute−named−command** "*command−string*" esc x

DESCRIPTION

execute−named−command command prompts the user for the name of a command to execute and then executes the command *n* times. MicroEmacs '02 offers command completion and history facilities, see ml−bind−key(2).

SEE ALSO

execute−buffer(2), describe−bindings(2), ml−bind−key(2).

execute−string(2)

NAME

execute−string − Execute a string as a command

SYNOPSIS

n **execute−string** "*string*"

DESCRIPTION

execute−string executes the given *string n* times as if it is being typed. This is the writable format of a keyboard macro, it can be placed in any **emf** file. Any characters may form the *string* (unprintables as `\xXX`) and key−strokes that are bound to a command will execute that command. This command is used by macros to store user defined keyboard macros.

EXAMPLE

The following example uses keyboard strokes with **execute−string** in a macro to format **nroff(1)** text located between `.` commands:

```
define−macro nroff−para
    beginning−of−line
    !if &not &sequal @wc "."
        1 buffer−mode "magic"
        execute−string "\CXS^\\.\CM\CB\CM\CX\CH\CN\CM"
        −1 fill−paragraph
        execute−string "\CD\CX\CH\CN\CD\CXH\CB"
    !endif
    forward−line
!emacro
```

execute−string has the advantage that execution is very fast as the amount of parsing and decoding to be performed is limited. The disadvantage is that you cannot quickly discern which operations are being performed !!

NOTES

Try to avoid using named key, such as "`up`" and "`return`", as the keyboard macro equivalent is not readable and is likely to change in future releases.

For this reason the following special abbreviations may be used

\E

The "**escape**" key.

\N

The "**return**" key.

\T

The "**tab**" key.

\b

The backspace character (0x08).

\d

The delete character (0x7f).

\e

The escape character (0x1b).

\f

The form−feed character (0x0c).

\n

The carriage−return character (0x0a).

\r

The line−feed character (0x0d). **SEE ALSO**

buffer−abbrev−file(2), global−abbrev−file(2), insert−macro(2), name−kbd−macro(2), start−kbd−macro(2).

execute−tool(3)

NAME

execute−tool − Execute a user defined shell tool

SYNOPSIS

n **execute−tool** "*tool−name*"

DESCRIPTION

execute−tool launches a predefined shell tool, the tools are typically defined by the user−setup(3) Tools page and executed using the MicroEmacs main Tools menu. See help on user−setup(3) for more information on the basic facilities given by execute−tool.

If the numeric argument *n* is supplied it is used as the tool name to be executed, otherwise the argument "*tool−name*" must be given.

A tool with a numeric name can be executed via a key binding, for example, to execute tool **3** (as defined by **user−setup**) to '`C−3`' add the following line to the user setup file:−

```
3 global-bind-key execute-tool "C-3"
```

NOTES

The registry entries for a tool must be located in registry directory "`/history/`**$platform/tool/tool−name**" where **$platform** is the current setting of variable $platform(5) and **tool−name** is the name of the tool as given to the command. The following registry entries are used:−

name

The name of the tool as displayed in the user−setup Tools dialog and the Main Tools menu. This is only used for tools `0` to `9`.

command

The command−line to be launched when the tool is executed, the following special tokens may be used in the command−line which are substituted at execution:−

%ff

The current buffer's full file name, including the path.

%fp

The current buffer's file path.

%fn

The current buffer's file name without the path.

%fb

The current buffer's file base name, i.e. the file name without the path or the extension.

%fe

The current buffer's file extension with the '.' (e.g. "*.emf*"), set to the empty string if the file name does not have an extension.

Note that "**%ff**" is always the same as "**%fp%fn**" and "**%fp%fb%fe**". If any of these tokens are used, the tool will fail to execute if the current buffer does not have a file name.

flag

A bit based flag setting the tool characteristics, where:−

0x01

Enable current buffer saving.

0x02

Enable prompt before saving current buffer.

0x04

Enable all edited buffers saving.

0x08

Enable prompt before saving an edited buffer.

0x10

Enable output capturing.

0x20

Enable concurrent running, not available on all platforms, see variable $system(5).

bname

The name of the buffer to be used if the output is captured. The following special tokens may be used in the buffer name which are substituted at execution:−

%fn

The current buffer's file name without the path, set to the buffer name if the current buffer does not have a file name.

%fb

The current buffer's file base name, i.e. the file name without the path or the extension. Set to the buffer name if the current buffer does not have a file name.

%fe

The current buffer's file extension with the '.' (e.g. "*.emf*"), set to the empty string if the current buffer does not have a file name or it does not have an extension.Note that "**%fn**" is always the same as "**%fb%fe**". Default buffer name when this field is left empty is "**command**", or "**icommand**" if `Run Concurrently` is enabled.

If more than 10 tools are required (maximum number definable by **user−setup**) or names are preferred, it is recommended that the **user−setup** dialog is used to define the tool and then use the registry copy utility bound to '`c`' in a list−registry(2) buffer.

SEE ALSO

user−setup(3), ipipe−shell−command(2), pipe−shell−command(2), shell−command(2), system(5).

exit−emacs(2)

NAME

exit−emacs − Exit MicroEmacs

SYNOPSIS

n **exit−emacs**

DESCRIPTION

Exit MicroEmacs back to the operating system. If no argument *n* is given and there are any unwritten, changed buffers, the editor prompts the user to discard changes. If an argument is specified then MicroEmacs exits immediately.

NOTES

All buffers with a name starting with a '*' are assumed to be system buffers (i.e. ***scratch***) and are not saved.

SEE ALSO

quick−exit(2), save−buffers−exit−emacs(2).

expand−abbrev(2)

NAME

expand−abbrev − Expand an abbreviation

SYNOPSIS

expand−abbrev

DESCRIPTION

expand−abbrev expands an abbreviation to an alternate form. The abbreviation must be an alpha−numeric string and the cursor must be one position to the right of the abbreviation (which must not be alpha−numeric) when this command is called. If the abbreviation is found, it is deleted and the alternate form is inserted leaving the cursor at the end of the insertion unless `\p` is used. If not found, a space is inserted.

SEE ALSO

buffer−abbrev−file(2), global−abbrev−file(2), expand−abbrev−handle(3), eaf(8).

expand−abbrev−handle(3)

NAME

expand−abbrev−handle − Expand an abbreviation handler

SYNOPSIS

expand−abbrev−handle (**esc esc**)

DESCRIPTION

expand−abbrev−handle pulls together all forms of abbreviation expansion into a single command so that it can be bound to a single key. The abbreviation must be an alpha−numeric string and the cursor must be one position to the right of the abbreviation (which must not be alpha−numeric) when this command is called. The command attempts to expand the abbreviation using the following commands in turn:

expand−abbrev(2)

Uses a buffer specific and global abbreviation files, if set, to look up the abbreviation. The use of the abbreviation file can be disabled using buffer−setup(3).

expand−iso−accents(3)

Expands ISO accent letter if the expansion mode is enabled via either the user−setup(3) General Page or by using the iso−accents−mode(3) command.

expand−look−back(3)

Looks for a word starting the same in the current buffer's last 100 lines, this can be enabled in the user−setup(3) General page.

Buffer specific expansion

Executes a buffer specific abbreviation expansion if the current buffer's file hook supports abbreviation expansion.

Word expansion

If the current buffer does not support file type specific expansion and Word Expansion is enabled via the user−setup(3) General page (`Dict'n` setting) expansion is attempted using the expand−word(3) command which expands the current partial word using the dictionary of the user's current language; warning − this can be slow!

The command exits after first command to successfully expand or if none expand the command fails. See the help in the individual expansion commands for more help.

SEE ALSO

user−setup(3), expand−abbrev(2), expand−iso−accents(3), expand−look−back(3), expand−word(3).

expand−look−back(3)

NAME

expand−look−back − Complete a word by looking back for a similar word

SYNOPSIS

expand−look−back

DESCRIPTION

expand−look−back attempts to complete the word at the current position by looking backward for another word which starts the same. If such a word is found within 100 lines of the current cursor position the current partial word is replaced with the word found.

expand−look−back is automatically invoked from the expand−abbrev−handle(3) macro in response to an expansion command, it is only invoked if enabled in the user−setup(3) => General => Abbrev Expansion => Lookbk setting is enabled.

NOTES

expand−look−back is a macro implemented in `abbrev.emf`.

The **user−setup** configuration simply sets the macro variable `.expand−look−back.on` to TRUE, i.e.:

```
set-variable .expand−look−back.on 1
```

It may be subsequently disabled by setting the variable back to 0.

SEE ALSO

expand−abbrev−handle(3), user−setup(3).

expand−word(3)

NAME

expand−word − Complete a word by invocation of the speller

SYNOPSIS

expand−word

DESCRIPTION

expand−word attempts to complete the word at the current position through the use of the current language dictionary. The user is presented with a list of endings for the given word portion. These may be selected with the cursor or mouse.

expand−word is automatically invoked from the expand−abbrev−handle(3) macro in response to an expansion command, it is only invoked if enabled in the user−setup(3) => General => Abbrev Expansion => Dict'n setting is enabled.

NOTES

expand−word is a macro implemented in `abbrev.emf`.

The **user−setup** configuration simply sets the macro variable `.expand−word.on` to TRUE, i.e.:

```
set-variable .expand-word.on 1
```

It may be subsequently disabled by setting the variable back to 0.

SEE ALSO

expand−abbrev−handle(3), spell−buffer(3), find−word(3).

file−attrib(3)

NAME

file−attrib − Set the current buffers system file attributes

SYNOPSIS

file−attrib

DESCRIPTION

file−attrib opens a dialog enabling the user to change the system properties of the current buffer's file. Top of the dialog give the current buffer name and its file name. The `Save Changes` button writes the current buffer out with any current edits and changes to its file attributes. The `Ok` button closes the file−attrib dialog, any changes made to the file attributes will be applied next time the buffer is written.

The type allow the changing between UNIX, MS Windows and DOS text file formats. UNIX has a single new line character ('`\n`') where as Windows and Dos have a double new line character ('`\r\n`'). Also a Dos text file is terminated with a `C−z` (0x1A) character which the other two do not. These attribute are set in MicroEmacs by using buffer modes crlf(2m) and ctrlz(2m).

The central part of the dialog contains system dependent attributes which are defined as follows:

UNIX Platforms

Allow the setting of user, group and global, read, write and execute permissions, see man pages on **chmod(1)** for more information. This is a front end to setting the variable $buffer−fmod(5).

Win32 Platforms

Allow the setting of MS Windows file attributes, i.e. read−only, hidden, archive etc. Note that the directory attribute is displayed but cannot be altered. This is a front end to setting the variable $buffer−fmod(5).

DOS Platform

Allow the setting of MS Dos file attributes, i.e. read−only, hidden, archive etc. Note that the directory attribute is displayed but cannot be altered. **NOTES**

file−attrib is a macro implemented in `fattrib.emf`.

SEE ALSO

find−file(2), write−buffer(2), crlf(2m), ctrlz(2m), $buffer−fmod(5).

file−browser(3)

NAME

file−browser − Browse the file system file−browser−close − Close the file−browser
file−browser−swap−buffers − Swap between file−browser windows

SYNOPSIS

file−browser (**f10**)
file−browser−close
file−browser−swap−buffers

DESCRIPTION

file−browser can be used to browse around the file system. When first executed **file−browser** creates 2 buffers, "`*directory*`" displaying the directory structure and "`*files*`" listing the files in the current directory with information on each file. **file−browser** displays these buffers side by side, splitting the current window horizontally if required.

Once open the user can browse through the system using the following keys in the `*directory*` buffer:

`space`

Selects the directory on the current line and up−dates the `*files*` buffer with the information on this directory. This can also be done by clicking the left mouse button on the directory name.

`return`

Selects the directory on the current line, if open (sub−directories displayed) then closes it or if closed it is opened. The `*files*` buffer is up−dated with the information on the directory. This can also be done by clicking the left mouse button on the '`+`' or '`−`' symbol just before the directory name.

`C-return`

As with `return` expect sub−directories are recursively opened or closed, note that this could take some time on large file systems. This can also be done by clicking the right mouse button on the '`+`' or '`−`' symbol just before the directory name.

`tab`

Move to the `*files*` buffer.

`delete`

Closes file−browser.

The following keys can be used in the `*files*` buffer:

`return`

If the current line is a directory, this because the current directory, updating both the `*directory*` and `*files*` buffers. If the line is a file then it is opened using find−file(2). This can also be done by clicking the left mouse button on the file name.

`space`

Toggles the tag state of the file on the current line, see `X` command. This can also be done by clicking the left mouse button anywhere before the file name, or for multiple files drag a region with the left mouse button.

`X` or `x`

Executes a shell−command(2) on all tagged files. The user is prompted for the command line which can contain the following special tokens:

`%p`	Full file name, including path.
`%f`	The file name without the path.

As the **shell−command** is executed in the directory `%f` is safe to use in a command such as "`del %f`".

`D` or `d`

Deletes all the tags in the buffer.

`tab`

Move to the `*directory*` buffer.

`delete`

Closes file−browser.

file−browser−swap−buffers swaps between the `*directory*` and `*file*` windows, making the other the current window, this is usually locally bound to the `tab` key.

file−browser−close hides both the `*directory*` and `*file*` windows, closing the file−browser, this is usually locally bound to the `delete` key.

SEE ALSO

directory−tree(2), find−file(2), shell−command(2).

file−op(2)

NAME

file−op − File system operations command

SYNOPSIS

n **file−op** [(["*from−file*" "*to−file*"]) |

(["*delete−file*"]) | (["*dir−name*"])] **DESCRIPTION**

file−op can be used to perform numerous file system operations. The given argument *n* must be used to determine the required operation, the value is a bit based flag denoting the operation as follows:

0x010

Log−off and close down the current ftp connect (not a file system operation but functionality was required and it had to go somewhere).

0x020

When this bit is set the command functionality is changed to delete−file, the single argument *delete−file* is deleted.

0x040

When this bit is set the command functionality is changed to move−file, the specified *from−file* is moved to *to−file*.

0x080

When this bit is set the command functionality is changed to copy−file, the specified *from−file* is copied to *to−file*.

0x100

When this bit is set the command functionality is changed to making a new directory, the specified *dir−name* is the name of the new directory. A file or directory of the given name must not already exist.

Only one operation can be performed per invocation. The following bits in the given argument *n* can be used to effect the behaviour of these operations:

0x01

Enables validity checks, these include a check that the proposed file does not already exist, if so confirmation of writing is requested from the user. Also MicroEmacs checks all other current buffers for one with the proposed file name, if found, again confirmation is requested. Without this flag the command will always succeed wherever possible.

0x02

Creates a backup of any file about to be deleted or over−written. Set help on $buffer−backup(5) for backup file−name generation. **NOTES**

http files are not supported except as the source file when copying. **ftp** files are fully supported with the restriction that the from and to files cannot both be url (http or ftp) files.

The command is used by file−browser(3) and ftp(3) which provides an easy to use interfaces for file manipulation.

SEE ALSO

file−browser(3), ftp(3), find−file(2), write−buffer(2), $temp−name(5).

fill−paragraph(2)

NAME

fill−paragraph − Format a paragraph

SYNOPSIS

n **fill−paragraph** (**esc o**)

DESCRIPTION

fill−paragraph this takes all the text in the current paragraph (as defined by surrounding blank lines, or a leading indent) and attempts to fill it from the left margin to the current fill column as defined by $fill−col(5). When an argument *n* is supplied *n* paragraphs are filled. If *n* is positive then MicroEmacs '02 performs indented filling (i.e. indentation for a bullet mark etc). If *n* is negative then indented filling is disabled. If no argument *n* is supplied then the paragraph is filled and the *point* and *mark* positions are retained. This allows paragraphs to be filled, whilst in the middle of the paragraph and the word position is maintained.

If **justify mode** is enabled the variable $fill−mode(5) determines how the paragraph is filled (i.e. *left*, *right*, *both* or *center*). The variable $fill−eos−len(5) determines the trailing space used after a period (`.`) character (the trailing characters are specified by $fill−eos(5)), typically defined as 2.

A set of characters defined by $fill−bullet(5) enable bullet markers to be placed in the text at the beginning of the paragraph causing the left margin to be moved to the right of the bullet. The search depth for fill to locate a bullet character is defined by $fill−bullet−len(5). When the paragraph is formatted and one of the bullet characters is encountered then the user is prompted as to whether the paragraph should be indented following the marker or not. The point of indentation is shown with a **<<<<** marker.

Filling is automatically disabled on paragraphs which start with characters in the $fill−ignore(5) set.

The simple text formatting is generally used for mail messages, ASCII text README files etc.

EXAMPLE

The following examples show how the text is formatted with indented filling enabled and both justification enabled:−

```
This  is  regular  text  that  is on the
margin

    This is a regular  paragraph that is
    offset  from  the  margin.  Note how
    MicroEmacs '02 retains the indent.
```

```
* With  the  introduction  of one of
  the  special  characters,  in this
  case a  bullet,  a  format  of the
  paragraph  offsets  the text  from
  the bullet.

1) Numbered  lists   are  the  same.
   Note that the  paragraphs are all
   separated with a blank line.

1. Numbered  lists  ending  with   a
   period.

label − Or labeled lists, separated
        with a dash.

>  '>' might be an ignore
>  character so it skips the paragraph
>
>          it is up to the user to
>   format these.
```

SEE ALSO

$fill−bullet(5), $fill−bullet−len(5), $fill−col(5), $fill−eos(5), $fill−eos−len(5), $fill−ignore(5), $fill−mode(5), ifill−paragraph(3), paragraph−to−line(3).

filter−buffer(2)

NAME

filter−buffer − Filter the current buffer through an O/S command

SYNOPSIS

filter−buffer (**C−x #**)

DESCRIPTION

filter−buffer executes one operating system command, using the contents of the current buffer as input, sending the results back to the same buffer, replacing the original text.

This would typically be used in conjunction with **sort(1)**, **awk(1)** or **sed(1)** to translate the contents of the buffer.

SEE ALSO

pipe−shell−command(2).

find−bfile(3)

NAME

find−bfile − Load a file as binary data
find−cfile − Load a crypted file

SYNOPSIS

n **find−bfile** "*file−name*" (**C−x 9**)
n **find−cfile** "*file−name*"

DESCRIPTION

find−bfile and **find−cfile** provide a simple interface to loading files in binary(2m) and crypt(2m) modes respectively. The numeric argument has the same effect as with the find−file(2) command except the respective modes are always enabled. See documentation on the modes an **find−file** command for more information.

NOTES

find−bfile and **find−cfile** are macros defined in file `tools.emf`.

The command find−file(2) is bound to key "`C−x 9`" with a numeric argument of 2, this is equivalent to executing **find−bfile** with no argument.

SEE ALSO

find−file(2), binary(2m), crypt(2m).

next−buffer(2)

NAME

next−buffer − Switch to the next buffer
find−buffer − Switch to the next buffer

SYNOPSIS

n **next−buffer** (**C−x x**)
n **find−buffer** "*buffer−name*" (**C−x b**)

DESCRIPTION

next−buffer switches to the *n*th next buffer in the buffer list in the current window, the default *n* is `1`, if *n* is negative then the `0−`*n*th previous buffer is selected. If `0` or a number greater than the number of buffers is specified then the command fails.

find−buffer switches to buffer "*buffer−name*" in the current window. If the buffer does not exist and a zero argument *n* is supplied then the command fails. If the buffer does not exist but no argument or a +ve argument *n* is specified then a new buffer is created, at which point the file−hook is evaluated.

If a −ve argument *n* is given to **find−buffer** then the buffer will be hidden. Any window displaying "*buffer−name*" will find another buffer to display. This functionality is often used with the hide(2m) buffer mode. If a value of −1 is given then the buffer will not be hidden in a window whose $window−flags(5) are set to lock the buffer to the window. If a value of less than −1 is given then the buffer is hidden from all windows.

If the current buffer has an *$buffer−ehook* command set then this command is executed before the new buffer is switched in. If the new buffer has a $*buffer−bhook* command set then this command is automatically executed after the new buffer is switched in but before control returns to the user.

SEE ALSO

next−window−find−buffer(2), hide(2m).

find−file(2)

NAME

find−file − Load a file

SYNOPSIS

n **find−file** "*file−name*" (**C−x C−f**)

DESCRIPTION

find−file finds the named file *file−name*. If it is already in a buffer, make that buffer active in the current window, otherwise attempt to create a new buffer and read the file into it.

The numeric argument *n* can be used to modify the default behaviour of the command, where the bits are defined as follows:

0x01

If the file does not exist and this bit is not set the command fails at this point. If the file does not exist and this bit is set (or no argument is specified as the default argument is `1`) then a new empty buffer is created with the given file name, saving the buffer subsequently creates a new file.

0x02

If this bit is set the file will be loaded with binary(2m) mode enabled. See help on **binary** mode for more information on editing binary data files.

0x04

If this bit is set the file will be loaded with crypt(2m) mode enabled. See help on **crypt** mode for more information on editing encrypted files.

0x08

If this bit is set the file will be loaded with rbin(2m) mode enabled. See help on **rbin** mode for more information on efficient editing of binary data files.

Text files are usually thought of as named collections of text residing on disk (or some other storage medium). In MicroEmacs '02 the disk based versions of files come into play only when reading into or writing out buffers. The link between the physical file and the buffer is through the associated file name.

MicroEmacs '02 permits full file names, i.e. you can specify:

```
        disk:\directories\filename.extension
```

or (UNIX)

```
        /directories/filename.extension
```

If the disk and directories are not specified, the current buffers disk/ directory is used. Several points should be noted in respect to the methods that MicroEmacs utilizes in the handling of files:−

- Without explicitly saving the buffer(s) to file, all edits would be lost upon leaving MicroEmacs − you are asked to confirm whenever you are about to lose edits.
- MicroEmacs has a mechanism for "protecting" your disk−based files from overwriting when it saves files. When instructed to save a file, it proceeds to dump the file to disk, making a backup of the existing file when backup(2m) mode is enabled.
- Auto−saving files can be performed on edited buffers by setting the $auto−time(5) variable. The file is saved in the same place with a '#' appended to the file name. This can be used directly by the user or in the unlikely event of MicroEmacs crashing (or system crash), the files are automatically recovered next time it is edited.

If you do not wish to perform any edits but merely browse the file(s), add the view(2m) mode to the buffer or ask for the file to be read in for viewing only.

RCS Support

If the file does not exist and the variable $rcs−file(5) is set then the existence of the RCS file is tested. If the rcs file exists then it will be checked out using a command−line created from the variable $rcs−co−com(5). If the check−out is successful then this file is loaded.

This raw interface for supporting file revision control systems has been adapted to support SCCS and Visual Source Safe see help on variable **$rcs−file** for more information and examples.

HTTP Support

MicroEmacs supports http file loading, this is available by default on UNIX systems but must be compiled in on win32 platforms (socket libraries not available on all win95 machines so cannot be compiled in by default). When available a http file can be loaded by simply executing **find−file** and giving the http file name, i.e. "`http://user:password@address:port/file`". Only the `http://`, `address` and `/file` components are mandatory, the rest can usually be omitted. e.g.:

```
        find−file "http://members.xoom.com/jasspa/index.html"
```

See help page on %http−proxy−addr(5) for information on HTTP proxy server support.

FTP support

MicroEmacs supports ftp file loading, this is identical to http except the prefix `ftp://` is used as opposed to `http://`. The user name and password defaults to *guest* in the absence of both these fields. If the user name is supplied but not the password the password will be prompted for; this can be useful as the password will not be stored or written to the history file. Connection is by default on port 21.

```
    find−file "ftp://<me>:<password>@members.xoom.com/jasspa/index.html"
```

See also ftp(3).

The progress of the FTP transfer, and the FTP commands issued, may be viewed in the `*ftp−console*` buffer. This is popped up depending on the setting of the %ftp−flags(5) variable.

NOTES

The base name part (i.e. not the path) of `file−name` can contain wild−card characters compatible with most file systems, namely:−

?

Match any character.

[abc]

Match character only if it is *a*, *b* or *c*.

[a−d]

Match character only if it is *a*, *b*, *c* or *d*.

[^abc]

Match character only if it is not *a*, *b* or *c*.

*****

Match any number of characters.

If the name matches more than one file, a buffer will be created for each matching file. Note that these are not the same wild−card characters used by regex.

For *ftp* and *http* then a ftp console window is opened up to show the progress of the transfer (when configured), this is described in ftp(3).

SEE ALSO

auto(2m), binary(2m), crypt(2m), rbin(2m), time(2m), view(2m), buffer−mode(2), find−bfile(3), ftp(3), $rcs−file(5), %ftp−flags(5), %http−flags(5), %http−proxy−addr(5), next−window−find−file(2), read−file(2), save−buffer(2), view−file(2), write−buffer(2), file−op(2), file−attrib(3).

find−registry(2)

NAME

find−registry − Index search of a registry sub−tree.

SYNOPSIS

find−registry "*root*" "*subkey*" *index*

DESCRIPTION

find−registry performs an indexed search of a registry sub−tree allowing the caller to determine the names of the children that exist as sub−nodes of the specified node. *root* and *sub−key* form the root whose children are to be determined, *subkey* may be specified as the null−string (`""`) if an absolute registry path is specified. *index* is a value from `0..n` and identifies the index number of the child node. The name of the child node is returned in $result(5) if one exists, otherwise an error status is returned.

EXAMPLE

The following example comes from `addrbook.emf` and shows how **find−registry** is used to iterate through entries in the address book. Note that **find−registry** is used with !force(4) and the $status(5) of the call is tested to determine if the invocation succeeded.

```
!force find-registry "/AddressBook" "Names" #l0
!if $status
    set-variable #l1 $result
    76 insert-string "_"
    2 newline
    insert-string &spr "Section: %s" #l1
    newline
    ; Iterate through all of the entries.
    set-variable #l2 0

    !repeat
        !force #l2 ab-buffer
        !if $status
            set-variable #l2 &add #l2 1
        !endif
    !until &not $status
    set-variable #l0 &add #l0 1
    !goto next
!endif
```

SEE ALSO

get−registry(2), list−registry(2), read−registry(2), set−registry(2), erf(8).

find−tag(2)

NAME

find−tag − Find tag, auto−load file and move to tag position

SYNOPSIS

n **find−tag** "*string*" (**esc t**)

DESCRIPTION

find−tag finds the current or given tag (*string*) in a **tags** file and goes to the given point, loading the file if necessary. The tag is either the current word under the cursor or a user supplied word if the cursor is not in a word. The buffer containing the tag is popped up in another window and the cursor moved to the tag in the new window.

A **tags** file is usually created by an external program (e.g. **ctags(1)**) which stores word references (or tags) and the name of the file containing the tag, with a search string to go to its local. It is an indexing system which is often used in programming.

The argument *n* can be used to change the default behavior of find−tag described above, *n* is a bit based flag where:−

0x01

Use popup−window to display the tag in a different window (default) when this flag is not given the current window is used to display the tag.

0x02

Disable the use of the current cursor position to determine the tag. Instead the tag must always be supplied through "*string*".

0x04

Find the next definition of the last tag (multiple tag support). This feature can only be used if multiple tag support is enabled (see flag '**m**' of variable %tag−option(5)) and **find−tag** has already been successfully executed. In this situation the last invocation of find−tag defines the current tag and executing again with an argument of `4` will jump to the next definition of the current tag or return the message "`[No more "<current>" tags]`".

The next tag is typically bound to `M−C−t`.

The **tags** file is, by default, assumed to reside in the current directory of the currently viewed file. The user variable %tag−option(5) may be specified with a value of '`r`' (recursive) and '`c`' (continue) flags, which ascends the directory tree from the current directory and attempts to locate a *recursively* generated tags file at a higher directory level. Recursive tag files are generally easier to maintain where project source files are located in a number of project sub−directories, and enable the whole of the project tree to be taggable.

Two user variables must be defined before **find−tag** will execute, if either %tag−file(5) or %tag−template(5) are not defined the error message "**[tags not setup]**" is signaled.

NOTES

A **tags** file may be generated by MicroEmacs '02 from the menu (*Tools−>XX Tools−>Create Tags File*). Alternatively a **tags** file may be generated by the **ctags(1)** utility. This is typically standard on UNIX platforms. For Windows and DOS platforms then the **Exuberant Ctags** is recommended, this is available from:−

```
    http://darren.hiebert.com
```

A MicroEmacs '02 compatible tags file may be generated using the command line "`ctags −N --format=1 .`" cataloging the current directory. To generate **tags** for a directory tree then use "`ctags −NR −format=1 .`". Refer to the **Exuberant Ctags** documentation for a more detailed description of the utility.

SEE ALSO

%tag−file(5), %tag−option(5), %tag−template(5), generate−tags−file(3), **ctags(1)**.

spell−buffer(3)

NAME

spell−buffer − Spell check the current buffer
spell−word − Spell check a single word
spell−edit−word − Edits a spell word entry
find−word − Find a using spelling dictionaries

SYNOPSIS

spell−buffer
n **spell−word** ["*word*"] (**esc $**)
spell−edit−word ["*word*"]
find−word ["*word*"]

DESCRIPTION

MicroEmacs '02 provides an integrated spell checker with the following features:−

- Different languages.
- Dialog control of the speller.
- Best guess capability.
- *Replace* and *Replace all*, *Ignore* and *Ignore All*
- Undo capability.
- Adding new words and endings to speller.
- Auto correct of commonly occurring mistakes.
- Word finder, allows words to be searched with wild cards.

spell−buffer spell checks the current buffer, from the current position, to the end of the buffer. On invocation, an osd(2) dialog is opened and any corrections are made through this interface. If an error dialog opens instead the current language is not setup, please see the Language setting in user−setup(3) and Locale Support.

The dialog provides the user with an interface from which a new spelling may be selected, in addition new words may be added to the spelling dictionary. The dialog entries are defined as follows:−

Word

The **word** entry contains the erroneous word, this is presented in a text dialog box which may be manually edited to correct. If the word is manually corrected, then it is spell checked prior to insertion, and a new guess list is created. The user may elect to replace the word, take one of the suggestions or re−edit the misspelled word.

Meaning

The meaning button provides a convenient interface to describe−word(3) for looking up the meaning of the current word. The **Insert** button within the describe−word dialog will replace the current word in the spell−buffer.

Suggestions

The suggestions entry contains a list of suggestions as to the correct spelling of the word. The list is ranked in order of the best match, typically the misspelled word appears at (or near) the top of the list, unless the word is unknown or there are gross errors in the spelling. Selecting the word in the list with a single click of the mouse selects the word as the replacement, the actual replacement is performed by the **Replace** or **Replace All** buttons. Alternatively, double selecting a guess word replaces the word.

Language

The **language** entry allows the user to select the current spelling language. The new language is chosen from the dialog box. The language may be changed at any time during the spell operation and is effective immediately. The **Ext** languages are extended dictionaries that contain additional words, it is recommended that all spelling is performed with the extended dictionaries (where available).

Replace

The **replace** button is activated when a new word has been edited or selected as a candidate for replacement. Selecting **replace** modifies the erroneous word in the buffer with the newly selected word.

Replace All

The **Replace All** button is similar to the **Replace** button, except that it automatically replaces any subsequent occurrences of the erroneous word with the newly selected word. The replacement words are retained for the MicroEmacs edit session and are lost when the editor is closed.

Ignore

The **ignore** button requests that the speller ignore the erroneous word and continue to spell the buffer.

Ignore All

The **Ignore All** button is similar to the **Ingore** button, except that it automatically ignores the erroneous word thereafter. The ignore words are retained for the MicroEmacs edit session and are lost when the editor is closed.

Add

Add adds the current erroneous word to the dictionary, thereafter the word is recognized as a valid word. **Add** should only be used for words which have no derivatives, it is generally better to add a new word through the **Edit** interface where a new *base* word may be specified with it's derivatives.

Edit

The **Edit** button executes **spell−edit−word** giving the current erroneous word. This allows new words and auto−corrections to be defined as well as existing words to be altered, see full description below.

Find

The **Find** button executes **find−word** giving the current word as a starting seed. This allows the user to search for the word using a simple search criteria, see full description below.

Undo Last

The **undo Last** button restores the user to the previous spelling so that it may be re−entered, any replacement text that was made is restored to it's original spelling.

Exit

Exits the speller and returns the user to the buffer.

spell−word checks a single word which is either supplied by the user, or if an argument is given, the word under (or to the left of) the cursor position. If the word is correct, a simple message−line print−out is given, otherwise the main spell **osd** dialog is opened and the user may check the spelling within the context of the spell dialog as described above.

The default key binding of "`esc $`" supplies an argument forcing **spell−word** to check the current buffer word. **spell−word** is often used to check the spelling of a word outside of the context of the editor (i.e. when working on paper, or when doing at that prize crossword !!).

spell−edit−word allows words in dictionaries to be altered as well as new words and auto−corrections to be defined. On invocation, an **osd** dialog is opened and changes are made through this interface, defined as follows:−

Word

The **word** entry to be changed or added. If **spell−edit−word** was executed via spell−buffer **Edit** button, this will be set to the current word.

No word set

The word entry is empty, most of the functionallity will not be available until a word is entered.

New Word

To add a new word, the derivatives of the new word should be selected using the prefix and suffix options. Note that not all derivatives are listed, only one example derivative of each spell rule is given.

> **BE CAREFUL WITH THE CASE OF THE BASE WORD:** new words that are entered are case sensitive, as a general rule the *word* in the **Word** text box should be edited to it's base form and should be presented in lower case characters (unless it is a proper name, in which case it should be capitalized, or is an abbreviation or acronym when it might be upper

case).

When the appropriate derivatives of the new word have been selected, it may be added to the dictionary using the **Add** button. This adds the word to the users personal dictionary. Please note that if there are numerous standard words missing then check that an *extended* dictionary (designated by **Ext** in the language) is being used, the extended dictionaries more than double the repertoire of words available.

Words added to the dictionary may be subsequently removed using the **Delete** button, typing the existing word in the **Word** entry and selecting **Delete** button removes the existing word.

Auto−Correct

Selection of the **Auto−Correct** button allows a replacement word to be entered in the **To** text entry. Selecting **Add** adds the automatic correction to the speller. Thereafter, whenever the erroneous word is encountered the replacement word is always used to replace it, without user intervention.

Entering an exiting *auto−correct* word into the dialog and selecting **Delete** removes an existing auto−correct entry.

Exit

Exits the **Edit** dialog.

find−word opens the word finder dialog. This allows the user to search for a word using a simple search criteria. (This is ideal for cheating at crosswords !!). The word to be searched for is entered into the **Word Mask** and may use wild cards ***** to represent any number of characters, **?** to represent an unknown character and **[..]** for a range of characters.

For example, searching for `t?e?e` presents the list *theme*, *there* and *these*. Searching for `t*n` lists all of the words beginning with `t` and ending in `n`. See $find−words(5) for a full discription of the format used by search engine.

The words that match are returned in the scrolling dialog, and may be selected with the mouse (or cursor keys). The **Insert** button inserts the selected word into the current buffer or into the **Word** entry if executed from the **spell−buffer** dialog. Note that the list presented is limited to 200 words, selecting **next** gets the next 200 words, and so on. The **Exit** button exits the dialog.

NOTES

The words added to the speller during a MicroEmacs session are saved when the editor is closed. The user is prompted to save the dictionary, if the dictionary is not saved then any words added are lost.

All *ignore* words accumulated during a spell session are lost when the editor is closed. In order to retain *ignore* words, it is suggested that they are added to the personal dictionary rather than be ignored.

The personal spelling dictionary is typically called *<user><type>*.edf, and is stored in the default user location. The dictionary names are specified in the user−setup(3) dialog.

find−word may claim to have found more words than are actually listed. The use of derivatives in the spell algorithm allows a single word to be present several times. **find−word** counts each occurrence but it is only listed once.

SEE ALSO

user−setup(3), Locale Support, osd(2), spell(2), describe−word(3), $find−words(5).

find−zfile(3)

NAME

find−zfile − Compressed file support
zfile−setup − Compressed file support setup

SYNOPSIS

find−zfile "*file−name*"
zfile−setup "*extension*" "*list−command*" "*cut−to*"

"*column*" "*file−end*" "*extract−command*"
"*remove−command*" **DESCRIPTION**

find−zfile provides generic support for listing and extracting the contents of compressed files. **find−zfile** also supports the extraction of the internal files into another buffer.

find−zfile must be configured for each compression format using **zfile−setup**. It relies on command−line programs to generate content lists which are used to generate the main file listing, and subsequently, the ability to extract individual files for file extraction support.

For basic content listing support the first 3 arguments must be given to zfile−setup. The first argument "*extension*" is used as the compressed file id string. The compressed file type is derived from the file extension, e.g. "`zip`" or "`Z`" for UNIX compressed files. The exact case of the extension is checked first, followed by the lower case and upper case string.

The compressed file contents list is generated from executing the user supplied "*list−command*" and dumping the output into the list buffer. The command is run from the directory containing the compressed file and the following special tags may be used within the "*list−command*" which get substituted as follows:−

%zb

The token is replaced with the compressed files base name, i.e. the file name without the path.

%zf

The token is replaced with the compressed files absolute file name, i.e. the file complete with the path.

The head of the list output is often unwanted verbose printout, this can be automatically be removed by the use of the "*cut−to*" argument. The argument, if supplied (not an empty string), must be a regex search string matching the start of the required list. If found, all text before it is removed.

For single file extraction support the last 4 arguments must be specified by **zfile−setup**. The file to extract is selected either by selecting the file name using the left mouse button or by moving the cursor to the line containing the file name and pressing the "`return`" key.

find−zfile assumes that the file name starts at a fixed column number, specified with the "*column*" argument. The end of the file name is obtained by searching for the regular expression "*file−end*" string, the file name is assumed to end at the start of the search string match.

The file is then extracted by executing the supplied "*extract−command*" and then loading the extracted file into a new buffer. The command is run from the system temporary directory (i.e. "`/tmp/`" on UNIX or `$TEMP` on Windows etc.). The following special tags may be used within the "*extract−command*" which get substituted as follows:−

%zb

The token is replaced with the compressed files base name, i.e. the file name without the path.

%zf

The token is replaced with the compressed files absolute file name, i.e. the file name complete with the path.

%fb

The name of the file to be extracted.

The file is assumed to be extracted to the temp directory due to the way the command is run, this file is then loaded into a new buffer. The temporary file should then be removed using the supplied "*remove−command*" with is run from the temp directory, the "**%fb**" special tag may be used in the command. This argument may be given as an empty string, thereby disabling the removal.

EXAMPLE

For zip file support the freely available **unzip(1)** command can be used, following is the list of arguments with suitable entries:

```
extension           zip
list−command        unzip −v %zb
cut−to              ^ Length
column              58
file−end            $
extract−command     unzip −o %zf %fb
remove−command      rm %fb
```

For the zip file "*/usr/jasspa/memacros.zip*", after substitution the list command becomes "`unzip −v memacros.zip`" which will be executed in the "*/usr/jasspa/*" directory. This will produce the following form of output:

```
Archive:  memacros.zip
 Length  Method   Size  Ratio   Date    Time   CRC−32     Name
```

```
      ------  ------   ----  -----   ----    ----   ------     ----
         565  Defl:N    258  54%  02-27-99  22:56  018a7f70   american.emf
        3409  Defl:N    872  74%  02-28-99  01:37  6a6f9722   americar.emf
        4201  Defl:N    772  82%  03-01-99  12:58  d4e3bc4a   benchmrk.emf
         565  Defl:N    258  54%  02-27-99  22:56  dd394e24   british.emf
        3408  Defl:N    872  74%  02-28-99  01:37  32f3eeca   britishr.emf
        7239  Defl:N   1923  73%  02-28-99  15:13  d408f0da   calc.emf
        7292  Defl:N   2072  72%  01-23-99  12:49  5979d6b2   cbox.emf
        7104  Defl:N   1402  80%  02-28-99  15:13  6faf4fc5   cmacros.emf
        5967  Defl:N   1239  79%  02-13-99  16:38  27601523   ctags.emf
        1097  Defl:N    489  55%  02-16-99  10:58  53a55e36   dos.emf
         562  Defl:N    310  45%  01-16-98  07:54  ec24f65e   dos2unix.emf
 .
 .
 .
```

The top Archive line is not require, this is automatically removed by setting the "*cut−to*" to "`^ Length`" which matches the start of the next line.

For file extract, consider the file "`ctags.emf`", the first character '`c`' is at column 58 and the first character after the end of the file name is the end−of−line character ('`\n`') which is matched by the regex string "$", hence the settings on "*column*" and "*file−end*". When this and the zip file name are substituted into the extract−command, it becomes "`unzip -o /usr/jasspa/memacros.zip calc.emf`" and is run from the "*/tmp.*" directory. Note that the "`-o`" option disables any overwrite prompts, these are not required as tests and prompting have already been performed by **find−zfile**. The extracted file "*/tmp/calc.emf*" is then loaded into a new buffer.

The temporary file is removed by executing the substituted remove−command which becomes "`rm calc.emf`" from the "`/tmp/`" directory.

For gzipped tar files, extension "**tgz**" the following setup can be used on UNIX platforms:

```
extension           tgz
list-command        unzip -v %zb
cut-to
column              43
file-end            $
extract-command     gunzip -c %zf | tar xof - %fb
remove-command      rm %fb
```

For the tgz file "*/usr/jasspa/memacros.tgz*", this will produce the following listing:

```
tgz file: /usr/jasspa/memacros.tgz


rw-rw-r-- 211/200    565 Feb 27 22:56 1999 american.emf
rw-rw-r-- 211/200   3409 Feb 28 01:37 1999 americar.emf
rw-rw-r-- 211/200   4201 Mar  1 12:58 1999 benchmrk.emf
rw-rw-r-- 211/200    565 Feb 27 22:56 1999 british.emf
rw-rw-r-- 211/200   3408 Feb 28 01:37 1999 britishr.emf
rw-rw-r-- 211/200   7239 Feb 28 15:13 1999 calc.emf
rw-rw-r-- 211/200   7292 Jan 23 12:49 1999 cbox.emf
rw-rw-r-- 211/200   7104 Feb 28 15:13 1999 cmacros.emf
rw-rw-r-- 211/200   5967 Feb 13 16:38 1999 ctags.emf
rw-rw-r-- 211/200   1097 Feb 16 10:58 1999 dos.emf
```

```
rw-rw-r-- 211/200    562 Jan 16 07:54 1998 dos2unix.emf
.
.
.
```

NOTES

find−zfile and **zfile−setup** are macros defined in `zfile.emf`.

SEE ALSO

find−file(2).

fold−current(3)

NAME

fold−current − (un)Fold a region in the current buffer
fold−all − (Un)Fold all regions in the current buffer

SYNOPSIS

fold−current
fold−all

DESCRIPTION

MicroEmacs '02 provides a generic, albeit course, folding mechanism which is applied to some of the well known file modes. The folding mechanism allows parts of the buffer to be scrolled up and hidden, leaving a residue hilighting marker within the buffer indicating a folded region. A folded buffer typically allows a summary of the buffer contents to be viewed within several windows, hiding the detail of the buffer.

The folding mechanism uses well defined *start* and *end* markers which form part of the syntax of the well known file mode. i.e. in 'C' this is the open and closed braces that appear on the left−hand margin (`{` .. `}`). The intention is that the natural syntax of the text is used to determine the fold positions, requiring no additional text formating or special text tags to be inserted by the user.

fold−current opens and closes a folded region within the buffer. If the current cursor position lies between a *start* and *end* marker then the region between the start and end is folded out and hidden from view, leaving a highlight marker in the buffer. If the fold already exists then, moving the cursor to the folded line and invoking **fold−current** removes the fold marker and reveals the text.

fold−all opens and closes all folded regions within the buffer, if the current state is unfolded then all of the *start*/*end* markers are located and their regions folded. Conversely, if the buffer is currently folded and **fold−all** is invoked, then all folds are removed and the associated text revealed.

CONFIGURATION

In order to utilize the **fold−current/all** commands within a buffer, the *start* and *end* markers have to be initialized for the syntactical contents of the buffer. This is performed within the hook function for the buffer, using the hook−name. Buffer specific variables are defined within the context of the buffer to configure that start and end fold handling. The buffer specific variables are defined as follows, where *xxxx* is the file hook base name.

xxxx−**fold−open**

A regular expression search string used to locate the start of the string. For speed the search string should include a regular expression start or end of line character whenever possible. i.e. in C the open is defined as "`^{`".

xxxx−**fold−close**

A regular expression search string used to locate the end of the string. For speed the search string should include a regular expression start or end line character whenever possible. i.e. in C the close is defined as "`^}`".

xxxx−**fold−mopen**

An integer value that denotes the number of lines to move in a forward or (−ve) backward direction from the *start* line located by the search string to the position in the buffer to be folded. If default value when **mopen** is omitted is 0, starting the fold from the search string line.

xxxx−**fold−mclose**

The relative displacement from the close fold line to the fold position, this is a positive or negative displacement depending on where the fold is to be positioned.

xxxx−**fold−mnext**

Specifies the number of lines to advance before the next search is continued on the fold operation. This is only used by **fold−all**. **EXAMPLE**

The following examples show how the fold variables are set up in each of the buffer modes.

C and C++

C and **C++** fold on the open and close brace appearing in the left−hand margin. The fold variables are defined in `hkc`/`hkcpp.emf` as follows:−

```
set−variable %c−fold−open  "^{"
set−variable %c−fold−close "^}"
```

Given a 'C' function definition:−

```
static void
myfunc (int a, int b)
{
    /* Function body */
}
```

the folded version appears as follows:−

```
static void
myfunc (int a, int b)
}
```

emf

MicroEmacs macro files **emf** support folding of macro definitions, the fold variables are defined in `hkemf.emf` as follows:−

```
set-variable %emf-fold-open  "^0? ?define-macro"
set-variable %emf-fold-close "^!emacro"
set-variable %emf-fold-mopen "1"
```

Given a macro definition:−

```
0 define-macro mymacro
; This is the body of the macro
; ... and some more ...
!emacro
```

the folded version of the macro is defined as:−

```
0 define-macro mymacro
!emacro
```

nroff

nroff is configured for manual pages only and folds between `.SH` and `.SS` sections, the hook variables are defined as follows:−

```
set-variable %nroff-fold-open  "^\.S[SH]"
set-variable %nroff-fold-close "^\.S[SH]"
set-variable %nroff-fold-mopen "1"
set-variable %nroff-fold-mnext "-1"
```

Given an nroff block of text defined as:−

```
.SH SYNOPSIS
.\" Some text
.\" Some more text
.SH DESCRIPTION
```

Then the folded version appears as:

```
.SH SYNOPSIS
.SH DESCRIPTION
```

tcl/tk

tcl/tk is configured to fold procedures. The fold variables are defined as:−

```
set-variable %tcl-fold-open  "^proc "
set-variable %tcl-fold-close "^}"
set-variable %tcl-fold-mopen "1"
```

Given a tcl procedure definition:−

```
proc tixControl:InitWidgetRec {w} {
    upvar #0 $w data

    tixChainMethod $w InitWidgetRec

    set data(varInited)   0
    set data(serial)    0
}
```

The folded version of the same section appears as:−

```
proc tixControl:InitWidgetRec {w} {
}
```

NOTES

fold−current and **fold−all** are macros implemented in `fold.emf`. The folding is performed using the narrow−buffer(2) command.

fold−current may also be bound to the mouse using the user−setup(3). The typical binding is `C−mouse−drop−1`.

SEE ALSO

File Hooks, user−setup(3), narrow−buffer(2).

ftp(3)

NAME

ftp − Initiate an FTP connection

SYNOPSIS

ftp

DESCRIPTION

ftp initiates a File Transfer Protocol (FTP) connection to a remote host on the network. Using FTP, editing of files may be performed in much the same way as on the local file system. Directory listings may be retrieved and traversed using the mouse or cursor keys. Using the directory listing, files may be transfered to/from the remote host to the local machine.

On issuing the command then a dialog is presented to the user which is used to open the connection. The dialog entries are defined as follows:−

Registry File

The name of a MicroEmacs registry file which is used to store the FTP information. If a registry name is provided then all FTP address information is stored in the registry file and saved for later sessions. Be aware that password information is saved in this file as plain text if a password is entered into the site information.

If the registry information is omitted then the information is not saved between sessions.

Site Name

An ASCII pseudo name for the remote host. The pull−down menu may be used to select existing sites that have been previously entered.

Host Address

The address of the host, this may be an IP address (`111.222.333.444`) or a DNS name (i.e. `ftp.mysite.com`).

User Name

The login name for the site. If this is omitted then `guest` is used by default.

Password

The password used to enter the site for the given login name. If the password is NOT supplied then the user is prompted for the password when a transaction takes place. If the password is omitted and left to promt then it is not stored in the registry.

Take note of the comments provided above regarding the password information.

Initial Host Path

The starting directory at the remote host. If this is omitted then the root directory ('`/`') is used by default.

On selecting **Connect** then a FTP connection is opened and the initial directory appears as a directory listing, if the initial path is a file then the file is loaded into the editor.

Thereafter the file may be edited within the editor as normal, on a write operation then the file is written back to the host, via FTP.

On opening a FTP connection the progress of the transfer, and the FTP commands issued, may be viewed in the `*ftp−console*` buffer. This buffer may automatically appear depending upon the value of the %ftp−flags(5) variable.

NOTES

ftp is a macro implemented in `ftp.emf`. This uses the underlying command find−file(2) to implement the FTP transfer.

FTP files can be directly loaded and edited using the standard file commands such as find−file(2).

The FTP addresses are retained in a registry file (see erf(8)). The registry file is automatically loaded when MicroEmacs starts up each session. The current site information may be viewed using list−registry(2) and is located at the following registry addresses:−

/url

Data value is file system location of the FTP registry file.

/url/ftp/<*hostName*>

The name of the host to which the connection is to be made.

/url/ftp/<*hostName*>**/host**

The name or IP address of the remote host

/url/ftp/<*hostName*>**/user**

The user name used to log into the remote host.

/url/ftp/*<hostName>***/pass**

The user password to the remote host. If this entry is empty then the user is always prompted for the password when the connection is made.

/url/ftp/*<hostName>***/path**

The initial path at the remote site.

When a FTP connection is initiated then the connection (socket) remains open for a period of approximately 4 minutes from the last transfer time, after that the connection is automatically closed and is re−initiated if required again.

NOTE: For windows platforms then the resultant executable must be built with URL support enabled, for UNIX platforms socket support is automatically enabled.

BUGS

Directory completion is not available when the current working directory is an FTP address. To work around this from the command line, select `<RETURN>` to get a directory listing of the current directory and select the file(s) from the directory to load.

SEE ALSO

%ftp−flags(5), erf(8), find−file(2), file−op(2), list−registry(2).

gdiff(3)

NAME

gdiff − Graphical file difference
%gdiff−com − Gdiff diff(1) command line

SYNOPSIS

gdiff "*version1*" "*version2*"

`%gdiff-com` "*string*"; Default is "`diff -c -w`"

DESCRIPTION

gdiff is a macro utility that facilitates the merging of two files (typically with different modification revisions). The changes between the revisions are hilighted with color, allowing modification regions and lines to be selected for the generation of a newer revision file, which might encompass selected modifications from each of the base revisions.

gdiff executes the **diff(1)** command with the command line set by the %gdiff−com(5) variable and the user supplied *version1* and *version2*. The output is displayed in two buffer windows, side by side, and the differences in the lines are hilighted to show the changes. In addition the content of the two buffers is *normalized* such that both windows are aligned at the same line position, allowing the changes in the text to be viewed in both windows at the same time.

Whilst in **gdiff** view mode then both scroll bars (if visible) are *locked*, such that either scrolls BOTH windows at the same time. Other key commands are disabled, as are the menu interactions. The short cut keys are defined as follows:−

`esc h/A-h` − View the help page.

Invokes the display of a OSD help box, summarizing the interaction commands

`C-up` − Move to previous difference

Moves to the previous changed region above the current cursor position.

`C-down` − Move to next difference

Moves to the next changed region below the current cursor position.

`left mouse button`
`space`
`enter`

`r` − Select difference version

Selects the difference version of the currently selected window. The region is hilighted as the required region to be incorporated into the new revision.

`R` − Select neither version.

Marks both regions as not required.

`l` − Line select current version

Selects the current line from the region as being included, without including ALL of the region modifications.

`L` − Line select neither version

Discards lines from both revisions of the file.

`g` − Globally selects the current version.

Shortcut allows ALL modifications to the current side to be accepted. This is typically the fastest method to select all changes, minor region adjustment may then be performed on those regions which are inappropriately included by the selection.

`G` − Globally selects neither version.

Marks all regions as not being acceptable.

`C−x C−s` − Save current side

Saves the current window to the specified file, merging the selected changes between the two revisions. Note that the save only operates iff all hilighted changes have been selected.

`C−x C−w` − Save current side as

Same as **Save current side** except the user is prompted to enter a new filename to which the modifications are written.

`C−x k` − Exit graphical diff

Exits the **gdiff** utility. **Hilighting**

The hilighting within the windows is dependent upon the color scheme selected, in general the following hilights apply:−

normal text

No change

cyan/grey

Addition/removal of line(s)/region(s) between files.

yellow

Modification in line(s)/region(s).

green/red

Selected region, red or green is attributed to a selection for each window. **NOTES**

gdiff is a macro defined in `gdiff.emf`, inspired by the GNU utility of the same name **gdiff(1)**

diff(1) must be executable on the system before **gdiff** can function. The **diff(1)** invocation must include the *context* difference, which annotates the differences with a `+`, `−` or `!` markers. **diff(1)** is typically invoked with the options **−c −w**.

diff(1) is a standard utility on UNIX systems. For Windows 95/NT a version of GNU **diff** may be found at:

> *<ftp.winsite.com/ftp/pub/pc/winnt/misc/gnudiff.zip>*

For MS−DOS users, a DJGPP port of GNU **diff** is also available on the net. A commercial version of **diff** is also available from MKS.

SEE ALSO

compare−windows(2), compile(3), **diff(1)**, gdiff(3f), grep(3), %grep−com(5).

generate−tags−file(3)

NAME

generate−tags−file − Generate a tags file

SYNOPSIS

n **generate−tags−file** ["*tag−command*"]

DESCRIPTION

The **generate−tags−file** command provides an interface to tag file generation. Typically the "*tag−command*" argument will not be required as the current buffer will automatically configure **generate−tags−file** on how tags are generated for the current buffer's file type. See the notes below for more information on configuration.

generate−tags−file supports two different methods of tag generation, firstly via a MicroEmacs macro file and secondly by an external shell command (such as **ctags(1)**). It is generally configured in the current buffer's setup hook.

If a macro file is used a setup dialog is opened if an argument of 0 is given to **generate−tags**. This dialog can be used to configure which type of tags are required and the starting directory (useful when using recursive tags over a source tree). Note that not all tag types are available for all file types.

The generated tags file can then be used by the find−tag(2) command.

NOTES

generate−tags−file is a macro defined in file `gentags.emf`.

generate−tags−file can be configured in one of 2 ways:

When a MicroEmacs macro file (such as `ctags.emf`) is to be used, simply give the name of the macro file to be run as the "*tag−command*" argument. Alternatively set the variable **.**<*$buffer−fhook*>**.tags** to this name, e.g. for C files

```
set-variable .fhook-c.tags "ctags"
```

Note the "`.emf`" extension is assumed.

When an external shell command is to be used, set the *tag−command* to the shell command−line prefixed with a '`!`' character, for example to use **ctags(1)** try the following:

```
set-variable .fhook-c.tags "!ctags *.c *h"
```

Note that the generate−tags dialog is not available in this mode of execution.

SEE ALSO

find−tag(2).

get−next−line(2)

NAME

get−next−line − Find the next command line

SYNOPSIS

get−next−line (**C−x** `)

DESCRIPTION

get−next−line is typically used in conjunction with the compile(3) and grep(3) commands to enable the user to step through errors/locations one by one. The command looks for lines in the form defined by add−next−line(2) in the order of definition. If a match is found the command attempts to find the next error or warning found from the current location (See compile(3)). If the buffer was not found then the next buffer set is searched for, and if found then the next expression from the cursor is automatically located. The command fails if none of the buffers exist, or the end of the buffer is reached.

SEE ALSO

$file−template(5), $line−template(5), add−next−line(2), compile(3), grep(3).

get−registry(2)

NAME

get−registry − Retrieve a node value from the registry.
set−registry − Modify a node value in the registry.

SYNOPSIS

get−registry "*root*" "*subkey*"
set−registry "*root*" *"subkey*" *"value*"

DESCRIPTION

get−registry retrieves the value of a node defined by *root*/*subkey* from the registry into the variable $result(5).

The node name is specified in two components, typically required when iterating over a registry tree, where the *root* component is static and the *subkey* is dynamic, *subkey* may be specified as the null string (`""`) if an absolute registry path is specified.

set−registry adds (or modifies) a new value to the registry. *root* is the root of the new entry and MUST exist or the call fails. *subkey* is the node name (or path) if the path does not exist then it is created. *value* is the value to assign to the node.

DIAGNOSTICS

get−registry fails if the node does not exist, otherwise the registry string is returned in $result(5).

set−registry fails if the *root* node does not exist.

EXAMPLE

The following call

```
set−registry "/history" "foo/win32/printer" "foo−bar"
```

constructs a registry hierarchy of the form:−

```
"history" {
  "foo" {
    "win32" {
      "printer"="foo−bar";
    }
  }
```

```
        }
```

The value of the registry node may be retrieved using:−

```
        get−registry "/history" "foo/win32/printer"
```

which would return "`foo−bar`".

SEE ALSO

find−registry(2), list−registry(2), read−registry(2), &reg(4), erf(8).

global−bind−key(2)

NAME

global−bind−key − Bind a key to a named command or macro
global−unbind−key − "Unbind a key from a named command or macro"

SYNOPSIS

n **global−bind−key** "*command*" "*key*" (**esc k**)
n **global−unbind−key** "*key*" (**esc C−k**)

DESCRIPTION

global−bind−key takes one of the named commands and binds it to a key. Thereafter, whenever that key is struck, the bound command is executed. If an argument *n* is given then the bound command is executed *n* times when the key is struck. (i.e. the command is passed the numeric argument '*n*').

global−unbind−key unbinds (detaches) a user entered *key* sequence (i.e. `C−x C−f`) from any command to which it may be bound. This does not work with buffer or message line key bindings, see buffer−unbind−key(2) and ml−unbind−key(2). If an argument of 0 is given to **global−unbind−key**, only a single key is obtained for the user, if the character is currently bound to the prefix command, the prefix binding and any sub−bindings are removed. **global−bind−key** calls **global−unbind−key** first if the key to be bound is already bound to something else.

If a −ve argument is given to **global−unbind−key** then all bindings are removed, **caution** − removing all bindings interactively will render the current MicroEmacs session unusable. This can only be used within macro development where new bindings are created immediately afterwards.

The **global−bind−key** command, currently bound to `esc k`, prompts the user for the named command and the key to which it is to be bound. This help file gives a complete list of all built in commands, and some useful macros, a complete list of all commands and macros can be obtained by using the command completion (type `esc x tab tab`, see ml−bind−key(2)) or using the command describe−bindings(2).

The mouse buttons are considered to be *keys*, there is a *key* for each button press and release event, use describe−key(2) to get the binding key string.

The non−ASCII standard keys such as the cursor keys have 'standard' key names to make cross platform binding support easy. Some systems such as *termcap* do not have fixed key−bindings, for these key the users must use the command translate−key(2) to convert the system key binding to the standard key binding.

Permanent changes are done indirectly through the `me.emf` file. This is a file that MicroEmacs '02 reads and executes (see execute−file(2)) during startup and hence results in the appearance of a permanent change in the key bindings. The syntax of commands in the `me.emf` file is described under the execute−file command. Of principal concern here are the two commands **global−bind−key** and **global−unbind−key**. The primary difference between the way parameters are passed to these commands in the `me.emf` file is that the keys are not typed in directly (as in the *control−I* key when you want `C−i`) but by symbolic names. Every key has a unique name which can be easily obtained with the current binding by using the command describe−key(2).

See help on Key Names for a description of the symbolic naming system and a complete list of valid key names. Also see Bindings for a complete list of default key bindings.

EXAMPLE

Alt P

```
global-bind-key "func" "A-p"
```

Control F2

```
global-bind-key "func" "C-f3"
```

Shift Alt Left Cursor

```
global-bind-key "func" "A-S-left"
```

Control Alt Delete

```
global-bind-key "func" "C-A-delete"
```

Note that binding **Control−Alt−Delete** is not recommended for MS−DOS systems for obvious reasons.

NOTES

Some ASCII keys, such as `<CR>` (C−m), `<tab>` (C−i), `<BACKSPACE>` (C−h) have non−ASCII key bindings, namely "**return**", "**tab**", "**backspace**" etc. this is to allow separate key−bindings for the real "**C−m**" etc.

Be very careful in binding and unbinding keys since you could get into some very peculiar situations such as being unable to abort out of a command (if you unbind CTRL−G or bind it to something else) or recover from the bad binding/unbinding if you unbind execute−named−command(2) or the **global−unbind−key** command. As long as you leave yourself the opportunity to do both of the last two commands, you can recover from disastrous bindings/unbindings.

SEE ALSO

buffer−bind−key(2), buffer−unbind−key(2), describe−bindings(2), describe−key(2), ml−bind−key(2), ml−unbind−key(2), translate−key(2).

goto−alpha−mark(2)

NAME

goto−alpha−mark − Move the cursor to a alpha marked location

SYNOPSIS

goto−alpha−mark "*?*" (**C−x a**)

DESCRIPTION

goto−alpha−mark prompts user for an alpha character and sets the cursor position to the preset location. Alpha marks are specified on a per buffer basis, thus the current buffer is not changed, merely the current location in the buffer. The alpha mark must already be defined using set−alpha−mark(2). This functionality is useful for rapidly returning back to locations in large files.

SEE ALSO

set−alpha−mark(2).

goto−line(2)

NAME

goto−line − Move the cursor to specified line

SYNOPSIS

n **goto−line** (**esc g**)
goto−line "*num*"

DESCRIPTION

goto−line moves the cursor to the specified line in the buffer. The user is prompted for the new line number on the command line, which may be entered as a relative displacement ([+|−]*number*) from the current position, or as an absolute line number (*number*). If the number is preceded by + or − then this is treated as a relative displacement from the current line, otherwise it is an absolute line number.

If a +ve argument *n* is supplied, **goto−line** moves to this line, e.g. to move the cursor to line 240:

```
240 goto-line
```

A special case of **goto−line** is operative if an argument of 0 is supplied, argument "*num*" must also be given as above except **goto−line** treats the line number or displacement as an absolute move, i.e. includes *narrowed out* sections when calculating the new position. If the new line lies within a narrowed out section (i.e. a section that has been hidden and is not visible on the screen) the narrow is automatically expanded. See narrow−buffer(2) for more information on narrowing.

Supplying a −ve argument to goto−line results in an error.

NOTES

After successfully calling goto−line, variable $window−line(5) is set to the required line number.

SEE ALSO

goto−alpha−mark(2), goto−matching−fence(2), narrow−buffer(2), $window−line(5).

goto−matching−fence(2)

NAME

goto−matching−fence − Move the cursor to specified line

SYNOPSIS

goto−matching−fence (**esc C−f**)

DESCRIPTION

goto−matching−fence moves the cursor to the opposing fence character of the character currently under the cursor. The set of fence characters include `[]`, `{}` and `()`. i.e. if the character under the cursor is `` `{' `` then **goto−matching−fence** moves the cursor to the opening fence `` `}' ``, and visa versa.

goto−matching−fence can also be used to move the cursor to matching C/C++ `#if`, `#elif`, `#else` and `#endif` constructs, cycling through them in the given order.

When the fence(2m) buffer mode is enabled the matching open fence is automatically displayed when the closing fence is typed. The length of time the matching fence is displayed for can be controlled by the $fmatchdelay(5) variable.

SEE ALSO

fence(2m), $fmatchdelay(5), goto−line(2).

set−position(2)

NAME

set−position − Store the current position
goto−position − Restore a stored position

SYNOPSIS

n **set−position** "*label*"
n **goto−position** "*label*"

DESCRIPTION

set−position stores current window, buffer, cursor and mark position information against the given '`label`' (a single alpha−numeric character). **goto−position** takes the positional information stored against the given '`label`' and restores the window, buffer and cursor positions from those previously **set**.

A call to **set−position** with the same label over−writes the previous stored information, a call to **goto−position** does not alter the information and may be restored multiple times.

The numerical argument to **set−position** is used to define the information that is stored in the position item. The argument is intrepreted as a bitmask, flagging what information is to be stored. The bit mask is defined as follows:

`0x001`

Store the current window.

`0x002`

Store the current window's horizonal scroll (value of $window−x−scroll(5)).

`0x004`

Store the current window's current line horizonal scroll (value of $window−xcl−scroll(5)).

`0x008`

Store the current window's vertical scroll (value of $window−y−scroll(5)).

`0x010`

Store the current buffer.

`0x020`

Store the current window's current line using an alpha mark.

`0x040`

Store the current window's current line number (value of $window−line(5)).

`0x080`

Store the current window's current column offset (value of $window−col(5)).

`0x100`

Store the current window's mark line using an alpha mark.

`0x200`

Store the current window's mark line number (value of $window−line(5) when mark was set).

`0x400`

Store the current window's mark column offset (value of $window−col(5) when mark was set).

When *n* is not specified, the default value is `0x0bf`, i.e. store all information required to return to the window, buffer and cursor position.

The argument supplied to **goto−position** similarly interpreted as a bitmask, restoring the positional information. When the numerical argument *n* is omitted the same default is used when omitted on the store. On restoring a position, information stored during the call to **set−position** which is not requested in corresponding **goto** is ignored, similarly information requested in a **goto** which was not stored in the **set** is also ignored.

EXAMPLE

The following example shows the typical use of these commands:

```
set−position "a"
    .
    .
goto−position "a"
```

The following example stores the current position at the start of a macro sequence, if `my−command` is not successful (**$status** equals 0) the original position is restored:

```
set−position "\x80"
!force my−command
!if &equ $status 0
    ; command failed, return to the original position
    goto−position "\x80"
```

```
        !endif
```

Note '`\x80`' is interpreted as the character with the ASCII value of 0x80 which is a non−alphanumeric character, this is permitted in macros to avoid using alphanumerics.

The following example shows how the current position can be restored after re−reading a file:

```
        0xce set−position
        read−file $buffer−fname @mna
        ; a numeric argument of 0xce is not
        ; required as this is the default
        goto−position
```

NOTES

The position item may store and restore the current line by using an alpha mark or the line number. The restrore strategy will determine what is required, as follows:−

The main benefit from using an alpha mark is that the position is maintained even when the buffer is edited, for example if the position is stored at line 10 and a line is subsequently inserted at the top of the buffer, if the line number was used then it would return back to the 10th line which is the old 9th line whereas if an alpha mark were used it would correctly return to the 11th line, as expected.

The disadvantage of using an alpha mark is that it is only associated with that buffer. In some cases a position may need to be restored in another buffer (e.g. when re−reading a buffer the original buffer may be deleted first), in this situation the buffer line number must be used.

Commands **set−window** and **goto−window**, which simple stored and returned to the current window, were replaced by set−position and goto−position in August 2000. The following macro implementations can be used as a replacement:

```
        define−macro set−window
            1 set−position "\x80"
        !emacro

        define−macro goto−window
            goto−position "\x80"
        !emacro
```

SEE ALSO

set−alpha−mark(2), find−buffer(2), $window−x−scroll(5), $window−xcl−scroll(5), $window−y−scroll(5), $window−line(5), $window−col(5).

grep(3)

NAME

grep − Execute grep command rgrep − Execute recursive grep command

SYNOPSIS

grep "*expression files...*" **rgrep** "*expression*" "*base−path*" "*file−mask*"

DESCRIPTION

grep executes the **grep(1)** command with the command line set by the %grep−com(5) variable and the user supplied *expression* and file list *files...*. The output of the command is piped into the ***grep*** buffer ready for the get−next−line(2) command to step through all matched lines. The syntax from the grep output must be setup using add−next−line(2).

If an argument is given then a pipe−shell−command(2) is used instead of ipipe−shell−command(2), this is useful when used in macros as it ensures that **grep** has finished before the command returns.

rgrep is simpler to **grep** in that it uses **grep(1)** to search for all occurrences of *expression*, but **rgrep** also uses **find(1)** to search for *expression* in all files matching the *file−mask* in all directories from *base−path* down.

NOTES

grep is a macro defined in `tools.emf`.

grep(1) must be executable on the system before grep or rgrep can function, **find(1)** must also be available for rgrep to work.

EXAMPLE

The **grep** command is generally set up in the startup files as follows:−

```
;
; setup the next-error stuff including grep and compiling
;
set-variable $line-template "[0-9]+"
set-variable $file-template "[a-zA-Z:]*[0-9a-zA-Z\_.]+"
;
; Definitions for GNU grep utility.
;
set-variable %grep-com "grep -n "
0 add-next-line "*grep*"
```

```
add-next-line "*grep*" "%f:%l:"
```

SEE ALSO

grep(1), %grep−com(5), add−next−line(2), get−next−line(2), compile(3).

help(2)

NAME

help − Help; high level introduction to help
help−command − Help; command information
help−variable − Help; variable information
help−item − Help; item information

SYNOPSIS

n **help** (**esc ?**)
help−command "*command*" (**C−h C−c**)
help−variable "*variable*" (**C−h C−v**)
help−item "*item*" (**C−h C−i**)

DESCRIPTION

The help commands provide a quick on−line help facility within MicroEmacs '02 without invoking a third party documentation system (e.g. a browser such as **Netscape(1)** or **winhelp(1)**).

The on−line help is assisted by a set of macros which enable key words within the help buffers to be located by either tagging (`esc t`) or by selection with the left mouse button. The tag mechanism supports command completion.

help provides general help on the philosophy and functionality of MicroEmacs '02, if an argument *n* of 0 is given to the command it changes the current buffer to the internal help buffer, typically named `"*on-line help*"`. This is a hidden system buffer used to store all the on−line help and can be used for a variety of things. Note that access to this buffer must be via the **help** command, not **find−buffer** and the help command will also ensure the system help file is loaded first.

help−command describes the purpose of the given *command*.

help−variable Describes the purpose of the given *variable*, similar to **help−command**, only for variables.

help−item Describes the purpose of any given item, where item could be a command, variable or any aspect of MicroEmacs '02.

FILES

The help files are ASCII text files located in the MicroEmacs '02 home directory. The files are defined as follows:−

`me.ehf` − Help text file.
`hkehf.emf` − Help macros.

SEE ALSO

osd−help(3), command−apropos(2), describe−bindings(2), describe−key(2), list−commands(2), list−variables(2).

hilight(2)

NAME

hilight − Manage the buffer hilighting schemes

SYNOPSIS

0 **hilight** "*hil−no*" "*flags*" ["*nol*"] ["*buffer−scheme*" ["*trunc−scheme*"]]

hilight "*hil−no*" "*type*" "*token*" [["*rtoken*"]

[(["*close*" ["*rclose*"] "*ignore*"]) |

(["*continue*"]) |
(["*b−hil−no*"])]
"*schemeNum*"
hilight "*hil−no*" "*0x200*" "*token*"
hilight "*hil−no*" "*0x400*" "*from−col*" "*to−col*" "*schemeNum*"

−1 **hilight** "*hil−no*" "*type*" "*token*"

DESCRIPTION

The **hilight** command creates and manages the buffer hilighting, the process of creating a new hilighting scheme is best described in File Language Templates. The command takes various forms as defined by the arguments. Each of the argument configurations is defined as follows:−

Hilight Scheme Creation

0 **hilight** "*hil−no*" "*flags*" ["*nol*"] ["*buffer−scheme*" ["*trunc−scheme*"]]

With an argument of `0`, **hilight** initializes or re−initializes the hilight scheme *hil−no* (1−255). Every buffer has a hilight scheme, the default is 0 which means no hi−lighting and only the $global−scheme(5) etc. are used. The hilighting scheme must be defined before use and is used to specify how the buffer is to be hilighted. MicroEmacs '02 supports the following hilighting concepts:−

- **hilight string**, a user specified string is hilighted in any color scheme.
- **Tokens**, same as a hilight string except that the string must be enclosed in non alpha−numeric characters.
- **Start−of−line hilights**, the start of the hilight must be the first non−white character of the line.
- **End−of−Line hilights**, the hilight starts from the current position and continues until the end of the line. Optionally, the hilight may continue onto the next line if the current line ends in a

given string. A bracket may also be searched for within the line.
- **Bracket hilight**, hi−lights from the current position until the closing bracket token is found.
- **Replace string** , allows the hilight string to be replaced with a different user specified string. (i.e. the displayed representation is different from the buffer contents)

Terminals that cannot display color directly may still be able to take advantage of the hi−lighting. A terminal that has fonts (i.e. *Termcap*) can use them in the same way using the add−color−scheme(2) command. The hi−light scheme is also used in printing (see print−buffer(2)). If your terminal cannot display color in any way, it is recommended that hi−lighting is disabled (except when printing) as it does take CPU time.

The "*hil−no*" argument specifies which hi−lighting scheme is being initialized. Once a hilighting scheme has been initialized, hi−light tokens can be added to it and it can be used by setting the current buffer's $buffer−hilight(5) variable to "*hil−no*". The "*flags*" argument is a bit based flag setting global hi−light characteristics, where:−

0x01

The hi−light scheme is case insensitive, i.e. the following tokens become equivalent:−

`house` == `HOUSE` == `hOuSe`

When the hilight scheme is attributed as case insensitive then the tokens must **all** be specified in **lower** case.

0x02

Set a hi−light look−back. During the process of determining the window hilighting then the hilight process has to determine whether the top of the window starts in a hi−light bracket or not. The look−back command tries looking "*nol*" lines backwards for an open bracket. If an open bracket is found then the top of the window is assumed to start with that bracket, else it is assumed that the top of the window is not in a bracket. For example, in `C', a comment starts with "`/*`" and ends with "`*/`" so if the hilight was initialized with

```
0 hilight 1 2 10 $global-scheme
```

of the following, only the first would begin hi−lighted which is correct (assuming the "`/*`" is 10 or less lines away).

```
 /* ........        /*.........          .........
    ........          .........*/        .........
----------------   ---------------    --------------- top of
    ........*/        .........          .........    window
```

The optional argument "*buffer−scheme*" specifies the default scheme to use if there is no specific hi−light, when omitted the value of $global−scheme(5) is used. The *buffer−scheme* is a reference to a set of foreground and background color pairs previously defined with add−color−scheme(2). The last argument "*trunc−scheme*" is also optional and specifies the line truncation scheme, when omitted the value of $trunc−scheme(5) is used.

The hi−lighting scheme required is based on the type of file being edited and so is usually directly related to the file extension, thus it can be automatically set using file hooks (see add−file−hook(2)).

Hilight Scheme Token Creation

hilight "*hil−no*" "*type*" "*token*" [["*rtoken*"]

[(["*close*" ["*rclose*"] "*ignore*"]) |

(["*continue*" ["*rcontinue*"]]) |
(["*b−hil−no*"])]
"*schemeNum*"
hilight "*hil−no*" "*0x200*" "*token*"
hilight "*hil−no*" "*0x400*" "*from−col*" "*to−col*" "*schemeNum*"

With the default argument of `1`, **hilight** creates a hilight token to be used in hilight color scheme identified by "*hil−no*" (1−255) (see the section on **Hilight Scheme Creation** for a overview of hi−lighting). The second argument "*type*" specifies the token type and must always be specified, it determines which other arguments required.

Typically the last argument, *schemeNum*, is also required. This identifies the color scheme to use when hilighting the token, defining the foreground, background and selection color schemes. This is an index generated from add−color−scheme(2). If the *schemeNum* argument is omitted the default hilght color scheme is used.

The token "**type**" is a bit based flag of which 0, 1 or more of the bits may be set, the effect of the bits are defined as follows:

0x001

The "*token*" must be surrounded by non−word characters (word characters are typically the alpha−numeric characters), e.g. the following defines "`if`" as a token:

```
hilight 1 1 "if" .scheme.keyword
```

this hilights the 'if' in " `if` " but not in "`aifa`".

0x002

Color this to the end of the line, often used for comments etc. For example in MicroEmacs macro language a '`;`' character signifies the rest of the line as a comment, hilighting is defined as follows:

```
; this is a comment line
hilight 1 2 ";" .scheme.comment
```

0x004

This is a bracket token, the closing bracket string "*close*" and an ignore character "*ignore*" must also be supplied. The ignore character indicates that when found it should ignore the next character; this prevents an early end on bracket miss−match. For example, in C a '"' character can be inserted into a string by 'protecting' it with a '\' character, such as "*this is a string with a \" in it*". In this example the ignore character should be '\' so the mid string '"' is correctly ignored, as follows:

```
hilight 1 4 "\"" "\"" "\\" .scheme.string
```

An empty value, "", effectively disables the ignore feature. If replacing bit `0x040` is set the replacement close bracket "*rclose*" must be supplied.

0x008

The token has a continuation string, usually used with 0x02 but cannot be used with token types `0x004` and `0x080`. The argument "*continue*" must be supplied and if the replacing bit `0x040` is set the replacement continue string "*rcontinue*" must also be supplied. The best example of its use can again be found in C; macros defined using the `#define` pre−processor construct may be constructed on single or multiple lines. The macro continues onto another line if the current line ends with a backslash '\' character, e.g.:

```
#define a_single_line_macro() printf("hello world\n")

#define a_four_lined_macro()          \
do {                                  \
    printf("hello world\n") ;         \
} while(0)
```

This can be correctly hilighted with the pre−processor scheme using:

```
; use to−end−of−line (2) and continuation (8), i.e. 2+8=10
hilight 1 10 "#" "\\" .scheme.prepro
```

0x010

If this is an end of line token (`0x002`) then

The rest of the line is checked for any valid brackets.

Else if this is a bracket token (`0x004`) then

This is still searched for after an end of line token is found.

Else

Ignored

This feature enables the searching and hilighting of specific brackets contained within a to−end−of−line scheme. For example, consider the following C code:

```
#define My_Token 0x01  /* This is a multi−lined comment
```

```
                                        * describing My_Token */
```

With the '#' pre−processor hilight (see bit 0x08 above) the #define line would all be hilighted with the pre−process scheme, the comment would be missed causing incorrect hilighting of the next line. Instead this feature may be used by both the pre−processor and comment hilight tokens to correctly hilight the above example:

```
hilight 1 26 "#" "\\" .scheme.prepro
hilight 1 20 "/\\*" "*/" "" .scheme.comment
```

0x020

This token must be the first non−white character of the line.

0x040

The token (and closing bracket tokens) are to be replaced by the given replacement strings. This is often utilized when displaying formated text such as MicroEmacs on−line help ehf(8) pages, the output from UNIX **man(1)** etc. In MicroEmacs help pages, the start of bold text is delimited with the character sequence "`\C[cD`" and ends with the character sequence "`\C[cA`", e.g.

```
"the word \C[cDbold\C[cA is in \C[cDbold\C[cA"
```

Obviously the hilight delimiters should not appear so the character sequence may be correctly drawn using a bracket token, starting with "`\C[cD`" and ending with "`\C[cA`", replacing both with an empty string:

```
hilight 1 0x44 "\C[cD" "" "\C[cA" "" "" .scheme.bold
```

0x080

This is a branch token. When this token is found, the token (or the replace string) is colored using the given color *schemeNum* and then the current hilighting scheme is changed to "*b−hil−no*" (which MUST be defined by the time it is first used). The "*b−hil−no*" hi−light scheme should also contain a branch token which branches back to "*hil−no*" or "`0`" (which branches to $buffer−hilight(5)). A branch does not have to branch back to "*hil−no*", it may branch to any other hi−light scheme. The branches are not stacked and there is no limit on the nesting.

0x100

The token must be at the start of the line.

0x200

This is an invalid token in its own right, which is used for optimizing a hi−lighting scheme.

This has the second highest precedence (see **0x400**) and all other bits are ignored. Only the first 3 arguments are required. For example, if there are 11 tokens starting with "`delete−`" as with the hi−lighting of this buffer, then adding the token "`delete−`", while invalid in its own right, means that "`delete-`" is only checked for once. This also reduces the size of the internal hilighting tables so if the message "**Table full**" appears, the hilighting scheme should be reduced by removal of the common components.

0x400

This is a column hilighting token, which allows absolute columns within a window to be hilighted (irrespective of the contents). This bit takes precedence over all other bits and all other bits are ignored. Column highlighting is a different concept to token in that it requires a "*from−col*" and a "*to−col*" column positions and a line will be hilighted in the given scheme between these two columns.

0x800

The flag is used with bracket tokens (`0x04`) and indicates that the bracket is typically contained on a single line. This information is used by MicroEmacs in trying to avoid hilighting anomalies caused when the start and end tokens of the bracket are the same (e.g. a string's start and end token is '`"`'). Problems arise when the bracket starts on one line and closes on a later line, even with a large look−back, eventually the start bracket will become too far back and only the end bracket is found. But as this is the same as the open token it is mistaken for an open bracket and the strings become out of synch. This test can reset this if further down the file an open and close bracket is found on the same line. For this to have any effect, the hilighting scheme must use look−back (flag `0x02` of **Hilight Creation**).

Note that `0x004`, `0x008` and `0x080` are mutually exclusive and more than 1 should not be set in any one hilight token, if 2 or more are set the effect is undefined. Other than this there is no restrictions placed on the types of token used, although strange combinations like `0x006` may lead to unexpected results −− hopefully not a core dump, but not guaranteed !

The token and close token of brackets may contain a limited subset of regular expression tokens as follows:−

^

When specified as the first character of the token, the token must be at the start of the line.

$

The token must be at the end of the line, must be the last character.

\{

Indicates the start of the hilighted part of the token, only one may be used per token. This token use is different from regex.

\}

Indicates the end of the hilighted part of the token, only one may be used per token. The rest of the token must be matched for it to be used but is not considered part of the token, i.e. hilighting continues on the character immediately after the "\}", not at the end of the token. Similar to the \< token, the length of the rest of the token must be fixed. This token use is different from regex.

\(.\)

Groups are supported in hilighting, but they must only enclose a single character, closures ('*', '?' and '+') must come after the closure, i.e. use "\(.\)*", not "\(.*\)". Alternatives ("`\|`") are not supported.

.

Matches any character.

[...]

Matches a single buffer character to a range of characters, for example to hilight MicroEmacs register variables (i.e. `#g0−#g9`, `#p0−#p9`, `#l0−#l9`) the following regex string may be used:

```
hilight 1 1 "#[gpl][0−9]"
```

This matches a token which starts with a '#', followed by a '`g`', '`p`' or '`l`' character and ends with a numerical digit. If the user required the replacement (bit 0x40) of the "#" to "#register" to aid readability, the replacement string some now needs to know whether the second character was a '`g`', '`p`' or '`l`' and which digit. Up to 9 groups ("`\(.\)`") can be use to store a store a single search character, which can be used later in the search string and in the replacement string by using the form "\#", where # is the range test number counting from the left, e.g. for the given example use:

```
hilight 1 65 "#\\([gpl]\\)\\([0−9]\\)" "#register\\1\\2"
```

The content of the brackets (**[...]**) include a set of special short cuts and regular expression syntax definitions as follows:−

`[abc]`

A list of characters.

`[a−z]`

A range of characters.

`[−.0−9]`

A combination of character lists and ranges.

`[[:space:]]`

A white space character. See set−char−mask(2) for a full description on MicroEmacs character range support.

`[[:digit:]]`

A digit, 0−9.

`[[:xdigit:]]`

A hexadecimal digit, 0−9, a−f, A−F.

`[[:lower:]]`

A lower case letter, by default a−z.

`[[:upper:]]`

An upper case letter, by default A−Z.

`[[:alpha:]]`

A lower or upper case letter.

`[[:alnum:]]`

A lower or upper case letter or a digit.

`[[:sword:]]`

A spell word character.

[^...]

Matches all characters except the given range of characters, e.g. "`[^[:space:]]`".

`\#`

The same character which matched the #th group token. This functionality is best explained using UNIX **man(1)** output as an example, to create a bold character '**X**' it produces "`X\CHX`" where `\CH` is a backspace character thereby overstriking the first '`X`' with another creating a bold character. This can be checked for and simulated in MicroEmacs using the following:

```
hilight 1 64 "\\(.\\)\CH\\1" "\\1" .scheme.bold
```

The use of "`\1`" in the search string ensures that the second character is the same as the first. This is replace by a single character drawn in the bold scheme.

? + *

Matches 0 or 1, 1 or more and 0 or more of the previous character or character range respectively.

Following is a list of hilighting regular expression restrictions:

The number of characters to the left of a **\{** and to the right of a **\}** token must be fixed, i.e. the '**?**', '**+**' and '*****' tokens cannot be used before this token. Consider the hilighting of a C function name defined to be a token at the start of a line followed by 0 or more spaces followed by a '('. The following hilight token looks valid but the variable space match is incorrect as it is to the right of the **\}**:

```
hilight 1 0 "^\\w+\\}\\s-*(" .scheme.function
```

Instead either the space match must be include in the function token hilighting (which may cause problems, particularly if printing with underlining) or by fixing the number of spaces as follows:

```
; include the spaces in the function hilighting
hilight 1 0 "^\\w+\\s-*\\}(" .scheme.function
; or fix the number of spaces to 0, 1 ...
hilight .hilight.c    0 "^\\w+\\}(" .scheme.function
hilight .hilight.c    0 "^\\w+\}\\s-(" .scheme.function
```

The **+** and ***** tokens match the longest string and do not narrow, e.g. consider the hilighting of a C goto label which takes the form of an alpha−numerical name at the start of a line followed by a ':' character. The token "`^.*:`" cannot be used as `.` will also match and move past the ending ':', ending only at the end of the line. As no narrowing is performed the final '`:`' in the token will not match and the label will not be hilighted. Instead a character range which excludes a ':' character must be used, e.g. "**^[^:]*:**".

A group should not be followed by a **?** or ***** closure, it should either be a fixed single character or followed by a **+** closure (in which case the last matching character is stored).

Following is a list of hilight type bit / token regex equivalents:

0x01

`"[^word]\{????\}[^word]"`

0x02

`"????.*"`

0x20

`"^\s-*\{????"` − (note that this is strictly incorrect as the `\s-*` is to the left of the `\{`, it is correctly handled for the ease of use).

0x100

`"^????"` **Hilight Scheme Token Deletion**

−1 **hilight** "*hil−no*" "*type*" "*token*" With a −ve argument **hilight** deletes the given "*token*" from a hi−light color scheme identified by "*hil−no*". The token "*type*" must also be specified to distinguish between normal and column token types.

EXAMPLE

Example 1

Hilighting a MicroEmacs character given in hex form, checking its validity (i.e. "`\x??`" where `?` is a hex digit):

```
hilight 1 0 "\\x[[:xdigit:]][[:xdigit:]]" .hilight.variable
```

Hilighting a C style variable length hex number (i.e. "`0x???`"):

```
hilight 1 1 "0[xX][[:xdigit:]]+" .hilight.variable
```

Example 2

Replacing a quoted character with just the character (i.e. `'x'` −> `x`)

```
hilight 1 64 "'\\(.\\)'" "\\1" %magenta
```

Example 3

The following example uses the branch hilighting feature to hilight each window line a different color to its neighbors by cycle through 3 different color schemes:

```
0 hilight .hilight.line1 0                        $global−scheme
  hilight .hilight.line1 0x80 "\\n" .hilight.line2 .scheme.no1
0 hilight .hilight.line2 0                        .scheme.no1
  hilight .hilight.line2 0x80 "\\n" .hilight.line3 .scheme.no2
0 hilight .hilight.line3 0                        .scheme.no2
  hilight .hilight.line3 0x80 "\\n" .hilight.line1 $global−scheme
```

Example 4

Simulate the hilighting from the output of a UNIX man page (taken from hkman.emf):

```
0 hilight  .hilight.man 0                                   $global−scheme
; ignore
hilight .hilight.man 64 ".\CH" ""                           $global−scheme
; normal underline/italic
hilight .hilight.man 64 "_\CH\\(.\\)\\}[^\CH]" "\\1"      .scheme.italic
hilight .hilight.man 64 "\\(.\\)\CH_\\}[^\CH]" "\\1"      .scheme.italic
; bold − first is for nroff −man
hilight .hilight.man 64 "\\(.\\)\CH\\1\\}[^\CH]" "\\1"    .scheme.bold
hilight .hilight.man 64 "_\CH_\CH_\CH_\\}[^\CH]" "_"      .scheme.header
hilight .hilight.man 64 "\\(.\\)\CH\\1\CH\\1\CH\\1\\}[^\CH]" "\\1" .scheme.header
; bold underline
hilight .hilight.man 64 "_\CH_\CH_\CH_\CH_\\}[^\CH]" "_" .scheme.italic
hilight .hilight.man 64 "_\CH\\(.\\)\CH\\1\CH\\1\CH\\1\\}[^\CH]" "\\1" .scheme.ita
```

This replaces the complex nroff character string with a single hi−lighted character (don't believe me, try it!).

NOTES

MicroEmacs hilight was written with speed and flexibility in mind, as a result the user is assumed to know what they are doing, if not the effects can be fatal.

SEE ALSO

File Language Templates, $buffer−hilight(5), add−file−hook(2), add−color−scheme(2), print−scheme(2), indent(2), $system(5), print−buffer(2).

hunt−forward(2)

NAME

hunt−forward − Resume previous search in forward direction hunt−backward − Resume previous search in backward direction

SYNOPSIS

n **hunt−forward** (**C−x h**)
n **hunt−backward** (**C−x C−h**)

DESCRIPTION

hunt−forward repeats the last search with the last search string in a forwards direction, from the current cursor position. magic(2m) and exact(2m) modes are operational.

hunt−backward repeats the last search with the last search string in a backwards direction, as per **hunt−forward**.

The numeric argument *n* is interpreted as follows:−

n > **0**

The *n*th occurrence of the pattern is located.

n < **0**

The first occurrence of the pattern is located in the next *n* lines. **DIAGNOSTICS**

The command returns a status of `FALSE` if no previous search string has been established, or if the pattern could not be located (or *n*th pattern where *n* occurrences are requested). If the pattern is found within the given search criteria the return status is `TRUE`.

SEE ALSO

exact(2m), isearch−forward(2), magic(2m), search−backward(2), search−forward(2), Regular Expressions

ifill−paragraph(3)

NAME

ifill−paragraph − Format a paragraph

SYNOPSIS

n **ifill−paragraph** (**esc q**)

DESCRIPTION

ifill−paragraph, like **fill−paragraph**, fills the current paragraph from the left margin to the current fill column. In addition ifill−paragraph also recognizes joined bullet lists and fills each bullet paragraph separately.

See fill−paragraph(2) for more information on the process of filling paragraphs.

EXAMPLE

Following are 2 copies of the same paragraph, the first has been filled using **ifill−paragraph**:

```
This  is the  main  paragraph  which  can be as long as  required,
following is a list of bullets, some with a sub−bullet  list. Here
is the list:
    a) The bullet paragraph can also be as long as required and it
       also can have a bullet  list  following  (sub−bullet  list)
       which will also be filled correctly. Here is the sub−bullet
       list:
       1. First  sub−bullet − again no length  restrictions,  this
          will be filled correctly.
       2. second sub−bullet − no problems.
       3. Third sub−bullet − again no length restrictions, this is
          getting boring.
    b) This is the second  major bullet and this can just carry on
       for ever, but all things must come to an
```

The following version has been filled using the normal **fill−paragraph**:

```
This  is the  main  paragraph  which  can be as long as  required,
following is a list of bullets, some with a sub−bullet  list. Here
is the  list:  a) The  bullet  paragraph  can  also  be as long as
required and it also can have a bullet list following  (sub−bullet
list) which will also be filled  correctly. Here is the sub−bullet
list: 1. First  sub−bullet  − again no length  restrictions,  this
will be filled  correctly. 2. second  sub−bullet − no problems. 3.
Third sub−bullet − again no length  restrictions,  this is getting
boring. b) This is the second major bullet and this can just carry
on for ever, but all things must come to an
```

NOTES

ifill−paragraph is a macro defined in `format.emf`.

SEE ALSO

fill−paragraph(2), paragraph−to−line(3).

indent(2)

NAME

indent − Manage the auto−indentation methods

SYNOPSIS

0 **indent** "*ind−no*" "*flags*" "*look−back*"

indent "*ind−no*" "*type*" "*token*" ["*close*" ["*ignore*"]] ["*indent*"]

DESCRIPTION

The **indent** command creates and manages the auto−indenting methods, the process of creating a new indentation method is best described in File Language Templates. The command takes various forms as defined by the arguments. Each of the argument configurations is defined as follows:−

Indentation Method Creation

0 **indent** "*ind−no*" "*flags*" "*look−back*"

With an argument of `0`, **indent** creates a new indentation method with the integer handle *ind−no*. The indentation method is assigned to a buffer by setting $buffer−indent(5) to *ind−no*. *ind−no* cannot be 0 as setting **$buffer−indent** to zero disables indentation. If the indentation method with the same *ind−no* already exists, then the existing method is deleted and a new method may be created.

flags Sets the indent bit flags where:−

`0x01`

Indent method is case insensitive. Note that **indent** tokens must be specified in lower case.

look−back specifies the maximum number of lines, prior to the current line, considered when calculating the indentation of a line, i.e. if there are *look−back* number of lines between the line to be indented and the previous non−blank line then the current indentation is lost.

If *look−back* is set to 0 then the indentation is effectively disabled as the current indentation can never be found. The value may be specified in the range 0−255, a value of 10 is typically sufficient.

Indentation Rule Creation

indent "*ind−no*" "*type*" "*token*" ["*close*" ["*ignore*"]] ["*indent*"]

With the default argument of `1`, **indent** creates a new rule for the indentation method *ind−no* which must have previously been defined and initialized.

The indentation of a line in a buffer, which is using an indentation method, is affected by the token types matched on the line (*type* `f`, `o`, `s`) and the current indentation (if line is not of type `f`).

The current indentation is determined by searching the previous lines (look−back) for the indentation of the last indented line. This may not simply be the indentation of the last non−blank line, the exact indentation is determined by searching for tokens in the line and assessing their effect on the indentation of the current line.

The format of the regex valid in the "*token*" and "*close*" arguments are the same as at used by hilight token creation, see hilight(2) for more information.

The indent tokens may be assigned one of the following types, using the *type* argument. If the type is specified in upper case then the token must be surrounded by non−alpha−numeric characters:

Fixed (*type* = '`f`' or '`F`')

A line containing a fixed indent token will be indented to the given *indent* column from the left−hand edge. *indent* is the only argument specified. e.g. MicroEmacs macro `!goto` labels:−

```
indent .hilight.emf f "*" 0
```

producing

```
        .....
*label
        .....
```

The fixed token must be the first non−white character on the line, the rest of the line is ignored. The indentation of the previous line has no effect.

Indent−from−next−line−onward (*type* = '`n`' or '`N`')

The indentation changes by *indent* from the next line onwards from the current line. *indent* is the only argument specified. e.g. MicroEmacs macro `!if`:−

```
indent .hilight.emf n "!if" 4
```

Keeps the indentation of the `!if` line the same as the previous indentation, change the indentation on the following lines by an extra 4 characters, to produce:

```
....
!if
    ....
```

Indent−from−current−line−onward (*type* = '`o`' or '`O`')

Increment the current and following lines indentation by *indent*. *indent* is the only argument specified. e.g. MicroEmacs macro `!endif`

```
indent .hilight.emf o "!endif" -4
```

decrement the indent of the `!endif` line and following lines by 4 spaces producing:

```
    ....
!endif
....
```

Indent−single (*type* = '`s`' or '`S`')

Changes the indentation of the current line ONLY by *indent*. *indent* is the only argument specified. e.g. MicroEmacs macro `!elif`:−

```
indent .hilight.emf o "!elif" -4
```

decrements the indentation of the `!elif` line by 4 characters, but restores the previous indentation after the current line, producing:

```
    ....
!elif
    ....
```

Bracket (*type* = '`b`' or '`B`')

A bracket should be used when a starting token pairs with a closing token which may span multiple lines. i.e. the opening and closing braces of a programming language. Note that the opening and closing tokens must be different otherwise they cannot be differentiated. A bracket has two main effects:

When the previous line has an unmatched open bracket

In this situation the current line is indented to the right of the mismatched bracket.

When the previous line has an unmatched close bracket

In this situation the matching open bracket is hunted for in previous lines until either the *look−back* limit (See **Indentation Method Creation**) is exhausted or the bracket is matched, in which case the indent of that line is used.

For a bracket the only other argument given is the *close*. e.g. tcl's '`(`' and '`)`' brackets

```
indent .hilight.tcl b "(" ")"
```

Which produces:

```
....
.... (....
      ....
```

```
            ....)
    ....
```

Continue (*type* = 'c' or 'C')

Indicates that when *token* is found on the current line, the next line is a continuation of the current line. The indentation of the next line is the indentation of the first continuation line plus the given *indent*. *indent* is the only argument specified. e.g. tcl's '\'

```
indent .hilight.tcl c "\\" 10
```

A simple example is

```
....
12345678901234567890      \
          ....
....
```

When used in conjunction with brackets, the following effect is observed:

```
....
12345678901234567890      \
          ....(....       \
               ....)      \
          ....            \
          ....
....
```

This shows why the first continuation line (the `123456...` line) must be located and used as the base line from which the indentation is derived; again the *look−back* limits the search for this line.

Exclusion (*type* = 'e' or 'E')

Used to exclude text between start *token* and *close* token from the indentation calculation, typically used for quotes. The *ignore* argument is also specified (see hilight(2) `type 0x004` type bracket) e.g. MicroEmacs macro quotes:−

```
indent .hilight.emf e "\"" "\"" "\\"
```

e.g. tcl's quotes

```
indent .hilight.tcl e "\"" "\"" "\\"
```

producing:−

```
....
".... ignore { ... \" ... ignore another { token ... "
....
```

Ignore (*type* = 'i' or 'I')

Text to the right of a line containing *token* is to be ignored; typically used for comments. e.g. MicroEmacs macro '`;`' comment:−

```
indent .hilight.emf i ";"
```

Or tcl's '#' comment

```
indent .hilight.tcl i "#"
```

producing

```
....
# ... ignore this { indent token
....
```

EXAMPLE

Examples of indentation method creations can be found in macro files `hkemf.emf`, `hktcl.emf` and `hkvrml.emf`. The following example is taken from `hkemf.emf`:−

```
!if &sequal .hilight.emf "ERROR"
    set−variable .hilight.emf &pinc .hilight.next 1
!endif

...

0 indent  .hilight.emf 0 10
indent .hilight.emf N "define−macro" 4
indent .hilight.emf n "!if" 4
indent .hilight.emf s "!eli" −4
indent .hilight.emf s "!els" −4
indent .hilight.emf o "!end" −4
indent .hilight.emf n "!whi" 4
indent .hilight.emf o "!don" −4
indent .hilight.emf n "!rep" 4
indent .hilight.emf o "!until" −4
indent .hilight.emf o "!ema" −4
indent .hilight.emf e "\"" "\"" "\\"
indent .hilight.emf i ";"
indent .hilight.emf f "*" 0
```

SEE ALSO

File Language Templates, $buffer−indent(5), add−file−hook(2), hilight(2).

info(3)

NAME

info − Display a GNU Info database
info−on − Display Info on a given topic
info−goto−link − Display Info on a given link
$INFOPATH − GNU info files base directory
.info.path − Cached info search path

SYNOPSIS

info

info−on *topic−str*

info−goto−link *link−str*

$INFOPATH *string*

.info.path *string*

DESCRIPTION

info interprets the GNU *info* pages, and presents the info file information within a buffer window called `*info XXXXX`, where `XXXXX` is the name of the info file. The root of the info page is displayed and may be traversed by selecting the links with the mouse, or by using the standard *info* traversal keys.

The root of the *info* tree is, by default, a file called **dir**, which points to the other information sources. The default search paths for the *info* directories are:−

`c:/info` − MS−DOS and MS−Windows (all).
`/usr/local/info` − All UNIX platforms.

The root directory may also be specified with the `$INFOPATH` environment variable. This is a colon (`:`) or semi−colon (`;`) separated list of directory paths which specify the locations of the info files, for UNIX and Microsoft DOS/Windows environment's, respectively.

info−on gets info on a user specified top level topic, e.g. "`gcc`", the info file "*topic−str*`.info`" must be found in the info search path.

info−goto−link gets and displays info on a user specified link or subject. The link may be within the currently displayed topic (the *link−str* need only specify the subject node name) or a subject within another topic (in which case the *link−str* takes the following form "`(`*topic*`)` *subject*").

NOTES

info is a macro implemented in file `info.emf`.

When an **info** command is run for the first time, the info search path is constructed and stored locally in the command variable **.info.path**. This variable must be directly changed by the user if changes to the info search path are required.

SEE ALSO

info(9).

insert−file(2)

NAME

insert−file − Insert file into current buffer

SYNOPSIS

n **insert−file** "*file−name*" (**C−x C−i**)

DESCRIPTION

insert−file inserts the named file *file−name n* times into the current buffer at the beginning of the current line. The buffer mark is set to the start of the insertion and the cursor is moved to the end.

SEE ALSO

set−mark(2), find−file(2), insert−file−name(2), view−file(2).

insert−file−name(2)

NAME

insert−file−name − Insert filename into current buffer

SYNOPSIS

insert−file−name (**C−x C−y**)

DESCRIPTION

insert−file−name inserts the current buffer's file name into the current buffer or, if entering text on the message line then enters the file name into the message line buffer.

SEE ALSO

insert−file(2), yank(2).

insert−macro(2)

NAME

insert−macro − Insert keyboard macro into buffer

SYNOPSIS

insert−macro "*command*"

DESCRIPTION

insert−macro inserts the named *command* into the current buffer in the MicroEmacs '02 macro language, thus enables it to be saved, re−load and therefore re−used at a later date. This is often used in conjunction with start−kbd−macro(2), end−kbd−macro(2) and name−kbd−macro(2). The given *command* must have been defined either by a keyboard macro or in MicroEmacs '02 macro code.

NOTES

The **insert−macro** provides a good method of identifying unknown low level key codes. Simply record the unknown key as a macro and insert the macro into the scratch buffer. The low level key code appears within the string.

SEE ALSO

start−kbd−macro(2), name−kbd−macro(2), define−macro(2), execute−file(2).

insert−newline(2)

NAME

insert−newline − Move the cursor to the next word

SYNOPSIS

n **insert−newline** (**C−o**)

DESCRIPTION

insert−newline inserts *n* new lines at the current cursor position, but does not move the cursor. Any text following the cursor is moved to the newly created line.

SEE ALSO

newline(2).

insert−space(2)

NAME

insert−space − Insert space(s) into current buffer

SYNOPSIS

n **insert−space**

DESCRIPTION

insert−space inserts *n* spaces at the current cursor position, moving the cursor position.

SEE ALSO

insert−string(2), insert−tab(2), insert−newline(2).

insert−string(2)

NAME

insert−string − Insert character string into current buffer

SYNOPSIS

n **insert−string** "*string*"

DESCRIPTION

insert−string inserts a string *n* times into the current buffer, moving the cursor position.

insert−string allows text to be built in a buffer without reading it from a file. Some special escape characters are interpreted in the *string*, as follows:

`\n` − Enters a new line
`\t` − A tab character
`\b` − Backspace
`\f` − Form−feed
`\\` − Literal backslash character '`\`'
`\xXX` − Hexadecimal value of character ASCII value

SEE ALSO

insert−file(2), insert−newline(2), insert−space(2), insert−tab(2), newline(2).

insert−tab(2)

NAME

insert−tab − Insert tab(s) into current buffer

SYNOPSIS

n **insert−tab** (**C−i**)

DESCRIPTION

insert−tab inserts *n* tab characters at the current cursor position, moving the cursor. The command is not affected by the tab(2m) mode as it always inserts literal tab characters.

SEE ALSO

insert−space(2), insert−string(2), insert−newline(2), tab(2), normal−tab(3), tab(2m).

ipipe−shell−command(2)

NAME

ipipe−shell−command − Incremental pipe (non−suspending system call)
ipipe−kill − Kill a incremental pipe
ipipe−write − Write a string to an incremental pipe

SYNOPSIS

n **ipipe−shell−command** "*command*" ["*buffer−name*"] (**esc backslash**)
n **ipipe−write** "*string*"
n **ipipe−kill**

PLATFORM

UNIX − *irix*, *hpux*, *sunos*, *freebsd*, *linux*.

Windows NT − *win32*.

DESCRIPTION

ipipe−shell−command executes the given system command *command*, opening up a ***icommand*** buffer into which the results of the command execution are displayed. Unlike the pipe−shell−command(2), the user may continue editing during command execution. The command may be terminated by deleting the buffer or issuing a **ipipe−kill** command.

The argument *n* can be used to change the default behavior of pipe−shell−command described above, *n* is a bit based flag where:−

0x01

Enables the use of the default buffer name ***icommand*** (default). If this bit is clear the user must supply a buffer name. This enables another command to be started without effecting any other command buffer.

0x02

Hides the output buffer, default action pops up a window and displays the output buffer in the new window.

0x04

Disable the use of the command−line processor to launch the program (win32 versions only).

By default the "**command**" is launched by executing the command:

```
%COMSPEC% /c command
```

Where `%COMSPEC%` is typically command.com. If this bit is set, the "**command**" is launched directly.

0x08

Detach the launched process from MicroEmacs (win32 versions only). By default the command is launched as a child process of MicroEmacs with a new console. With this bit set the process is completely detached from MicroEmacs instead.

0x10

Disable the command name mangling (win32 versions only). By default any '`/`' characters found in the command name (the first argument only) are converted to '`\`' characters to make it Windows compliant.

0x20

Displays the new process window, by default this window is hidden.

Many other macro commands (see compile(3), grep(3)etc.) use this command.

ipipe−write writes a string *string* to an open ipipe, *n* times.

ipipe−kill terminates an open ipipe, this is automatically called when the ipipe buffer is deleted using delete−buffer(2) or when MicroEmacs is exited.. The numeric argument *n* can be used to change the signal generated, where *n* can take the following values:

1

Sends a Terminate process signal, literally a `SIGKILL` signal on unix or a `WM_CLOSE` on windows platforms. This is the default signal and is typically bound to `C−c C−k`.

2

Sends an interrupt signal, writes a Ctrl−C to the <stdin> pipe on unix or sends Ctrl−C key events on windows platforms. This is typically bound to `C−c C−c`. **NOTES**

On UNIX platforms the TERM environment variable of the new process can be set by setting the user variable **%ipipe−term** to the required value, e.g.:

```
set-variable %ipipe-term "TERM=vt100-nam"
```

Ipipe shells support a large sub−set of vt100 terminal commands, notable exceptions are color and font support and the support of auto−margins. Using the terminal type "`vt100−nam`" disables the use of auto−margins, providing better support.

On platforms which do not support **ipipe−shell−command**, such as MS−DOS, executing **ipipe−shell−command** automatically invokes pipe−shell−command, hence macros may safely use ipipes without explicitly checking the platform type. **ipipe−shell−command** does not run reliably on Windows 3.11 and Windows 95; Windows NT does support ipipes.

While the pipe command is running, mode pipe(2m) is enabled. Modes lock(2m) and wrap(2m) effect the output behavior of an **ipipe−shell−command**.

EXAMPLE

The following example is the grep(3) command macro which utilizes the **ipipe−shell−command**, diverting the output to a buffer called ***grep***.

```
define-macro grep
    !if &seq %grep-com "ERROR"
        set-variable %grep-com "grep "
    !endif
    !force set-variable #l0 @1
    !if &not $status
        set-variable #l0 @ml00 %grep-com
    !endif
    !if @?
        1 pipe-shell-command &cat %grep-com #l0 "*grep*" @mna
    !else
        1 ipipe-shell-command &cat %grep-com #l0 "*grep*" @mna
    !endif
!emacro
```

Note that if an argument is passed to **grep** then it uses pipe−shell−command instead. This is useful if another command is using **grep** which must finish before the calling command can continue, see replace−all−string(3) for an example.

BUGS

On MicroSoft Windows platforms, **ipipe−shell−command** spawns the shell (e.g. `command.com`) with the appropriate command line to make it execute the given command. If the command to be run detaches from the shell and creates its own window, for example `me.exe`, **ipipe−kill** will only kill the shell, it will not kill the actual process, i.e. the `me.exe`.

On MicroSoft Windows platforms **ipipe−shell−command** does not work on Novell's Intranet Client v2.2 networked drives, version 2.5 does appear to work.

SEE ALSO

$buffer−ipipe(5), compile(3), grep(3), pipe−shell−command(2), replace−all−string(3), shell−command(2), pipe(2m), lock(2m), wrap(2m).

isearch−forward(2)

NAME

isearch−forward − Search forward incrementally (interactive)
isearch−backward − Search backwards incrementally (interactive)

SYNOPSIS

isearch−forward (**C−s**)
isearch−backward (**C−r**)

DESCRIPTION

isearch−forward provides an interactive search in the forward direction. This command is similar to search−forward(2), but it processes the search as each character of the input string is typed in. This allows the user to only use as many key−strokes as are needed to uniquely specify the string being searched.

The follow keys can be used at the start of an incremental search only:

`C−s` − Search for last string.
`C−m` − Perform a search−forward instead.
`esc p`,
`esc n` − Scroll through history list etc (See ml−bind−key(2)).

Several control characters are active while isearching:

C−s or **C−x**

Skip to the next occurrence of the current string

C−r

Skip to the last occurrence of the current string

C−h

Back up to the last match (possibly deleting the last character on the search string)

C−w

Insert the next word into the search string.

C−g

Abort the search, return to start.

esc or **C−m**

End the search, stay here

isearch−backward is the same as **isearch−forward**, but it searches in the reverse direction.

For both commands when the end of the buffer is reached, an alarm is raised (bell etc.) a further search request (`C−s`) causes the search to commence from the start of the buffer.

NOTES

The ml−bind−key(2) bindings are used.

The incremental search supports buffer modes exact(2m) and magic(2m).

BUGS

Due to the dynamic nature of active ipipe−shell−command(2) buffers the search history cannot be stored in the same way (list of fixed locations). As a result the search history is stored as a list of searches which are not guaranteed to be consistent.

SEE ALSO

exact(2m), hunt−forward(2), magic(2m), ml−bind−key(2), search−forward(2).
Regular Expressions

ishell(3)

NAME

ishell − Open a interactive shell window
$ME_ISHELL − Windows ishell command comspec

PLATFORM

Windows '95/'98/NT − win32
Unix − All variants.

SYNOPSIS

ishell

[Windows Only]
$ME_ISHELL = <*comspec*>

DESCRIPTION

ishell creates an interactive shell window within the a MicroEmacs buffer window, providing access to the native operating systems command shell. Within the window commands may be entered and executed, the results are shown in the window.

On running **ishell** a new buffer is created called `*shell*` which contains the shell. Executing the command again creates a new shell window called `*shell1*`, and so on. If a `*shell*` window is killed off then the available window is used next time the command is run.

Additional controls are available within the shell window to control the editors interaction with the window. The operating mode is shown as a digit on the buffer mode line, this should typically show "3", which corresponds to *F3*. The operating modes are mapped to keys as follows:−

F2

Locks the window and allows local editing to be performed. All commands entered into the window are interpreted by the editors. **F2** mode is typically entered to cut and paste from the window, search for text strings etc. In mode 2, a **2** is shown on the mode line.

F3

The normal operating mode, text typed into the window is presented to the shell window. Translation of MicroEmacs commands (i.e. beginning−of−word) are translated and passed to the shell. For interactive use this is the default mode. In mode 3, a **3** is shown on the mode line.

F4

All input is passed to the shell, no MicroEmacs commands are interpreted and keys are passed straight to the shell window. This mode is used where none of the keys to be entered are to be interpreted by MicroEmacs. Note that you have to un−toggle the F4 mode before you can swap buffers as this effectively locks the editor into the window.

F5

Clears the buffer contents. This simply erases all of the historical information in the buffer. The operation of the shell is unaffected.

To exit the shell then end the shell session using the normal exit command i.e. "`exit`" or "`C−d`" as normal and then close the buffer. A short cut "`C−c C−k`" is available to kill off the pipe. However, it is not recommended that this method is used as it effectively performs a hard kill of the buffer and attached process

UNIX

The UNIX environment uses the native **pty** support of the operating system. The shell that is opened is determined by the conventional `$SHELL` environment variable.

The shell window assumes that the user is running some sort of Emacs emulation on the command line (i.e. `VISUAL=emacs` for **ksh(1)**, **zsh(1)**, **bash(1)**, **tsch(1)**) and passes Emacs controls for command line editing.

The shell window understands re−size operations and provides a limited decoding of the *termio* characters for a VT100 screen. From within the shell window it is possible to run the likes of **top(1)** correctly. It is even possible to run another MicroEmacs terminal session !!

WINDOWS

The Windows environment provides a very poor command shell facility, this is more of a fundamental problem with the operating system than anything else. Unfortunately NT is no better than Windows '95/'98, stemming from the fact that the Windows is not actually an O/S but a huge window manager, hindered by legacy issues of MS−DOS.

For those familiar with the UNIX command shell then it is strongly recommended that the cygnus(3) BASH shell is used as an alternative. This is a far more responsive shell window and provides the familiar Emacs editing of the command line.

The command shell under Windows is slow and very unresponsive, this would appear to be a problem with the *command.com* as the same problems are not apparent with the cygwin environment. However, the shell window is good for kicking off command line utilities (such as *make*), or any command line processes that generate output on *stdout* as all of the output is captured in the buffer window which can be scrolled backwards for post analysis. For this very reason it is more preferable to the standard MS−DOS box.

It is not possible to run any utilities that use embedded screen control characters as these are not interpreted by the editor.

Changing the Shell

The default shell that is executed is defined by the environment variable **$COMSPEC**. Where the user is using a different command shell (i.e. 4−DOS), then problems may arise if this is an old 16−bit executable. The shell that MicroEmacs executes may be overridden by setting the environment variable **$ME_ISHELL**. This is typically set in the me32.ini(8) file i.e.

```
[username]
ME_ISHELL=c:\windows\command.com
```

Bugs

WinOldAp

Winoldap is created by the Microsoft environment whenever a shell is created. On occasions where processes have terminated badly the user may be prompted to kill these off; this is the normal behaviour of Windows. It is strongly advised that the shell is always exited correctly (i.e. `exit`) before leaving the editor. The Windows operating system for '95/'98 is not particularly resilient to erroneous processes can bring the whole system down. I believe that NT does not suffer from these problems (much).

Locked Input

There are occasions after killing a process the editor appears to lock up. This is typically a case that the old application has not shut down correctly. Kill off the erroneous task (`Alt−Ctrl−Del` − *End Task*) then bring the editor under control using a few `C−g` abort−command(2) sequences. **NOTES**

The **ishell** command uses the ipipe−shell−command(2) to manage the pipe between the editor and the shell. The window is controlled by the macro file `hkipipe.emf` which controls the interaction with the shell.

SEE ALSO

ipipe−shell−command(2), cygnus(3), me32.ini(8).

kbd−macro−query(2)

NAME

kbd−macro−query − Query termination of keyboard macro

SYNOPSIS

[Definition]
kbd−macro−query (**C−x q**)

[Execution]
kbd−macro−query "**y**"|"**n**"|"**C−g**"

DESCRIPTION

kbd−macro−query queries the termination state of keyboard macro recording. If the command is executed during a keyboard macro definition, at that point during its execution the user is prompted as to whether to continue the macro execution. A reply of "**y**" continues the execution as normal, "**n**" stops execution at that point once, if executing the macro *n* times the macro will still executed a further *n−1* times. If the "`C−g`" abort command is entered then all keyboard macro execution is aborted, regardless of the number of repetitions.

SEE ALSO

start−kbd−macro(2), execute−kbd−macro(2).

kill−line(2)

NAME

kill−line − Delete all characters to the end of the line

SYNOPSIS

n **kill−line** (**C−k**)

DESCRIPTION

kill−line, when used with no argument *n*, deletes all text from the cursor to the end of a line, the end of line character is also deleted if the cursor is in the first column and the line(2m) mode is disabled. The deleted text is placed in the kill buffer, see yank(2) for more information on the kill buffer. When used on a blank line, it always deletes it.

If a +ve argument *n* is supplied the specified number of lines is deleted, the setting of the **line** mode is ignore. If *n* is `0` the command has no effect. If a −ve argument is given, +*n* lines are deleted but the text is NOT added to the kill buffer.

NOTES

If a line is accidentally removed then yank the text back immediately or use undo(2).

The −ve argument is typically used in macro scripts where the yank buffer is more precisely controlled by the script.

SEE ALSO

kill−region(2), line(2m), undo(2), yank(2), forward−kill−word(2).

kill−paragraph(2)

NAME

kill−paragraph − Delete a paragraph

SYNOPSIS

n **kill−paragraph**

DESCRIPTION

kill−paragraph deletes the next *n* paragraphs, if *n* is +ve then the paragraph the cursor is currently in and the next *n*−1 paragraphs are killed. If *n* is −ve then the current paragraph and the previous *n*−1 paragraphs are killed. If *n* is zero the command simply returns. The default value for *n* is 1.

DIAGNOSTICS

The following errors can be generated, in each case the command returns a FALSE status:

[end of buffer]

The given argument *n* was greater that the number of remaining paragraphs, all the remaining paragraphs are still removed.

[top of buffer]

A negative argument *n* was given requesting more paragraphs to be killed then are present before the cursor. All the paragraphs before the cursor are still removed. **NOTES**

A paragraph is terminated by a blank line. All text residing between two blank lines is considered to be a paragraph − regardless of the text layout.

The distinction between killed text and deleted text is that text which is killed is placed into the yank buffer so that it can be pasted into any buffer using yank(2).

SEE ALSO

backward−paragraph(2), forward−paragraph(2), kill−region(2).

kill−rectangle(2)

NAME

kill−rectangle − Delete a column of text
yank−rectangle − Insert a column of text

SYNOPSIS

kill−rectangle (**esc C−w**)
n **yank−rectangle** (**esc C−y**)

DESCRIPTION

kill−rectangle deletes a rectangle (or column) of text defined be the cursor and the set−mark position. The text between the mark column and the cursor column is removed from every line between the mark line and the cursor line inclusive and copied to the kill buffer. The delete text may then be extracted from the kill buffer using yank(2) or **yank−rectangle**.

The mark position may be ahead or behind the current cursor position. If the rectangle column boundary divides a tab character which spans multiple columns, the tab character is replaces with the equivalent number of spaces. Similarly if the boundary divides an unprintable character which is displayed using multiple characters (e.g. '^A' for character 0x01) then spaces are inserted before the character to move it to the right of the boundary.

yank−rectangle inserts the current kill buffer (which may or may not have been generated using **kill−rectangle**) into the current buffer in a column fashion. That is to say that the first line of text in the kill buffer is inserted into the current line of text in the current buffer from the current cursor column, the cursor is then moved the the next line and placed at the same column. The process is then repeated for the second line of text in the kill buffer etc.

NOTES

The command copy−rectangle is not provided by default as this command is rarely required. If this command is required, the following macro definition can be used:

```
define−macro copy−rectangle
    set−alpha−mark "T"
    set−variable #l0 &bmod "view"
    set−variable #l1 &bmod "edit"
    set−variable #l2 &bmod "undo"
    −1 buffer−mode view
    1 buffer−mode undo
    kill−rectangle
    ; undo the kill and restore the buffer state
    undo
```

```
                &cond #l2 1 −1 buffer−mode "undo"
                &cond #l1 1 −1 buffer−mode "edit"
                &cond #l0 1 −1 buffer−mode "view"
                goto−alpha−mark "T"
                ; flag the command to be a copy−region type command
                set−variable @cl copy−region
            !emacro
```

SEE ALSO

set−mark(2), kill−region(2), yank(2), copy−region(2), reyank(2), undo(2).

kill−region(2)

NAME

kill−region − Delete all characters in the marked region

SYNOPSIS

n **kill−region** (**C−w**)

DESCRIPTION

kill−region deletes all characters from the cursor to the mark set with the set−mark(2) command. The characters removed are copied into the kill buffer and may be extracted using yank(2). If a numeric argument of `0` is given the command has no effect. If a −ve argument is given the characters are not placed in the kill buffer, therefore the text is effectively lost (this does not effect the undo(2) operation).

The mark position may be ahead or behind the current cursor position.

USAGE

To move text from one place to another:

- Move to the beginning of the text you want to move.
- Set the mark there with the set−mark (**esc space**) command.
- Move the point (cursor) to the end of the text.
- Use the **kill−region** command to delete the region you just defined. The text will be saved in the kill buffer.
- Move the point to the place you want the text to appear.
- Use the yank (**C−y**) command to copy the text from the kill buffer to the current point.

Repeat the last two steps to insert further copies of the same text.

NOTES

If a region is accidentally removed then yank the text back immediately or use undo(2).

Windowing systems such as X−Windows and Microsoft Windows utilize a global windowing kill buffer allowing data to be moved between windowing applications (*cut buffer* and *clipboard*, respectively). Within these environments MicroEmacs '02 automatically interacts with the windowing systems kill buffer, the last MicroEmacs '02 **kill−region** entry is immediately available for a paste operation into another windowing application.

SEE ALSO

copy−region(2), kill−rectangle(2), reyank(2), set−mark(2), undo(2), yank(2).

line−scheme−search(3)

NAME

line−scheme−search − Search and annotate the current buffer

SYNOPSIS

line−scheme−search

DESCRIPTION

line−scheme−search provides a method of searching for text patterns within the current buffer and annotating any matches through colored line hilighting. A selection of line colors are provided to allow different search patterns to be assigned their own color.

line−scheme−search is generally used for annotating log files and alike, where indevidual lines are of interest in addition to the context about that line. The hilighting draws attention to the line, by providing a visual cue, allowing the contents of the file to be breifly scanned.

On invocation of **line−scheme−search** a osd(2) dialog is presented to the user, search patterns and their associated hilighting assignment are selected through this interface. The dialog entries are defined as follows:−

Search for

The text dialog entry box allows the search pattern to be entered. This may be a regular expression or plain text.

Color

The **Color** allows the line hilighting color scheme to be selected from a pop−up menu. The color **Remove** is special and allows previously applied line hilighting to be removed.

Case Sensitive

A check box that allows the search to be case sensitive or insensitive. This modifies the exact(2m) mode.

Magic Mode

A check box that enables/disables regular expression pattern matching. This modifies the magic(2m) mode.

Below

Searches and hilights lines matching the search pattern from the current cursor position to the end of the buffer.

Above

Searches and hilights lines matching the search pattern from the current cursor position to the top of the buffer.

All

Searches and hilights lines matching the search pattern for the whole buffer.

Clear All

Removes all line hilighting from the current buffer.

First

Moves to the top of the buffer and hilights the first line that matches the search pattern.

Next

Hilights the next line that matches the search pattern.

Reverse

Hilights the previous line that matches the search pattern.

Exit

Exits the hilighting search dialog. **NOTES**

line−scheme−search is a macro implemented in `hiline.emf`.

SEE ALSO

osd(2), $line−scheme(5).

list−buffers(2)

NAME

list−buffers − List all buffers and show their status

SYNOPSIS

list−buffers (**C−x C−b**)

DESCRIPTION

list−buffers splits the current window and in one half brings up a list of all the buffers currently existing in the editor. The active modes, change flag, and active flag for each buffer is displayed. (The change flag is a ***** character if the buffer has been changed and not written out. The active flag is not an **@** if the file had been specified on the command line, but has not been read in yet since nothing has switched to that buffer.)

The buffer list has some special command keys associated with it which allow the state of the buffers to be edited from the buffer list, the editing allows buffers to be killed and saved to disk. The key codes are defined as follows:−

1 − Switch to buffer

Switch to that buffer and make it the only buffer.

2 − Move to buffer

Switch the buffer list window to that buffer.

D − delete buffer

Flag buffer for deletion. A buffer scheduled for deletion is marked with a '**D**' in first column. The delete status is enacted by the '**X**' command, or may be removed with the '**U**' command.

S − save buffer

Flag buffer for saving. A buffer scheduled from saving is marked with a '**S**' in the second column. Note that a buffer may be marked for saving and deletion, the save operation is performed before the delete.

U − unmark buffer

Unmark the '**D**' and '**S**' flags on current line.

X − execute

Execute all the '**D**' and '**S**' flags currently set. The **S**ave is enacted first.

For all but '**X**', the buffer selected is the buffer noted on the current cursor line. These keys are not remappable.

SEE ALSO

list−variables(2), list−commands(2), split−window−horizontally(2).

list−commands(2)

NAME

list−commands − List available commands

SYNOPSIS

list−commands (**C−h c**)

DESCRIPTION

list−commands constructs a list of all known built in commands and macros that are currently defined by MicroEmacs '02 and presents a list of those commands in the buffer "***commands***". Each entry is formatted as:−

command .... keyCode

Where multiple keys are bound to the same command, then each of the *keyCode*'s is shown.

list−commands is similar to describe−bindings(2) except that the commands are presented in alphabetical order (as opposed to key binding order).

EXAMPLE

The following is an example of the output of **list−commands**:−

```
backward-char ................. "C-b"
                                "left"
backward-delete-char .......... "backspace"
                                "S-backspace"
backward-delete-tab ........... "S-tab"
backward-kill-word ............ "esc backspace"
backward-line ................. "C-p"
                                "up"
                                "C-up"
backward-paragraph ............ "esc ["
                                "esc p"
backward-word ................. "esc b"
                                "C-left"
beginning-of-buffer ........... "esc <"
                                "home"
beginning-of-line ............. "C-a"
buffer-bind-key
buffer-info ................... "C-x ="
buffer-mode ................... "esc ~"
                                "C-x m"
                                "insert"
```

```
        buffer−unbind−key
        :
        :
```

SEE ALSO

describe−bindings(2), list−variables(2).

list−registry(2)

NAME

list−registry − Display the registry in a buffer

SYNOPSIS

list−registry

DESCRIPTION

list−registry lists the contents of the registry in the a buffer in a hierarchical format. The key name and any associated string is shown as a hierarchical tree.

The registry listing is generated in the buffer "`*registry*`".

SEE ALSO

read−registry(2), erf(8).

list−variables(2)

NAME

list−variables − List defined variables

SYNOPSIS

list−variables (**C−h v**)

DESCRIPTION

list−variables pops up a window with a list of all register, buffer, user and global variables with their current setting. The variables are shown for the current buffer from which the command was invoked

list−variables provides a good alternative to describe−variable(2) where the value of multiple variables is to be interrogated.

The output is displayed in four sections:−

Register variables

The current settings of the global register variables ('**#**' prefix).

Buffer Variables

The current setting of the buffer variables ('**:**' prefix). This variables relate to the current buffer from which the command was invoked.

System Variables

The current settings of the system variables ('**$**' prefix).

Global Variables

The current setting of the global variables ('**%**' prefix). **EXAMPLE**

An example output from **list−variables** is shown below:−

```
Register variables:

    #g0 ........................... "29"
    #g1 ........................... ""
    #g2 ........................... "ERROR"
    :
    :
    #g8 ............................ "ERROR"
    #g9 ............................ "ERROR"

Buffer [m2cmd086.2] variables:


System variables:

    $auto-time ..................... "300"
    $buffer-bhook .................. "bhook-nroff"
    $buffer-bname .................. "m2cmd086.2"
    $buffer-ehook .................. "ehook-nroff"
    $buffer-fhook .................. "fhook-nroff"
    $buffer-fmod ................... "040"
    $buffer-fname .................. "d:/emacs/doc/m2cmd086.2"
    $buffer-hilight ................ "3"
    :
    :
    $window-width .................. "80"
    $window-x-scroll ............... "0"
    $window-xcl-scroll ............. "0"
    $window-y-scroll ............... "52"

Global variables:

    %black ......................... "0"
    %blue .......................... "4"
    %compile-com ................... "nmake "
    %cyan .......................... "6"
    %green ......................... "2"
    %grep-com ...................... "grep -n "
    :
    :
    %usr1mode ...................... "off"
    %white ......................... "7"
    %yellow ........................ "3"
```

SEE ALSO

describe−variable(2), list−commands(2).

Mahjongg(3)

NAME

Mahjongg − MicroEmacs '02 version of the solitaire Mah Jongg game

SYNOPSIS

Mahjongg

DESCRIPTION

Mah Jongg is an ancient Chinese game usually played by four players with tiles similar to dominos. This is a MicroEmacs '02 version which was inspired by the X−Windows version of the same game. The X−Windows version for the solitaire game originally seen on the PC and later ported to SunView.

Theory Of Play

The object of the game is to remove all the tiles from the board. Tiles are removed by matching two identical tiles which have either an open left edge or open right edge. The only exception to this rule is that any open "*flower*" tile (bamboo `[BAMB]`, orchid `[ORCH]`, plum `[PLUM]`, or chrysanthemum `[CHRY]`) matches any other open "*flower*" tile and any open "*season*" tile (spring, summer, autumn, or winter) matches any other open "*season*" tile.

Tiles are stacked on the board, the height of the tile is indicated by the color coding as follows:−

`Level 5` − White
`Level 4` − Red
`Level 3` − Yellow
`Level 2` − Green
`Level 1` − Cyan

To remove a pair of tiles, click the left mouse button on a tile (which will show in the selection color) and then click the left mouse button on the matching tile. At this point, both tiles will disappear from the board. If after selecting the first tile, you decide that you don't wish to play that tile, simply reclick the left button on the selected tile, alternatively click the right button to deselect any selected tile.

To the right of the board are a number of control buttons. To select an option, click the left mouse button on it.

NEW

Start a new game (keyboard n).

SAME

Start the same game again (keyboard s).

QUIT

Exit the game (keyboard q).

HELP

This help page (keyboard esc h).

The counter shows the number of remaining tiles on the board, at the start of the game there are 144 tiles.

NOTES

Mahjongg is a macro defined in `mahjongg.emf`.

Mah Jongg may only be played with a mouse, there is no keyboard support, with the exception of the re−start keys.

ACKNOWLEDGEMENT

Thanks to Jeff S. Young who (I think) wrote the original X−Windows version, and whose manual page formed the basis of this page.

The tile patterns were inspired from the X−Windows tile patterns. The X−Windows tile patterns themselves are copyright 1988 by Mark A. Holm <tektronix!tessi!exc!markh>.

SEE ALSO

Games, Match−It(3), Patience(3).

MainMenu(3)

NAME

Main Menu − The top main menu

SYNOPSIS

n osd

DESCRIPTION

The main menu is provided to give an easier access to parts of MicroEmacs functionality, the menu is not burnt into MicroEmacs but defined on start−up in `me.emf` and `osd.emf`. The user−setup(3) command can be used to set whether the menu is always visible and if the Alt−Hotkeys are enabled (i.e. '`A−f`' to open the **File** menu).

The main menu is osd(2) dialog number `0` so key bindings can be made which will open the main menu, an argument of `0` will simply open the main menu, an argument of `0x0n0000` will not only open the main menu but also the `n`th sub menu, e.g. to open the edit menu use:

```
0x020000 osd
```

Following is a brief description of the main menu items:

File Menu

`New`

Changes the current buffer to a new buffer.

`Open`

Opens a dialog enabling the user to select files for opening into MicroEmacs. By default the dialog opens the selected file using command find−file(2), but if the view option is selected the view−file(2) command is used. The binary or encrypt options configure whether the files are to be loaded with binary(2m) or crypt(2m) modes enabled.

`Quick Open`

Opens a sub−menu list all user file types (defined in user−setup(3)). Selecting one will open another sub−dialog list all files of that type in the current directory, selecting a file will open it using command find−file(2).

`Favorites`

Opens a sub−menu enabling the user to add new favorite files, edit the existing list of favorite files, or select an existing favorite file in which case the file is opened using command find−file(2). The favorite file using to store the list is "**$MENAME**`.eff`" and is saved in the first path given in the $search−path(5). Each favorite file takes 2 lines in the file, the first is the text displayed in the dialog (note that characters '\' and '&' must be protected with a '\' and the '&' can be used to set the Hot key) and the second line is the file name. A line with a single '**−**' character creates a separater line in the dialog.

`Find Tag`

Only visible when a `tags` file is found in the current directory, the command jumps to the current tag or if not on a tag or the tag is not found, opens a dialog enabling the user to select a tag. See command find−tag(2) for more information.

`Find File`

Executes command file−browser(3).

`FTP`

Executes command ftp(3).

`Close`

Executes a dialog form of the command delete−buffer(2).

`Attributes`

Opens a dialog enabling the user to set the current buffers file attributes. See command file−attrib(3) for more information.

`Save`

Executes a dialog form of the command save−buffer(2).

`Save As`

Executes a dialog form of the command write−buffer(2).

`Save All`

Executes a dialog form of the command save−all(3).

`Printer Setup`

Opens a dialog which enables the user to configure the printer driver, output location and page layout (executes command print−setup(3)).

`Print`

Executes command print−buffer(2).

`Buffer`

Opens a sub−menu listing all created buffers, selecting one will change the current buffer to the selected one.

`Exit`

Executes command save−buffers−exit−emacs(2). **Edit Menu**

`Undo`

Undoes the last edit in the current buffer (executes command undo(2)).

`Redo`

Redo the last undo, only available immediately after an undo. This is also done via the undo(2) command.

`Undo All`

Undo all edits in the current buffer until the last save or no more undo history is available. Executes the command undo(2) with a 0 numerical argument.

`Set Mark`

Executes command set−mark(2).

`Cut`

Executes command kill−region(2).

`Copy`

Executes command copy−region(2).

`Paste`

Executes command yank(2).

`Narrow Out`

Executes command narrow−buffer(2) with a numeric argument of 4.

`Narrow To`

Executes command narrow−buffer(2) with a numeric argument of 3.

`Remove Single Narrow`

Executes command narrow−buffer(2) with a numeric argument of `2`.

`Remove All Narrows`

Executes command narrow−buffer(2) with a numeric argument of `1`. **Search Menu**

`Search`

Executes a dialog form of the command isearch−forward(2).

`Replace`

Executes a dialog form of the command query−replace−string(2).

`Hilight Search`

Opens another dialog which can be used to add and remove hilighting of individual lines in the current buffer. Note that setting a line hilight is a temporary change, it will not effect any files etc and will be lost when the buffer is deleted.

`Goto Line`

Executes a dialog form of the command goto−line(2).

`Goto Fence`

Executes command goto−matching−fence(2).

`Set Bookmark`

Executes command set−alpha−mark(2).

`Goto Bookmark`

Executes command goto−alpha−mark(2). **Insert Menu**

`Symbol`

Executes command symbol(3).

`Date & Time`

Opens a dialog with the current date and time in a selection of common formats; selecting one of these will insert the string into the current buffer at the current position. Note that the format text strings depend on the current language (Default and American languages use the order MM−DD−YY

etc whereas the rest use DD−MM−YY). The names used for the day and month names can be defined using the Setup page of Organizer(3).

`File`

Executes command insert−file(2).

`File Name`

Executes command insert−file−name(2).

`Macro...`

Executes command insert−macro(2). **Format Menu**

`Restyle Buffer`

Executes command restyle−buffer(3).

`Restyle Region`

Executes command restyle−region(3).

`Clean Buffer`

Executes command clean(3).

`Change Buffer Char Set`

Executes command charset−change(3).

`IQ Fill Paragraph`

Executes command ifill−paragraph(3).

`Fill Paragraph`

Executes command fill−paragraph(2).

`Fill All Paragraphs`

Executes command fill−paragraph(2) with a very large positive numerical argument. Note that this only effects paragraphs from the current position onwards.

`Paragraph to Line`

Executes command paragraph−to−line(3).

```
All Paragraphs to Line
```

Executes command paragraph−to−line(3) with a very large positive numerical argument. Note that this only effects paragraphs from the current position onwards.

```
Sort Lines
```

Executes command sort−lines(2).

```
Ignore Case Sort Lines
```

Executes command sort−lines−ignore−case(3).

```
Capitalize Word
```

Executes command capitalize−word(2).

```
Lower Case Word
```

Executes command lower−case−word(2).

```
Lower Case Region
```

Executes command lower−case−region(2).

```
Upper Case Word
```

Executes command upper−case−word(2).

```
Upper Case Region
```

Executes command upper−case−region(2). **Execute Menu**

```
Execute Command
```

Executes command execute−named−command(2).

```
Execute Buffer
```

Executes command execute−buffer(2).

```
Execute File
```

Executes command execute−file(2).

```
Start Kbd Macro
```

Executes command start−kbd−macro(2).

`Query Kbd Macro`

Executes command kbd−macro−query(2).

`End Kbd Macro`

Executes command end−kbd−macro(2).

`Execute Kbd Macro`

Executes command execute−kbd−macro(2).

`Name Kbd Macro`

Executes command name−kbd−macro(2).

`Ipipe command`

Executes command ipipe−shell−command(2).

`Shell`

Executes command shell(2). **Tools Menu**

`Current Buffer Tools`

For some file formats MicroEmacs provides a file format specific set of tools, see the file type help page for more specific information.

`Count Words`

Executes command count−words(2).

`Spell Word`

Executes command spell−word(3).

`Spell Buffer`

Executes command spell−buffer(3).

`Word Complete`

Takes the incomplete word to the left of the cursor and attempts to complete the word by using the users current language dictionary. Executes command expand−word(3).

`Compare Windows`

Executes command compare−windows(2).

`Compile`

Executes command compile(3).

`Grep`

Executes command grep(3).

`Graphical Diff`

Executes command gdiff(3).

`Diff`

Executes command diff(3).

`Diff Changes`

Executes command diff−changes(3).

`Organizer`

Executes command organizer(3).

`Mail`

Executes command mail(3).

`View Mail`

Executes command vm(3).

`More...`

Opens a sub−menu with a collection of other useful miscellaneous tools. **Window Menu**

`Split Window V`

Executes command split−window−vertically(2).

`Grow Window V`

Executes command change−window−depth(2) with an argument of 1.

`Shrink Window V`

Executes command change−window−depth(2) with an argument of −1.

`Split Window H`

Executes command split−window−horizontally(2).

`Grow Window H`

Executes command change−window−width(2) with an argument of 1.

`Shrink Window H`

Executes command change−window−width(2) with an argument of −1.

`One Window`

Executes command delete−other−windows(2).

`Delete Window`

Executes command delete−window(2).

`Previous Window`

Executes command previous−window(2).

`Next Window`

Executes command next−window(2).

`Create New Frame`

Create an new external frame, only available on version which support multiple−window frames. Executes command create−frame(2).

`Create New Frame`

Closes the current frame, only available on version which support multiple−window frames. The command will fail if this is the only frame, use File −> Exit to exit MicroEmacs, executes command delete−frame(2).
Help Menu

`Curr Buffer Help`

For some file formats MicroEmacs provides a file format specific help page giving details of key−bindings and tools specific to the current buffers file type.

`General Help`

Executes command osd−help(3).

```
Help on Command
```

Executes command help−command(2).

```
Help on Variable
```

Executes command help−variable(2).

```
Describe Bindings
```

Executes command describe−bindings(2).

```
Describe key
```

Executes command describe−key(2).

```
Describe Variable
```

Executes command describe−variable(2).

```
Describe Word
```

Executes command describe−word(3).

```
List Buffers
```

Executes command list−buffers(2).

```
List Commands
```

Executes command list−commands(2).

```
List Registry
```

Executes command list−registry(2).

```
List Variables
```

Executes command list−variables(2).

```
Command Apropos
```

Executes command command−apropos(2).

```
Buffer Setup
```

Executes command buffer−setup(3).

`User Setup`

Executes command user−setup(3).

`Scheme Editor`

Executes command scheme−editor(3).

`Games`

Opens a sub−menu listing all available games, see Games for more information.

`Product Support`

Opens on−line Contact information.

`About MicroEmacs`

Executes command about(2). **NOTES**

The main menu is defined using osd(2) in macro files me.emf and osd.emf.

General user extensions to the main menu can be added to the user file `myosd.emf` which is executed once when the main menu is first opened. The macro file can add new items to any of the main sub menus and can delete most existing items (some are dynamically added when appropriate, these should not be deleted). See `osd.emf` for examples of how to add items to the menu.

New sub−menus should be added in the company or user setup files as this must be done at start−up. The content on the menu is not required until the main menu is used so populating the new sub−menu can be done in `myosd.emf`.

SEE ALSO

user−setup(3).

Match−It(3)

NAME

Match−It − MicroEmacs '02 version of the Match−It game

SYNOPSIS

Match−It

DESCRIPTION

The object of the game is to score the largest number of points, to do this the player must complete as many sheets as possible. A sheet is completed when all the tiles are removed from the board within the given time limit − ALL sheet are possible. If the player fails to remove all the tiles before the time runs out a life is lost, if all lives have been lost then the game is over.

Tiles are removed from the board by matching two identical tiles which have an 'extraction' path between them. The only exception to this rule is that any open "*flower*" tile (bamboo `[BAMB]`, orchid `[ORCH]`, plum `[PLUM]`, or chrysanthemum `[CHRY]`) matches any other open "*flower*" tile and any open "*season*" tile (spring, summer, autumn, or winter) matches any other open "*season*" tile.

An 'extraction' path is a straight line which uses 2 or less right angles, the following are legal extraction paths, '`*`'s denote the right angles:

```
                A---*       *-----*     A----*
A----A              A       AXXXXXA     XXXXX|
                                        A----*
```

The following are illegal paths:

```
*----*          *---*
AXXXX|          |XXXA
XXXXA*      A---*XXXX
```

2 points are added to the score whenever a pair is successfully removed, a point is deducted whenever a pair is selected which can not be removed because there is no valid extraction path. There are 2 aids, pressing the right button on a tile when no other tile is selected will hilight all tiles of matching type, this costs 4 points. The other help is activated by a button at the top right of the screen and it removes a random removable pair (or informs the user that there are no removable pairs), there are a limited number of these helps.

At the end of a successful sheet the score is increased be the time left, the number of lives and helps remaining and by the Pedigree and Internal bonuses if they were achieved.

The Pedigree bonus is obtained when only identical tiles are paired, i.e. no differing flowers or seasons were paired, 50 points are awarded when achieved. Its status is indicated by a 'P' to the left of the 'Help' button and the top of the window.

The internal bonus is obtained when the outer 4 margins are not used. If the left or right margins are not used then 10 points are awarded for each, if the top or bottom are not used then 20 points are awarded for each and if none are used then 400 points are awarded! The status on the Internal bonus is indicated by an '`I`' surrounded by '`*`'s, one for each margin. This can be found next to the Pedigree bonus '`P`'.

GAME CONTROLS

To the right of the high score table on the main menu there are a number of control buttons. To select an option, click the left mouse button on it.

NEW

Start a new game.

QUIT

Exit Match−It.

HELP

This help page (keyboard esc h).

During a sheet, to remove a pair of tiles, click the left mouse button on a tile (which will show in the selection color) and then click the left mouse button on the matching tile. At this point, if the tiles can be removed, the extraction path is drawn and both tiles will disappear from the board. If after selecting the first tile, you decide that you don't wish to play that tile, simply reclick the left button on the selected tile, alternatively click the right button to deselect any selected tile.

To the top right of the sheet there are a number of control buttons:−

HELP

Removes a tile pair.

QUIT

Exit the game.

BOSS

Hides Match−It, also acts as a pause. Execute Match−It again to return to the game.

The top left shows the number of remaining lives, the current sheet level, the current score, time remaining for the current sheet and the status of the Internal and Pedigree bonuses.

NOTES

Match−It is a macro defined in `matchit.emf`.

Match−It may only be played with a mouse, there is no keyboard support, with the exception of the re−start keys.

The sheet database file matchit.edf must be accessible for Match−It to work.

SEE ALSO

Games, Mahjongg(3), Metris(3).

Metris(3)

NAME

Metris − MicroEmacs '02 version of the falling blocks game

SYNOPSIS

Metris

DESCRIPTION

Traditional falling blocks game, make solid horizontal lines out of the falling blocks. The blocks can be rotated and moved left or right by the user as they fall. Once a horizontal line is completely solid it will disappear and everything above it will drop down. A bonus is given if 3 solid rows are made at the same time, i.e. using one block.

Every line you make the game speeds up until it gets too fast!! The game ends when there is no more room to put a block.

The keys used to control Metris are:

left or **j**

Move the block left one character.

right or **l**

Move the block right one character.

down or **k**

Rotate the block counter−clockwise 90 degrees.

space

Drop the current block.

p

Pause the current game.

q

Quit the current game.

C−l

Redraw the display.

return

Start a new game.

esc h

View this help page. **NOTES**

Metris is a macro defined in `metris.emf`.

SEE ALSO

Games, Match−It(3), Patience(3).

vm(3)

NAME

vm − Email viewer
mail−check − Check for new email
stop−mail−check − Disable the check for new email
mail − Compose and send an email

SYNOPSIS

vm
mail−check
stop−mail−check
mail

DESCRIPTION

vm is a simple email manager, it is configured to send and receive emails using the user−setup(3) Mail dialog.

mail−check tests the size of this incoming mail box, a non−zero length indicates that new mail has arrived and **mail−check** informs the user by inserting a '`M`' in the mode−line (2nd character for the left) and ringing the system bell. **mail−check** uses create−callback(2) to check for new mail every 10 minutes, this can be disabled by executing **stop−mail−check**.

When **vm** is executed it checks for new mail, if found it first copies the new mail to a file called "`new_mail`" in the users mail directory. The incoming box is then emptied by truncating the file to zero length. The users main mail box is then loaded and the new mail (if any) is appended. The mail box is then processed after which 2 windows are created the bottom window listing all messages in the box and the top displaying the current message.

vm is capable of:

- Scrolling through the mail box displaying each message (`up`, `p`, `down`, `n`, `return`, `space`).
- Check and get new mail messages (`g`).
- Extract and cut embedded data files (`x`, `C`, `c`).
- Reply to and forward mail messages (`R`, `r`, `z`).
- Delete mail messages (`d`, `u`).
- Archive messages to other mail boxes (`A`, `a`).
- Save changes to the current mail box (`S`, `s`).
- Delete the current mail box (`D`).
- Visit another mail box (`v`).
- Send a mail message (`m`).
- Hide vm windows (`delete`).

Use the vm help page (bound to "`esc h`") for further information.

vm supports two types of embedded data, uuencode and mime encoding and uses ipipe−shell−command(2) to extract the data, the commanding to use must be supplied by the user using the setup dialog, which can contain the following special tokens:

`%i`

Temporary file name, if used, the embedded data is written to the this file first.

`%o`

User supplied output file name, if `%i` is not used, the embedded data is written to this file first.

`%b`

The output base name, i.e. `%o` without the path.

If no command line is supplied then the embedded data is written to the user supplied file name as a text file in the form found in the mail message.

mail can be used to compose and send an email, it can insert embedded data in a similar way to **vm**'s data extraction, the following special tokens can be used:

`%i`

The user supplied data file to be embedded.

`%b`

The input base name, i.e. `%i` without the path.

`%o`

Temporary file name used to output the processed data file, this file is inserted into the mail message using insert−file(2).

mail also uses **ipipe−shell−command** to send the mail message, the following special tokens can be used:

`%f`

The from user name.

`%s`

The email subject.

`%t`

A comma separated list of 'To' recipients.

%c

A comma separated list of 'Cc' recipients.

%o

A file name of the mail message.

Any field not used in the command−line is left at the head of the mail message.

EXAMPLE − UNIX

The following command−line can be used on most UNIX systems to extract uuencoded data:

```
rm −f %o ; uudecode %i ; rm −f %i
```

The following command−line can be used on most UNIX systems to extract mime encoded data:

```
rm −f /tmp/%b ; metamail −B −d −q −w −x −z %i ; mv −f /tmp/%b %o
```

The following command−line can be used on most UNIX systems to uuencode a data file ready for it to be embedded, the original file is not changed:

```
uuencode %b < %i > %o
```

The following command−line can be used on most UNIX systems to send an email:

```
/usr/lib/sendmail −oi −oem −odi −t < %o
```

EXAMPLE − WIN32

Typically the **cygnus(1)** utilities can be used for data insertion and extraction. These have the advantage of being very similar to the unix ones so only minor changes are required, i.e. try the following for data insertion and mime & uuencode extraction respectively:

```
del %o ^ uudecode %i ^ del %i
del c:\tmp\%b ^ metamail −B −d −q −w −x −z %i ^ move c:\tmp\%b %o
uuencode %b < %i > %o
```

This assumes that the shell you are using supports the '^' multiple commands on a single line feature, this is supported by **4dos(1)** and **4nt(1)**. If your shell does not support this feature a simple batch file command could be used instead.

postie(1) is a freely available pop3/smpt e−mail support program, available on the net, which can be used to provide a fully working **vm** on windows systems. As it is typically used in a dial−up connect environment, the **user−setup** 'Queue Outgoing Mail' option will be enabled while the 'Check Mail' and 'VM Gets Mail' will be disabled. This ensures that a connection is only made when the **vm** 'g' command is used which sets all queued outgoing mail and gets any incoming mail.

The following command−line can be used to get mail from your pop server using postie:

```
postie −host:pop−mail−addr −user:user−addr −pass:password −file:inbox
        "−sep:From root Mon Jan 11 20:02:02 1999" −raw −rm
```

Where the `inbox` is the 'Incoming Mail Box' file specified in user−setup. The `−sep` option is used to partition each mail message from the previous message, this string is used as it is in a unix standard form so the resulting mail box could be understood by unix mail systems such as netscape etc.

NOTE: The **−rm** option is used to remove the incoming mail messages from the server. It is strongly recommended that the system is thoroughly tested without this option first.

The following command−line can be used to send mail to your smtp server using postie:

```
postie −host:smtp−mail−addr "−from:user@mail−addr" −use_mime:0
        "−to:%t" "−s:%s" "−cc:%c" "−file:%o"
```

blat(1) is another freely available windows program which can be used to send mail with the following command−line:

```
blat %o −f %f −s \"%s\" −t \"%t\" −c \"%c\"
```

NOTES

vm is a macro defined in `vm.emf`, **mail−check**, **stop−mail−check** and **mail** are macros defined in `mail.emf`.

vm has only been tested in a couple of environments, the author will not except any responsibility for any loss of data, i.e. use at your own peril. You have been warned! Back−up all data files and test **vm** THOROUGHLY before using it.

SEE ALSO

user−setup(3), ipipe−shell−command(2), create−callback(2), **sendmail(1)**.

man(3)

NAME

man − UNIX manual page viewer. man−clean − Clean UNIX manual page.

SYNOPSIS

man
man−clean

DESCRIPTION

man provides a mechanism to display a UNIX manual page within the MicroEmacs window. On invoking **man** the user is prompted for the name of the manual page to display:−

```
Man on ?
```

The name of the manual page (and any options) are entered on the command line. The macro invokes the UNIX utility **man(1)** to generate the page and displays the results in a window.

Another manual page can be selected by either moving the cursor to the link and pressing return or double clicking on it with the left mouse button. MicroEmacs will then attempt to load and display the selected manual page.

man−clean removes any man−page formatting codes from the current buffer reducing a manual page to plain text. The formatting codes are used to create the bold and underline fonts. This allows the page to be treated as a normal buffer, i.e. string searches and other similar command will work as expected.

NOTES

man and **man−clean** are macros defined in `hkman.emf`.

man is only made available within UNIX environments, the UNIX start up file `unixterm.emf` links in the macro. If the **man** utility is required on other platforms then the following definition is required in a start−up file.

```
define−macro−file hkman man
```

SEE ALSO

man(9), user−setup(3), spell−buffer(3).

mark−registry(2)

NAME

mark−registry − Modify the operating mode of a registry node

SYNOPSIS

n **mark−registry** "*root*" "*mode*"

DESCRIPTION

mark−registry modifies the *mode* of a registry node *root*. If an argument *n* is supplied then the *n*th register node down from **root** (as viewed from list−registry(2) output) is modified instead. The *mode* is string specifying the modes, each mode is represented by a character. Lower case characters add a mode, upper case characters delete a mode. The modes are defined as:−

? − Query Name

Returns the full name, including path, of the given registry node in the variable $result(5). This does not alter the registry.

! − Hide Value

Hides the value of the given registry node, i.e. its value will not be displayed in the output of list−registry(2). Once set, this mode cannot be removed.

a − Autosave

Automatically saves the registry when it is deleted or unloaded from the registry. The user is not prompted for a save.

b − Backup

Automatically performs a backup of the registry file whenever a save operation is performed.

c − Create

If the registry file cannot be loaded then the *root* node is created and the invocation succeeds. If this mode is omitted then the call fails if the *file* cannot be found.

d − Discard

Marks the registry as discardable. This is typically used for registries that are not saved.

f − File

The registry node is marked as a file root, the value must be set to the registry file name.

g − Get Modes

Returns the list of modes currently set on the given registry node in the variable $result(5). This does not alter the registry.

h − Hidden

The registry node is marked as *Hidden*, i.e. its children will not be shown in list−registry(2) output.

u − Updated

Marks the registry as modified. The modified bit is removed when the registry file is saved. If the modified bit is applied to a registry node the user will be prompted to save the registry when it is deleted (or it will be automatically saved when the *Autosave* mode is used).

Multiple modes may be applied.

EXAMPLE

A history registry can be hidden with the following invocation:−

```
mark-registry "/history" "h"
```

It could then be made visible again using:−

```
mark-registry "/history" "H"
```

BUGS

At exit only registry nodes attached to the root are saved.

DIAGNOSTICS

mark−registry fails if *root* does not exist.

SEE ALSO

get−registry(2), list−registry(2), read−registry(2), set−registry(2), erf(8).

ml−bind−key(2)

NAME

ml−bind−key − Create key binding for message line
ml−unbind−key − Remove key binding from message line

SYNOPSIS

n **ml−bind−key** "*command*" "*key*"
n **ml−unbind−key** "*key*"

DESCRIPTION

ml−bind−key creates a key binding local to the message line input buffer. There are several commands that can be used in message line input, each command is associated with a main buffer editing command and inherits all that commands global bindings, i.e. moving forward 1 character is associated with the command forward−char(2) and thus inherits the binding `C−f` (as well as any other like the right cursor key). The following is a list of available commands, what they do and their associated commands

Cursor Movement

- move backwards 1 character, command: backward−char (**C−b**)
- move forwards 1 character, command: forward−char (**C−f**)
- move backwards 1 word, command: backward−word
- move forwards 1 word, command: forward−word
- move to the beginning of buffer, command: beginning−of−line (**C−a**)
- move to the end of buffer, command: end−of−line (**C−e**)

Input

- Quote a character, command: quote−char (**C−q**)
- Yank kill buffer into message line, command: yank (**C−y**)
- insert current buffers current line into the buffer, command: insert−newline (**C−o**)
- insert current buffers file name into the buffer, command: insert−file−name (**C−x C−y**).
- insert current buffers buffer name into the buffer, command: reyank (**esc y**)

Deletion

- delete backward 1 character, command: backward−delete−char (**C−h**)
- delete forward 1 character, command: forward−delete−char

- kill text from current position to end of line, command: kill−line (**C−k**).
- erase whole line, command kill−region (**C−w**). Note that in incremental searches this is used to add the current word to the search string.

History

MicroEmacs '02 stores the last 20 entries of each kind (command, buffer, file, search and general which is also saved in the history file so the state of the history is retained when next loaded. The following commands can be used to manipulate the history.

- next history list entry (loop through history), command: forward−paragraph (**esc n**)
- previous history list entry, command: forward−paragraph (**esc p**)

Completion

When entering a command, file, buffer or a mode name MicroEmacs '02 creates a list of possible completions the following operations can be performed on this list.

- expand. This completes the given input until the first ambiguous character, command: a space (' ') or tab (**C−i**).
- expand to the previous completion (loops through the completion list, command: backward−line (**C−p**)
- expand to the next completion (loops through the completion list, command: forward−line (**C−n**)
- create a listing of all completions, command: a double expansion, i.e. 2 spaces or tabs. The first expands and the second creates the list.
- page up the completion list buffer, scroll−up (**C−z**)
- page down the completion list buffer, scroll−down (**C−v**)

Miscellaneous

- abort input, returning failure to the input, abort−command (**C−g**)
- re−fresh the message line, command: recenter (**C−l**)
- finish input, command newline (**C−m**, return)
- transpose previous character with current character, command: transpose−chars (**C−t**)
- capitalize the next word, command: capitalize−word (**esc c**)
- Turn the whole of the next word to lower case letters, command: upper−case−word (**esc u**)
- Turn the whole of the next word to upper case letters, command: lower−case−word (**esc l**)

ml−unbind−key unbinds a user created message line key binding, this command effects only the message line key bindings. If a −ve argument is given to **ml−unbind−key** then all message line bindings are removed.

EXAMPLE

If expansion was required on the **esc esc** key binding then use the following:−

```
        ml−bind−key tab esc esc
```

NOTES

The prefix commands cannot be rebound with this command.

Command key response time will linearly increase with each local binding.

SEE ALSO

global−bind−key(2), buffer−bind−key(2), describe−bindings(2), osd−bind−key(2), global−unbind−key(2).

ml−clear(2)

NAME

ml−clear − Clear the message line

SYNOPSIS

ml−clear

DESCRIPTION

ml−clear clears the message line during script execution. This is useful so as not to leave a confusing message from the last command(s) in a script.

Callback macros which may interrupt the user at any point in time are handled by **ml−clear**. The callback macro for instance may interrupt the user while entering a new file name, and any ml−write(2) erases the message−line which may currently be in use. MicroEmacs '02 stores the line and when ml−clear(2) is invoked, instead of clearing the message line the current input line is restored.

SEE ALSO

create−callback(2), ml−write(2).

ml−write(2)

NAME

ml−write − Write message on message line

SYNOPSIS

n **ml−write** "*message*"

DESCRIPTION

ml−write writes the given *message* to the message line. If a positive argument *n* is given then there will be an *n* millisecond uninterruptible delay, giving the user time to see the message.

A call to **ml−write** from a callback macro can erase a message line which is currently being used (to enter a buffer name say). A call to ml−clear(2) restores the previous message−line.

EXAMPLE

The following call displays a message on the message−line with a fixed 2 second pause:

```
2000 ml-write "Hello World!"
```

SEE ALSO

ml−clear(2), command−wait(2), create−callback(2).

name−kbd−macro(2)

NAME

name−kbd−macro − Assign a name to the last keyboard macro

SYNOPSIS

name−kbd−macro "*command*"

DESCRIPTION

name−kbd−macro labels the last defined keyboard macro with the given *command* name. The command name must be either unique or the name of an existing macro. A keyboard macro is deleted when another keyboard macro is defined, but when named, it is preserved. A named keyboard macro can also be bound to its own command key sequence, and may be inserted into a buffer enabling it to be saved and thus re−loaded and re−used at a later date.

SEE ALSO

execute−file(2), execute−kbd−macro(2), global−bind−key(2), insert−macro(2), start−kbd−macro(2).

narrow−buffer(2)

NAME

narrow−buffer − Hide buffer lines

SYNOPSIS

n **narrow−buffer**

DESCRIPTION

The effect of **narrow−buffer** depends on the argument given, defined as follows:−

1

Removes all narrows in the current buffer (Default).

2

Removes the current line's narrow.

3

Narrow to region. Hides all but the lines of test in the current buffer from the mark position to the current cursor position, effectively 'narrowing' the buffer to the remaining text.

4

Narrow out region. Hides the lines of test in the current buffer from the mark position to the current cursor position, opposite to argument **3**.

When a narrow is created the buffer mode narrow(2m) is automatically set, when the last is removed this mode is deleted.

For example, if the buffer contains the following text:

```
1 Richmond
2 Lafayette
3 Bloomington
4 Indianapolis
5 Gary
6
```

If the mark is on line 2 and the current point is on line 4, executing:−

```
4 narrow−buffer
```

Creates one narrow, narrowing out line 2 and 3. Line 4 becomes the narrow anchor line, when the narrow is removed lines 2 and 3 will be inserted before line 4. The buffer will become:−

```
1 Richmond
4 Indianapolis
5 Gary
```

If instead the following was executed:−

```
3 narrow−buffer
```

Two narrows are created, the first narrowing out line 4 and 5 (line 6, the last line, being the anchor line) the second narrowing out line 1 (line 2 being the anchor line). The buffer will become:−

```
2 Lafayette
3 Bloomington
6
```

Executing **narrow−buffer** with an argument of **2** will only work on the anchor lines, namely 4 in the first example and 2 and 6 in the second.

NOTES

Alpha mark set by set−alpha−mark(2) in text which is subsequently narrowed out will automatically remove the narrow if the user returns to the mark using goto−alpha−mark(2).

get−next−line(2) operates on the unnarrowed buffer and will remove any narrows hiding the 'next' line.

EXAMPLE

c−hash−eval(3) macro defined in cmacros.emf uses narrow−buffer to hide regions of source code which has been #defined out, improving readability.

vm(3) defined in vm.emf uses narrow−buffer with appropriate arguments to append−buffer(2) and write−buffer(2) to write out only parts of the current buffer.

SEE ALSO

narrow(2m), set−mark(2), set−alpha−mark(2), get−next−line(2), c−hash−eval(3), vm(3).

newline(2)

NAME

newline − Insert a new line

SYNOPSIS

n **newline** (**return**)

DESCRIPTION

newline inserts *n* new lines into the text, move the cursor down to the beginning of the next physical line, carrying any text that was after it with it. The next line may automatically be indented depending on the current buffer mode, see cmode(2m), indent(2m), and wrap(2m).

SEE ALSO

cmode(2m), indent(2m), wrap(2m), buffer−mode(2).

next−frame(2)

NAME

next−frame − Change the focus to the next frame

SYNOPSIS

n **next−frame**

DESCRIPTION

next−frame changes the focus to the next frame. The numerical argument *n* can be used to select the type of frame to change to, it is a bit based flag defined as follows:

0x01

Allow the selection of an external frame.

0x02

Allow the selection of an internal frame. The default operation when *n* is omitted is to allow the selection of either type of frame (equivalent to an argument of 3). **SEE ALSO**

create−frame(2), delete−frame(2).

next−window(2)

NAME

next−window − Move the cursor to the next window
previous−window − Move the cursor to the previous window

SYNOPSIS

n **next−window** (**C−x o**)
n **previous−window** (**C−x p**)

DESCRIPTION

next−window makes the next window down the current window, if the current window is the last one in the frame the first one is selected. The numeric argument *n* can be used to modify this default behaviour, it is a bitwise flag where the bits are defined as follows:

0x01

If there is no 'next' window because this is the last then if this bit is set the search for the next window is allow to continue with the first window of the frame. As the default argument *n* is `1` this is the default behaviour.

0x02

When this bit is set windows whose $window−flags(5) are set to be ignored by this command are not skipped. The setting of bit `0x010` of a windows **$window−flags** will make the default action of this command skip it which means the the command may not simply select the next window but the next window without this flag set. Setting this bit of the numeric argument will force the command to always select the next window.

0x04

When set the search for the next window starts at the first window instead of the current window, this can be used to find the first window in the current frame.

previous−window makes the next window up the current window. The numeric argument *n* has the same effect on this command as for **next−window** except bit **0x04** starts the search at the last window of the frame.

EXAMPLE

The following example visits every window in the current frame printing the buffer it displays with a second pause between each one:

```
; go to the first window
!force 6 next-window
!while $status
    1000 ml-write $buffer-bname
    ; go to the next window - fail if this is the last
    !force 2 next-window
!done
```

NOTES

Both commands fail if a suitable window cannot be for, see the example on how this can be used.

SEE ALSO

next−window−find−buffer(2), next−window−find−file(2), set−position(2), goto−position(2), $window−flags(5).

next−window−find−buffer(2)

NAME

next−window−find−buffer − Split the current window and show new buffer

SYNOPSIS

next−window−find−buffer "*buffer*" (**C−x 3**)

DESCRIPTION

next−window−find−buffer splits the current window into two near equal windows, and swaps the current windows buffer to the given *buffer*. It is effectively a split−window−vertically(2) command followed by a find−buffer(2). When there is insufficient space in the current window to perform the split, then the current window is replaced. The requested *buffer* is always displayed, if the buffer does not already exist it is created.

SEE ALSO

find−buffer(2), split−window−vertically(2), next−window−find−file(2).

next−window−find−file(2)

NAME

next−window−find−file − Split the current window and find file

SYNOPSIS

next−window−find−file "*file*" (**C−x 4**)

DESCRIPTION

next−window−find−file splits the current window into two near equal windows, and loads the given *file* into the current window. It is effectively a split−window−vertically(2) command followed by a find−file(2).

When there is insufficient space in the current window to perform the split, then the current window is replaced. The requested *file* is always displayed, if the file does not already exist it is effectively created within MicroEmacs (although it will not exist on the disk until a save operation is performed).

The numeric argument *n* can be used to modify the default behaviour of the command, where the bits are defined as follows:

0x01

If the file does not exist and this bit is not set the command fails at this point. If the file does not exist and this bit is set (or no argument is specified as the default argument is `1`) then a new empty buffer is created with the given file name, saving the buffer subsequently creates a new file.

0x02

If this bit is set the file will be loaded with binary(2m) mode enabled. See help on **binary** mode for more information on editing binary data files.

0x04

If this bit is set the file will be loaded with crypt(2m) mode enabled. See help on **crypt** mode for more information on editing encrypted files.

0x08

If this bit is set the file will be loaded with rbin(2m) mode enabled. See help on **rbin** mode for more information on efficient editing of binary data files. **SEE ALSO** find−file(2), next−window−find−buffer(2), split−window−vertically(2), binary(2m), crypt(2m), rbin(2m).

normal−tab(3)

NAME

normal−tab − Insert a normal tab

SYNOPSIS

n **normal−tab**

DESCRIPTION

normal−tab insert a tab into the current buffer by temporarily disabling any auto indentation schemes. The macro first disables any indentation rules by setting $buffer−indent(5) to 0 and disabling the cmode(2m), the command tab(2) is then called with the given argument *n*. This means that the buffer's tab(2m) mode setting will be respected, i.e. whether a tab character or spaces are inserted. The initial indentation rules are restored on exit.

NOTES

normal−tab is a macro implemented in `format.emf`.

SEE ALSO

tab(2), insert−tab(2), tab(2m).

organizer(3)

NAME

organizer − Calendar and address organizer

SYNOPSIS

organizer

DESCRIPTION

organizer is a calendar and address organizer, enabling notes to be stored against the calendar days; addresses may be archived into an address book.

organizer uses the MicroEmacs '02 in−built registry to store information within a registry file called *<username>*`.eof`. **organizer** may be entered directly from the command line, or via the menu (via **Tools**).

organizer is displayed within a single osd dialog box, tab selections at the top of the window enable the different forms of information to be displayed. Navigation is typically performed using the mouse, where the mouse is absent then the `TAB` key may be used to move between the fields. The information presented is defined as follows:−

Month

Shows the calendar month, starting with the current month, the current day is hi−lighted and any notes that have been entered are displayed in the **Notes** entry box at the bottom of the page.

The default mode of operation is note entries for the current month, however specifying the *<year>* as the wild card '*' (star) enables annual events to be entered into the organizer. Annual events are automatically inserted into the calendar each year, typically used for birthdays etc.

The entry controls to the dialog are defined as follows:−

<−

Advances to the previous month.

−>

Advances to the next month.

<Month>

A pull down dialog enabling month selection.

<year>

A text entry field specifying the current year as a 4 digit number. A value of ***** is the wild card year for specifying annual events.

Notes

A free form text entry box allowing a note to be attached to the currently selected day.

Save

Saves the entry back to file.

Month To Buffer

Dumps a view of the month to the currently active buffer, any notes are also dumped to the buffer.

Exit

Exits the **organizer**. **Week**

Shows the calendar week in the current buffer, the days of the week are shown in a column ordering. Note that selection of the week is typically performed from the **Month** view, moving to the **Week** view (via the tab) selects the week appropriate to the previously selected day within the month view.

The entry controls on the dialog are defined as follows:−

<−

Advances to the previous week.

−>

Advances to the next week.

<year>

A text entry field specifying the current year as a 4 digit number. The value of ***** for viewing and setting annual events is not valid in this view.

Notes

A free form text entry box allowing a note to be attached to the currently selected day.

<day>

Selecting a date in the *day* column changes the view to the **Day** view.

Save

Saves the entry back to file.

Week To Buffer

Dumps a view of the week to the currently active buffer, any notes are also dumped to the buffer.

Exit

Exits the **organizer**.

Note: The start day in the week view may be configured to commence on a day other than Sunday from the **Setup** tab.

Day

Shows an extended view of the notes attached to the current day, day selection is typically performed from the **Month** or **Week** views. The entry controls on the dialog are defined as follows:−

<−

Advances to the previous day.

−>

Advances to the next day.

<year>

A text entry field specifying the current year as a 4 digit number. A value of ***** is the wild card year for specifying annual events.

<month>

A pull down dialog enabling month selection.

<day>

A text entry enabling the current day to be entered.

Notes

A free form text entry box allowing a note to be attached to the currently selected day.

Save

Saves the entry back to file.

Day To Buffer

Dumps a view of the day to the currently active buffer, any notes are printed in the buffer.

Exit

Exits the **organizer**. **Lists**

The lists pane provides support for multiple list generation and manipulation. Each list consists of zero or more ordered items each of which has a text field in which the user can enter information.

Entry to the dialog is defined as follows:−

List

Selects a list.

New

Creates a new list.

Lines Per Item

Sets the number of lines to use when displaying a list item.

New

Creates a new list item at the end of the current list.

Up

Moves the currently selected item (left click on the item number) up the list.

Down

Moves the currently selected item down the list.

Insert

Inserts a new list item before the currently selected item.

Delete

Deletes the currently selected item.

Save

Saves the entry back to file.

List To Buffer

Dumps a view of the list to the currently active buffer.

Exit

Exits the **organizer**. **Address**

The address pane provides entry to the address book, enabling personal and business details to be retained against a single name, tabbed selection of **Work** or **Home** selects the information that is displayed. A search engine is provide to locate names within the database, and provision is made to save some text against a name. Entries in the database are, by default, organized by record number, sorting may be explicitly performed from the **Sort** button.

Entry to the dialog is defined as follows:−

<Record No>

The identity number of the record, a value of ***** denotes that this is a new record that is being inserted.

<<

Moves to the start of the database.

>>

Moves to the end of the database, showing record *****, a new entry may be entered.

<

Moves to the previous record.

>

Moves to the next record.

Name

The name of the individual, entered as *fore−name* and *surname*.

Nickname

A pseudo name assigned to an individual.

Partner

Shown in the **Home** view only. The *forename* and *surname* of any partner.

Chld

Shown in the **Home** view only in the **Extended Address Book Mode**. The names of any children (up to 3).

DOB

Date of Birth, shown in the **Home** view only in the **Extended Address Book Mode**. The dates of birth of the parents, any children in addition to an anniversary date.

Company

Shown in the **Work** view only. The name of the company.

Address

The address of the individual/company.

Tel/Fax/Mobile

Telecommunication information.

Email/WWW/FTP

Electronic communication information.

Notes

Notes associated with the individual.

Save

Saves the address information to file.

Dup

Duplicates the currently selected address entry, creating a new record card. Typically used to construct a similar entry for a different individual.

Delete

Deletes the currently selected entry.

Addr to Buffer

Dumps the currently selected address to the current buffer.

Exit

Exits the organizer.

Find

find provides access to a search engine, enabling addresses to be located in the address book.

Search For

The string to search for.

In Field

Pull−down menu allowing the selection of the field to be searched in.

Match

Selects how strict the search should be; typically **Any Part** is used as this is the least in−exact search. The default mode is configured in the **Setup** tab.

Case/magic

Selects the search criteria. The default mode is configured in the **Setup** tab.

First

Finds the first record that matches the search criteria

Next

Finds the next record that matches the search criteria, from the currently displayed record.

Reverse

Searches in reverse order.

Exit

Exits the search

Sort

sort provides a mechanism to re−sort the data base into a different order. The sort is performed on up to 3 different keys enabling conflicting primary sort fields to be resolved by the secondary sort criteria. The default sort order is *<Record No>*, *<None>*, *<None>*.

Sort Keys

The *Primary*, *Secondary* and *Tertiary* sort fields are selected by a pull down menu. The fields to be used for sorting are selected from the list.

Sort

Performs the sort, based on the settings of the *Sort Keys*.

Exit

Exits the sort dialog. **Setup**

The **setup** pane configures a number of general settings of the organizer.

Current Organizer File

The full pathname of the organizer file. By default this is set to *<userpath><userName>***.eof** and can be altered using user−setup(3).

Change Name

Allows the displayed name of the month and the day to be modified.

First Day of the week

Selects the first day of the week, this sets the first day to be displayed in the **Week** view and the first column in the **Month** view.

Min New Year Days

The number of days that must appear in the first week of the New Year for the week to be considered week 1. Modifying the value of this field modifies the week number.

The **Calendar** section allows the wordy representation of the calendar date to be modified. Typically used to modify the names to the native language.

Change Month Name

Select the existing month representation from the left−hand box and type in a new selection into the right−hand box.

Change Week Day Name

As *Change Month Name*, enables the day of the week representation to be modified.

First Day Of The Week

Selects the first day that appears in the **Week** view.

Minimum Days of New Year in first week

Specifies the number of days that must appear in the first week of the New Year for the week to be designated as week 1. This value allows the week number to be aligned with the calender weeks of standard diaries. The default value is 7 days; but may be reduced to 5 or 6 for typical alignment.

The **Address Book** section allows the operation of the address book to be modified.

Use Extended Address Book

The extended address book allows additional information to be added to the personal address book. The extended information is limited to the amount of personal information attributed to an individual, including *Date of Birth* and *Child* information.

Import From File

> The **Import** from file allows the address book to be imported from a file. The import data format is a single line per entry, comma `,` separated. The field order is defined as follows, the ***** entries indicate the **Extended Address Book** fields:−
>
> > *Record No*, *First Name*, *Surname*, *Nick Name*, *Selected*, *Notes*, *Partner First Name*, *Partner Surname*, *Home Address*, *Home Telephone*, *Home Fax*, *Home Mobile*, *Home E−Mail*, *Home WWW Page*, *Home FTP Site*, *Work Company*, *Work Address*, *Work Telephone*, *Work Fax*, *Work Mobile*, *Work E−Mail*, *Work WWW Page*, *Work FTP Site*, *Date−Of−Birth**, *Partner DOB**, *Date−Of−Marriage**, *Child1 Name**, *Child1 DOB**, *Child2 Name**, *Child2 DOB**, *Child3 Name**, *Child3 DOB**.

Export To File

Exports the address book to a file, the address book is exported in the current sort order, with the fields defined as above. The exported address book may then be imported into a 3rd party package i.e. Microsoft Access, etc.

The **Default Address Find Settings** section defines the default search criteria used in the address book search function.

Whole/Start/Any Part

> Radio buttons determine how the search is performed on the string.
>
> · **Whole** matches the whole string exactly.
> · **Start** matches the first part of the string only (i.e. `Ab*`).
> · **Any Part** finds entries that include the search string at any position within the data base search field.

Case Insensitive

Checked, matches the strings regardless of case. (default).

Magic Mode

Allows magic strings to be included in the search string. **NOTES**

organizer is a macro that is implemented in `organiz*.emf` files. Organizer uses osd(2) to create and manage the dialogs.

The maximum size of a text note is 1024 characters.

With an new address is created it is added to the end of the address list regardless of the current sort criteria.

Organizer replaces the original **Calendar** utility.

SEE ALSO

user−setup(3), osd(2).

osd(2)

NAME

osd − Manage the On−Screen Display

SYNOPSIS

osd
−1 **osd**
−2 **osd**
n **osd**
−1 osd *n*
osd −1 *flag*
osd *n* **0** *flags* ["*scheme*"] ["*x−pos*" "*y−pos*"] ["*min−width*" "*min−depth*" "*max−wid*" "*max−dep*"] ["*default*"] [["*title−bar−scheme*"] ["*Text*"]] ["*resize−command*"] ["*control−command*"] ["*init−command*"]
osd *n i flags* ["*tab−no*"] ["*item−scheme*"] ["*width*" "*depth*"] ["*text*"] ["*argument*" "*command*"]

DESCRIPTION

The **osd** command manages the On−Screen Display, menu and dialogs. The command takes various forms as defined by the arguments. Each of the argument configurations is defined as follows:−

Main Menu−Bar Status

osd −1 *flag*

This invocation determines the state of the top main menu bar. The state is set by the argument *flag* defined as:−

`1` − enable.
`0` − disable.
`−1` − disable and destroy.

Dialog Creation and Redefinition

osd *n* **0** *flags* ["*scheme*"] ["*x−pos*" "*y−pos*"] ["*min−width*" "*min−depth*" "*max−wid*" "*max−dep*"] ["*default*"] [["*title−bar−scheme*"] ["*Text*"]] ["*resize−command*"] ["*control−command*"] ["*init−command*"]

This invocation creates or resets the base properties of dialog *n*. The *flags* argument determines the arguments and are defined as follows:

A

Defines dialog as an alpha type dialog, items are added according to their string text value. Alpha dialogs may not have separator or child items.

i

Used with the **A** flag, sets the alpha ordering to be case insensitive.

G

Create a Grid dialog. Every item in the dialog is given a single character boarder around it. If one of the dialogs items is also given a '**G**' flag, the boarder is drawn as a box around it, otherwise spaces are used.

N

Create a Note−Book (or tabs) dialog. The dialog can only contain one dialog inclusion item ('**I**') and Note−Book pages ('**P**'). Pages added before the Inclusion item (page item number is lass than the inclusion page item number) will be drawn at the top of the note−book, those added after will be drawn at the bottom.

b

Draw boarder, draws a boarder around the outside of the dialog. See also *flag* **t** (title) as flag effects the boarder.

a

Defines the absolute start−up position of the dialog in the arguments *x−pos* and *y−pos*, which are the column and row positions respectively of the dialog from the top left−hand corner of the display. The arguments must be specified. e.g. the main menu is defined with an absolute position of (0,0). If the dialog can not be fully drawn on the screen at the given position it will be moved to a position which shows the most.

o

Specifies an offset to the dialog position calculated by MicroEmacs in the arguments *x−pos* and *y−pos*, which are the column and row offsets. This flag is ignored when flag **a** is also specified. If the dialog can not be fully drawn on the screen at the new position it will be moved to a position which shows the most.

s

Sets the size of the dialog. **osd** automatically resizes a dialog to fit the contents, this flag should be considered as a size hint for **osd**, and is not guaranteed to be honored. If the dialog has a boarder (flag **b**) the size given should include the boarder size.

The arguments, *min−width*, *min−depth*, *max−width* and *max−depth* must be specified, as

+ve

The actual size of the dialog, minimum and maximum sizes.

0

min value should be specified as desired window size, *max* may be 0 which specifies the screen size.

−ve

min defines the maximum size. *max* is unlimited.

The following table shows possible combination of the sing parameters and their effect:−

min=0, *max*=0

Default setting, makes dialog as small as possible, with a maximum size of the screen.

min=0, *max*=50

Make dialog as small as possible with a max of 50 characters.

min=50, *max*=0

Make dialog as small as possible, but make it at least 50 characters big and no larger than the screen.

min=30, *max*=−1

Make dialog at least 30 characters big with no upper limit, very useful for dialogs being used as scrolled children.

min=−1, *max*=50

Make dialog 50 characters big.

min=−1, *max*=0

Make dialog the same size as the screen.

min=−1, *max*=−1

Make dialog as big as possible (do not do this unless you have a large amount of memory to throw away).

S

Sets the main dialog scheme, The default scheme when not specified is $osd−scheme(5) See macro file `fileopen.emf` for an example.

d

Sets default item to select to "*default*". This item is selected when the dialog is first opened, if this item is an automatically opened sub−menu then the child menu will also be opened.

t

Title bar is present − draws the title bar. The *text* argument is optional Also see flags **H**, **c** and **r**.

H

Defines the title bar color scheme if flag **t** is specified. If *t* is absent the option is ignored.

c

Centers the title bar text if specified. Option **t** must be specified, otherwise the option is ignored.

r

Right justifies the title bar text if specified. Option **t** must be specified, otherwise the option is ignored.

R

Defines the dialog as re−sizable. The *resize−command* argument must be specified and the command should resize the dialog to the sizes given in $result(5) in the format "`wwwwdddd`", where `w` is width and `d` the depth. If the *resize−command* is aborted then that resize operation is abandoned.

M

Identifies the dialog as the main menu dialog.

C

Binds a command to the dialog, which is automatically executed when the dialog is opened. When the dialog with a **C** attribute is opened, it is rendered on the screen and then a command, defined by *control−command* is invoked, when the command completes the dialog is closed.

The command dialog is typically used to create status messages. e.g. a "`Busy - Please Wait`" dialog box, such a dialog may be implemented when saving the current buffer then create the simple dialog and sent the *control−command* to save−buffer(2). The dialog would be defined as:−

```
osd 200 0 "btcHC" %osd-title-scheme "Saving Buffer" save-buffer
osd 200 1 ""
osd 200 2 "" "Busy - Please Wait"
osd 200 3 ""
200 osd
```

If the dialog has buttons which need to become active then control can be returned to **osd** by calling **osd** with no arguments, e.g. in the above example the dialog can be made to stay on the screen until the user selects `okay` by:

```
define-macro test-osd
    save-buffer
    osd 200 2 "" "Save Complete"
    osd 200 4 "BcfH" %osd-ebtt-scheme "  &Okay  " f void
    osd
!emacro

osd 200 0 "btcHC" %osd-title-scheme "Saving Buffer" test-osd
osd 200 1 ""
osd 200 2 "" "Busy - Please Wait"
osd 200 3 ""
osd 200 4 "BcfHS" %osd-dbtt-scheme "  Okay  "

200 osd
```

The above mechanism is how spell−buffer(3) operates.

k

Disables hot−keys for the dialog. All text strings are copied literally. This is useful for dialogs like the tags child dialog as many tags have '&'s in them.

B

Makes the mouse right Button have the same behaviour as the left, by default the right mouse button simply closes the dialog. This is useful for some dialogs which are opened using the right mouse button.

f

Automatically uses the first letter of an item's test as the hot key. Unlike the normal hot keys, the letter is not hi−lighted and when typed by the user the item is only selected, not executed. This flag also disables the normal hot−keys for the dialog, so all text strings are copied literally.

n

Disables '\n' characters in text fields leading to multi lines. By default a text item of "Hello\nWorld" will create an item 5 by 2 characters big.

If "*init−command*" is given then this function is always called just prior to the dialog being displayed so it can be used to configure the dialog.

Dialog Destruction

−1 osd *n*

This invocation destructs a dialog *n*. The dialog is only destroyed if it is not currently being displayed.

Dialog Item Creation and Redefinition

osd *n i flags* ["*tab−no*"] ["*item−scheme*"] ["*width*" "*depth*"] ["*text*"] ["*argument*" "*command*"]

This invocation type adds a new item *i* to a dialog *n*, the operation of the invocation is controlled by the *flags* as follows:−

D

Disable item *i*, the item is ignored and is not rendered in the dialog.

I

Include dialog "*argument*" into this dialog. If "*command*" is specified then it is called prior to the child being constructed and can be used to define the child. This is similar to the **M**s command. See also flag **b**.

P

Item is a Note−Book page, the item must have text and have an argument which is the osd dialog to be show when the page is activated.

M

Item is a sub−menu, The argument "*argument*" specifies the sub−menus osd dialog number. A "*command*" may also be specified which is executed first, this can actually re−define the item and set the dialog number, e.g.

```
; To start with the dialog number is unknown
osd 1 1 "M" f submenu−setup

define−macro submenu−setup
    osd 200 0 ....
    ....
    ; Now the sub−menu number is known redefine parent item,
    ; note the setup  command  is not given as we have now set
    ; it up!
```

```
            osd 1 1 "M" 200
        !emacro
```

See also options **m**, **n**, **e**, **s**, **w** and **d**.

m

Sub−menu must be manually opened, using hot−key, the return key or the left mouse button.

n, **e**, **s**, **w**

Specify where a sub−menu is to be placed relative to the parent item. The letter indicates the direction as points on a compass, North, East, South and West, respectively. The default when omitted is East.

d

Display sub−menu type, i.e. "`..`" for auto opening and " >" for a manual opening sub−menu.

−

Fill a non−defined chars with '**−**'s instead of ' 's, used to draw the lines across menus, typically with no text given, e.g.

```
osd 200 5 "−"
```

But could also be specified as:

```
osd 200 5 "−c" "Lined"
```

C

Item is a check−box. The setting of the check−box is evaluated when the dialog is first drawn, re−draw and whenever any item is executed. A "*command*" must be specified which must both return the current setting when the given argument (of 1) is given (!abort if false, !return if true) and change the value if the argument value is negated. The text string must also be specified, the first 6 characters are used in the drawing of the check box. The format can be shown as follows:−

```
String\State       Off          On
"123456"            "12356"      "12456"
" (−+)^"            " (−)"       " (+)"
"^[ *] "            "[ ] "       "[*] "
"^^NY^^"            "N"          "Y"
"^^^^^^"            ""           ""
```

Note that no character is rendered when a '^' character is used. See also **p** for prepending the check−box.

p

Prepend the check−box box. By default a check box is drawn as:

```
"Check box12?56"
```

This option changes it to:

```
"12?56Check box"
```

x

When the item is executed do not exit the dialog. Often used with Check−boxes.

i

The command given is a command line string which is executed in a similar fashion to execute−line(2). Note that if an argument is required it is usually specified in the string, i.e.

```
osd "i" "text" 5 "1000 ml−write @#"
```

writes the argument (i.e. 5) for 1 second.

```
osd "i" "text" 5 "my−command"
```

in this case *my−command* will not be given an argument,

```
osd "i" "text" 5 "10 my−command"
```

in this case *my−command* will be given an argument of 10,

```
osd "i" "text" 5 "@# my−command"
```

in this case *my−command* will be given an argument of 5.

h

Horizontally add the next item, e.g.

```
osd "h" "1st on line "
osd ""  "2nd on line"
```

Will produce "`1st on line 2nd on line`". If there is not enough room on a single dialog line to display all the horizontally added items then the line is split and as many lines as needed are used.

c

Center the text for the item in the middle of the dialog.

r

Right hand justify the text for the item.

t

Set the items tab order in the dialog.

b

Child inclusion is a scroll box type. By default a child inclusion simply draws the whole child dialog at the position. If this flag is specified then arguments "*width*" and "*depth*" must also be supplied and a window displaying "*width*" by "*depth*" of the child is created. The size of this item will be "*width*"+1+*ss* by "*depth*"+1+*ss* where *ss* is the scroll bar size which is 1 or 2 depending on the setting of $scroll−bar(5). It is up to the user to ensure that the child dialog being displayed is at least "*width*" by "*depth*" characters in size, if this is not true then the effect is undefined, (a crash dump is not out of the question).

f

Fix the item size to the given "*size*", by default an item is expanded to the width of the dialog.

E

Item is an entry box type. Use a string of #'s to set the position and size of the entry text box. Similar to Check−boxes, the command given must both return and set the value depending on value of the argument given. The value must be returned in $result(5) if the given argument (or 1 for '`f`') is given, and the value must be set (usually using @ml(4) or @mc(4)) if the argument is negated. The absolute value of the argument is maintained.

```
set−variable %entry−value "Hello world"

define−macro my−entry−set
    !if &equ @# −1
        set−variable %entry−value @ml "" %entry−value
    !else
        set−variable $result %entry−value
    !endif
!emacro

osd 200 1  "S" " &Enter text" 2
osd 200 2  "ExHf" %osd−entry−scheme "########" 1 my−entry−set
```

B

Item is a Button type. Add the last 2 characters of $window−chars(5) to the text string given, one on each side, i.e. if the last two chars are "[]" then:

```
osd "B" " Okay "
```

will be drawn as "`[ Okay ]`". See also flag **T**.

T

Item is a repeat type, this is typically used with buttons, altering their execution behavior. By default an item is only executed when the left mouse button is released while over the item. However when this flag is specified the item is executed as soon as the left mouse button is pressed and is repeatedly executed until the button is release or the mouse moves off the item. The delay between repeated executions is determined by the variables $delay−time(5) and $repeat−time(5).

S

Item is a separator type. This is not often required as any item without anything to execute is automatically set to be a separator. Occasionally a mouse−insensitive item which can be executed is required, typically a text string with a hot key, e.g.

```
osd 200 1  "S" " &Enter text" 2
osd 200 2  "ExHf" %osd−entry−scheme "########" 1 my−entry−set
```

will be drawn as "`[ Okay ]`"

Item 1 will have a hot−key which executes item 2 (as no command is given), but it will not hi−light if the mouse is placed over it.

R

Redraw dialog. Forces a redraw of the dialog when the item is executed. This is not usually required as **osd** generally works out for itself whether a redraw is needed, however, sometimes it does not, most notably when the item sets a variable that is displayed by another item as an entry, e.g.

```
set−variable %entry−value "Hello world"

define−macro my−entry−set
    !if &equ @# −1
        set−variable %entry−value @ml "" %entry−value
    !else
        set−variable $result %entry−value
    !endif
!emacro

osd 200 1  "S" " &Enter text" 2
osd 200 2  "ExHf" %osd−entry−scheme "########" 1 my−entry−set
osd 200 3  "BxHcfiR" %osd−ebtt−scheme  " &Reset " f "set−variable %er
```

If item 3 did not have flag **R** set when executed, **osd** would not realize that the change to value `%entry−value` affects the display and the button would not appear to operate.

H

Sets the item color scheme. Note that for scrolled child items this only sets the scroll−box color scheme, the dialog scheme is used for the rest of the boarder.

G

This flag is only applicable in grid dialogs (see flag **G** in dialog creation). The current item will be drawn with a box around it using $box−chars(5).

z

Sets the item size, arguments "*width*" and "*depth*" must be given.

N

This flag only has an effect on entry item types, it selects 'New−line' style text entry which allows the user to enter multiple line of text using the return key and to end the input using the tab key.

Note that for a non−sub−menu item type, if an argument is given with no command then it is assumed that the number given is the item number to be executed, see flag **S** for an example.

Dialog Exacution

n **osd**

This invocation with a single positive numeric argument executes the *n*th dialog.

Returning Command Control

osd

An invocation of **osd** with no arguments returns control back to the **osd** from a *control−command*. Refer to the **C** flag in the create/reset dialog property for information and an example.

Current Dialog Redraw

−1 **osd**

Calling osd with an argument of −1 forces the complete redrawing of current dialog and any sub−dialogs. This is very useful when the execution of one item may effect the appearance of another.

Redraw All Active Dialogs

−2 **osd**

Calling osd with an argument of −2 forces the complete redrawing of all currently active osd dialogs. This is better than calling screen−update(2) when only the osd dialogs need updating as it suffers less from flickering.

EXAMPLE

Refer to `osd.emf`, `userstp.emf`, `search.emf`, `spell.emf` and `organize.emf` for examples of the OSD.

SEE ALSO

$osd−scheme(5), $result(5), $scroll−bar(5), $window−chars(5).

osd−bind−key(2)

NAME

osd−bind−key − Create key binding for OSD dialog
osd−unbind−key − Remove key binding from OSD dialog

SYNOPSIS

osd−bind−key *n* "*command*" "*key*"
osd−unbind−key *n* "*key*"

DESCRIPTION

osd−bind−key creates a local key binding for a given osd dialog, binding the command *command* to the keyboard input *key*. Only the current root dialog's local bindings are used, local bindings of included dialogs or other root dialogs currently displayed are ignored.

Osd local bindings take priority over default osd bindings, local bindings created using ml−bind−key(2) are also used, but any current buffer local bindings created using buffer−bind−key(2) are ignored.

NOTES

The prefix commands cannot be rebound with this command.

Key response time linearly increases with each osd binding added.

As only the root dialog's bindings are used, creating note−book page specific bindings can be awkward. Typically all required keys are bound to the same command which, depending on the page that is currently being displayed, checks if the key pressed is bound on the current page and if so calls the required command. See organizer(3), defined in `organize.emf` for an example of this operation.

SEE ALSO

osd(2), global−bind−key(2), ml−bind−key(2), buffer−bind−key(2), global−unbind−key(2).

osd−dialog(3)

NAME

osd−dialog − OSD dialog box
osd−xdialog − OSD Extended dialog box
osd−entry − OSD entry dialog box

SYNOPSIS

n **osd−dialog** "*title*" "*prompt*" ["*x−pos*" "*y−pos*"] "*but1*"
n **osd−xdialog** "*title*" "*prompt*" *default* ["*x−pos*" "*y−pos*"]

"*but1*" "*but2*" ...
n **osd−entry** "*title*" "*prompt*" *variable* ["*x−pos*" "*y−pos*"]

[["*entry−xsize*" | "*entry−xsizexentry−ysize*"] ["*type*"]] **DESCRIPTION**

osd−dialog constructs a OSD dialog prompt with a title string *title*, a prompt string within the dialog of *prompt*. A single button, with text rendering *but1*, is placed within the dialog. The dialog remains on the screen until the button is selected or the user aborts.

osd−xdialog creates an extended dialog with multiple buttons similar to **osd−dialog**, the number of buttons created (*#*) is determined from the number of *but* arguments. The *default* integer argument specifies the default button (1..*#*), a value of 0 specifies that there is no default button.

The commands return the button pressed in the variable $result(5).

osd−entry constructs a simple OSD entry dialog which prompts the user to type in a value. The value of the supplied variable is used as an initial entry value, the variable is set to the entered value when the user presses the "`Okay`" button but remains unchanged if the user Cancel or aborts.

The size of the entry defaults to 30 characters if not specified by the user, when a size parameter is given it can take one of two forms, either simply "`w`" specifying the width, the height defaulting to 1, or "`wxh`" (i.e. "`40x5`") specifying both. The last optional argument *type* sets the type of value being entered (e.g. file name, buffer name, etc) see flag **h** on the help page for @ml(4) for a list of entry types and the numerical value to be supplied.

The argument *n* can be used to change the default behavior of the commands described above, *n* is a bit based flag where:−

0x01

Enables command abort (default), except **osd−entry** which ignores the setting of this bit. When enabled, if the user abort by either closing the dialog (top right button) or using the **abort−command** the dialog command will also abort. If bit 0x01 is not set the command will not abort and **$result** will be set to −1.

0x02

When set, flags that a dialog position has also been provided, extra arguments **x−pos** and **y−pos** must also be given. By default the dialog is placed under the mouse. **EXAMPLE**

A simple query dialog is typically constructed using **osd−dialog**, as follows:−

```
!if &seq %osd-search-str ""
    osd-dialog "Replace" "Error: Search string is empty!" "  &OK  "
    !return
!endif
```

The following example uses multiple buttons within a single dialog, using **osd−xdialog**, as follows:−

```
0 define-macro osd-close
    !if &bmod "edit"
        set-variable #l0 &spr "Buffer \"%s\" changed" $buffer-bname
        osd-xdialog "Buffer Close" #l0 1 "&Save First" \
                                        "&Loose Changes" "&Cancel"
        !if &equ $result 3
            !abort
        !elif &equ $result 2
            -1 buffer-mode "edit"
        !else
            !if &seq $buffer-fname ""
                !nma write-buffer
            !else
                !nma save-buffer
            !endif
        !endif
    !endif
    delete-buffer $buffer-bname @mna
!emacro
```

The next example macro can be used to change the value of a user variable to a user supplied file name:

```
set-variable %source-root "~/"

define-macro set-source-root
    osd-entry "Source Root" "&Path : " %source-root 35 1
!emacro
```

NOTES

osd−dialog, **osd−xdialog** and **osd−entry** are macros defined in `osd.emf`, using osd(2) to create the dialog.

SEE ALSO

$result(5), osd(2).

osd−help(3)

NAME

osd−help − GUI based on−line help

SYNOPSIS

osd−help

DESCRIPTION

osd−help provides a GUI front end to the on−line help manual, the dialog consists of 3 pages which are defined as follows:−

Contents

The contents page displays a list on contents similar to the help(2) high level help page. Selecting an item will display the help page in a buffer, selecting **Exit** will exit the dialog.

Index

The index page gives a list of help items, the **Scope** menu can be used to narrow the index list to the required item type.

Search

The search page provides a way of searching the on−line help for a given topic. Similarly to the Index page, the **Scope** menu is provided to narrow the search to the required area.

The search strings is considered to be made up of items separated by spaces, an item can be enclosed in quotes ('**"**') so that the item can include a space. If the first letter of an item is a '**+**' the given item must be found in a page for it to match, if the character is a '**−**' the item must NOT be found on a page for it to match, or other items are considered optional. At least one item must be found on a page for it to be a match, the numbers to the right of each found page is the number of items found.

NOTES

See Help! for help on the on−line help pages.

osd−help is a macro using osd(2), defined in osdhelp.emf.

SEE ALSO

help(2).

Patience(3)

NAME

Patience − MicroEmacs '02 version of Patience (or Solitaire)

SYNOPSIS

Patience

DESCRIPTION

Patience (or Solitaire) is a solitaire game using a standard set of playing cards. The object of the game is to use all of the cards in the deck to build up four suit stacks from Ace to King.

The board is laid out with the dealer pile at the top right hand corner, to the left are four suit stacks onto which cards of the same suit are placed, in ascending order from the Ace. Below these two areas of the board are seven row stacks, organized in a triangular shape with zero to six downward facing cards.

Cards may be moved around the playing area by stacking alternative red and black cards in descending order on the row stacks. When a row stack has no upturned cards on the stack then the top card may be turned over and may be played. If a stack becomes empty then only a King may be moved into the vacant position. Cards may be removed from the dealer, they are over−turned in sets of three cards, the underlying 2 cards are visible, but are not accessible, only the top card may be removed and played from the dealer.

Cards are moved around the board using the mouse. Cards may be moved from the dealer or between the row stacks by placing the mouse over the card to be moved and pressing the left mouse button. Move the cursor to the new card position and release the left mouse button. If the move is legal then the card(s) are moved to the new stack. Multiple cards may be moved from the row stacks, the appropriate card(s) to be moved is automatically determined.

Cards may be moved onto the suit stacks by a single left mouse press and release on the same card, the card is moved to the appropriate suit stack. The same technique is used to turn cards over in the suit stacks, and to deal the next set of cards by the dealer. To deal, then click on the down−turned card stack, if there are no further cards at the dealer then click on the empty position and the dealer will turn over the dealer stack and deal from the top again.

Note that once a card is played onto the suit stacks then it cannot be removed.

To the right of the board are a number of control buttons. To select an option, click the left mouse button on it, the buttons are labeled:

DEAL

Start a new game by dealing new cards.

QUIT

Exit the game

HELP

This help page

Note that the screen may be updated at any time using "*C−l*".

NOTES

Patience is a macro defined in `patience.emf`.

The game is best played with a mouse, it is possible to play with the keyboard, as follows:−

"*esc h*" for help

To move a card between stacks enter the source and destination column number ("*1*","*2*",.."*7*"). To move from the dealer pile then the source is the "*space*" key.

"*tab*" deals the next cards.

To overturn a card on the row stacks then enter the card column twice i.e. source and destination are the same.

To move a card from the row to the suit stacks then either enter the card column twice, or enter the destination as "*h*","*d*","*c*","*s*" (i.e. "*2 2*" or "*2 s*" to move the card in column 2 to the spades stack).

"*C−c C−c*" to deal the cards again.

"*C−l*" redraw the screen.

"*q*" to quit the game.

SEE ALSO

Games, Triangle(3), Mahjongg(3).

paragraph−to−line(3)

NAME

paragraph−to−line − Convert a paragraph to a single line

SYNOPSIS

n **paragraph−to−line**

DESCRIPTION

paragraph−to−line is a variation of fill−paragraph(2). **paragraph−to−line** reduces each of the next *n* paragraphs of text to single lines. This command is typically used to prepare text for import into a word processor such as **Microsoft Word** or **Word Perfect**. Reduction of text to a single line allows the word processor to import the raw text file and keep the text within paragraph blocks. If the text is not prepared then all of the line−feeds have to be manually deleted.

paragraph−to−line allows text based documents to be prepared in MicroEmacs '02 and imported into the word processor at the final stage for formatting and layout.

NOTES

paragraph−to−line is a macro defined in `format.emf`.

SEE ALSO

fill−paragraph(2).

pipe−shell−command(2)

NAME

pipe−shell−command − Execute a single operating system command
$ME_PIPE_STDERR − Command line diversion to stderr symbol

SYNOPSIS

n **pipe−shell−command** "*command*" ["*buffer−name*"] (**esc @**)

[MS−DOS and Win32s Only]
$ME_PIPE_STDERR "*string*"; Default is undefined.

DESCRIPTION

pipe−shell−command executes one operating system command *command* and pipes the resulting output into a buffer with the name of ***command***.

The argument *n* can be used to change the default behavior of pipe−shell−command described above, *n* is a bit based flag where:−

0x01

Enables the use of the default buffer name ***command*** (default). If this bit is clear the user must supply a buffer name. This enables another command to be started without effecting any other command buffer.

0x02

Hides the output buffer, default action pops up a window and displays the output buffer in the new window.

0x04

Disable the use of the command−line processor to launch the program (win32 versions only). By default the "**command**" is launched by executing the command:

```
%COMSPEC% /c command
```

Where `%COMSPEC%` is typically command.com. If this bit is set, the "**command**" is launched directly.

0x08

Detach the launched process from MicroEmacs (win32 versions only). By default the command is launched as a child process of MicroEmacs with a new console. With this bit set the process is completely detached from MicroEmacs instead.

0x10

Disable the command name mangling (win32 versions only). By default any '`/`' characters found in the command name (the first argument only) are converted to '`\`' characters to make it Windows compliant.

NOTES

On MS−DOS and *Win32s* the standard shell **command.com(1)** does not support the piping of *stderr* to a file. Other shells, such as **4Dos.com(1)**, do, using the command−line argument "`>&`". If the environment variable "`ME_PIPE_STDERR`" is defined (the value is not used) then MicroEmacs assumes that the current shell supports piping of stderr.

SEE ALSO

ipipe−shell−command(2), shell−command(2).

popup−window(2)

NAME

popup−window − Pop−up a window on the screen

SYNOPSIS

n **popup−window** "*name*"

DESCRIPTION

popup−window manages the display of a new window on the screen. If only one window exists then it will be split else the current window will changed to one of the other existing visible windows. If the given buffer name "*name*" is not null ("") then the buffer is created, if it does not exist, and swapped in.

If an argument *n* of zero is given then the command only succeeds if the given buffer is already being displayed in an existing window, this window is made current. If an non−zero argument is given to the command and the given buffer is not visible then a window displaying a system buffer is chosen in preference. A system buffer is one who's name starts with a `'*'` character, e.g. `"*help*"`. window used to display

SEE ALSO

find−buffer(2).

prefix(2)

NAME

prefix − Key prefix command
prefix2 − Control(2) prefix
prefix3 − Control(3) prefix
prefix4 − Control(4) prefix

SYNOPSIS

n **prefix**

Default prefix bindings:

prefix 1 (**esc**)
prefix 2 (**C−x**)
prefix 3 (**C−h**)
prefix 4 (**C−c**)

DESCRIPTION

prefix sets up to 8 prefix key sequences, allowing two stoke key bindings. The command does not do anything, it is used to create double barrel key bindings such as such as goto−line(2) (**esc g**). This binding may be redefined, redefining ALL meta bindings. If the meta bindings are not required the command should first be unbound using the global−unbind−key(2).

The prefix key can only be defined using the global−bind−key(2), passing the command the prefix number required, for example:

```
1 global-bind-key "prefix" "esc "
2 global-bind-key "prefix" "C-x"
```

Binds the first prefix to the Escape key and the second prefix to Control−x.

The first prefix key (**prefix 1**) differs from the other prefixes since it permits entry of the numeric argument at the message line, e.g. "`esc 1 0 C-f`" will move forward 10 characters.

NOTES

Invocating this command via execute−named−command(2) or by a macro has no effect. It can be bound to only one key sequence which must be a single key stroke such as **C−x** etc. Re−binding the command to another key will not only unbind the new key but also the current **prefix ?** key bindings.

SEE ALSO

global−bind−key(2), global−unbind−key(2).

print−buffer(2)

NAME

print−buffer − Print buffer, with formatting
print−region − Print region, with formatting

SYNOPSIS

n **print−buffer**
n **print−region**

DESCRIPTION

print−buffer and **print−region** print the current buffer or region, respectively, using high−lighting where appropriate. The hilighting assigned to a buffer is defined by the variable $buffer−hilight(5) the print scheme is defined with print−scheme(2), the scheme−editor(3) should be used to create printer schemes.

The printing is typically configured using print−setup(3), which can be found in the main menu under **File−>Printer Setup**.

The numerical argument *n* is generally used for macro development, it changes the default behaviour of these commands as follows:

−2

Configures the printer and, on win32 platforms, opens a Windows printing dialog box enabling the user to configure the printer, font and page layout. The configuration is stored in the "`/print`" registry.

−1

Configures the printer, the configuration is stored in the "`/print`" registry.

0

Configures the printer and, on win32 platforms, using the Windows printer, opens a Windows printing dialog box enabling the user to configure the printer, font and page layout. The required printing is then performed.

1

Configures the printer and performs the required printing. **Printing Process**

When either of these commands are executed the macro file `print.emf` is executed to configure the printer (in a same vain as `me.emf` is executed to configure MicroEmacs for general usage). After the macro file has been executed the "`/print`" registry must contain the information required for printing. Following is a list of registry entries and their use:

flags (*integer*)

The setup flags, defined as a bit mask as follows:−

`0x0f` − Destination of the printer output.

`0x00` − Buffer only.
`0x01` − Internal queue.
`0x02` − To file only.
`0x03` − To file and command line.
`0x10` − Bit set, header enabled.
`0x20` − Bit set, footer
`0x40` − Bit set, enable line numbers.
`0x80` − Bit set, Enable truncated line character (typically \).

paper−x (*integer*)

Paper page width in character cells.

paper−y (*integer*)

Paper page depth in character cells.

page−x (*integer*)

The logical page width in character cells.

page−y (*integer*)

The logical paper depth in character cells.

specifier−x (*integer*)

Windows only.

specifier−y (*integer*)

Windows only.

font−face (*string*)

The name of the font face (Windows only).

rows (*integer*)

Number rows per output page.

cols (*integer*)

Number of columns per output page.

mtop (*integer*)

The size of the top margin in character cells (i.e. where printing may commence).

mbottom (*integer*)

The size of the bottom margin in character cells (i.e. where printing stops).

mleft (*integer*)

The number of characters of space forming the left magin of the physical page.

mright (*integer*)

The number of characters of space forming the right magin of the physical page.

header (*string*)

The ASCII text string for the header line.

footer (*string*)

The ASCII text string for the footer line.

port (*string*)

Printer port identity.

buffer (*string*)

The name of the destination buffer.

file (*string*)

The name of the destination file.

strip (*integer*)

If *integer* value strip spaces from eol.

device (*string*)

The ASCII name of the device (i.e. `/dev/lp`).

eof (*string*)

The printer codes for the end of the file, may be the empty string if not reqired.

eol (*string*)

The printer codes for the end of line character.

eop (*string*)

The printer codes for the end of a page.

sof (*string*)

The printer codes for the start of a file, may be the empty string if not required.

sol (*string*)

The printer codes for the start of a line.

sop (*string*)

The printer codes for the start of a page.

scont (*string*)

The printer codes for a start of row continuation.

econt (*string*)

The printer codes for the end of row continuation.

hsep (*string*)

The horizonal logical page separator character.

vsep (*string*)

The vertical logical page separator character.

wsep (*string*)

The depth in character cells of the vertical logical page separator.

xsep (*string*)

The width in character cells of the logical horizontal separator.

bg−color (*integer*)

The background colour number.

command−line (*string*)

The command line to perform a print operation. **Printing Under Microsoft Windows Environments**

Printing under Microsoft Windows Environments automatically invokes a dialog box to assign and configure the printer page characteristics. The dialog box allows the printer to be selected, enables line numbering, headers and footers.

The dialog allows the user to select the font size, by defining the number of characters that appear on a logical page, and the number of logical pages that appear on a physical page. Selecting the logical and physical page characteristics determine the size of the font. For dense pages with a small typeface then a point size of 6 is appropriate. For clarity, a larger typeface of 10 or 12 points is advised.

NOTES

The last printer configuration selected by the user is held in the registry file "`print.erf`" which is loaded into the */print−history* registry section. This feature is implemented in the macro file `print.emf`.

BUGS

Landscape printing under Microsoft Windows environments is temperamental.

Font selection under Microsoft Windows environments does not always determine the most appropriate font size.

The printer interface does not support native postscript generation. (In progress).

SEE ALSO

print−setup(3), scheme−editor(3), print−scheme(2), hilight(2), printall(3f), $buffer−hilight(5).

print−color(2)

NAME

print−color − Create a new printer color
print−scheme − Create a new printer color and font scheme

SYNOPSIS

n **print−color** "*col−no*" "*red*" "*green*" "*blue*"
n **print−scheme** "*schemeNum*" "*fore*" "*back*" "*font−mask*"

DESCRIPTION

print−color and **print−scheme** are similar to add−color(2) and add−color−scheme(2) except they configure MicroEmacs's printer scheme.

print−color creates a new printer color and inserts it into the printer color table, where *red*, *green* and *blue* are the color components and *col−no* is the printer color index. The printer color table contains 256 entries indexed by *col−no* in the range 0−255. **print−color** may also be used to modify an existing *col−no* index by re−assignment, the existing color definition is over−written with the new color definition.

An argument *n* of 0 to **print−color** resets the printer color table, removing all currently defined colors.

print−scheme creates a new printer scheme. A printer scheme maps the hilight(2) buffer's text into a print scheme. For example key words could be printed in *bold* or in *blue* etc. **print−scheme** arguments comprise an identifying index number "*schemeNum*", two color values, "*fore*" and "*back*" (defined by **print−color**) and a font setting "*font−mask*". The *font−mask* is a bit mask where each bit is defined as follows:

`0x01`	Enable bold font.
`0x02`	Enable italic font.
`0x04`	Enable light font.
`0x08`	Enable reverse font.
`0x10`	Enable underlining.

An argument *n* of 0 to **print−scheme** resets the printer scheme table, removing all currently defined printer schemes.

NOTES

Printer schemes may be created and altered using the scheme−editor(3) dialog, the created printer scheme may then be used directly in the print−setup(3) dialog. Therefore direct use of these commands is largely redundant.

SEE ALSO

scheme−editor(3), print−setup(3), print−buffer(2), hilight(2), $buffer−hilight(5).

print−setup(3)

NAME

print−setup − Configure MicroEmacs's printer interface

SYNOPSIS

print−setup

DESCRIPTION

print−setup provides a dialog interface for configuring MicroEmacs's printing interface. **print−setup** may be invoked from the main *File* menu or directly from the command line using execute−named−command(2).

The **print−setup** dialog is broken down into three pages of configuration options, on all pages the following buttons are available at the bottom of the dialog:−

`Print`

Prints the current buffer using the current configuration.

`Exit`

Quits **print−setup**, changes made to the configuration will be saved.

The following pages appear in the dialog:−

Printer

The **Printer** page is used to configure the type, style and location of the printer, the items on this page are defined as follows:−

`Driver`

Sets the printer type to be used, selecting this item creates a drop down list of available printer drivers. The drivers inform MicroEmacs which fonts and colors are available and how to enable/disable them, these are usually special character sequences. The following special drivers are defined:−

Default Plain Text

This driver does not use any special character sequences so the output it produces is plain text. This should work with most printers, but it does not support any colors or fonts.

HTML

This is a virtual printer driver as no printer uses HTML directly. However the files produced by this driver can be loaded by a web−browser and rendered with full color and font support so provides an efficient way of testing printer schemes. In addition may be used to convert the text rendered in MicroEmacs into HTML content.

Windows

This utilizes MicroEmacs's built−in Windows printer interface (Windows platforms only). When selected MicroEmacs communicates directly to the MS Printer Manager.

`Print Scheme`

Sets the color and font scheme to be used, selecting this item creates a drop down list of available printer schemes − choose one appropriated for your printer. The Default Plain Text scheme does not use any color or fonts so should work for all drivers. see the next item for scheme creation and editing.

`Edit`

Opens the scheme−editor(3) dialog box to edit the currently selected printer scheme, the editor may also be used to create and install new printer schemes.

`Destination`

Specifies the resultant print output, when selected a drop down menu appears containing the following items:

To buffer only

Creates a "`*printer*`" buffer and prints to the buffer.

To file only

Creates a new temporary file and prints to it.

To file & print

Prints to a temporary file and then executes the command−line (see next item) to print the resultant file (option not available when using the Windows printer driver).

Direct to printer

Output is sent directly to the printer, option only available when using the Windows driver.

`Command−line`

Sets the command−line required to print a generated print file (option not available when the Windows driver is selected as printing is done by talking to MS Print Manager directly). The command−line should be a single shell command using "`%f`" whenever the name of the file to be printed is required, e.g. on UNIX systems **lp(1)** or **lpr(1)** can usually be used as follows:−

```
lp −s %f
```

On MS−DOS machines this can usually be achieved by copying the file to the `PRN` device, as follows:

```
copy %f PRN
```

`Page Size`

Displays the currently configured page size in the form:

*Columns***x***Rows Chars−Wide***x***Chars−High*

the field cannot be edited directly, the settings **Page Setup** affect these values.

Page Setup

`Paper Size`

Sets the size of the printer paper, selecting this item will produce a pop down menu listing all available paper sizes unless the Windows printer driver is being used in which case this field cannot be selected and the **Edit** button must be used.

`Character Size`

Sets the size of a character within the page, expressed in terms of the number of characters which will fit on the paper (*width***x***height*). When selected a drop down menu lists all available sizes for the current paper size unless the Windows driver is selected in which case this field cannot be selected and the **Edit** button must be used.

`Edit` (Windows only)

Opens a Windows printer dialog box allowing the user to specify the windows printer, paper size and character size etc.

`No. of Columns` and `Rows`

Sets the number of sub−columns and rows to divide the page into, creating pages within a page.

`Line Numbers`

When enabled, prints the line number at the left hand edge for each line.

`Split Line ID`

When enabled the last right hand text column is reserved for a split identifier. Whenever a line is too long to fit on a single line it is split over two or more lines, if this option is enabled the right edge will be set to the split character (usually a '\' char) to clearly indicate that the line is split.

`Page Size`

As with the **Printer Page Size** it displays the current page size, the field cannot be edited. **Layout**

`Margins`

Configures the top, bottom, left and right margins in characters.

`Header`

Sets whether a header should be printed and if so what it should be, the following special strings can be used:

`%%`

Print a '`%`' character.

`%b`

Print the current buffer's name.

`%D`

Print the current day of the month.

`%f`

Print the current buffer's file name.

`%h`

Print the current hour.

`%M`

Print the current month of the year.

`%m`

Print the current minute of the hour.

`%p`

Print the current page number.

`%s`

Print the current seconds.

`%Y`

Print the current year as a 2 digit number.

`%y`

Print the current year as a 4 digit number.

`Footer`

Sets whether a footer should be printed and if so what it should be, the same special strings can be used as for the header. **NOTES**

user−setup is a macro using osd(2), defined in `printstp.emf`.

The list of available printer drivers and print schemes is stored in the macro file `printers.emf`. Using the **Install** option of the scheme−editor(3) automatically adds the new scheme to the print schemes list. To create a new printer driver a new configuration registry file (`erf` file − see `print*.erf` for examples) must be created and added to the printer driver lists within `printer.emf`.

SEE ALSO

print−buffer(2), scheme−editor(3), osd(2).

query−replace−all−string(3)

NAME

query−replace−all−string − Query replace string in a list of files

SYNOPSIS

n **query−replace−all−string** "*from*" "*to*" "*files*" ["*grep−from*"]

DESCRIPTION

query−replace−all−string, similar to query−replace−string(2), replaces all occurrences of "*from*" to "*to*" in the given list of files prompting the user before replacing each occurrence.

The command finds all occurrences of "*from*" by calling the command grep(3) to search for string "*from*" in files "*files*". Thus all relevant edited files must be saved or **grep** may return the wrong line numbers. This is achieved by a call to save−some−buffers(2) which prompts the user to save any changed buffers one at a time.

Each occurrence of "*from*" is jumped to using get−next−line(2) and the string is replaced by the call:

```
-1 query-replace-string "from" "to"
```

This query−replaces all occurrences of "*from*" to "*to*" on the current line only, hence the line numbers must be correct. This also means that the "*from*" search string must be correctly formatted for both grep and query−replace−string, unless bit 0x02 is set (see below).

The given argument *n* is a bit based flag which changes the default behavior described above. The bits have the following effect:−

0x01

Prompt before saving any changed buffer, enabled by default. If this bit is not set then any changed buffer is automatically saved before the **grep** is performed.

0x02

If set then a fourth argument "*grep−from*" must also be given. This string is used in place of the "*from*" string for the **grep** only. **NOTES**

query−replace−all−string is a macro defined in `search.emf`.

The **grep** command must be working before this command can function properly.

It is not recommended to use a "from" or "to" string which uses more that one line as the results may be unpredictable.

As the change is likely to be over several files a single call to undo(2) at the end of execution will not undo all the changes made. To undo all the changes made, use get−next−line(2) to loop through all the occurrences and call **undo** for each occurrence

SEE ALSO

query−replace−string(2), save−some−buffers(2), grep(3), get−next−line(2), undo(2), replace−all−string(3), search−forward(2).
Regular Expressions

query−replace−string(2)

NAME

query−replace−string − Search and replace a string − with query

SYNOPSIS

query−replace−string (**esc C−r**)

DESCRIPTION

query−replace−string operates like the replace−string(2) command. replacing one string with another. However, it allows you to step through each string and ask you if you wish to make the replacement. The user is prompted for a replacement response as follows:−

Y

Make the replacement and continue on to the next string.

N

Do not make the replacement, and continue.

!

Replace the rest of the strings without asking.

^**G**

Stop the command.

.

Go back to place the command started

u

Undo last replacement.

l

Last replacement, do next and stop.

?

Help − get a list of options. **SEE ALSO**

Refer to search−forward(2) for a description of the magic mode search characters.

replace−string(2).
Regular Expressions

quick−exit(2)

NAME

quick−exit − Exit the editor writing changes
save−buffers−exit−emacs − Exit the editor prompt user to write changes

SYNOPSIS

quick−exit (**esc z**)
save−buffers−exit−emacs (**C−x C−c**)

DESCRIPTION

quick−exit writes out all changed buffers to the files they were read from, saves all changed dictionaries, killing any running commands and exits the editor.

save−buffers−exit−emacs operates a **quick−exit** only prompts the user before saving any files.

NOTES

All buffers with a name starting with a '*****' are assumed to be system buffer (i.e. ***scratch***) and are not saved.

SEE ALSO

exit−emacs(2), save−buffer(2).

quote−char(2)

NAME

quote−char − Insert literal character

SYNOPSIS

n **quote−char** "*key*" (**C−q**)

DESCRIPTION

quote−char inserts the next typed character *n* times, default is 1, ignoring the fact that it may be a command character. **quote−char** obeys the current buffer setting of over(2m) mode.

SEE ALSO

insert−string(2), Symbol(3).

rcs−file(2)

NAME

rcs−file − Handle Revision Control System (RCS) files

SYNOPSIS

n **rcs−file** (**C−x C−q**)

DESCRIPTION

MicroEmacs '02 RCS support command. The action of this command depends on the current buffer view(2m) mode state, the argument *n*, and the existence of an RCS file.

view−mode ON; RCS file does not exist

Removes buffer view mode to enable the user to edit the file.

view−mode ON; RCS file exists

MicroEmacs attempts to check out the file using the command line given by the variable $rcs−cou−com(5) (co unlock). The file is then reloaded and the view mode status re−evaluated.

view−mode OFF; RCS file does not exist

MicroEmacs attempts to check−in the file into RCS for the first time using the command−line given by the variable $rcs−cif−com(5) (ci first). The file is then reload.

view−mode OFF; RCS file exists

MicroEmacs attempts to check−in the file into RCS using the command−line given by the variable $rcs−ci−com(5). The file is then reload.

−ve argument given

MicroEmacs attempts to unedit any changes made to the file using the command−line given by the variable $rcs−ue−com(5). The file is then reload. **SEE ALSO**

rcs(1). $rcs−file(5), buffer−mode(2), find−file(2), view(2m).

read−file(2)

NAME

read−file − Find and load file replacing current buffer

SYNOPSIS

n **read−file** "*file−name*" (**C−x C−r**)

DESCRIPTION

read−file operates like find−file(2), this command either finds the file in a buffer, or creates a new buffer and reads the file in. The command destroys the current buffer before the new buffer is created making this command ideal to use when the wrong file was entered on a find−file(2). This command is also useful for re−loading files that have changed on disk.

The numeric argument *n* can be used to modify the default behaviour of the command, where the bits are defined as follows:

0x01

If the file does not exist and this bit is not set the command fails at this point. If the file does not exist and this bit is set (or no argument is specified as the default argument is `1`) then a new empty buffer is created with the given file name, saving the buffer subsequently creates a new file.

0x02

If this bit is set the file will be loaded with binary(2m) mode enabled. See help on **binary** mode for more information on editing binary data files.

0x04

If this bit is set the file will be loaded with crypt(2m) mode enabled. See help on **crypt** mode for more information on editing encrypted files.

0x08

If this bit is set the file will be loaded with rbin(2m) mode enabled. See help on **rbin** mode for more information on efficient editing of binary data files. **SEE ALSO**

reread−file(3), find−file(2), view−file(2), binary(2m), crypt(2m), rbin(2m).

read−history(2)

NAME

read−history − Read in session history information

SYNOPSIS

n **read−history** ["*hist−file*"]

DESCRIPTION

read−history reads in a MicroEmacs '02 history file, setting the current history information. If argument **n** is not given then the given "*hist−file*" is simply read in. If a non−zero argument is specified then default history is set to the given file−name and the file is read. If an argument of zero is given then the default history is re−read. Information read in (and saved) from the history file includes:−

- Searching and replacing history.
- Buffer name history.
- Command name history.
- File name history.
- General (all the rest) history.
- Buffer and file list with line numbers.

MicroEmacs '02's environment may be retained almost intact by the use of the default history and using the **−c** (continue) command−line option to re−load all files that were being edited in a previous session.

NOTES

When running multiple MicroEmacs '02 sessions on the same work−station (or different workstations sharing the same home directory), the default history is saved when MicroEmacs '02 exits. As a result the last MicroEmacs '02 sessions that terminates writes the history information used next time.

The history information is saved in a registry format file (see erf(8)). Reference should be made to the notes included in erf(8) as to how the history file may be edited and effected in the same MicroEmacs '02 session.

SEE ALSO

erf(8), save−history(2).

read−registry(2)

NAME

read−registry − Read in a registry definition file

SYNOPSIS

read−registry "*root*" "*file*" "*mode*"

DESCRIPTION

read−registry loads a registry file erf(8) into the internal registry memory, where the information may be queried via the registry macro commands. The arguments are defined as follows:−

root

The root node in the registry to into which the registry contents are attached. The root name is limited to 32 characters in length and is specified without a leading forward slash '`/`'. The node *root* is created at the root of the registry.

file

The name of the registry file erf(8) to load. This may be an absolute, relative or `$MEPATH` specified file; typically it is located on `$MEPATH`.

mode

The *mode* is string specifying the registry node loading and saving modes, each mode is represented by a character. Lower case characters add a mode, upper case characters delete a mode. The modes are defined as follows:−

a − Autosave

Automatically saves the registry when it is deleted or unloaded from the registry. The user is not prompted for a save.

b − Backup

Automatically performs a backup of the registry file whenever a save operation is performed.

c − Create

If the registry file cannot be loaded then the *root* node is created and the invocation succeeds. If this mode is omitted then the call fails if the *file* cannot be found.

d − Discard

Marks the registry as discardable. This is typically used for registries that are not saved.

r − Reload

If the registry node already exists then it is deleted and reloaded, see also the merge flag (**m**). By default, when both the **r** and **m** flags are omitted and the registry node already exists the read operation is not performed and the existing node is used.

m − Merge

The registry file is merged with the contents of any existing registry node. (i.e. the existing registry tree nodes are not deleted if they already exist). See also the reload flag (**r**).

h − Hidden

The registry node is created in the *Hidden* state. (i.e. children will not be shown in list−registry(2) output).

u − Updated

Marks the registry as modified. The modified bit is removed when the registry file is saved. If the modified bit is applied to a registry node the user will be prompted to save the registry when it is deleted (or it will be automatically saved when the *Autosave* mode is used).

Multiple modes may be applied.

EXAMPLE

The following example is a typical call made from a macro using a registry file where the user may edit the registry file. In this case this a reload of the registry is forced to ensure that the most up−to−date contents are retrieved. Note that the name of the registry file is actually retrieved from the *history* registry.

```
set−variable #l1 &reg "/history" "address" $MENAME
!if &seq &set #l0 &find #l1 ".ab" "ERROR"
    set−variable #l0 &reg "/" "history" ""
    set−variable #l0 &spr "%s%s.ab" &lef #l0 &rsin "/" #l0 #l1
!endif
read−registry "AddressBook" #l0 "rc"
```

BUGS

At exit only registry nodes attached to the root are saved.

SEE ALSO

save−registry(2), list−registry(2), mark−registry(2), erf(8).

recenter(2)

NAME

recenter − Recenter the window (refresh the screen)

SYNOPSIS

n **recenter** (**C−l**)

DESCRIPTION

recenter scrolls the current window so that the cursor position is at the center of the window and redraws the whole screen. If *n* is given then scrolls the window so that the cursor is *n* lines from the top if *n* is positive or from the bottom if negative.

recenter is typically used to refresh the screen if it is out of date (i.e. needs to be redrawn).

SEE ALSO

screen−update(2).

regex−forward(3)

NAME

regex−forward − Search for a magic string in the forward direction
regex−backward − Search for a magic string in the backward direction

SYNOPSIS

n **regex−forward** "*string*"
n **regex−backward** "*string*"

DESCRIPTION

regex−forward searches for a regular expression string from the current cursor position to the end of the file. A case insensitive regular expression search is performed regardless of the magic(2m) and exact(2m) mode settings.

The numeric argument *n* is interpreted as follows:−

n > **0**

The *n*th occurrence of the *string* is located.

n < **0**

The first occurrence of the *string* is located in the next *n* lines.

regex−backward searches backwards in the file. In all other ways it is like **regex−forward**.

DIAGNOSTICS

The command returns a status of `FALSE` if the *string* could not be located (or *n*th *string* where *n* occurrences are requested). If the *string* is found within the given search criteria the return status is `TRUE`.

NOTES

The **regex−forward** and **regex−backward** commands are not publically available from the command line, but may be used within macros to perform regular expression searches regardless of the user mode settings.

These commands are implemented as macros in `utils.emf`.

SEE ALSO

buffer−mode(2), exact(2m), isearch−forward(2), magic(2m), replace−string(2), search−backward(2), search−forward(2).
Regular Expressions

replace−all−pairs(3)

NAME

replace−all−pairs − Replace string pairs in a list of files

SYNOPSIS

n **replace−all−pairs** "*files*"

DESCRIPTION

replace−all−pairs uses the current buffer to extract "*from*" and "*to*" pairs and then replaces all occurrences of "*from*" to "*to*" in the given list of files without prompting the user. An optional third argument "*grep*" can be given which will be used as the grep string, if not given the "*from*" string is used. The format of the current buffer must be:

```
/from1/to1/
Xfrom2Xto2X
?from3?to3?
/from4/to4/grep4/
    .
    .
/fromN/toN/
```

For each pair the command finds all occurrences of "*from*" (or "*grep*" if specified) by calling the command grep(3) to search for string "*from*" in files "*files*". Thus all relevant edited files must be saved or **grep** may return the wrong line numbers. This is achieved by a call to save−some−buffers(2) between each replace pair, it is called with an argument of 0 to ensure that any changed buffers are automatically saved.

Each occurrence of "*from*" is jumped to using get−next−line(2) and the string is replaced by the call:

```
−1 replace−string "from" "to"
```

This replaces all occurrences of "*from*" to "*to*" on the current line only, hence the line numbers must be correct. This also means that the "*from*" search string must be correctly formatted for both grep and replace−string.

The given argument *n* is a bit based flag which changes the default behavior described above. The bits have the following effect:−

0x01

Prompt before saving any changed buffers FIRST time ONLY, enabled by default. If set then the user is also prompted to continue before any changes are made. If this bit is not set then the command executes without any user input. **NOTES**

replace−all−pairs is a macro defined in `search.emf`.

The **grep** command must be working before this command can function properly.

It is not recommended to use a "from" or "to" string which uses more that one line as the results may be unpredictable.

As the change is likely to be several pair strings with each changed buffer being saved between pairs undo(2) cannot be used to undo the changes. Neither can the backups be relied on as a buffer may be saved more than once in this process, therefore it is strongly recommend that a backup of the files is made before commencing with this command.

SEE ALSO

replace−all−string(3), replace−string(2), save−some−buffers(2), grep(3), get−next−line(2), undo(2), query−replace−all−string(3), search−forward(2).
Regular Expressions

replace−all−string(3)

NAME

replace−all−string − Replace string with new string in a list of files

SYNOPSIS

n **replace−all−string** "*from*" "*to*" "*files*" ["*grep−from*"]

DESCRIPTION

replace−all−string, similar to replace−string(2), replaces all occurrences of "*from*" to "*to*" in the given list of files without prompting the user.

The command finds all occurrences of "*from*" by calling the command grep(3) to search for string "*from*" in files "*files*". Thus all relevant edited files must be saved or **grep** may return the wrong line numbers. This is achieved by a call to save−some−buffers(2) which prompts the user to save any changed buffers one at a time.

Each occurrence of "*from*" is jumped to using get−next−line(2) and the string is replaced by the call:

```
    −1 replace-string "from" "to"
```

This replaces all occurrences of "*from*" to "*to*" on the current line only, hence the line numbers must be correct. This also means that the "*from*" search string must be correctly formatted for both grep and replace−string, unless bit 0x02 is set (see below).

The given argument *n* is a bit based flag which changes the default behavior described above. The bits have the following effect:−

0x01

Prompt before saving any changed buffer, enabled by default. If this bit is not set then any changed buffer is automatically saved before the **grep** is performed.

0x02

If set then a fourth argument "*grep−from*" must also be given. This string is used in place of the "*from*" string for the **grep** only. **NOTES**

replace−all−string is a macro defined in `search.emf`.

The **grep** command must be working before this command can function properly.

It is not recommended to use a "from" or "to" string which uses more that one line as the results may be unpredictable.

As the change is likely to be over several files a single call to undo(2) at the end of execution will not undo all the changes made. To undo all the changes made, use get−next−line(2) to loop through all the occurrences and call **undo** for each occurrence

SEE ALSO

replace−string(2), save−some−buffers(2), grep(3), get−next−line(2), undo(2), query−replace−all−string(3), replace−all−pairs(3), search−forward(2).

replace−string(2)

NAME

replace−string − Replace string with new string

SYNOPSIS

n **replace−string** (**esc r**)

DESCRIPTION

replace−string replaces all occurrences of one string with another string. The replacement starts at the current location of the cursor and goes to the end of the current buffer.

A numeric argument positive *n* limits the number of strings replaced to *n*. A negative argument *n* limits the number of lines in which the replacement may take place, e.g. a value of −15 restricts the replacement of the string to the next 15 lines from the current cursor position.

SEE ALSO

See Operating Modes for a description of the magic(2m) and exact(2m) modes which change the search space.

buffer−mode(2), query−replace−string(2), search−forward(2).
Regular Expressions

reread−file(3)

NAME

reread−file − Reload the current buffer's file

SYNOPSIS

reread−file

DESCRIPTION

reread−file reloads from disk the file associated with the current buffer, this command is particularly useful when the file is continually updated by an external program. If the buffer has been edited and its name does not start with a `'*'` then the user is prompted as to whether the changes should be discarded. Also if the buffer has an active process running in it then confirmation is sort from the user before the process is killed.

NOTES

reread−file is a macro implemented in `tool.emf`.

SEE ALSO

find−file(2), read−file(2), view−file(2).

resize−all−windows(2)

NAME

resize−all−windows − Automatically resize the windows

SYNOPSIS

n **resize−all−windows**

DESCRIPTION

resize−all−windows performs an automatic layout of the windows on the screen, reorganizing the windows such that each window has an equal amount of space. The argument *n* determines which axes reorganization is performed in.

- A +ve argument reorganizes the windows vertically, leaving the horizontal arrangement as is.
- A −ve argument rearranges the windows horizontally, leaving the vertical arrangement as is.
- An argument of zero performs no vertical or horizontal arrangement.
- No argument re−arranges both the vertical and horizontal window layout.

SEE ALSO

resize−window−vertically(2), resize−window−horizontally(2), split−window−vertically(2).

restyle−buffer(3)

NAME

restyle−buffer − Automatically reformat a buffer's indentation.
restyle−region − Automatically reformat a regions indentation.

SYNOPSIS

restyle−buffer
restyle−region

DESCRIPTION

restyle−buffer automatically re−formats the indentation of a buffer. The indentation only operates if the indentation method is defined with cmode(2m) or $buffer−indent(5), otherwise the command has no effect.

restyle−region modifies the indentation between *point* and *mark*.

NOTES

restyle−buffer and **restyle−region** are macros defined in `format.emf`.

SEE ALSO

cmode(2m), indent(2), $buffer−indent(5).

reyank(2)

NAME

reyank − Restore next yank buffer

SYNOPSIS

n **reyank** (**esc y**)

DESCRIPTION

Every region killed goes onto a stack, with the most recent at the top. Immediately after yanking text out into the current buffer using yank(2), the user may **reyank** which deletes the region just yanked and replaces it with *n* insertions of the next region on the kill stack. Another call to reyank deletes that region and replaces it with the next in the stack etc.

The last 15 kills are stored.

SEE ALSO

copy−region(2), kill−region(2), set−mark(2), yank(2).

save−all(3)

NAME

save−all − Save all modified files (with query)

SYNOPSIS

n **save−all**

DESCRIPTION

save−all cycles through all buffers, dictionaries and registry files writing back any changes made. For each buffer, dictionary or registry file which has been modified the user is prompted before the changes are saved, a value of **y** initiates the save, **n** skips the save.

The argument *n* can be used to change the default behavior of save−all described above, *n* is a bit based flag where:−

0x01

Enables the user prompt before the file is saved (default). If this flag is not supplied then all modified files will automatically be written. **NOTES**

save−all is a macro defined in `me.emf`, using commands save−some−buffers(2), save−dictionary(2) and save−registry(2).

SEE ALSO

save−some−buffers(2), save−dictionary(2), save−registry(2).

save−buffer(2)

NAME

save−buffer − Save contents of changed buffer to file

SYNOPSIS

n **save−buffer** (**C−x C−s**)

DESCRIPTION

save−buffer saves the contents of the current buffer if the contents have been changed, writing the buffer back to the file it was read from.

On saving the file, if time(2m) mode is enabled then the time stamp string is searched for in the file and modified if located, to reflect the modification date and time.

If backup(2m) mode is enabled then a backup copy of the file existing is created and the contents of the buffer are written to the file. Any automatic save copies of the file are deleted.

If the buffer contains a narrow(2m) it will automatically be removed before saving so that the whole buffer is saved and restored when saving is complete

If auto(2m) mode is enabled the the file is written out in the style indicated by modes crlf(2m) and ctrlz(2m). Otherwise the file is written out in the style on the current platform.

The argument *n* can be used to change the default behavior of save−buffer described above, *n* is a bit based flag where:−

0x01

Enables validity checks (default). These include check that the buffer has been modified, if not an error occurs. Also the time stamp of the file to be written is checked, if the file systems file exists and is newer the confirmation of writing is requested from the user. If this flag is not supplied then the buffer is written whenever possible and without any prompts to the user.

0x02

Disables the expansion of any narrows (see narrow−buffer(2)) before saving the buffer. **NOTES**

- undo(2) information is discarded when the file is saved.
- Refer to $auto−time(5) for a description of the file extensions used by MicroEmacs '02 for backup and temporary files.

- Buffers may also be saved via the list−buffers(2) command.

SEE ALSO

$auto−time(5), $timestamp(5), buffer−mode(2), find−file(2), narrow−buffer(2), save−some−buffers(2), undo(2), backup(2m), time(2m), undo(2m), narrow(2m), auto(2m), crlf(2m), ctrlz(2m), write−buffer(2), append−buffer(2).

save−dictionary(2)

NAME

save−dictionary − Save changed spelling dictionaries

SYNOPSIS

n **save−dictionary** ["*dictionary*"]

DESCRIPTION

save−dictionary may be used to save one, or all changed, dictionaries back to disk. By default **save−dictionary** prompts for a single dictionary, which is then saved. If the dictionary to be saved has been created within the session (rather than read from disk) the user is always prompted to save and enter a full dictionary file name (pathname) to save to. If the dictionary was not created then the user is only prompted to save if,

- a non−zero argument is supplied
- and the users history registry node "*/history/spell/autosave*" does not exist or its value is zero.

Otherwise the dictionary is automatically saved.

The argument *n* may be used to control the effect of the command, *n* is a bit based flag defined as follows:−

0x01

Enables prompting before saving, only used when saving all dictionaries.

0x02

Save all changed dictionaries. **NOTES**

This command is called to save all dictionary changes whenever MicroEmacs is exited.

The dictionary auto−save registry value can be changed via the user−setup(3) dialog.

SEE ALSO

add−dictionary(2), delete−dictionary(2), spell(2).

save−history(2)

NAME

save−history − Write history information to history file

SYNOPSIS

n **save−history** "*hist−file*"

DESCRIPTION

save−history writes out MicroEmacs '02's current history information into the given history file.

The command read−history(2) can set a default history file in which case the history is automatically written out to this file if an argument of zero is given; the user is not prompted for a file. MicroEmacs '02 automatically tries to write the default history whenever it is exited.

NOTES

The history information is saved in a registry format file (see erf(8)). Reference should be made to the notes included in erf(8) as to how the history file may be edited and effected in the same MicroEmacs '02 session.

SEE ALSO

erf(8), read−history(2).

save−registry(2)

NAME

save−registry − Write a registry definition file

SYNOPSIS

n **save−registry** ["*root*" "*file*"]

DESCRIPTION

save−registry saves a registry tree, defined by *root*, to a registry file *file* in the erf(8) format. By default the user is prompted for the registry *root* to save, which must already exist. If the *file* given is the empty string **""**, the registry node *root* must be a root node with an associated file name stored, this file name is used.

The argument *n* may be used to control the effect of the command, *n* is a bit based flag defined as follows:−

0x01

Enables prompting before saving, only used when saving all registries.

0x02

Save all changed registries except the history node which should be saved using the command save−history(2). **NOTES**

This command is called to save all registry changes whenever MicroEmacs is exited.

SEE ALSO

read−registry(2), save−history(2), erf(8).

save−some−buffers(2)

NAME

save−some−buffers − Save contents of all changed buffers to file (with query)

SYNOPSIS

n **save−some−buffers**

DESCRIPTION

save−some−buffers cycles through all visible buffers (buffers without mode hide(2m) set) and attempts to save all modified ones, writing the contents back to the file from where it was read. For each buffer that has been modified the user is prompted to save the buffer, a value of **y** initiates a save for the buffer, **n** skips the buffer.

The argument *n* can be used to change the default behavior of save−some−buffers described above, *n* is a bit based flag where:−

0x01

Enables the user prompt before the buffer is saved (default). If this flag is not supplied then all modified visible buffers will be written. **SEE ALSO**

save−buffer(2), save−buffers−exit−emacs(2), write−buffer(2), hide(2m).

scheme−editor(3)

NAME

scheme−editor − Color Scheme Editor

SYNOPSIS

scheme−editor

DESCRIPTION

scheme−editor is a color and font scheme editor that provides a dialog interface to configure the display schemes used by the editor. The schemes may be created or modified within the scheme editor and then committed to the configuration files for general use.

The editor can be used to create both screen and printer color/font schemes, they are typically stored in the `macros` directory and are executed as macro files at start up or when printing. The standard screen schemes are called `scheme`*X*`.emf` and printer ones `print`*X*`.emf`.

The **scheme−editor** is displayed within a single dialog box, tab selections at the top of the dialog box enable **color** and **scheme** creation and/or modification. Navigation is typically performed using the mouse, where the mouse is absent then the `TAB` key may be used to move between the fields. The information presented is defined as follows:−

File Name

The name of the color scheme to be modified. This is the name of the **scheme*X*.emf** file, omitting the file extension. See the **FILES** section below for a list of standard screen and printer scheme supplied with MicroEmacs '02.

Type

Defines whether the scheme is a screen or printer type.

Description

An ASCII description of the color scheme, used to identify the color scheme.

Buffer Hilight

Available when scheme is a screen type. Defines whether buffer hilighting should be enabled, when *Completely Disable* all buffers are displayed character for character in the standard text scheme, this will ensure maximum update performance but some file formats such as the on−line help will become unreadable so this option is really selected. Similarly *Reformat Only* disables the majority of buffers, hilighting is only enabled when the file would be unreadable without it, such as the on−line help or man page files. The default *Fully Enabled* setting enables all buffer hilighting.

Print Option

Available when scheme is a printer type. Defines what components of a scheme is to be used when printing. **Colors**

The **colors** tab allows the basic palette colors of the editor to be created and modified. The left−hand side of the dialog contains a scrolling window containing the existing color entries. The right−hand side of the dialog provides the controls to add and change the color assignment. The controls operate on the currently selected palette entry.

Add

Creates and adds a new color entry into the palette. The new palette entry is created with a default color that may be subsequently modified.

Change

Commits the current selection color to the palette.

Red/**Green**/**Blue**

The color entries allow the currently selected palette color entry to be modified. The color values may be changed by direct numeric entry (0..255) or via the ^/v controls; the color is committed to the palette using the **Add** or **Change** button. **Schemes**

The **schemes** tab allows the schemes to be edited. The left−hand side of the dialog contains a scrolling window of the available color palette (created from the **Colors** tab). The right−hand side of the window shows the variants of the scheme.

Selection

The **selection** item provides a pull−down menu containing gross scheme categories used by the editor.

Scheme

A pull−down menu containing the schemes of the selection, modifying this entry shows the variants of the scheme in the **Normal**, **Current**, **Select** and **Sel−Cur** dialogs.

There are 4 variants, or styles, for a single scheme; each style is comprised of a foreground and background color, and a row of toggle button to enable/disable fonts, defined as follows.

`B` − Bold.
`I` − Italic.
`L` − Light (typically not supported).

R − Reverse video (fore/back−ground swapped).
U − Underline.
V − Toggle reverse video when inverted.

The last mode **V** needs a little more explanation; commands such as screen−poke(2) are able to invert the color scheme, i.e. use the fore color for the background etc. Enabling this mode will toggle the reverse video mode (**R**) when this feature is used.

The style displayed by a particular scheme depends upon the selection/current status of the text:

Normal

The normal style, when the text object is not selected or current (i.e. out of focus).

Current

The style used when the text object is current (i.e. in focus)

Select

The style used when the text object is selected (i.e. by the mouse) and is not current.

Sel−Cur

The style used when the text object is selected and is current.

Note that a printer scheme only uses the Normal style.

Setting of the **selection** and **scheme** shows the current scheme in the **Normal**, **Current**, **Select** and **Sel−Cur** dialogs. New colors are assigned by selecting a color in the palette area and making it current. The current color is applied by selecting the **Fore** / **Back** boxes of the scheme dialog. The assigned color is displayed in the text box *The big brown fox...*.

Controls

The controls at the bottom of the dialog apply the edits to the configuration files.

Current

Makes the changes to the palette and schemes current, they are applied to the current editing session but are not committed to file. This allows the palette changes to be used prior to commitment. Note that all modifications are lost if they are not saved and the editing session is terminated.

Save

Saves the scheme modifications to file, effectively making the changes permanent. Note however that the scheme macro file will be saved in the first directory in the $search−path(5), regardless of the location of the original. For network systems this typically means that the changes will only effect the current user.

Install

Installs the current color scheme into the configuration files, making the color scheme accessible to the user−setup(3) dialog.

Exit

Quits the scheme editor without modifying the settings. **FILES**

`scheme.emf` − Defines the standard scheme variables, including the available scheme list, and associated text.
`schemed.emf` − Default white on black color scheme.
`schemej.emf` − Black on cream color scheme.
`schemevi` − Sandy shores.
`schemesf` − Sherwood Forest.
`schemebh` − Blue Hue.
`schemepd` − Plain Black on Cream.
`schemepl` − Plain White on Black.
`schemel` − Black on grey.
`schememd` − Microsoft Developer Studio Colors.
`printers.emf` − Defines the list of available printer schemes and drivers.
`printd` − Default plain print−out.
`printf` − Print using fonts.
`printepc` − Print using Epson base colors and fonts.

NOTES

scheme−editor is a macro that is implemented in file `schemosd.emf`. The scheme editor uses osd(2) to create and manage the dialogs.

Only the Normal scheme style is used by printer schemes.

The setting of **Buffer Hilight** can effect the way buffer hooks are load so changing from one scheme to another with differing Buffer Hilight settings may not fully work. This can be rectified by restart MicroEmacs with the new scheme as default.

The current screen scheme can effect the printing due to the **Buffer Hilight** setting, e.g. if the screen scheme is set to completely disable hilighting then any print−out will also have no hilighting.

SEE ALSO

user−setup(3), add−color−scheme(2), print−scheme(2), osd(2).

screen−poke(2)

NAME

screen−poke − Immediate write string to the screen

SYNOPSIS

n **screen−poke** *row column colorScheme* "*string*"

DESCRIPTION

screen−poke writes a *string* to the screen at position (*row*, *column*) using the given color scheme. The screen coordinates are defined with (0,0) at the top left of the screen.

screen−poke by−passes the conventional buffer update and writes directly to the screen buffer. The command has no effect on buffers already showing on the screen and is erased on the next screen update. The *string* is clipped to the screen area hence the caller need not continually check on the size of the client area.

The numeric argument *n* is a bitwise flag which has the following meaning
`0x01` Don't mark the poke area for update.
`0x02` Don't flush poke to screen.
`0x04` colorScheme is an array of values, one for each letter.
`0xf0` colorScheme pair offset to use.

If the **0x01** flag is absent then the parts of the screen over written by **screen−poke** are marked and refreshed on the next **screen−update** operation, thereby erasing the poked information. If the flag is present the poked information remains on the screen until a forced refresh is performed (i.e. **1 screen−update**) or the window information under the poked screen data is modified.

In macros using many consecutive screen−pokes (e.g. Patience(3) to display a pack of cards) most pokes use the 'No flush' flag to improve performance and look on some platforms.

The use of **screen−poke** has largely been reduced to games such as Metris(3) since the introduction of osd(2) to create dialogs.

NOTES

Some platforms do not allow all character values to be poked, illegal characters are replaced with a '`.`'.

SEE ALSO

osd(2), screen−update(2), Mahjongg(3), Metris(3).

screen−update(2)

NAME

screen−update − Force screen update

SYNOPSIS

n **screen−update** (**redraw**)

DESCRIPTION

screen−update updates the current screen, usually used in macros. The argument *n* can be used to change the behaviour of this command as follows:

`−ve`

Disables the next −*n* screen updates, i.e. if *n* is `−1` then the next time the screen needs to be redrawn nothing will happen.

`0`

Resets the screen update disable count to zero, useful to remember when the the disable feature has been used incorrectly.

`1`

Full screen update (default), the screen is completely cleared and redrawn (as if garbled).

`2`

Partial screen update, only the parts of the screen which require updating are redrawn.

`3`

No screen redraw, only window variables are up−dated. This feature is provided for macros which manipulate the screen view and need to know where the cursor is in the window without redrawing the screen (which may cause unwanted flickering). Note that as the screen is not redrawn not all variables may have the correct value, for example the frame store variable @fs(4) could be out of date. **EXAMPLES**

The following macro demonstrates the problems encountered when trying to use screen variables in macros after the current position has changed. The first value printed is the starting cursor Y position and the next value should be one less than the first value due to the call to backward−line(2). But it is the same as the first because the screen (and its variables) have not been updated. The subsequent call to screen−update ensures that the third value is the correct one although by giving it an argument of 3 the screen is not visibly updated thus avoiding any annoying screen flicker:

```
define−macro test−screen−update
    set−variable #l0 $cursor−y
    backward−line
    set−variable #l1 $cursor−y
    3 screen−update
    set−variable #l2 $cursor−y
    forward−line
    ml−write &spr "%d %d %d" #l0 #l1 #l2
!emacro
```

NOTES

Every time the screen requires updating, MicroEmacs executes the *redraw* key, it is similar in mechanism to the user pressing *C−l* to refresh the screen. The user can therefore re−bind the *redraw* key to another command or macro, thereby allowing the user complete control of what is displayed. For example if *redraw* was bound to void(2) the screen would not be up−dated (**Note**: this is difficult to get out of and may require MicroEmacs to be killed).

This feature is often exploited by macros which take control of the input and output, such macros include gdiff(3), Metris(3), and Mahjongg(3).

SEE ALSO

recenter(2), screen−poke(2).

scroll−down(2)

NAME

scroll−down − Move the window down (scrolling)
scroll−up − Move the window up (scrolling)

SYNOPSIS

n **scroll−down** (**C−n**)
n **scroll−up** (**C−p**)

DESCRIPTION

scroll−down moves the window in the current buffer down by *n* lines, the default when *n* is omitted is 1 windows worth of lines i.e. a next page operation. A −ve value of *n* causes the window to move up.

scroll−up moves the window in the current buffer up by *n* lines, default when *n* is omitted is 1 windows worth of lines, i.e. a previous page operation. A −ve value of *n* causes the window to move down.

SEE ALSO

scroll−left(2), scroll−right(2), $window−y−scroll(5).

scroll−left(2)

NAME

scroll−left − Move the window left (scrolling)
scroll−right − Move the window right (scrolling)

SYNOPSIS

n **scroll−left** (**C−x** <)
n **scroll−right** (**C−x** >)

DESCRIPTION

scroll−left moves the window in current buffer left by 1 screen width. If an argument *n* is supplied then the resolution of movement is specified in characters relative to the current displacement. Moving the window in the current buffer left by *n* characters (that is if the current left−hand margin of the screen is column 0, the left hand margin becomes column *n*).

scroll−right moves the window in current buffer right by 1 screen width. If an argument *n* is supplied then the resolution of movement is specified in characters relative to the current displacement.

The ends of the lines of a scrolled screen are delimited with a dollar (**$**) character indicating that the text continues. When no scroll is in effect the left hand margin of the screen does not show the **$** symbol. i.e. The line `This text is scrolled on this line` with a current scroll offset of 2 in a 22 column window would appear as follows:

```
          22
|<-------------------->|

|$s text is scrolled $|
```

The amount of scroll (*n*) is effectively unlimited, it is possible to scroll all of the text in a buffer out of the window, when only **$**'s appear in the left margin, in the last highlighting color of the line (blank lines always remain blank and are not delimited with a **$**). Text on the current line is handled according to the value of $scroll(5) as follows:

$scroll 0

The current line ONLY is scrolled (about the current scroll position) to enable the current buffers cursor position to be viewed. To enable the user to determine where the current line is in relation to the scrolled lines then the first character of the current line is interpreted as follows:−

All of user text appears

```
|$f line of te$|
|At start of l$|
|$f line of te$|
```

Surrounding lines commence with "$" indicates at the start of the line.

$ in column 0

```
|$f line of te$|
|$f line of te$|
|$f line of te$|
```

Text column is the same as the surrounding text i.e. the line and window scroll are the same.

> Left of scroll position

```
|$f line of te$|
|>f line of te$|
|$f line of te$|
```

The current line is to the left of the scrolled position. forward−char (i.e. interpret as **−−>** indicating the direction of travel) moves the cursor, and therefore the line, towards the natural scroll position (**$** in column).

< Right of scroll position

```
|$f line of te$|
|<f line of te$|
|$f line of te$|
```

The current line is to the right of the scrolled position. backward−char (i.e. interpret as **<−−** indicating the direction of travel) moves the cursor, and therefore the line, towards the natural scroll position (**$** in column).

$scroll 1

The position of the cursor on the line determines the scrolled position. In this case all lines in the window are scrolled to ensure that the cursor is always visible. This mode is only useful when dealing with large blocks of text whose line lengths do not vary. **NOTES**

The scrolling is an attribute of the WINDOW and not the BUFFER. If the window is closed, or contents swapped to a different buffer then the scroll setting is reset for the next buffer. A return to the previous buffer does not restore the scroll setting. The only case where scrolling is inherited is when a window is split (see split−window−vertically(2)).

When binding **scroll−left** to the keyboard then it is important to note that when no argument is specified the resolution is *frame−width*'s. A key binding would operate on character multiples, hence the command should be bound with a numeric argument to perform the perform the keyboard action. e.g.

```
        1 global-bind-key scroll-left  "A-left"
        1 global-bind-key scroll-right "A-right"
```

To move 5 columns on a key stroke, for an accelerated scroll, then the binding may be re−written as:−

```
        5 global-bind-key scroll-left  "A-left"
        5 global-bind-key scroll-right "A-right"
```

SEE ALSO

$scroll(5), scroll−up(2), scroll−down(2), $window−x−scroll(5).

scroll−next−window−down(2)

NAME

scroll−next−window−down − Scroll next window down
scroll−next−window−up − Scroll next window up

SYNOPSIS

n **scroll−next−window−down** (**esc C−v**)
n **scroll−next−window−up** (**esc C−z**)

DESCRIPTION

scroll−next−window−down scrolls the next window down *n* lines, if *n* is omitted then the next window is scrolled by *window* number of lines (i.e. next screen page).

scroll−next−window−up scrolls the next window up *n* lines, as **scroll−next−window−down**.

These commands are useful in macros to control other windows.

SEE ALSO

scroll−up(2), scroll−down(2).

search−forward(2)

NAME

search−forward − Search for a string in the forward direction
search−backward − Search for a string in the backward direction

SYNOPSIS

n **search−forward** "*string*" (**C−x s**)
n **search−backward** "*string*" (**C−x r**)

DESCRIPTION

search−forward searches for a string from the current cursor position to the end of the file. The string is typed on the bottom line of the screen, and terminated with the `<ESC>` key. Special characters can be typed in by preceding them with a `^Q`. A single `^Q` indicates a null string. On successive searches, hitting `<ESC>` alone causes the last search string to be reused.

Searching is affected by magic(2m) mode, which allows regular expression pattern matching, and exact(2m) mode which makes the search case sensitive.

The numeric argument *n* is interpreted as follows:−

n > **0**

The *n*th occurrence of the *string* is located.

n < **0**

The first occurrence of the *string* is located in the next *n* lines.

search−backward searches backwards in the file. In all other ways it is like **search−forward**.

DIAGNOSTICS

The command returns a status of `FALSE` if the *string* could not be located (or *n*th *string* where *n* occurrences are requested). If the *string* is found within the given search criteria the return status is `TRUE`.

SEE ALSO

buffer−mode(2), exact(2m), hunt−backward(2), hunt−forward(2), isearch−forward(2), magic(2m),

replace−string(2).
Regular Expressions

set−alpha−mark(2)

NAME

set−alpha−mark − Place an alphabetic marker in the buffer

SYNOPSIS

set−alpha−mark "*?*" (**C−x C−a**)

DESCRIPTION

set−alpha−mark places an alpha mark at the current location in the buffer which can be returned to from anywhere in the buffer using the command goto−alpha−mark(2). The user is prompted for a mark name which can be any alphabetic character. the mark is destroyed if the line is deleted.

SEE ALSO

goto−alpha−mark(2).

set−char−mask(2)

NAME

set−char−mask − Set character word mask

SYNOPSIS

n **set−char−mask** "*flags*" ["*value*"]

DESCRIPTION

set−char−mask returns or modifies the setting of MicroEmacs internal character tables. The argument *n* defines the action to be taken, as follows:−

−1

Removes characters from the given set.

0

Returns characters in the given set in $result(5).

1

Adds characters to the given set.

The first argument "*flags*" determines the required character set as follows:−

d

Is Displayable. Characters in this set can be directly displayed to the screen (as a single character) when occurring in a buffer. When a character not in this set is to be displayed it is performed using more than one character. Characters in the range 1−31 are displayed as "`^?`" where `?` is the ASCII character plus 64, (e.g. 0x01 −> 65, i.e. "`^A`") otherwise the character is displayed in the form "`\xhh`" where `hh` is the hex form of the ASCII value. One notable exception is the tab character (0x09), by default this character is not displayable, instead it is displayed as a sequence of one or more spaces up to the next tab stop.

p

Is Pokable. Similar to **d**, characters in this set can be poked to the screen when using screen−poke(2). When found in a binary file the character is displayed in the right hand column. Unlike **d**, any character outside this set will be displayed as a single period '`.`', indicating that it cannot be displayed.

P

Is Printable. Similar to **d**, characters in this set may be printed as a single character when using print−buffer(2) or print−region(2). Any character not in this set is printed in a similar fashion to **d**.

M

Character font Map. Internally MicroEmacs uses ISO−8859−1 (Latin 1) to configure alphabetic classes and the spell−checker, however the system font being by the native platform may not be the same, for example a small 'e' acute is character 0xe9 in ISO−8859−1 but character 0x82 in Windows OEM fonts. To change the characteristics of the 'e' acute character (such as making it an alphabetic character), the ISO−8859−1 character should always be used, but a correct mapping of ISO−8859−1 to the display font (such as Windows OEM) must also be supplied.

> Unlike other sets, this set cannot be incrementally altered, any calls to alter this set leads to the resetting of all the character tables so the character mapping must be performed first and in a single call. No other set may be altered in the same call. When setting, the "*value*" must supply pairs of characters, an ISO−8859−1 character followed by its system font equivalent.

L

ISO−8859−1 (Latin 1) character map list. This set cannot be altered using this flag, character mappings must be set up using flag **M**. The order of the characters in the returned **$result** string is the same as the order for flag **U**.

U

User font character map list. This set cannot be altered using this flag, character mappings must be set up using flag **M**. The order of the characters in **$result** when returned is the same as the order for flag **L**.

a

Is Alphabetic letter. Characters in this set are alphabetical characters, used by many MicroEmacs commands such as forward−word(2). When setting, the "*value*" must specify pairs of ISO−8859−1 (Latin 1) characters, an Upper−case character followed by its lower−case equivalent. This enables commands such as lower−case−word(2) to operate correctly regardless of the font and language being used. Some fonts may not have all the characters available for rendering, for instance PC Code page 437 does not have an upper−case 'e' grave. In this case an ordinary 'E' should be used as a sensible replacement, i.e. "E`e" (where `e is an 'e' grave). However, this will lead to all upper−case 'E's to map to a lower−case 'e' grave in a case changing operation, this may be corrected by adding a further mapping of 'E' to 'e' to over−ride the 'e' grave mapping, i.e. "E`eEe". This technique does fail when changing the case more than once, when all lower case 'e' graves will be lost.

> Note that the returned character list will pair all lower−case characters with their upper−case equivalent letters first.

l

Is Lower case letter. This set cannot be altered using this flag, alterations to the alphabetic set must be performed using flag **a**. Characters in this set are all the lower−case letters, typically the characters '`a`' to '`z`'. The order may not be the same as returned by flag **u**.

u

Is Upper case letter. This set cannot be altered using this flag, alterations to the alphabetic set must be performed using flag **a**. Characters in this set are all the upper−case letters, typically the characters '`A`' to '`Z`'. The order may not be the same as returned by

h

Is Hex−decimal Digit. The set is rarely used as it is invariably the digits '`0`' to '`9`' and the letters '`a`' to '`f`' in upper and lower case. It is often used in the setting of $buffer−mask(5).

A

Is Alpha−numeric. This set cannot be altered using this flag, alterations to the alphabetic set must be performed using flag **a**. Characters in this set are either alphabetic characters or the digits 0−9.

s

Is Spell extended word character. The characters in this set are recognized by the spell checker as characters which may be considered part of a word, for example the period '`.`'s in e.g. or the hyphen '`−`' in hyphenated−words. Typically this set contains the characters '`'`', '`−`' and '`.`'.

1, **2**, **3** & **4**

Is in Word. These user definable sets are used to add characters to a buffer's word character set, affecting the operation of commands like forward−word(2). Many different file types operate better with a different word character set, e.g. it is preferable to include the '_' character when editing C files. See variable $buffer−mask(5).

Unless stated otherwise, multiple flags may be specified at the same time returning a combined character set or setting multiple properties for the given "*value*" characters.

EXAMPLE

For many UNIX XTerm fonts the best characters to use for $box−chars(5) (used in drawing osd(2) dialogs) lie in the range 0x0B to 0x19. For example the vertical bar is '`\x19`', the top left hand corner is '`\x0D`' etc. These characters are by default set to be not displayable or pokable which renders them useless. They can be made displayable and pokable as follows:−

```
set−char−mask "dp" "\x19\x0D\x0C\x0E\x0B\x18\x15\x0F\x16\x17\x12"
```

MicroEmacs variables have either '`$`', '`#`', '`%`', '`:`' or a '`.`' character prepended to their name, they may also contain a '`−`' character in the body of their name. It is preferable for these characters to be part of the variable 'word' so commands like forward−kill−word(2) can work correctly. This may be achieved by adding these characters to user set **2** and setting the **buffer−mask** variable to include set **2**, as follows:

```
        set-char-mask "2" "$#%:.-"

        define-macro fhook-emf
            set-variable $buffer-mask "luh2"
            .
            .
        !emacro
```

For the examples below only the following subset of characters will be used:−

```
        Character               ISO-8859-1    Windows OEM    PC Page 437

        Capital A (A)           A             A              A
        Capital A grave (`A)    \xC0          \xB7           No equivalent
        Capital A acute ('A)    \xC1          \x90           No equivalent
        Small a (a)             a             a              a
        Small A grave (`a)      \xE0          \x85           \x85
        Small A acute ('a)      \xE1          \xA0           \xA0
```

As the spell checker only operates in ISO−8859−1 (Latin 1), the character font mapping (flag **M**) must be correctly setup for spell checking to operate correctly. For ISO−8859−1 (ISO) this is an empty string as the default mapping is correct, but for both Windows OEM (OEM) and PC Code Page 437 (PC−437) the mappings should be set as follows:−

```
        ; OEM font mapping setup
        set-char-mask "M" "\xC0\xB7\xC1\x90\xE0\x85\xE1\xA0"
        ; PC-437 font mapping setup
        set-char-mask "M" "\xC0A\xC1AAA\xE0\x85\xE1\xA0"
```

As all the characters in ISO have equivalents in OEM, the mapping for OEM is a simple ISO to OEM character list. However the missing capital **A**'s in PC−437 cause problems, for the command charset−iso−to−user(3) it is preferable for a mapping of **`A** to be given, otherwise the document being converted may remain unreadable. Therefore a mapping of **`A** to **A** is given to alleviate this problem, similarly **'A** is also mapped to **A**.

This leads to a similar problem with the conversion of PC−437 back to ISO (the operation of command charset−user−to−iso(3)). If only the mapping of "`\xC0A\xC1A`" was given, the last mapping (**'A** to **A**) would also be the back conversion for **A**, i.e. ALL **A**'s would be converted back to **'A**'s. To solve this problem, a further seemingly pointless mapping of **A** to **A** is given to correct the back conversion.

For languages which use these characters, the alphabetic character set must be extended to include these characters for letter based commands like forward−word(2) and upper−case−word(2) to operate correctly. The addition of extra letters must achieve two goals, firstly to define whether a character is a letter, enabling commands like **forward−word** to work correctly. The second is to provide an upper case to lower case character mapping, enabling commands like **upper−case−word** to work correctly. This is achieved with a single call to **set−char−mask** using the **a** flag as follows:−

```
        set-char-mask "a" "\xC0\xE0\xC1\xE1"
```

Note that this flag always expects a ISO−8859−1 character, this allows the same map character list to be used regardless of the font set being used, i.e. the above line can be used for ISO, OEM and PC−437 fonts. But it does mean that the ISO to user font character mapping (flag **M**) must already have been performed.

Similar problems are encountered with the **M** flag with font PC−437. This problem is not immediately obvious because the mapping is given in ISO, but when this is converted to PC−437, the mapping string becomes "`A\x85A\xA0`". As can be seen, **A** is mapped last to **'a** so an upper to lower character operation will convert a **A** to **'a**. A similar solution is used, a further mapping of **A** to **a** is given to correct the default case mapping for both **A** and **a**, i.e. the following line should always be used instead:−

```
set−char−mask "a" "\xC0\xE0\xC1\xE1Aa"
```

SEE ALSO

forward−word(2), $buffer−mask(5), screen−poke(2), spell(2), $tabwidth(5).

set−cursor−to−mouse(2)

NAME

set−cursor−to−mouse − Move the cursor to the current mouse position

SYNOPSIS

n **set−cursor−to−mouse**

DESCRIPTION

set−cursor−to−mouse sets the current window and cursor position to the location of the mouse on it's last event (button press or release). This command may change the current window. If the line on which the mouse was located was the message line then the no action is taken, if the line was a window mode line the that window is made the current window but the cursor location within the window remains the same. This is usually used in user defined macros that control the functionality of the mouse.

An argument *n* determines if the command is permitted to change windows, when omitted a window change is permitted on **set−cursor−to−mouse**. When specified, the mouse is not permitted to change windows and returns an error condition in $mouse−pos(5) indicating that the mouse is not within the current window.

Invocation of this command sets the variable $mouse−pos(5) which determines where the mouse is within the window. Interrogation of the variable following the command may be used to determine if the mouse is located on one of the more specialized window or screen regions.

When writing macros to cut and paste using the mouse, care should be taken to ensure that the window at the button release is the same is at the button press. If this is not undertaken, undesired effects could result. The use of set−position(2) and goto−position(2) are most usefully used with this command to restore existing window context.

SEE ALSO

$mouse−pos(5), $mouse−x(5), $mouse−y(5), $window−mode−line(5), $window−scroll−bar(5), set−scroll−with−mouse(2), set−position(2), goto−position(2).

set−encryption−key(2)

NAME

set−encryption−key − Define the encryption key

SYNOPSIS

set−encryption−key (**esc e**)

DESCRIPTION

set−encryption−key sets the encryption key for files loaded or saved with crypt(2m) mode enabled. This must be performed for each file, key is not entered into the history. The key can be set for each file on the command line using the **−k** flag. When saving a buffer in encryption mode the key will be prompted for if not already set.

SEE ALSO

buffer−mode(2), crypt(2m), find−file(2), find−cfile(3).

set−mark(2)

NAME

set−mark − Set starting point of region

SYNOPSIS

set−mark (**esc space**)

DESCRIPTION

set−mark is used to delimit the beginning of a marked region. Many commands are effective for a region of text. A region is defined as the text between the mark and the current cursor position. To delete a section of text, for example, one moves the cursor to the beginning of the text to be deleted, issues the **set−mark** command by typing **esc space**, moves the cursor to the end of the text to be deleted, and then deletes it by using the kill−region(2) (**C−w**) command. Only one mark can be set in one window or one buffer at a time, and MicroEmacs '02 will try to remember a mark set in an off screen buffer when it is called back on screen.

A region is a block of text to be acted upon by some MicroEmacs '02 commands. It is demarcated by the **POINT** on one end and the **MARK** at the other. The point is the primary location identifier where most of the action takes place and is always between two characters. The point is indicated by the cursor position in that it is just behind the cursor. The point is also significant in that it defines one end of the region. The mark, on the other hand, is invisible, and is used to demarcate the other end of the region and is set through **set−mark**.

SEE ALSO

copy−region(2), exchange−point−and−mark(2), kill−region(2). reyank(2), yank(2),

set−scroll−with−mouse(2)

NAME

set−scroll−with−mouse − Scroll the window with the mouse

SYNOPSIS

n **set−scroll−with−mouse**

DESCRIPTION

The **set−scroll−with−mouse** command controls the scrolling of a window by the mouse. This is a two stage process, the first stage locks the cursor to the mouse, the second stage scrolls the screen.

The first stage (locking) is performed when the mouse is located on the scroll−box (typically when the left button is depressed i.e. **pick−mouse−1**). **set−scroll−with−mouse** is invoked with an argument *n*, this causes the mouse position to be recorded ready for a scroll. Depending on the scroll method, the blank lines present at the end of the buffer are scrolled off the screen.

Subsequent calls to the **set−scroll−with−mouse** are made with no argument, the window is scrolled by the relative displacement of the mouse from it's locked position, motion is limited at the end of the scrolling region. Scrolling is proportional to the buffer length. The command is typically bound to **move−mouse−1** which results in an update whenever the mouse is moved by the user.

When the button is released **drop−mouse−1** then the scrolling is stopped by unbinding **move−mouse−1**, thereby breaking the binding between the mouse moving and the scroll command.

The scrolling utilizes fractional mouse positional information (i.e. units smaller than a character cell), if available, resulting in a smoother scrolling motion.

EXAMPLE

The following example shows how the command is used.

```
0 define−macro mouse−scroll−pick
    1 set−scroll−with−mouse         ; Lock mouse position to scroller
    global−bind−key set−scroll−with−mouse "mouse−move−1"
!emacro

0 define−macro mouse−scroll−drop
    global−unbind−key "mouse−move−1"
!emacro

global−bind−key mouse−scroll−pick "mouse−pick−1"
global−bind−key mouse−scroll−drop "mouse−drop−1"
```

When the left button is 'picked', **mouse−scroll−pick** lock the cursor to the mouse and binds mouse movement to **set−scroll−with−mouse** so that whenever the mouse is moved the cursor will be repositioned appropriately. When the button is 'dropped', the mouse movement is unbound so that the cursor will no longer be locked to the mouse.

SEE ALSO

$mouse−pos(5), $scroll−bar(5), set−cursor−to−mouse(2).

set−variable(2)

NAME

set−variable − Assign a new value to a variable
unset−variable − Delete a variable

SYNOPSIS

set−variable "*variable*" "*value*" (**C−x v**)
unset−variable "*variable*"

DESCRIPTION

set−variable sets the given register (**#** name), system (**$** name), global (**%** name), buffer (**:** name) or command (**.** name) variable to the given value, erasing its current value. The returned value of an undefined variable is the string "`ERROR`", this maybe used to determine whether a variable has been set.

unset−variable unsets the given variable so that it no longer exists. The variable must be a global (**%**), buffer (**:**) or command (**.**) variable, system (**$**) variables cannot be unset.

The *value* may be quoted or unquoted, if there are any white space characters, or characters open to other interpretation (e.g. **@wc**) in *value* then quotes should be used.

value may contain control characters which are delimited by a back slash (\) which include:−

`\n` newline
`\t` tab
`\\` backslash

Confusion sometimes arises in macros with the back slash, as the back slashes are dereferenced when set. Commands such as replace−string(2) where the command itself utilizes back slashes. In this case the number of back slashes should be doubled as the variable contents under go two stages of dereferencing.

SEE ALSO

describe−variable(2), list−variables(2), &set(4).

Variables
Introduction to Variable Functions
Register Variables

shell(2)

NAME

shell − Create a new command processor or shell

SYNOPSIS

shell (**C−x c**)

DESCRIPTION

shell−command creates a new command processor or shell. Upon exiting the shell, MicroEmacs '02 redraws its screen and continues editing. The exceptions to this are as follows:

X−Windows

A new **xterm** is spawned off and editing control is returned to MicroEmacs '02 once the **xterm** has initialized.

Microsoft Windows

A new MS−DOS shell is created and control is returned to MicroEmacs '02 once the DOS console window has initialized. The shell created is determined by the MS−DOS environment variable `COMSPEC`, this may be a replacement shell e.g. `4DOS`. **SEE ALSO**

ipipe−shell−command(2), pipe−shell−command(2), suspend−emacs(2).

shell−command(2)

NAME

shell−command − Perform an operating system command

SYNOPSIS

shell−command "*string*"

DESCRIPTION

shell−command performs an operating system call with the given *string* as its argument. The command only fails if the shell−command call returns −1. The $result(5) variable is set the return value and can be used to test the result.

SEE ALSO

$result(5), ipipe−shell−command(2), pipe−shell−command(2), suspend−emacs(2).

show−cursor(2)

NAME

show−cursor − Change the visibility of the cursor

SYNOPSIS

n **show−cursor**

DESCRIPTION

show−cursor hides the cursor if a negative argument is given and restores it if a positive or no argument is given. Note that this is not supported on all platforms.

show−cursor internally performs a counting operation, if the cursor is hidden *m* times then it must also be shown *m* times before the cursor becomes visible again, giving no argument will restore the count ensuring it is visible.

show−region(2)

NAME

show−region − Show the current copy region

SYNOPSIS

n **show−region**

DESCRIPTION

show−region manipulates the currently defined region, it can be used to inquire the state of the current region, if any. It can also be used to define a region, enable and disable the region hilighting, as well as move the cursor to the start or end of the region.

Region hilighting occurs between the *mark* (see set−mark(2)) and *point* (current cursor) positions within the current buffer. A region is defined when text is copied to the kill buffer, by using any of the kill commands such as kill−region(2), or copy−region(2). However, the kill region is only visible after a copy−region(2) or a yank(2) operation. A hilight region is also created on a successful search using commands like search−forward(2), the region encloses the search matching string. Spell(2) also creates a hilight region around the current spell word. The user can also define their own region using the numeric argument to **show−region**.

The argument *n* supplied to the command indicates the require functionality and can take the following values:−

```
-3 − Set the start position of the region.
-2 − Move the cursor the  Mark position.
-1 − Disable the hilighting of the current region.
 0 − Return the current status of the region in
$result(5).
 1 − Enable the hilighting of the current region.
 2 − Move the cursor the  Dot position.
 3 − Set the end position of the region.
 4 − Reactivate the current region.
```

Where an argument of 0 is used to return the current state the value of $result is a bit based flag where:−

0x01

Indicates a region is currently active (visible).

0x02

Indicates a region has been fixed (may not visible).

0x04

Indicates the region is in the current buffer.

0x08

Indicates the cursor is in the current region.

The color of the selection hilight is defined by add−color−scheme(2) and is determined by $buffer−scheme(5), $global−scheme(5) or $buffer−hilight(5).

DIAGNOSTICS

The following errors can be generated, in each case the command returns a FALSE status:

[No current region]

There is no current defined region on which to operate.

[Current region not in this buffer]

An argument of 2 or −2 was used and the defined region isn't in the current window so the cursor can not be moved to it. **NOTES**

If no argument is given to the command it hilights the current region, similar to an argument of 1. But the properties of the hilight, namely how long it will be hilighted for, are inherited from the setting of $show−region(5), whereas if an argument of 1 is passed in then the hilighting is set to be kept until the region becomes invalid (i.e. as if $show−region(5) is set to 3).

SEE ALSO

$show−region(5), $buffer−hilight(5), $buffer−scheme(5), $global−scheme(5), add−color−scheme(2), copy−region(2), yank(2), search−forward(2), spell(2), set−mark(2).

start−up(3)

NAME

start−up − Editor startup callback command
shut−down − Editor exit callback command

SYNOPSIS

start−up
shut−down

DESCRIPTION

By default **start−up** is not defined, if the command is defined (via a user macro) then it is executed immediately after MicroEmacs '02 has completed its initialization.

This command may initially seem redundant as the user may execute any command at start−up by editing the "`me.emf`" file or using the '**@**' command−line argument. At the point of "`me.emf`" file execution none of the files specified on the command−line will be loaded, thus any actions required on the given command−line files will not work (the only buffer present will be the "***scratch***" buffer).

The **start−up** command is executed AFTER the execution of "`me.emf`" and initialization of buffers, but before MicroEmacs '02 waits for user input.

The **shut−down** command is also not defined by default, but if it is defined during the running of MicroEmacs the command will be called when MicroEmacs exits. The command is not called if MicroEmacs has to perform an emergency exit (due to the system being shut down or process being killed etc).

SEE ALSO

me(1).

sort−lines(2)

NAME

sort−lines − Alphabetically sort lines

SYNOPSIS

n **sort−lines**

DESCRIPTION

sort−lines alphabetically sorts lines of text in the current buffer from the mark position to the current cursor position. If the buffer mode exact(2m) is enabled then the sort is case sensitive, otherwise the sort is case insensitive. By default the text is compared from left to right from column 0 (the left hand edge), if a positive argument *n* is given then the text is compared left to right from the *n*th column, any lines shorter than *n* characters are moved to the top and sorted from column 0.

If a negative argument *n* is given then the text is sorted in reverse order. The comparison starts at column `−1−n`, i.e. an argument of −1 sorts in reverse order from column 0.

EXAMPLE

The following table gives the results of **sort−lines** for different exact modes and values of *n*.

	Original	Sorted Lines					
exact	−	n	n	y	y	n	n
n	−	−	1	−	1	−1	−2
	B	a2	B	Aa	B	CA	Aa
	CA	Aa	c	B	c	c	CA
	b1	B	b1	CA	b1	b1	a2
	Aa	b1	a2	a2	a2	B	b1
	c	c	CA	b1	CA	Aa	c
	a2	CA	Aa	c	Aa	a2	B

NOTES

Typically MicroEmacs is executed with exact(2m) mode enabled, the macro command **sort−lines−ignore−case** provides a command to sort lines case insensitively while **exact** mode is enabled. The macro is defined as follows:−

```
define−macro sort−lines−ignore−case
    set−variable #l0 &bmod exact
    −1 buffer−mode "exact"
    !if @?
        @# sort−lines
    !else
        sort−lines
    !endif
    &cond #l0 1 −1 buffer−mode "exact"
!emacro
```

sort−lines−ignore−case(3) is a macro defined in format.emf.

SEE ALSO

buffer−mode(2), exact(2m), sort−lines−ignore−case(3), transpose−lines(2), uniq(3).

sort−lines−ignore−case(3)

NAME

sort−lines−ignore−case − Alphabetically sort lines ignoring case"

SYNOPSIS

n **sort−lines−ignore−case**

DESCRIPTION

sort−lines−ignore−case forces the current buffers exact(2m) mode to off and then calls sort−lines(2) which will perform a case insensitive alphabetical line sort from the mark position to the current cursor position. The state of the current buffers **exact** mode is restored on completion.

NOTES

sort−lines−ignore−case is a macro defined in `format.emf`, see help on command sort−lines(2) for a complete definition.

SEE ALSO

sort−lines(2), buffer−mode(2), exact(2m), transpose−lines(2).

spell(2)

NAME

spell − Spell checker service provider

SYNOPSIS

n **spell** ["*word*"] ["*rules*"] ["*correction*"] ["*rule*"]

DESCRIPTION

spell is a low level command which provides spell checking capabilities for MicroEmacs '02, it is not designed to be used directly. The action of **spell** depends on the argument given, which is a bitwise flag defined as follows:−

0x001

If set then gets the input word from the user, i.e. "*word*" must be supplied. Otherwise the word input is taken from the current buffer.

0x002

If set then keeps getting words from the current buffer until either the end of the buffer is reached or an error is found. If the end of the buffer is reached then the command succeeds setting $result(5) to the value "*F*". This bit is ignored if bit 0x001 is set. **spell** sets the current show−region to enclose the problematical word and the command show−region(2) can be used to move around the word.

0x004

Adds the given word to a dictionary determined by the state of bit 0x008. If the word is flagged as erroneous (see bit 0x010) then a "*correction*" word must be supplied, otherwise a list of "*rules*" which can be applied to the word must be given, this list can be empty. Note that if the word is not flagged as erroneous and it already exists in the dictionary, the word is not removed, instead a combined rule list is created.

0x008

When set flags that word additions (bit 0x004) and deletions (bit 0x200) should be made to the ignore dictionary. Otherwise word additions are made the last added dictionary and deletions are made to all main dictionaries.

0x010

When set flags that the given word is erroneous, used solely by word additions to create auto−corrections.

0x020

Returns a '|' separated guest guess list for the given word in **$result**.

0x040

If bit **0x100** is also set a complete list of valid words derivable from the given word are inserted into the current buffer. Otherwise spell returns $result(5) set to the derivative word created when the given "*rule*" is applied to "*word*". The rule applied is the first found of the given rule letter with a matching base ending (see add−spell−rule(2)). The word need not exist as not tests for the legality of the resultant word is used, for example in American, executing

```
65 spell "spelling" "V"
```

returns "`spellingive`" in **$result**. Returns the empty string if no rule could be applied.

0x080

Used with bit 0x002 to enable double word checking.

0x100

Return information in **$result** about the given word, or the word which is used to derive the given word. The information consists of the spell status, the word as stored in the dictionary, and either the list of valid rules, or the correction word. See also bit **0x040**.

0x200

Delete the given word from a dictionary determined by bit 0x008

If none of the main functions are used (bits 0x004, 0x020, 0x040 & 0x200) then the status flag is returned in the first column of **$result**. These are defined as follows:−

A

Auto−replace. The word was found and flagged as erroneous. The correction word is given in **$result**, either next to the flag, or if bit 0x100 is set then after the '>' character.

D

Double word. Indicates that the first problem found is a double occurrence of the same word one after the other.

E

Erroneous. The word was not found, so is Erroneous

N

Not a word. The current word found contains no alphabetic characters so is not deemed to be a word, e.g. 3.141593.

O

Okay. The word was found and is not an erroneous word. **SEE ALSO**

add−dictionary(2), add−spell−rule(2), delete−dictionary(2), save−dictionary(2), show−region(2), spell−buffer(3), spell−word(3), Locale Support.

spell−add−word(3)

NAME

spell−add−word − Add a word to the main dictionary

SYNOPSIS

n **spell−add−word** ["*word*"]

DESCRIPTION

spell−add−word adds words to the last dictionary added using the command add−dictionary(2). If no argument is supplied the user is prompted for the word and rule flags, only a 'Good' word can be added (see below). If an argument *n* is given then the next *n* words from the current buffer are added. The words must take one of the following three forms:

`xxxx` − Good word `xxxx` with no spell rules allowed
`xxxx/abc` − Good word `xxxx` with spell rules `abc` allowed
`xxxx>yyyy` − Erroneous word with an auto−replace to `yyyy`

NOTES

spell−add−word is a macro defined in file `spellutl.emf`. It is not defined by default so `spellutl.emf` must be executed first using execute−file(2).

SEE ALSO

add−dictionary(2), edit−dictionary(3), save−dictionary(2), delete−dictionary(2).

split−window−horizontally(2)

NAME

split−window−horizontally − Split current window into two (horizontally)

SYNOPSIS

n **split−window−horizontally** (**C−x 5**)

DESCRIPTION

split−window−horizontally splits the current window horizontally into two near equal windows, each displaying the buffer displayed by the original window.

A numeric argument *n* of 1 forces the left window to be the new current window, and an argument of 2 forces the right window to be the new current window. The default when omitted is the left window.

SEE ALSO

$scroll−bar(5), $scroll−bar−scheme(5), $window−chars(5), grow−window−horizontally(2), split−window−vertically(2).

split−window−vertically(2)

NAME

split−window−vertically − Split the current window into two

SYNOPSIS

n **split−window−vertically** (**C−x 2**)

DESCRIPTION

split−window−vertically splits the current window vertically into two near equal windows, each displaying the buffer displayed by the original window. A numeric argument *n* of 1 forces the upper window to be the new current window (default), and an argument of 2 forces the lower window to be the new current window.

SEE ALSO

grow−window−vertically(2), next−window−find−buffer(2), next−window−find−file(2), resize−window−vertically(2), split−window−horizontally(2).

suspend−emacs(2)

NAME

suspend−emacs − Suspend editor and place in background

SYNOPSIS

n **suspend−emacs**

PLATFORM

Supported on UNIX platforms − *irix*, *hpux*, *sunos*, *freebsd* or *linux*.

DESCRIPTION

suspend−emacs suspends the editing processor and puts it into the background. The "*fg*" command restarts MicroEmacs. The prompt to suspend is disabled if a `0` numeric argument *n* is given to the command.

SEE ALSO

shell(2).

symbol(3)

NAME

symbol − Insert an ASCII character

SYNOPSIS

symbol

DESCRIPTION

symbol draws the ASCII character table to the screen, displaying decimal, hexadecimal and character notations in a tabular form. A character is selected using the mouse or cursor characters inserting the selected character into the current buffer at the current position.

NOTES

symbol is a macro defined in `misc.emf`.

The dialog is created using osd(2).

SEE ALSO

insert−string(2), &atoi(4), osd(2).

Triangle(3)

NAME

Triangle − MicroEmacs '02 version of Triangle patience game

SYNOPSIS

Triangle

DESCRIPTION

Triangle is a solitaire game using a standard set of playing cards. The object of the game is to use all of the cards in the deck to build up four suit stacks from Ace to King.

The board is laid out so that every card is used to create a triangle shape. In the first column there is one up−turned card, in the second column there is one down−turned card and 2 up−turned, third has 2 down 3 up etc. The only break form this pattern is in the last 3 columns where there is an extra up−turned card so that all the deck is used.

Cards may be moved around the playing area by stacking the same suit cards in descending order on the row stacks. When a row stack has no up−turned cards on the stack then the top card may be turned over and may be played. If a stack becomes empty then only a King may be moved into the vacant position.

If the last card in a stack is an Ace then it can be moved to its suit stack, then the 2 of that suit etc. until finally the King is removed.

Cards are moved around the board using the mouse. Cards may be moved from one row stack to another row stack by placing the mouse over the 'from' stack and pressing the left mouse button. Move the cursor to the 'to' stack and release the left mouse button. If the move is legal then the card(s) are moved to the new stack. Multiple cards may be moved from the row stacks, the appropriate card(s) to be moved is automatically determined.

Cards may be moved onto the suit stacks by a single left mouse press and release on the same card, the card is moved to the appropriate suit stack. The same technique is used to turn cards over in the suit stacks.

Note that once a card is played onto the suit stacks then it cannot be removed.

To the right of the board are a number of control buttons. To select an option, click the left mouse button on it, the buttons are labeled:

DEAL

Start a new game by dealing new cards.

QUIT

Exit the game

HELP

This help page

Note that the screen may be updated at any time using "*C−l*".

NOTES

Triangle is a macro defined in `triangle.emf`.

The game is best played with a mouse, it is possible to play with the keyboard, as follows:−

"*esc h*" for help

To move a card between stacks enter the source and destination column number ("*1*","*2*",.."*7*").

To overturn a card on the row stacks then enter the card column twice i.e. source and destination are the same.

To move a card from the row to the suit stacks then either enter the card column twice, or enter the destination as "*h*","*d*","*c*","*s*" (i.e. "*2 2*" or "*2 s*" to move the card in column 2 to the spades stack).

"*C−c C−c*" to deal the cards again.

"*C−l*" redraw the screen.

"*q*" to quit the game.

SEE ALSO

Games, Patience(3), Mahjongg(3).

tab(2)

NAME

tab − Handle the tab key

SYNOPSIS

n **tab** (**tab**)

DESCRIPTION

tab manages the `tab` key, typically inserts *n* tabs. The effect of the command is determined by:

$buffer−indent

If $buffer−indent(5), is non−zero then the effect of tab is defined by the setting of bit `0x1000` of variable $system(5), typically it resets the current line indentation or inserts a tab.

cmode

If cmode is enabled then the effect of tab is defined by the setting of bit `0x1000` of variable $system(5), typically it resets the current line indentation or inserts a tab.

tab

If a tab is to be inserted and this mode is enabled then multiple spaces are used instead of tab characters, see tab(2m) mode. **SEE ALSO**

cmode(2m), $buffer−indent(5), tab(2m), backward−delete−tab(2), insert−tab(2), normal−tab(3), $tabsize(5), $tabwidth(5).

tabs−to−spaces(3)

NAME

tabs−to−spaces − Converts all tabs to spaces

SYNOPSIS

tabs−to−spaces

DESCRIPTION

tabs−to−spaces converts all tab characters found in the current buffer with spaces. The number of spaces a tab is replaced with depends on the column of the tab character and the setting of $tabwidth(5).

The cursor is restored to the start of the current line after completion.

NOTES

tabs−to−spaces is a macro defined in `format.emf`.

SEE ALSO

$tabwidth(5), tab(2), tab(2m), clean(3).

time(3)

NAME

time − Command time evaluator

SYNOPSIS

time "*string*"

DESCRIPTION

time evaluates the time take to execute line "*string*". **time** uses command execute−line(2) to execute the given string.

EXAMPLE

The following example simply times the time take to save the current buffer:−

```
time "save-buffer"
```

NOTES

time is a macro defined in `misc.emf`.

On multi−task systems like UNIX **time** cannot take into account the number of other processes running at the same time, it can only return the actual time elapse. This leads to inaccuracies and variation in results.

SEE ALSO

execute−line(2).

translate−key(2)

NAME

translate−key − Translate key

SYNOPSIS

n **translate−key** ["*from*" ["*to*"]]

DESCRIPTION

translate−key may be used to convert any given input key sequence to another single key. **translate−key** operates at a very low level, before MicroEmacs attempts to evaluate keyboard bindings, so it may be used to solve a variety of keyboard problems such as special language characters and UNIX termcap key sequence bindings (see below).

If a +ve numeric argument *n* is given it is used to set the time in milliseconds MicroEmacs waits for another key to be pressed before continuing, the default time use when no argument is supplied is 250ms.

If a numeric argument *n* of −1 is specified then the "*to*" argument is not required and the "*from*" character sequence is removed from the translate key table.

If a numeric argument *n* of 0 is specified then no arguments are required; the current translation table is dumped to buffer "**tcap−keys**". Following is a sample output:−

```
"C-h" ......................... "backspace"
"C-[" ......................... "esc"
"C-[ [ 1 ~" ................... "delete"
"C-[ [ 1 1 ~" ................. "f1"
"C-[ [ 1 2 ~" ................. "f2"
"C-[ [ 1 3 ~" ................. "f3"
"C-[ [ 1 4 ~" ................. "f4"
"C-[ [ B" ..................... "down"
"C-[ [ 4 ~" ................... "end"
"C-[ [ 2 ~" ................... "insert"
"C-[ [ 3 ~" ................... "home"
"C-[ [ D" ..................... "left"
"C-[ [ 6 ~" ................... "page-down"
"C-[ [ 5 ~" ................... "page-up"
"C-[ [ C" ..................... "right"
"C-[ [ A" ..................... "up"
"C-[ [ V" ..................... "page-up"
"C-[ [ U" ..................... "page-down"
"C-m" ......................... "return"
"C-i" ......................... "tab"
"\x7F" ........................ "backspace"
```

FOREIGN KEYBOARDS

Foreign keyboards (non−US/UK) use a variety of key sequences, not recognized by MicroEmacs, to expand the keyboard character range to cope with accented characters. For example, on a German keyboard '`AltGr−m`' (recognized as '`A−C−m`') is used to insert a Greek mu (or micro sign). On a Belgian keyboard '`AltGr−9`' inserts a '{' character.

Many foreign keyboards are already directly supported by MicroEmacs and the keyboard specifics of a country have been understood and resolved. In these cases the **Keyboard** configuration in user−setup(3) may be used for the country location.

If MicroEmacs does not support your keyboard, **translate−key** may be used to fix any key input problems. For the aforementioned examples the following **translate−key** commands would be required:

```
; translate AltGr−m to a Greek mu (char 0xb5)
translate−key "A−C−m" "\xB5"
; translate AltGr−9 to a '{'
translate−key "A−C−9" "{"
```

The problem is complicated further on Microsoft Window's platforms by the simultaneous generation of 2 keys for some Alt−Gr key combinations (this is a side effect of endeavoring to capture all key combinations in this environment). For the Belgian keyboard example, on Win32 platforms an '`AltGr−9`' generates an '`A−C−9`' key first followed immediately by an '`A−C−{`'. As both keys are generated in quick succession this is unexpected and confusing.

When the key is first pressed on a poorly configured system the error "*[Key not bound "A−C−{"]*" is given even when using the command describe−key(2) as the key described will be '`A−C−9`' and then the '`A−C−{`' key is generated and interpreted creating the error message.

The variable $recent−keys(5) can be used to diagnose this problem and to obtain the 2 keys generated; alternatively use the macro below:

```
define−macro report−2−keys
    ml−write "Press key 1"
    set−variable #l0 @cgk
    ml−write "Press key 2"
    set−variable #l1 @cgk
    ml−write &spr "[The following keys where pressed: \"%s\" \"%s\"]" #l0 #l1
!emacro
```

When executed the user is prompted for the first key; press the required key sequence (in this case '`AltGr−9`'), if you are not prompted for the second key and the result is immediately returned then the key you pressed has generated 2 keys, both of which will be given in the print out, i.e.:

```
"[The following keys where pressed: "A−C−9" "A−C−{"]"
```

The translate−key required to fix this type of problem would be:

```
translate−key "A−C−9 A−C−{" "{"
```

If your keyboard is not directly supported by MicroEmacs, please submit the keyboard name and platform with a working translate−key configuration to JASSPA as a **BUG**.

UNIX TERMCAP

translate−key may also be used to interpret non−standard key sequences for UNIX termcap platforms to standard MicroEmacs keys. Non−standard keys, such as the cursor keys, have system dependent key sequences. The output from these keys usually take the form:

`^[[X` or `^[[DX` or `^[[DDX` or `^[[DDD`

where **^[** is the escape key (27), **D** is a digit and **X** is any character. These keys may be bound to the standard keys, for example the typical output of the cursor keys may be translated as follows:−

`^[[A` = **up**, `^[[B` = **down**, `^[[C` = **right** and `^[[D` = **left**

The "*from*" string is specified as this key sequence and the "*to*" string is simply the key it is to be bound to, see global−bind−key(2) for a guide to the string format. For the above example the following set of translations are required:−

```
translate−key "esc [ A" "up"
translate−key "esc [ B" "down"
translate−key "esc [ C" "right"
translate−key "esc [ D" "left"
```

Note that MicroEmacs interprets `\e` as an escape key. More obscure keys tend to be very platform specific, following are some examples:

```
translate−key "esc [ 2 ~" "insert"
translate−key "esc [ 5 ~" "page−up"
translate−key "esc [ 5 ^" "C−page−up"
```

EXAMPLE

Using the +ve numeric argument it is possible to reduce the delay and there by increase usability is some features. For instance, in the Mouse configuration of **user−setup** there is an option to 'Simulate 3 Buttons' which translates a rapid left and right button press into a middle button press. This is implemented using **translate−key** as follows:

```
10 translate−key "mouse−pick−1 mouse−pick−3" "mouse−pick−2"
10 translate−key "mouse−pick−3 mouse−pick−1" "mouse−pick−2"
10 translate−key "mouse−drop−1 mouse−drop−3" "mouse−drop−2"
10 translate−key "mouse−drop−3 mouse−drop−1" "mouse−drop−2"
```

When a `mouse−pick−1` key is generated MicroEmacs must wait to see if a `mouse−pick−3` key is next and therefore translate both to a single `mouse−pick−2` key. This wait time is usually a quarter of a second but this makes the left button unusable for dragging regions etc as the delay is too long. By giving a argument of 10ms the delay is long enough for a simultaneous left and right button press but short enough for the left button to still be usable on its own.

The +ve numeric argument can be very useful for delaying MicroEmacs as well, for example, the character string "`'e`" can be converted to e−accute using expand−iso−accents(3). This could be performed automatically using translate−key as follows:

```
1000 translate-key "' e" "\xE9"
```

The larger 1 second delay give the user enough time to type the '`e`' after the '`'`' character.

NOTES

The concept of standardized key−bindings is very important for cross platform use and maintenance.

Refer to global−bind−key(2) for a list of standard bindings.

One of the easiest ways of obtaining a key sequence is to run **sh(1)** which does not attempt to interpret these keys so when a key is pressed (followed by `<RETURN>`) the following type of error message is usually generated:−

```
sh: ^[[2~:  not found.
```

where `^[[2~` is the required key sequence. Another method of obtaining these key sequences is to start MicroEmacs '02, use start−kbd−macro(2) to start a macro definition, press the required keys and then use end−kbd−macro(2) followed by name−kbd−macro(2) and insert−macro(2) to display the keys pressed.

The key sequences generated for these keys are dependent on the machine displaying MicroEmacs '02 as opposed to the machine running it. Often they are the same machine, but when they are not there is no easy method of determining the displaying machine and therefore correctly configuring MicroEmacs '02.

A better way of obtaining this cross platform consistency is to create an XTerm app−defaults setup file with the correct VT100 key translations, e.g. the setup file could contain the following

```
*vt100.translations: #override \
        Shift<Key>Tab:          string("\033[Z") \n\
        <Key>BackSpace:         string("\177") \n\
        <Key>Delete:            string("\033[1~") \n\
        <Key>Insert:            string("\033[2~") \n\
        <Key>Home:              string("\033[3~") \n\
        <Key>End:               string("\033[4~") \n\
        <Key>Prior:             string("\033[5~") \n\
        <Key>Next:              string("\033[6~") \n\
        Ctrl<Key>Up:            string("\033Oa") \n\
        Ctrl<Key>Down:          string("\033Ob") \n\
        Ctrl<Key>Right:         string("\033Oc") \n\
        Ctrl<Key>Left:          string("\033Od") \n\
        Shift<Key>Up:           string("\033[a") \n\
        Shift<Key>Down:         string("\033[b") \n\
        Shift<Key>Right:        string("\033[c") \n\
        Shift<Key>Left:         string("\033[d") \n
```

By using the environment variable *XUSERFILESEARCHPATH* to ensure that this configuration file is found instead of the system one (found in `/usr/lib/X11/app-defaults`), the key sequences will then be the same across all platforms. See manual page on **xterm(1)** for more information.

SEE ALSO

expand−iso−accents(3), user−setup(3), describe−key(2), global−bind−key(2), start−kbd−macro(2), **xterm(1)**, **sh(1)**.

transpose−chars(2)

NAME

transpose−chars − Exchange (swap) adjacent characters transpose−lines − Exchange (swap) adjacent lines

SYNOPSIS

transpose−chars (**C−t**)
n **transpose−lines** (**C−x C−t**)

DESCRIPTION

transpose−chars exchanges (swaps) the current character under the cursor with the previous character. **transpose−characters** does not operate in column 0 (since there is no previous character). If the cursor is at the end of a line when the command is initiated then the cursor is moved to the previous character and the operation performed from the new position.

transpose−lines swaps the next line for the current line and moves to the next line, effectively retaining the same text position. Repeating this *n* times moves the current line *n* lines down.

EXAMPLE

transpose−character performs the following operations (cursor at ^):−

```
abcde  => acbde         [Middle of line]
  ^         ^

abcde  => abced         [End of line]
     ^         ^
```

SEE ALSO

sort−lines(2).

undo(2)

NAME

undo − Undo the last edit

SYNOPSIS

n **undo** (**C−x u**)

DESCRIPTION

undo removes the last *n* edits made to the current buffer. The undo(2m) buffer mode must be enabled for this command to operate.

The undo information is retained up until the next save operation, at which point the undo information is discarded. When editing large files with gross changes then it is advisable to either disable undo mode, or save frequently to flush the undo buffer, thereby keeping MicroEmacs '02 memory requirements reasonable (most UNIX users have restrictions on the amount of memory that may be consumed by a single process. Windows is restricted by the amount of virtual memory (or swap space)).

SEE ALSO

buffer−mode(2), save−buffer(2), undo(2m).

uniq(3)

NAME

uniq − Make lines in a sorted list unique

SYNOPSIS

uniq

DESCRIPTION

uniq reduces a sorted lines of text in the current buffer to a unique list such that no entries are repeated. The list is made unique from the mark position to the current cursor position (point). The operation is case sensitive.

NOTES

uniq is a macro implemented in `tools.emf`.

For **uniq** to operate correctly then the list must have been previously sorted, see sort−lines(2).

SEE ALSO

sort−lines(2), sort−lines−ignore−case(3), transpose−lines(2),

universal−argument(2)

NAME

universal−argument − Set the command argument count

SYNOPSIS

universal−argument (**C−u**)

DESCRIPTION

universal−argument sets the argument number passed to a command to 4^*n* (4 to the power of *n*) where *n* is the number of calls to **universal−argument**, e.g. the key sequence "`C−uC−n`" moves down 4 lines, "`C−uC−uC−uC−n`" moves down 4*4*4 = 64 lines.

After invoking the **universal−command** a '−' character can be pressed to negate the argument value, and an alternative numeric argument can be entered using the '*0*' to '*9*' keys.

Invoking this command via execute−named−command(2) or by a macro has no effect. The command should be treated as a command key prefix (like prefix(2)) in that it may be bound to only one key sequence which must be a single key stroke. Re−binding this command to another key unbinds the new key and also the current **universal−argument** key.

The **prefix 1** key (by default bound to `esc`) may also be used to enter a numeric argument at the message line, e.g. "`esc 1 0 C−f`" will move forward 10 characters.

SEE ALSO

prefix(2).

user−setup(3)

NAME

user−setup − Configure MicroEmacs for a specific user

SYNOPSIS

user−setup

DESCRIPTION

user−setup provides a dialog interface to enable the user to configure the editor. **user−setup** may be invoked from the main *Help* menu or directly from the command line using execute−named−command(2). **user−setup** configures the user's setup registry file, "<*logname*>`.erf`" which is used by MicroEmacs to initialize the environment to a user's preference.

Note, if your screen is too small to display the whole dialog, it may be moved using any key bound to the scroll commands such as **scroll−up**, e.g. `A−up`, `C−z`, `A−down`, `C−v`, `A−left` etc. For systems without mouse support, the `tab` key may be used to move between fields.

On all pages the following buttons are available at the bottom of the dialog and have the following effect:

`Save`

Saves the changes made to the users registry file, i.e. "<*Log−Name*>`.erf`" but does not re−initialize MicroEmacs. Some changes, such as color scheme changes, only take effect when the **Current** button is used or when MicroEmacs is restarted.

`Current`

Makes the current user and the changes made Current to this MicroEmacs session, dismissing the **user−setup** dialog and reinitializing MicroEmacs. This also saves the registry file out!

`Exit`

Quits user−setup, if changes where not **Save**d or made **Current** they will be lost.

The following pages, which appear in the dialog, are defined as follows:−

Start−up

`Log Name`

Sets the name of the current user to setup, this can be set to any valid file base name (no extension) which need not be the current user. The rest of the **user−setup** entries are then initialized to the settings defined for the given user (or standard defaults if not defined).

`Default User`

Creates a small macro file, "`default.emf`", setting $MENAME(5) to the current setting of **Log Name**. This may be executed at start−up to determine the current user. See $MENAME(5) for more information.

`Setup Path`

Sets the location of the user files, the files are searched for and created in this directory. $MEPATH(5) should be defined to include this path.

`Setup File`

Sets the personal user setup macro file name which is executed at start−up. A user macro file should contain all personal settings such as preferred key bindings etc. See Setting Up A User Profile for more information. The **Edit** check box can be used to enable/disable the automatic loading of the setup file ready for editing when the **Current** button is used.

`Company File`

Sets the company setup macro file name which is executed at start−up. A company macro file should contain all company wide standard settings such as `%company−name`, No `.emf` extension is supplied. See Setting Up a Company Profile for more information.

`Emulate`

Sets an emulation mode which changes the behaviour on MicroEmacs to emulate another editor/program; this is done by executing a macro file at start−up. An emulation macro file should contain the macro code required to simulate the environment of the other editor. MicroEmacs '02 is released with two emulation modes, MicroEmacs v3.8 which executes macro file `meme3_8.emf` (See Compatibility for more information) and NEdit v5 which is at best a demonstration of what can be achieved, this executes macro file `menedit.emf`.

`MS Friendly Keys`

When enabled the following key bindings are created to ease frustration for MS users:

`home`

Bound to beginning−of−line instead of beginning−of−buffer.

`end`

Bound to end−of−line instead of end−of−buffer.

`C−home`

Bound to beginning−of−buffer.

`C−end`

Bound to end−of−buffer.

`C−v`

Bound to yank (paste).

`esc−v`

Bound to reyank.

Note that the "`C−x`" and "`C−c`" keys are just to intrinsic to MicroEmacs to rebind (sorry).

`MS Shift Region`

Enables/disables cursor key manipulation with the shift key similar to the conventional Microsoft region selection. When enabled, pressing the shift key in conjunction with the cursor movement keys selects a region which is hilighted. Once the region is selected then the `<DELETE>` or `<BACKSPACE>` key erases the selected region. This also enables a similar behaviour with the Mouse **Drag region** driver, see below. **Locale Setup**

`Keyboard`

Configures MicroEmacs to the user's keyboard. Accent character generation keys present on foreign keyboards cannot be automatically supported on Windows platforms. MicroEmacs must be informed of the keyboard being used to correctly interpret the keys. If a required keyboard is not supported please see FAQ38 on how to setup the keyboard, also see Locale Support.

`Language`

Sets the user language, this sets the word (or letter) characters and if available sets up spell(2) with appropriate spelling rules and dictionaries. For more information on adding support for a language see Locale Support.

NOTES

Earlier versions MicroEmacs had "`(Ext)`" languages which use extended language dictionaries, vastly increasing the word list. New versions automatically test for and use these dictionaries if available.

In earlier versions a personal dictionary name could be set in the next field, this option was removed on Oct 2001. Instead a personal dictionary for each language is automatically created for you, any words or auto−corrected words will be added to the current languages personal dictionary. The name of dictionary is "`lsdp`*<lang−id>*`.edf`" where "*<lang−id>*"

is the 4 letter MicroEmacs language name (e.g. "`enus`" for American), simply rename any existing personal dictionary to this new name.

`Auto Spell`

Enables Auto Spell Checking in file types which support this feature (usually text based files such as txt(9) or nroff(9) files etc). Auto spell detects word breaks as you type and checks the spelling of every completed word hilighting any erroneous words in the error color scheme (usually red). The feature can be manually enabled and disabled by invoking the auto−spell(3) command (usually bound to "`f5`").

`Auto Save Dics`

Enables auto−saving of any changed dictionaries on exit. If this is disabled the user is prompted to save for each changed dictionary. **General**

`Full Name`

This should be set to the user's name and is used in a variety of places, e.g. by etfinsrt(3) to set the "Created By" field in a template.

`Organizer`

Sets the organizer file base name, defaults to the **Log Name**. When notes and addresses are stored using organizer(3) the file "<*Organizer*>`.eof`" is used.

`Auto−Save Time`

Sets the length of time in seconds between buffer auto−saves, a setting of `0` or an empty string disables auto−saving. The default setting is 300 seconds or 5 minutes. This indirectly sets the auto−time(5) variable and the autosv(2m) global mode.

`Global Modes`

Sets the initial state of the global quiet(2m) mode. This indirectly executes global−mode(2) to set the required modes.

`Buffer Modes`

Sets the initial state of the global modes auto(2m), backup(2m), tab(2m) and undo(2m), any buffers created will inherit the state of these modes. However, as changing these modes directly effects only the global modes, any existing buffers (including ones re−created using the `−c` command−line option, see me(1)) will not be effect by the setting of these modes. For them to take effect, the buffers should be reloaded. These modes can be changed on a per file type basis using the command buffer−setup(3), also some file hooks override these global settings, such as the makefile(9) hook which overrides the **tab** mode. This indirectly executes global−mode(2) to set the required modes.

`Search Modes`

Sets the initial state of the global search modes exact(2m) and magic(2m). This indirectly executes global−mode(2) to set the required modes.

`Keep Undo History`

If this is enabled the undo history is kept after a save allowing the undo(2) command to back−up changes beyond the last save. When clear the undo history is discarded after the buffer is saved. This indirectly sets bit 0x8000 of the $system(5) variable.

`Hide Backups`

Enables hiding MicroEmacs generated backup files. On Windows and Dos platforms the Hidden file attribute is used to hide the file, whereas on UNIX the backup file name is prepended with a '`.`'. This indirectly sets bit 0x100000 of the $system(5) variable.

`Main Menu`

Enables the top main menu bar.

`Alt -> Main Menu`

If enabled the main menu Alt hot−key bindings are enabled. These are dynamic bindings automatically generated from the main menu. Typically the first item in the main menu is "`File`" with a hot key of '**F**', with this enabled '`A−f`' will open this menu item. Note that global and local key bindings override these. This indirectly sets bit 0x2000 of the $system(5) variable.

`Alt -> Esc Prefx`

If enabled the Alt key acts as a prefix 1 modifier key. By default '`A−n`' is not bound, with this bit set the key is inferred to '`esc n`' which is bound to **forward−paragraph**. Note that global, local and menu hot−key bindings override these. This indirectly sets bit 0x4000 of the $system(5) variable.

`Abbrev Expansion`

Configures which expansion methods are enabled by default when the expand−abbrev−handle(3) is executed. **Accent** enables expand−iso−accents(3), **Lookbk** enables expand−look−back(3) and **Dict'n** enables expand−word(3).

`Tab To Indent`

Sets the tab(2) behavior in a buffer which has cmode(2m) enabled or an indentation method. This indirectly sets bits `0x1000` and `0x200000` of the $system(5) variable.

`Show Modes`

Selects which modes are to be displayed on the mode−line whenever a "`%e`" token is used in the $mode−line(5) variable. This indirectly sets the $show−modes(5) variable. **Platform − UNIX Setup**

Only present on UNIX platforms using the X interface, see below for the Console setup.

`Font`

Sets the X font name to be used. This indirectly executes change−font(2) with the given font name. e.g.

```
    "-misc-fixed-bold-r-normal--13-*-*-*-c-70-iso8859-1"
```

`Display Char Set`

Selects the display character set being used by the system to render the MicroEmacs window, dependent on the **Font** being used. The setting of this option effects the configuration of MicroEmacs's internal character maps (using command set−char−mask(2)) enabling the character sets of foreign languages to be correctly supported. It also changes the definition of variables $box−chars(5) and $window−chars(5) to their best values for the given font.

`Extend Char Set`

When enabled MicroEmacs replaces the display of characters `0x00` to `0x1f` with forms which are useful for variables $box−chars(5) and $window−chars(5) greatly improving the look of osd(2) dialogs, the scroll bars etc.

`Use Fonts`

When enabled the bold, italic, light and underline characteristics of the font will be used depending on their availability and the Color Scheme being used. This indirectly sets bit `0x10` of the $system(5) variable.

`Draw White Spaces`

Enables the drawing of visible white spaces, i.e. space, tab and new−line characters. This indirectly sets bit `0x80000` of the $system(5) variable.

`Enable Toolbar`

Enables the Toolbar − configurable, managed windows giving easy access to many features and tools.

`Client Server`

The client/server enables the file based external macro command driver to be enabled − see Client−Server. This by default is disabled, when enabled it is used by command−line options **−m** and **−o**.

`DOS File Names`

DOS has a restricted 8.3 file naming system (i.e. "`BBBBBBBB.XXX`"), if this option is enabled the MicroEmacs '02 will adhere to this system for auto−save and backup file names whenever possible. See $auto−time(5) for more information on the naming convention used. This indirectly sets bit

0x400 of the $system(5) variable.

`# Backups`

This option only has an effect when **DOS File Names** is disabled. Setting this to a number greater than zero enables multiple backup files to be created, the number determined by this value. If set to zero (or less) then only a single backup file is created. This indirectly sets the $kept−versions(5) variable.

`Ignore Files`

Sets a list extensions of files to be ignored in file completion, e.g. MicroEmacs backup files (~). This indirectly sets the $file−ignore(5) variable.

`Cursor Blink Rate`

Sets the cursor blink period in millisecond. The first entry box sets the cursor visible time, a setting of zero disables blinking. The second box sets the hidden time. A visible time of 600 and hidden time of 200 gives a reasonable blink cycle. This indirectly sets the $cursor−blink(5) variable.

`Fence Display`

Sets the preferred method of displaying a matching fence, a fence is one of the following brackets:

```
{...}   (...)   [...]
```

Jumping to the opening fence only occurs when the closing brace is typed, whereas the drawing of matching fences occurs whenever the cursor is on an open fence or one character past the close fence. When this option is set to "`Never Display`" the buffer−setup(3) setting is ignored.

`Scroll Bars`

Selects the scroll bar support required. When Splitter is enabled, the first character of the scroll bar and mode−line is a split character used for splitting the window into two using the mouse. This indirectly sets the $scroll−bar(5) variable.

`Horizontal Scroll`

Selects the horizontal scrolling method used with the scroll−left(2) and scroll−right(2) commands. This indirectly sets the $scroll(5) variable.

`Vertical Scroll`

Selects the vertical scrolling method used with the forward−line(2) and backward−line(2) commands. This indirectly sets the $scroll(5) variable.

`Color Scheme`

Sets the color scheme setup macro file name which is executed at start−up. MicroEmacs by default comes with 4 color schemes. Color schemes can be created and altered using the scheme−editor(3) dialog. **Platform − UNIX Console Setup**

Only present on UNIX platforms when using the termcap interface, all the Console platform settings are kept independent of the X interface settings.

`Termcap Color`

This option determines whether Termcap based colors should be used. These are typically the standard eight colors and may not be supported on all terminals. If this option is disabled Termcap fonts (such as bold) are used instead to create a primitive hi−lighting. This indirectly sets bit 0x004 of the $system(5) variable.

`Use Fonts`

See **Platform UNIX Setup** above.

`Display Char Set`

See **Platform UNIX Setup** above.

`Draw White Spaces`

See **Platform UNIX Setup** above.

`Client Server`

See **Platform UNIX Setup** above.

`DOS File Names`

See **Platform UNIX Setup** above.

`# Backups`

See **Platform UNIX Setup** above.

`Ignore Files`

See **Platform UNIX Setup** above.

`Cursor Blink Rate`

See **Platform UNIX Setup** above.

`Scroll Bars`

See **Platform UNIX Setup** above.

`Horizontal Scroll`

See **Platform UNIX Setup** above.

`Vertical Scroll`

See **Platform UNIX Setup** above.

`Color Scheme`

See **Platform UNIX Setup** above. **Platform − Win32 Setup**

Only present on Microsoft Windows based machines.

`Font Name`

Sets the windows font name and size. This indirectly executes change−font(2) with the given font name. MicroEmacs may only use a Fixed Mono Font, either an OEM font as used by the MS−DOS command line, or the more conventional ANSI fonts. The fonts are selected using the **Change Font** button which invokes a dialog to allow the available fonts to be selected. True−Type mono fonts such as `Courier New` or `Lucida Console` are typically used.

`Weight & Size`

Allows the size and weight of the font to be selected, specified as *weight*, *width* and *height*. The *weight* is typically 4, this corresponds to a regular weighting, 7 is bold. *width* is the width of the font in pixels, this may be 0 when the height is specified as −ve. *height* is the height of the font, typically a −ve value (where the *width* is 0), which produces a proportionally sized font, values of in the range −11 .. −14 generally produce reasonably sized fonts. The *hight* and *width* may be specified as +ve values and allow explicit font dimensions to be specified, generally used to achieve a precise font size requirement.

`Use Fonts`

See **Platform UNIX Setup** above.

`Display Char Set`

See **Platform UNIX Setup** above.

`Extend Char Set`

See **Platform UNIX Setup** above.

`Choose Font`

Opens a windows dialog allowing the user to select a font, the selection is used to configure the above font fields.

`Draw White Spaces`

See **Platform UNIX Setup** above.

`Capture Alt Space`

Used to enable/disable the capture and interpretation of the '`A−space`' key sequence. If this key sequence is not captured by MicroEmacs it is passed back to Windows which opens the top left window menu, allow keyboard access to Window commands like Maximize.

`Client Server`

See **Platform UNIX Setup** above. Note that on windows based systems the client/server is also used by memsdev(1) to drive the editor from the Microsoft Developer environment.

`DOS File Names`

See **Platform UNIX Setup** above. Note that some early version of Windows '95 have problems with ~ extensions. Service release 2 fixed these problems − if you experience problems then return to 8.3 filename mode − note that MicroEmacs will still store longer file names, only the backup naming convention changes.

`# Backups`

See **Platform UNIX Setup** above.

`Ignore Files`

See **Platform UNIX Setup** above.

`Cursor Blink Rate`

See **Platform UNIX Setup** above.

`Scroll Bars`

See **Platform UNIX Setup** above.

`Horizontal Scroll`

See **Platform UNIX Setup** above.

`Vertical Scroll`

See **Platform UNIX Setup** above.

`Color Scheme`

See **Platform UNIX Setup** above. **Platform − Win32 Console Setup**

Only present on Windows NT and Win95+ platforms when using the console interface, all the Console platform settings are kept independent of the Window interface settings.

`Display Char Set`

See **Platform UNIX Setup** above.

`Draw White Spaces`

See **Platform UNIX Setup** above.

`Client Server`

See **Platform Win32 Setup** above.

`DOS File Names`

See **Platform Win32 Setup** above.

`# Backups`

See **Platform UNIX Setup** above.

`Ignore Files`

See **Platform UNIX Setup** above.

`Cursor Blink Rate`

See **Platform UNIX Setup** above.

`Scroll Bars`

See **Platform UNIX Setup** above.

`Horizontal Scroll`

See **Platform UNIX Setup** above.

`Vertical Scroll`

See **Platform UNIX Setup** above.

`Color Scheme`

See **Platform UNIX Setup** above. **Platform − DOS Setup**

Only present on DOS machines.

`Graphic Mode #` and `Double Lines`

Sets the DOS graphics mode number and whether the number of text lines can be doubled. This indirectly executes change−font(2) with the given font name.

`Display Char Set`

See **Platform UNIX Setup** above.

`Draw White Spaces`

See **Platform UNIX Setup** above.

`Ignore Files`

See **Platform UNIX Setup** above.

`Cursor Blink Rate`

See **Platform UNIX Setup** above.

`Scroll Bars`

See **Platform UNIX Setup** above.

`Horizontal Scroll`

See **Platform UNIX Setup** above.

`Vertical Scroll`

See **Platform UNIX Setup** above.

`Color Scheme`

See **Platform UNIX Setup** above. **Mouse**

The mouse device creates keys in a similar way to regular keyboard keys and, like keyboard keysm they must be bound before they are used. MicroEmacs '02 does not have the mouse functionality hard coded into the editor, it provides a macro interface to the mouse for ultimate flexibility and a set of default functionality which can be bound to the mouse in a variety of ways.

All the mouse controlling macros are stored in `mouse.emf` and `mouseosd.emf` although some buffers have local functionality over−rides, such as file−browser(3). The user can expand the range of mouse functionality but how this is achieved is beyond the scope of this documentation.

The **user−setup** dialog allows the user to configure the mouse to use the default functionality, as follows:−

`Enable Mouse`

Enables or disables the mouse, when disabled the mouse can not be used and will not generate any key events. This does not apply to UNIX Termcap systems as the mouse cut and paste operation is performed by the Xterm. This indirectly sets bit 0x010 of the $mouse(5) variable.

`Number Buttons`

Sets the number of buttons on the mouse, may be 1, 2 or 3. MicroEmacs usually obtains the correct number for the system, but sometimes this can be wrong. This entry can be used to correct this problem. For one button mice, the button is considered to be the `left` mouse button, two button mice have an `left` and `right` button. This indirectly sets the $mouse(5) variable.

`Swap Buttons`

If enabled then the `left` and `right` buttons are swapped, i.e. when the left button is pressed it executes the right button bindings. This indirectly sets bit 0x020 of the $mouse(5) variable.

`Simulate 3 Buttons`

If enabled then pressing the `left` and `right` buttons together with generate a middle button press event, this feature is for people with a 2 button mouse who want more. The two buttons must be pressed or release within 10 millisecond of each other.

The following four fields determine which mouse button binding the user wishes to view and change:−

`Button`

The mouse button, `Left`, `Right` or `Middle` for the normal buttons and `Whell Up` or `Whell Down` for the pilot wheel events.

`Shift Pressed`

The action of the mouse can be different for every modifier key setting, if this is enabled then the binding being modified is for the **Button** being pressed with the **Shift** key held down.

`Control Pressed`

If enabled then modifying the action when the **Button** is pressed with the **Control** key held down.

`Alt Pressed`

If enabled then modifying the action when the **Button** is pressed with the **Alt** key held down.

The following two fields determine the functionality of the button defined by the previous four fields:−

`Handle Scroll`

When enabled, if the button is pressed with the mouse on the main menu, a scroll bar or mode−line the standard action is performed, such as opening the main menu or scrolling up or down the window etc. The **bound To** command is only called if the mouse is in a main window. If disabled, the **Bound To** command is always called.

`Bound To`

The function to be performed. The functions available depend on the type of button being bound, the following is a list of functions available for normal buttons:−

`Not bound`

The Button is not bound.

`Drag region`

set−mark(2) is called at the pick location, until the button is dropped, the area of text between this point and the current mouse position is hi−lighted. When the mouse button is dropped, if the drop position is the same as the pick then the double click is tested for, if a double click is entered then the **Select Word** function is executed, otherwise the cursor is simply moved to the drop position. If the pick and drop position are different then the enclosed text is copied to the kill buffer using copy−region(2). Note this behaviour is altered by the setting of **MS Shift Region** on the **Start−Up** page.

`Select Word`

Also executed from a double click bound to **Drag Region**, **Select Word** copies the word under the mouse into the kill buffer using copy−region(2), unless a double click is entered in which case the whole line is copied.

`Default Pan`

While the mouse button is pressed the current buffer pans with any mouse movement.

`MS Pan`

MicroSoft style Pan; while the mouse button is pressed the current buffer pans vertically according to the mouse position relative to the point where the button was pressed.

`Find Tag`

Executes find−tag(2) with the word currently under the mouse.

`Find ME Help`

Executes help−item(2) with the word currently under the mouse.

`Undo`

Simply executes undo(2) without moving the cursor to the position of the mouse. Subsequent calls to this binding will undo multiple edits.

`No move yank`

Simply executes yank(2) without moving the cursor to the position of the mouse.

`Replace yank`

Simplar to "`No move yank`" except when the is a current region (typically defined by "`Drag region`" above), in which case the region is first deleted.

`Move to yank`

Moves the cursor to the current position of the mouse and executes yank(2).

`Reyank`

Executes reyank(2) without moving the cursor. Note, to enable this functionality some sanity checks have had to be removed, as a result it should not be misused as seeming bizarre things can occur.

`Fold current`

Toggles the fold status of the current block, only applicable in buffers supporting fold−current(3), such as c and emf files.

`Fold all`

Toggles the fold status of the whole buffer, opening or closing all found blocks. Only applicable in buffers supporting fold−all(3), such as c and emf files.

`Main menu`

Simply opens the main menu from any where on the screen.

`Multi−Menu`

Opens a context sensitive menu dependent on the position of the mouse, i.e. opens the main menu if over it, opens a different menu when executed on the mode−line etc.

The following is a list of functions available for pilot wheel events:−

`Not bound`

The Button is not bound.

`Scroll Up 1 Line` ....

Scrolls the current buffer by the specified amount.

`Defaults`

Rests the mouse configuration to the default settings. **File Types**

The file type list is used in two places, the main menu's `File => Quick Open` sub−menu list and the `File => Open => File Type` list. In each case the file type "`All Files`" is automatically added. The user can add, remove and change the list of file types by using this dialog. An entry can be selected for editing or deletion by simply selecting it with the left mouse button. A new entry may be added by simply filling in the 3 entry boxes and selecting Add. Items in the Dialog are as follows.

`No.`

The file type entry number. A new entry is always added to the end of the list, ignoring this value. The position of an existing entry can be changed by altering this field to the desired position and selecting the `Change` button to move it to its new position.

`Name`

The file type name, the string printed in the sub−menus.

`File Mask List`

A comma ('`,`') separated list of file masks which match the file type, e.g. for C and C++ source files use "`*.c,*.cc,*.cpp`".

`Add`

Adds a new entry to the list, only the **Name** and **FileMask List** fields are used, the **No.** field is ignored as the new entry is always added to the end of the list. The position can be altered by using the **Change** button.

`Change`

Alters an existing file type entry, all 3 fields must be set.

`Delete`

Deletes the current entry number, only the **No.** entry is used. **Tools**

The Tools dialog allows the user to configure up to 10 system commands, or tools, which can be executed via MicroEmacs Main Tools Menu. The dialog configures the user's registry for the command execute−tool(3) to be used. The execution of a tool can also be bound to a key, see **execute−tool** for more information.

The top half of the dialog consists of the 10 Tools (`0−9`) configuration buttons. Selecting one of these selects the current tool to be configured, the current tool is shown by the title in the middle of the dialog.

The lower half of the dialog configures the currently selected tool, as follows:−

`Tool Name`

Sets the displayed name of the tool. The tool name is used in the buttons in the top half of this dialog and in the MicroEmacs Main Tools Menu.

`Tool Command Line`

Sets the system command−line to be launched whenever the tool is executed, the following special tokens may be used in the command−line which are substituted at execution:−

%ff

The current buffer's full file name, including the path.

%fp

The current buffer's file path.

%fn

The current buffer's file name without the path.

%fb

The current buffer's file base name, i.e. the file name without the path or the extension.

%fe

The current buffer's file extension with the '.' (e.g. "*.emf*"), set to the empty string if the file name does not have an extension.

Note that "**%ff**" is always the same as "**%fp%fn**" and "**%fp%fb%fe**". If any of these tokens are used, the tool will fail to execute if the current buffer does not have a file name.

`Save Current Buffer` and `Prompt`

If the current buffer has been edited, enabling `Save Current Buffer` will automatically save the current buffer before executing the tool. This is particularly useful when the tool operates on the current buffer's file (e.g. compiles the file). If `Prompt` is also enabled the user will be prompted before the file is saved.

`Save All Buffers` and `Prompt`

If `Save All Buffers` is enabled, all edited buffers will be automatically saved before executing the tool. This is particularly useful when the tool may operate on multiple files (e.g. compilation of a project). If `Prompt` is also enabled the user will be prompted before each file is saved.

`Capture Output`

If enabled any output produced from the execution of the tool will be captured and inserted into a new buffer. When enabled the following two items, `Buffer` and `Hide`, may be specified. When disabled the command used to execute the tool is shell−command(2), otherwise the command used is either pipe−shell−command(2) or ipipe−shell−command(2) depending on the setting of `Run Concurrently`.

`Buffer`

Specifies the buffer name the captured output should be dumped to, this option is only visible when `Capture Output` is enabled. The following special tokens may be used in the buffer name which are substituted at execution:−

%fn

The current buffer's file name without the path, set to the buffer name if the current buffer does not have a file name.

%fb

The current buffer's file base name, i.e. the file name without the path or the extension. Set to the buffer name if the current buffer does not have a file name.

%fe

The current buffer's file extension with the '.' (e.g. "*.emf*"), set to the empty string if the current buffer does not have a file name or it does not have an extension.Note that "**%fn**" is always the same as "**%fb%fe**". Default buffer name when this field is left empty is "**command**", or "**icommand**" if `Run Concurrently` is enabled.

`Hide`

When enabled the tool output capture buffer is hidden, this option is only visible when `Capture Output` is enabled.

`Run Concurrently`

If enabled, when the tool is executed the command is launched and run concurrently, allowing the user to continue working in MicroEmacs during the tools execution. This option is not available for all versions on

MicroEmacs and forces the output to be captured. Enabling this option will force the use of command ipipe−shell−command(2) to launch the tool. **E−Mail**

MicroEmacs '02 provides a simple E−Mail manager, see vm(3) for more information and example entries. It must be stressed that **vm** has only been tested in one environment, caution should be used as system differences may cause problems, such as loss of data, which the author does not except any responsibility for.

The **E−Mail Setup** dialog configures a user to use part or all of the **vm** E−Mail manager, as follows:−

Platform ALL Mail Setup

The following field is used for both sending and receiving mail:

`User Mail Dir`

Sets the user mail−box directory where all files are to be found and stored (except usually the **Incoming Mail box**). The value of this field is platform independent and must be setup for each one.

The following fields are used for sending mail:

`Send Mail Signature`

Sets the signature file name which is inserted at the bottom of every out−going email message, if empty the no signature is inserted. The value of this field is platform independent, is value use by all. The file must be located in the **User Mail Dir** and no path entered for it to work across platforms.

`Carbon−Copy File`

Sets the sent−mail carbon−copy file, creating the "`Fcc:`" line of the mail buffer. All out−going emails are appended to the end of this file if the "`Fcc:`" line is not altered. If this field is left empty then no "`Fcc:`" line is created. The value of this field is platform independent, the file must be located in the **User Mail Dir**.

`Insert Data (^C^I)`

Sets the first embedded data command line, bound to "`C−c C−I`". The value of this field is platform dependent.

`Insert Data (^C^Z)`

Sets the second embedded data command line, bound to "`C−c C−z`". The value of this field is platform dependent.

`Send Mail Command`

Sets the command−line used for sending email messages. The value of this field is platform dependent.

The following fields are used for receiving mail:

`Check for mail`

Sets the time interval between the automatic checking for incoming mail in seconds, when set to 0 the automatic checking is disabled. When enabled, the check is performed by mail−check(3) which also sends any queued mail and gets any new mail if the **Get Mail Command** is used. The value of this field is platform dependent.

`Get Mail Command`

The command used to get new mail from the server, if empty it is assumed the **Incoming Mail Box** is automatically updated by the system. If used the command must append new mail to the end of the **Incoming Mail Box** specified below. The value of this field is platform dependent.

`Incoming Mail Box`

Sets the incoming mail box file which new incoming mail is appended to, either automatically by the system or by the **Get Mail Command**. The value of this field is platform dependent.

`VM Main In Box`

Sets the main current mail box, or inbox. The value of this field is platform independent, the file must be located in the **User Mail Dir**.

`VM Gets Mail`

When enabled, executing the command vm will not only create the mail box windows, it will also get and process any new mail. When disabled only the vm '`g`' command can be used to get and process new mail.

`Mime Data Extract`

Sets the command−line used for extracting Mime encoded embedded data. The value of this field is platform dependent.

`Uuencode Extract`

Sets the command−line used for extracting Uuencoded embedded data. The value of this field is platform dependent.

`Auto−Archive Setup`

Sets up the auto−archive of messages in the current inbox to other mail boxes. **NOTES**

user−setup is a macro using osd(2), defined in `userstp.emf`.

SEE ALSO

User Profiles, Company Profiles, Installation, buffer−setup(3), scheme−editor(3).

view−file(2)

NAME

view−file − Load a file read only

SYNOPSIS

n **view−file** "*file−name*" (**C−x C−v**)

DESCRIPTION

view−file is like find−file(2), and either finds the file in a buffer, or creates a new buffer and reads the file in. A new file is left in view(2m) mode if the file was found (i.e. cannot be edited).

The numeric argument *n* can be used to modify the default behaviour of the command, where the bits are defined as follows:

0x01

If the file does not exist and this bit is not set the command fails at this point. If the file does not exist and this bit is set (or no argument is specified as the default argument is `1`) then a new empty buffer is created with the given file name, saving the buffer subsequently creates a new file.

0x02

If this bit is set the file will be loaded with binary(2m) mode enabled. See help on **binary** mode for more information on editing binary data files.

0x04

If this bit is set the file will be loaded with crypt(2m) mode enabled. See help on **crypt** mode for more information on editing encrypted files.

0x08

If this bit is set the file will be loaded with rbin(2m) mode enabled. See help on **rbin** mode for more information on efficient editing of binary data files. **SEE ALSO**

buffer−mode(2), find−file(2), read−file(2), view(2m), binary(2m), crypt(2m), rbin(2m).

void(2)

NAME

void − Null command

SYNOPSIS

n **void**

DESCRIPTION

void does nothing except return FALSE if the given argument *n* is zero, TRUE otherwise. Used to bind any frequently miss hit keys to something harmless.

SEE ALSO

global−bind−key(2).

which(3)

NAME

which − Program finder
.which.result − Program path

SYNOPSIS

which "*progname*"
.which.result "*string*"

DESCRIPTION

which searches for the given program "*progname*" on the system path (set by the environment variable **$PATH**). If found the location is printed on the message line, otherwise an error message is printed and the command fails.

The variable **.which.result** is set to the last found program or the string "`ERROR`" if the program was not found.

NOTES

which is a macro defined in `tools.emf`, it used the &which macro directive.

SEE ALSO

&which(4).

wrap−word(2)

NAME

wrap−word − Wrap word onto next line

SYNOPSIS

wrap−word

DESCRIPTION

wrap−word wraps the current word onto the next line, justifying the current line if the justify(2m) mode is enabled. The justification method is defined by $fill−mode(5).

SEE ALSO

buffer−mode(2), fill−paragraph(2), $fill−mode(5), justify(2m).

write−buffer(2)

NAME

write−buffer − Write contents of buffer to named (new) file

SYNOPSIS

n **write−buffer** "*file−name*" (**C−x C−w**)

DESCRIPTION

write−buffer is used to write the contents of the buffer to a NEW file, use save−buffer(2) if the buffer is to be written to the existing file already associated with the buffer.

write−buffer writes the contents of the current buffer to the named file *file−name*. The action of the write also changes the file name associated with the current buffer to the new file name.

Unlike append−buffer(2), **write−buffer** always replaces an existing file and the new file inherits the buffers file characteristics instead of the old file's.

On writing the file, if time(2m) mode is enabled then the time stamp string is searched for in the file and modified if located, to reflect the modification date and time.

If the buffer contains a narrow(2m) it will automatically be removed before saving so that the whole buffer is saved and restored when saving is complete

If backup(2m) mode is enabled and the buffer is associated with a different file (compared with *file−name*) then any automatic save copies of the file associated with the *buffer* are deleted.

The argument *n* can be used to change the default behavior of write−buffer described above, *n* is a bit based flag where:−

0x01

Enables validity checks (default). These include a check that the proposed file does not already exist, if so confirmation of writing is requested from the user. Also MicroEmacs '02 checks all other current buffers for one with the proposed file name, if found, again confirmation is requested. Without this flag the command will always succeed wherever possible.

0x02

Disables the expansion of any narrows (see narrow−buffer(2)) before saving the buffer. **NOTES** undo(2) information is discarded when the file is written.

SEE ALSO

$auto−time(5), backup(2m), time(2m), buffer−mode(2), file−attrib(3), change−file−name(2), save−buffer(2), append−buffer(2).

yank(2)

NAME

yank − Paste (copy) kill buffer contents into buffer

SYNOPSIS

n **yank** (**C−y**)

DESCRIPTION

When a non negative argument is supplied to **yank**, the command copies the contents of the kill buffer *n* times into the current buffer at the current cursor position. This does not clear the kill buffer, and therefore may be used to make multiple copies of a section of text. On windowing systems which support clip−boards, such as windows and X−terms, MicroEmacs will also cut to and paste from the global clip−board.

If *yank* is IMMEDIATELY followed by a reyank(2) then the *yanked* text is replaced by text of the next entry in the kill ring. (another **reyank** replaces the text with the previous reyank text and so on).

If an −ve argument is given, **yank** removes the last 0−*n* items from the kill ring.

Text is inserted into the kill buffer by one of the following commands:−

> backward−kill−word(2), copy−region(2), forward−kill−word(2), kill−line(2), kill−paragraph(2), kill−region(2), forward−delete−char(2), backward−delete−char(2).

All the above commands (except **copy−region**) cut text out of the buffer, the last 2 commands require the letter(2m) mode enabled to add the text to the kill buffer. If any of these commands are executed immediately after any other (including itself) or the @cl(4) variable is set to one of these command, the new kill text is appended to the last kill buffer text.

NOTES

Windowing systems such as X−Windows and Microsoft Windows utilize a global windowing kill buffer allowing data to be moved between windowing applications (*cut buffer* and *clipboard*, respectively). Within these environments MicroEmacs '02 automatically interacts with the windowing systems kill buffer, the last MicroEmacs '02 kill buffer entry is immediately available for a *paste* operation into another application (regardless of how it was inserted into the kill buffer). Conversely, data placed in the windowing kill buffer is available to MicroEmacs '02, via **yank**, until a new item has been inserted into the kill buffer (the data may still be available via reyank(2)).

EXAMPLE

The following example is a basic macro code implementation of the transpose−lines(2) command,

```
beginning-of-line
kill−line
forward−line
yank
−1 yank
backward−line
```

Note that similar to **transpose−lines** it does not leave the moved line in the kill buffer, effectively tidying up after itself.

SEE ALSO

yank−rectangle(2), copy−region(2), kill−region(2), letter(2m), reyank(2), @y(4), @cc(4).

Variable Glossary

VARIABLE GLOSSARY

The following is an alphabetic list of **MicroEmacs '02** variables:−

$INFOPATH(5) GNU info files base directory
$LOGNAME(5) System user name (UNIX)
$MEBACKUPPATH(5) Backup file location
$MEBACKUPSUB(5) Backup file name modifier
$MENAME(5) MicroEmacs user name
$MEPATH(5) MicroEmacs search path
$ME_ISHELL(5) Windows ishell command.com
$ME_PIPE_STDERR(5) Command line diversion to stderr symbol
$auto−time(5) Automatic buffer save time
$box−chars(5) Characters used to draw lines
$buffer−backup(5) Buffer backup file name
$buffer−bhook(5) Buffer macro hook command name (buffer current)
$buffer−bname(5) Name of the current buffer
$buffer−dhook(5) Buffer macro hook command name (buffer deletion)
$buffer−ehook(5) Buffer macro hook command name (buffer swapped)
$buffer−fhook(5) Buffer macro hook command name (buffer creation)
$buffer−fmod(5) Buffer file modes (or attributes)
$buffer−fname(5) Name of the current buffer's file name
$buffer−hilight(5) Define current buffer hilighting scheme
$buffer−indent(5) Current buffer indentation scheme
$buffer−input(5) Divert buffer input through macro
$buffer−ipipe(5) Divert buffer incremental pipe input through macro
$buffer−mask(5) Current buffer word class mask
$buffer−mode−line(5) Buffer mode line string
$buffer−names(5) Filtered buffer name list
$buffer−scheme(5) Buffer color scheme
$c−brace(5) C−mode; brace indentation
$c−case(5) C−mode; case indentation
$c−contcomm(5) C−mode; comment continuation string
$c−continue(5) C−mode; line continuation indent
$c−contmax(5) C−mode; line continuation maximum indent
$c−margin(5) C−mode; trailing comment margin
$c−statement(5) C−mode; statement indentation
$c−switch(5) C−mode; switch indentation
$command−names(5) Filtered command name list
$cursor−blink(5) Cursor blink rate
$cursor−color(5) Cursor foreground color
$cursor−x(5) Mouse X (horizontal) position
$cursor−y(5) Mouse Y (vertical) position
$debug(5) Macro debugging flag
$delay−time(5) Mouse time event delay time

$file−ignore(5)	File extensions to ignore
$file−names(5)	Filtered file name list
$file−template(5)	Regular expression file search string
$fill−bullet(5)	Paragraph filling bullet character set
$fill−bullet−len(5)	Paragraph filling bullet search depth
$fill−col(5)	Paragraph Mode; right fill column
$fill−eos(5)	Paragraph filling; end of sentence fill characters
$fill−eos−len(5)	Paragraph filling; end of sentence padding length
$fill−ignore(5)	Ignore paragraph filling character(s)
$fill−mode(5)	Paragraph mode; justification method
$find−words(5)	Filtered word list
$fmatchdelay(5)	Fence matching delay time
$frame−depth(5)	Number of lines on the current frame canvas
$frame−width(5)	Number of columns on the current frame canvas
$global−fmod(5)	Global file modes (or attributes)
$global−scheme(5)	Global buffer color scheme
$home(5)	Users `home' directory location
$idle−time(5)	System idle event delay time
$kept−versions(5)	Number of backups to be kept
$line−scheme(5)	Set the current line color scheme
$line−template(5)	Command line regular expression search string
$ml−scheme(5)	Message line color scheme
$mode−line(5)	Mode line format
$mode−line−scheme(5)	Mode line color scheme
$mode−names(5)	Filtered mode name list
$mouse(5)	Mouse configuration variable
$mouse−pos(5)	Mouse position information
$mouse−x(5)	Mouse X (horizontal) position
$mouse−y(5)	Mouse Y (vertical) position
$osd−scheme(5)	OSD color scheme
$platform(5)	MicroEmacs host platform identifier
$progname(5)	Program file name
$random(5)	Generate a random number
$rcs−ci−com(5)	RCS (and SCCS) check in command
$rcs−cif−com(5)	RCS (and SCCS) check in first command
$rcs−co−com(5)	RCS (and SCCS) check out command
$rcs−cou−com(5)	RCS (and SCCS) check out unlock command
$rcs−file(5)	RCS (and SCCS) file name
$rcs−ue−com(5)	RCS (and SCCS) unedit file command
$recent−keys(5)	Recent key history
$repeat−time(5)	Mouse time event repeat time
$result(5)	Various command return values
$screen−depth(5)	Number of character lines on the screen canvas
$screen−width(5)	Number of character columns on the screen canvas
$scroll(5)	Screen scroll control
$scroll−bar(5)	Scroll bar configuration
$scroll−bar−scheme(5)	Scroll bar color scheme
$search−path(5)	MicroEmacs search path
$show−modes(5)	Select buffer modes to display

$show−region(5) Enable the hilighting of regions
$status(5) Macro command execution status
$system(5) System configuration variable
$tabsize(5) Tab character width
$tabwidth(5) Tab character interval
$temp−name(5) Temporary file name
$time(5) The current system time
$timestamp(5) Time stamp string
$trunc−scheme(5) Truncation color scheme
$variable−names(5) Filtered variable name list
$version(5) MicroEmacs version date−code
$window−acol(5) Window cursor actual column
$window−aline(5) Window cursor actual line
$window−chars(5) Character set used to render the windows
$window−col(5) Window cursor column (no expansion)
$window−depth(5) Number of text lines in a window
$window−flags(5) Current window setup flags
$window−line(5) Window cursor line
$window−mode−line(5) Window mode line position
$window−scroll−bar(5) Window scroll bar (or separator) position
$window−wcol(5) Window cursor column (historic)
$window−width(5) Number of character columns in a window
$window−wline(5) Window cursor line (historic)
$window−x−scroll(5) Current window X scroll
$window−xcl−scroll(5) Current window current line X scroll
$window−y−scroll(5) Current window Y scroll
%compile−com(5) Default system compile command line
%cygnus−bin−path(5) Cygwin BASH directory
%cygnus−hilight(5) Cygwin shell hilight enable flag
%cygnus−prompt(5) Cygwin shell prompt
%diff−com(5) Diff command line
%ftp−flags(5) Configure the FTP console
%gdiff−com(5) Gdiff command line
%grep−com(5) Grep command line
%http−flags(5) Configure the HTTP console
%http−proxy−addr(5) Set HTTP proxy server address
%http−proxy−port(5) Set HTTP proxy server port
%tag−file(5) Tag file name
%tag−option(5) Tag file search option
%tag−template(5) Tag file search string
.calc.result(5) Last calc calculation result

info(3)

NAME

info − Display a GNU Info database
info−on − Display Info on a given topic
info−goto−link − Display Info on a given link
$INFOPATH − GNU info files base directory
.info.path − Cached info search path

SYNOPSIS

info

info−on *topic−str*

info−goto−link *link−str*

$INFOPATH *string*

.info.path *string*

DESCRIPTION

info interprets the GNU *info* pages, and presents the info file information within a buffer window called `*info XXXXX`, where `XXXXX` is the name of the info file. The root of the info page is displayed and may be traversed by selecting the links with the mouse, or by using the standard *info* traversal keys.

The root of the *info* tree is, by default, a file called **dir**, which points to the other information sources. The default search paths for the *info* directories are:−

> `c:/info` − MS−DOS and MS−Windows (all).
> `/usr/local/info` − All UNIX platforms.

The root directory may also be specified with the `$INFOPATH` environment variable. This is a colon (`:`) or semi−colon (`;`) separated list of directory paths which specify the locations of the info files, for UNIX and Microsoft DOS/Windows environment's, respectively.

info−on gets info on a user specified top level topic, e.g. "`gcc`", the info file "*topic−str*`.info`" must be found in the info search path.

info−goto−link gets and displays info on a user specified link or subject. The link may be within the currently displayed topic (the *link−str* need only specify the subject node name) or a subject within another topic (in which case the *link−str* takes the following form "`(`*topic*`)` *subject*").

NOTES

info is a macro implemented in file `info.emf`.

When an **info** command is run for the first time, the info search path is constructed and stored locally in the command variable **.info.path**. This variable must be directly changed by the user if changes to the info search path are required.

SEE ALSO

info(9).

$MENAME(5)

NAME

$MENAME − MicroEmacs user name
$LOGNAME − System user name (UNIX)

SYNOPSIS

$MENAME *string*; Default is `guest`

$LOGNAME *string*

DESCRIPTION

$MENAME is an environment variable used to initialize the MicroEmacs '02 environment for a given user. At start−up, if **$MENAME** is defined then the user's configuration and history file "`name`.*erf*" is located and read, where `name` is the variable value.

If at start−up **$MENAME** is not defined then **$MENAME** is assigned the value of **$LOGNAME**, if **$LOGNAME** is not defined the file `default.emf` is located and executed. This macro file is created by user−setup(3) to set **$MENAME** to the default user. If this fails then **$MENAME** defaults to `guest` and a default configuration is used.

The user configuration and history file has many uses, see user−setup(3) and read−history(2) for more information.

Microsoft Windows Environments

Within Microsoft Windows environments, if *login* is enabled then the users login name is automatically used as the first choice login name. No environment variables need to be set. If login is not enabled then one of the aforementioned methods should be used.

UNIX

In UNIX environments, **$LOGNAME** is typically defined.

NOTES

The three variables must be defined before start−up for them to have any effect.

$LOGNAME is often defined by the system and should not be altered. If a different user name is required, setting of **$MENAME** is preferable.

SEE ALSO

user−setup(3), read−history(2), $MEPATH(5).

$buffer−backup(5)

NAME

$buffer−backup − Buffer backup file name

SYNOPSIS

$buffer−backup *FileName*

DESCRIPTION

$buffer−backup is automatically set to the file name the current buffer's file would be backed up to if required. If the current buffer has no file name the variable will be set to "".

The value depends on whether DOS compliant file names are being used (see $system(5)), whether multiple backups are being kept (see $kept−versions(5)) and the setting of the environment variables **$MEBACKUPPATH** and **$MEBACKUPSUB**. The variable does not take into consideration the current setting of the buffer's backup(2m) mode which determine whether a backup will be made.

The environment variable **$MEBACKUPPATH** can be used to change the location of the backup files, it can also be used to prepend the backup filename with a string. **$MEBACKUPPATH** can specify an absolute path (e.g. "`c:/temp/mebackup/`") or a relative path (e.g. "`mebackup/`" which will move all backup files into a sub−directory automatically in the files directory).

The trailing '`/`' is important as the file name is simple appended, i.e. is creating a backup for "c:/foo/bar.txt" and $MEBACKUPPATH is set the "backup" the backup file name will be "c:/foo/backupbar.txt".

The environment variable **$MEBACKUPSUB** can be used to substitute strings within the backup filename for another. The format of the value is a list of **sed(1)** string substitutions, i.e.

```
$MEBACKUPSUB="s/from1/to1/ s/from2/to2/ s/fr..."
```

The 3 divide characters do not have to be '/'s, they can be any character as long as they are the same, e.g. "sXfrom1Xto1X". When define MicroEmacs performs a simple search for string "`from1`" (i.e. no regex support) and replaces any match with the string "`to1`" etc.

EXAMPLE

The following example compares the differences between the current version and the bucked up version using the diff(3) macro. The **diff−changes** macro is defined in `tools.emf`.

```
define−macro diff−changes
    !if &seq $buffer−fname ""
```

```
                ml-write "[Current buffer has no file name]"
                !abort
            !endif
            !if &bmod "edit"
                !if &iseq @mc1 "Save buffer first [y/n]? " "nNyY" "y"
                    save-buffer
                !endif
            !endif
            ; get the real file name - this only has effect on unix, copes with symbolic l
            set-variable #l0 &stat "a" $buffer-fname
            ; get the backup name
            set-variable #l1 $buffer-backup
            diff #l1 #l0
        !emacro
```

NOTES

The variable **$buffer−backup** can not be set, any attempt to set it will result in an error.

On Windows and DOS platforms if the $MEBACKUPPATH and $MEBACKUPSUB variables are used all remaining '`:`' characters are changed to '`/`'s as these are illegal in the middle of a filename.

SEE ALSO

backup(2m), $system(5), $kept−versions(5).

$search−path(5)

NAME

$search−path − MicroEmacs search path $MEPATH − MicroEmacs search path

SYNOPSIS

$search−path *string*

[Microsoft Windows/MS−DOS]
MEPATH= *<path1>***;***<path2>***;....;***<pathn>*

[UNIX]
MEPATH= *<path1>***:***<path2>***:....:***<pathn>*

DESCRIPTION

$search−path is initialized to the environment variable **$MEPATH**, and identifies the search paths which are searched to locate editor specific files. Multiple search paths may be specified, separated by the platform path separator (semi−colon ('**;**') on Microsoft Windows or MS−DOS environments and a colon ('**:**') on UNIX environments). Where multiple search paths are defined then they are search left to right.

The search paths are generally ordered from highest priority to lowest priority and might be arranged such as:−

MEPATH=*<user>*:*<company>*:*<me>*

where *<user>* represents the users path; *<company>* is the company file path (e.g. template files) and *<me>* are the standard MicroEmacs '02 files.

This would correspond to a directory installation, of user **foo** such as:−

```
/usr/foo/microemacs   - User files.
/usr/group/microemacs - Company wide files
/usr/local/microemacs - MicroEmacs installation directory
```

and a **$MEPATH** such as:−

```
MEPATH=/usr/foo/microemacs:/usr/group/microemacs:/usr/local/microemacs
```

USAGE

The current working directory is checked first for the location of a file.

$search−path is used to locate all macro files, and other files located with operators such as &find(4).

NOTES

If **$MEPATH** is not set then **$search−path** is initialized to the environment variable **$PATH**.

On UNIX systems the path */usr/local/microemacs* is automatically added to the end of **$MEPATH**, or if not defined, to the beginning of **$PATH**.

SEE ALSO

Variable Functions, execute−file(2), $MENAME(5), &find(4).

ishell(3)

NAME

ishell − Open a interactive shell window
$ME_ISHELL − Windows ishell command comspec

PLATFORM

Windows '95/'98/NT − win32
Unix − All variants.

SYNOPSIS

ishell

[Windows Only]
$ME_ISHELL = <*comspec*>

DESCRIPTION

ishell creates an interactive shell window within the a MicroEmacs buffer window, providing access to the native operating systems command shell. Within the window commands may be entered and executed, the results are shown in the window.

On running **ishell** a new buffer is created called `*shell*` which contains the shell. Executing the command again creates a new shell window called `*shell1*`, and so on. If a `*shell*` window is killed off then the available window is used next time the command is run.

Additional controls are available within the shell window to control the editors interaction with the window. The operating mode is shown as a digit on the buffer mode line, this should typically show "3", which corresponds to *F3*. The operating modes are mapped to keys as follows:−

F2

Locks the window and allows local editing to be performed. All commands entered into the window are interpreted by the editors. **F2** mode is typically entered to cut and paste from the window, search for text strings etc. In mode 2, a **2** is shown on the mode line.

F3

The normal operating mode, text typed into the window is presented to the shell window. Translation of MicroEmacs commands (i.e. beginning−of−word) are translated and passed to the shell. For interactive use this is the default mode. In mode 3, a **3** is shown on the mode line.

F4

All input is passed to the shell, no MicroEmacs commands are interpreted and keys are passed straight to the shell window. This mode is used where none of the keys to be entered are to be interpreted by MicroEmacs. Note that you have to un−toggle the F4 mode before you can swap buffers as this effectively locks the editor into the window.

F5

Clears the buffer contents. This simply erases all of the historical information in the buffer. The operation of the shell is unaffected.

To exit the shell then end the shell session using the normal exit command i.e. "`exit`" or "`C−d`" as normal and then close the buffer. A short cut "`C−c C−k`" is available to kill off the pipe. However, it is not recommended that this method is used as it effectively performs a hard kill of the buffer and attached process

UNIX

The UNIX environment uses the native **pty** support of the operating system. The shell that is opened is determined by the conventional `$SHELL` environment variable.

The shell window assumes that the user is running some sort of Emacs emulation on the command line (i.e. `VISUAL=emacs` for **ksh(1)**, **zsh(1)**, **bash(1)**, **tsch(1)**) and passes Emacs controls for command line editing.

The shell window understands re−size operations and provides a limited decoding of the *termio* characters for a VT100 screen. From within the shell window it is possible to run the likes of **top(1)** correctly. It is even possible to run another MicroEmacs terminal session !!

WINDOWS

The Windows environment provides a very poor command shell facility, this is more of a fundamental problem with the operating system than anything else. Unfortunately NT is no better than Windows '95/'98, stemming from the fact that the Windows is not actually an O/S but a huge window manager, hindered by legacy issues of MS−DOS.

For those familiar with the UNIX command shell then it is strongly recommended that the cygnus(3) BASH shell is used as an alternative. This is a far more responsive shell window and provides the familiar Emacs editing of the command line.

The command shell under Windows is slow and very unresponsive, this would appear to be a problem with the *command.com* as the same problems are not apparent with the cygwin environment. However, the shell window is good for kicking off command line utilities (such as *make*), or any command line processes that generate output on *stdout* as all of the output is captured in the buffer window which can be scrolled backwards for post analysis. For this very reason it is more preferable to the standard MS−DOS box.

It is not possible to run any utilities that use embedded screen control characters as these are not interpreted by the editor.

Changing the Shell

The default shell that is executed is defined by the environment variable **$COMSPEC**. Where the user is using a different command shell (i.e. 4−DOS), then problems may arise if this is an old 16−bit executable. The shell that MicroEmacs executes may be overridden by setting the environment variable **$ME_ISHELL**. This is typically set in the me32.ini(8) file i.e.

```
[username]
ME_ISHELL=c:\windows\command.com
```

Bugs

WinOldAp

Winoldap is created by the Microsoft environment whenever a shell is created. On occasions where processes have terminated badly the user may be prompted to kill these off; this is the normal behaviour of Windows. It is strongly advised that the shell is always exited correctly (i.e. `exit`) before leaving the editor. The Windows operating system for '95/'98 is not particularly resilient to erroneous processes can bring the whole system down. I believe that NT does not suffer from these problems (much).

Locked Input

There are occasions after killing a process the editor appears to lock up. This is typically a case that the old application has not shut down correctly. Kill off the erroneous task (`Alt−Ctrl−Del` − *End Task*) then bring the editor under control using a few `C−g` abort−command(2) sequences. **NOTES**

The **ishell** command uses the ipipe−shell−command(2) to manage the pipe between the editor and the shell. The window is controlled by the macro file `hkipipe.emf` which controls the interaction with the shell.

SEE ALSO

ipipe−shell−command(2), cygnus(3), me32.ini(8).

pipe−shell−command(2)

NAME

pipe−shell−command − Execute a single operating system command
$ME_PIPE_STDERR − Command line diversion to stderr symbol

SYNOPSIS

n **pipe−shell−command** "*command*" ["*buffer−name*"] (**esc @**)

[MS−DOS and Win32s Only]
$ME_PIPE_STDERR "*string*"; Default is undefined.

DESCRIPTION

pipe−shell−command executes one operating system command *command* and pipes the resulting output into a buffer with the name of ***command***.

The argument *n* can be used to change the default behavior of pipe−shell−command described above, *n* is a bit based flag where:−

0x01

Enables the use of the default buffer name ***command*** (default). If this bit is clear the user must supply a buffer name. This enables another command to be started without effecting any other command buffer.

0x02

Hides the output buffer, default action pops up a window and displays the output buffer in the new window.

0x04

Disable the use of the command−line processor to launch the program (win32 versions only). By default the "**command**" is launched by executing the command:

```
%COMSPEC% /c command
```

Where `%COMSPEC%` is typically command.com. If this bit is set, the "**command**" is launched directly.

0x08

Detach the launched process from MicroEmacs (win32 versions only). By default the command is launched as a child process of MicroEmacs with a new console. With this bit set the process is completely detached from MicroEmacs instead.

0x10

Disable the command name mangling (win32 versions only). By default any '`/`' characters found in the command name (the first argument only) are converted to '`\`' characters to make it Windows compliant.

NOTES

On MS−DOS and *Win32s* the standard shell **command.com(1)** does not support the piping of *stderr* to a file. Other shells, such as **4Dos.com(1)**, do, using the command−line argument "`>&`". If the environment variable "`ME_PIPE_STDERR`" is defined (the value is not used) then MicroEmacs assumes that the current shell supports piping of stderr.

SEE ALSO

ipipe−shell−command(2), shell−command(2).

$auto−time(5)

NAME

$auto−time − Automatic buffer save time

SYNOPSIS

$auto−time *seconds*; Default is `300` seconds

0 <= *seconds* <= t

DESCRIPTION

Sets the number of seconds to wait until an edited buffer is auto−saved to temporary file to t seconds. A setting of 0 disables the auto−saving command. Auto−saving can be enabled and disabled on a per buffer basis using buffer mode autosv(2m).

The auto−save file naming convention is the same as the backup name only using hash ('#') instead of tilde ('~') and is automatically removed on saving a buffer.

On unlimited length file name systems (UNIX), the following file naming conventions are used for file `xxxxx`:

```
xxxxx -> xxxxx#
```

On systems with an `xxxxxxxx.yyy` file name (DOS etc), the following file naming conventions are used:

```
xxxxxxxx     -> xxxxxxxx.###
xxxxxxxx.y   -> xxxxxxxx.y##
xxxxxxxx.yy  -> xxxxxxxx.yy#
xxxxxxxx.yyy -> xxxxxxxx.yy#
```

NOTES

The user is warned to be extra careful if files ending in '~' or '#'s are used, it is advisable to disable backup creation (see global−mode(2)) and auto−saving (`$auto-time = 0`). The author denies all responsibility (yet again) for any loss of data! Please be careful.

Auto−save files of URL files (i.e. "`ftp://...`" and "`http://...`") are written to the system's temporary directory. This avoids potentially slow auto−saves. This can however lead to recovery problems as the buffer name must be used to avoid auto−saving conflict with other buffers with the same base file name but different paths.

SEE ALSO

autosv(2m), backup(2m), buffer−mode(2) find−file(2), ftp(3).

$box−chars(5)

NAME

$box−chars − Characters used to draw lines

SYNOPSIS

$box−chars "*string*"; Default is "|++++++++++−"

DESCRIPTION

$box−chars is a fixed length string that defines the set of characters used to render lines to the screen. Osd(2), directory−tree(2), list−registry(2) and many macros use these characters as a platform independent method of drawing lines. The characters have fixed indices defined as follows:−

Index 0

Line joining north to south (vertical line).

Index 1

Line joining south to east.

Index 2

Line joining south to west.

Index 3

Line joining north to east.

Index 4

Line joining north to west.

Index 5

Line joining east to south to west.

Index 6

Line joining north to east to south.

Index 7

Line joining north to east to south to west.

Index 8

Line joining north to south to west.

Index 9

Line joining north to east to south.

Index 10

Line joining east to west. **EXAMPLE**

The **$box−chars** is typically platform dependent, it's setting is determined by the characters available in character set of the hosting platform. MS−DOS and Microsoft Windows environments might use a string such as:−

```
"\xB3\xDA\xBF\xC0\xD9\xC2\xC3\xC5\xB4\xC1\xC4"
```

X−Windows environments might use a string such as:−

```
"\x19\x0D\x0C\x0E\x0B\x18\x15\x0F\x16\x17\x12"
```

Both utilize platform specific characters.

SEE ALSO

Osd(2), directory−tree(2), list−registry(2) $window−chars(5).

$buffer−fhook(5)

NAME

$buffer−fhook − Buffer macro hook command name (buffer creation)
$buffer−dhook − Buffer macro hook command name (buffer deletion)
$buffer−bhook − Buffer macro hook command name (buffer current)
$buffer−ehook − Buffer macro hook command name (buffer swapped)

SYNOPSIS

$buffer−fhook *FunctionName*
$buffer−dhook *FunctionName*
$buffer−bhook *FunctionName*
$buffer−ehook *FunctionName*

DESCRIPTION

Sets the buffer create, delete, begin and end hook command which are executed:

buffer−fhook

When the buffer is created.

buffer−dhook

When the buffer is deleted.

buffer−bhook

When the buffer becomes the current buffer.

buffer−ehook

When the buffer is swapped out from being the current buffer.

The variable **$buffer−fhook** is largely redundant as the file hook is executed only once and before it can be sent. Its main use is within macros which wish to ascertain what type of buffer it is executing on, i.e. if a command was to be executed only on c file then the follow ensures that this is the case:

```
!if &not &seq $buffer−fhook "fhook−cmode"
    !abort
!endif
```

Where the command *fhook−cmode* is the c file hook.

dhooks are executed when a buffer is deleted, but before the contents of the buffer are lost. Note that dhooks will not be called if the buffer never becomes active, or if MicroEmacs '02 quits due to the receipt of a panic signal.

bhooks and **ehooks** are usually used to set and restore global variables which require different setting in the current buffer.

The order of The default settings of these variable are determined by the command add−file−hook(2).

SEE ALSO

add−file−hook(2).

$buffer−bname(5)

NAME

$buffer−bname − Name of the current buffer
$buffer−fname − Name of the current buffer's file name

SYNOPSIS

$buffer−bname *BufferName*
$buffer−fname *FileName*

DESCRIPTION

$buffer−bname the string name of the current buffer. Buffer names are unrestricted in length, but must be unique. By default the buffer name is derived from the buffer's file name without the path. But this can lead to conflicts, caused by identical file names but different paths. In these situations a counter is appended to the end of the buffer name and is incremented until a unique buffer name is created. For example:

```
File Name                Buffer Name
_______________________________________

/etc/file.c              file.c
/tmp/file.c              file.c<1>
/usr/file.c              file.c<2>
```

$buffer−fname contains the name of the current buffer's file name complete with path.

SEE ALSO

change−buffer−name(2).

$buffer−fmod(5)

NAME

$buffer−fmod − Buffer file modes (or attributes)
$global−fmod − Global file modes (or attributes)

SYNOPSIS

$buffer−fmod *FileMode*
$global−fmod *FileMode*

DESCRIPTION

$buffer−fmod is bit based variable setting the buffers file system modes or attributes. If the buffer was loaded from an existing file then the value of **$buffer−fmod** is taken directly from the file. But if the buffer was created then the buffer inherits the default file modes, **$global−fmod**, which is determined from the users umask on UNIX or a default on others.

The definition of the file mode bits are platform specific and are considered independently, as follows:

UNIX

The file modes of Unix are the standard read, write and execute permissions for user, group and global. See **chmod(1)** for a full description of their use and effect.

The variable is displayed in octal.

Microsoft Windows and DOS

On Microsoft platforms each file attribute (see **attrib(1)**) is assigned a bit, on windows 95 and NT the new file attributes such as compressed are also represented. The bits are assigned as follows

Bit	Attrib Flag	Attribute
0x001	R	Read Only
0x002	H	Hidden
0x004	S	System
0x010		Directory
0x020	A	Archive
0x080		Normal
0x100		Temporary
0x800		Compressed

EXAMPLE

The following example changes the $buffer−fmod so that the file will be executable (UNIX only), useful when writing a shell script.

```
set−variable $buffer−fmod 0775
```

SEE ALSO

crlf(2m), ctrlz(2m), auto(2m).

$buffer−hilight(5)

NAME

$buffer−hilight − Define current buffer hilighting scheme.

SYNOPSIS

$buffer−hilight *hilightNum*; Default is `0`

0 <= *hilightNum* <= 255

DESCRIPTION

$buffer−hilight Sets the current buffer's hi−lighting scheme (see hilight(2) for a full description of hi−lighting). The default setting is 0 which specifies no hi−lighting, when set to a non−zero, the hi−light scheme of that number MUST already be defined.

Terminals that cannot display color directly may still be able to take benefit from hi−lighting. A terminal that has fonts can use them in the same way using the add−color−scheme(2) command. The hi−light scheme is also used in printing (see print−buffer(2)). If, however, your terminal cannot display color in any way, it is recommended that hi−lighting is disabled (except when printing) as it does take CPU time.

SEE ALSO

hilight(2), print−buffer(2), $buffer−scheme(5), $buffer−indent(5).

$buffer−indent(5)

NAME

$buffer−indent − Current buffer indentation scheme.

SYNOPSIS

$buffer−indent *indentNum*; Default is `0`

0 <= *indentNum* <= 255

DESCRIPTION

$buffer−indent sets the current buffers indentation scheme. *indentNum* is the identity of the indentation scheme, as defined by indent(2), which is typically the same value as the buffers hilighting scheme number (see $buffer−hilight(5)).

The default setting is 0 which specifies no indentation scheme is present (with the exception of cmode(2m)). When non−zero, the value identifies the indentation scheme.

A buffer assigned an indentation method, MicroEmacs performs automatic line re−styling, by moving the left indentation, according to the defined indentation method. The `tab` key is typically disabled. This behavior can be altered using bit `0x1000` of the $system(5) variable, which can be changed using user−setup(3).

The use of tab characters to create the required indentation is determined by the setting of the buffers tab(2m) mode. If the mode is disabled tab characters are used wherever possible, otherwise spaces are always used.

NOTES

The commands restyle−region(3) and restyle−buffer(3) use the indentation method when defined.

The buffer indentation scheme is typically assigned in the *fhook* macro, see Language Templates.

EXAMPLE

The following example sets up an indentation scheme for a buffer within the *fhook* macro.

```
!if &sequal .hilight.foo "ERROR"
    set-variable .hilight.foo &pinc .hilight.next 1
!endif
```

```
....

; Define the indentation scheme
0 indent  .hilight.foo 2 10
indent .hilight.foo n "then" 4
indent .hilight.foo s "else" -4
indent .hilight.foo o "endif" -4

....

; File hook - called when new file is loaded.
define-macro fhook-foo
    ; if arg is 0 this is a new file so add template
    !if &not @#
        etfinsrt "foo"
    !endif
    ; Assign the hilighting
    set-variable $buffer-hilight .hilight.foo
    ; Assign the buffer indentation
    set-variable $buffer-indent .hilight.foo
    ; Set the abbreviation file
    buffer-abbrev-file "foo"
    ; Temporary comment to make sure that it works.
    ml-write "Loaded a foo file"
!emacro
```

This provides an indentation of the form:−

```
if condition
then
    XXXX
else
    if condition
    then
        XXXX
    endif
endif
```

SEE ALSO

indent(2), tab(2m), $system(5), user−setup(3), restyle−buffer(3), restyle−region(3), $buffer−hilight(5).

$buffer−input(5)

NAME

$buffer−input − Divert buffer input through macro.

SYNOPSIS

$buffer−input *commandName*

DESCRIPTION

$buffer−input allows the buffer input mechanism to be diverted through a command macro defined by *commandName*. If this variable is set to a valid command, which may be a user defined macro, this command will be called instead. The command can access the actual key−code typed by the user via the command variable @cc(4), e.g. the following macro prints out the name of the command that the user presses until the abort−command(2) is executed.

```
define−macro test−input
    ml−write &spr "Current command: %s" @cc
    !if &seq @cc "abort−command"
        set−variable $buffer−input ""
    !endif
!emacro

set−variable $buffer−input test−input
```

WARNING

Caution is advised when using this, if there is no way of reseting the variable then **MicroEmacs '02** must be killed.

SEE ALSO

abort−command(2), @cc(4).

$buffer−ipipe(5)

NAME

$buffer−input − Divert buffer incremental pipe input through macro.

SYNOPSIS

$buffer−ipipe *commandName*

DESCRIPTION

$buffer−ipipe allows the buffer incremental pipe input mechanism to be diverted through a command macro defined by *commandName*. On a buffer running an ipipe−shell−command(2) the command, set by this variable, will be called whenever new text has been inserted by the executing process. Two *alpha−marks* will be set in the buffer, '`i`' denotes the start of the newly inserted text and '`I`' denotes the end.

SEE ALSO

goto−alpha−mark(2), ipipe−shell−command(2).

$buffer−mask(5)

NAME

$buffer−mask − Current buffer word class mask.

SYNOPSIS

$buffer−mask string; Default is `luh`

DESCRIPTION

$buffer−mask sets the current buffer word class mask. MicroEmacs '02 has an internal word lookup table which defines whether a given letter is considered to be part of a word. This functionality is used in many areas such as forward−word(2), forward−kill−word(2) hilighting etc. The mask is composed with any combination of the following flags, the order in which the flags are specified is not important:

l

All lower case letters.

u

All upper case letters.

h

All hexadecimal characters (used to include numerical digits).

s

Spell extended characters, typically set to accent (`'`), hyphen (`−`) and period (`.`).

1

User set **1**, usually set to just underscore (_) for many system and programming files such as 'C'.

2

User set **2**, usually set to '`−`', '$', '&', '#', '`!`', '%', '`:`' and '@' for MicroEmacs files.

3

User set **3**, not usually defined.

4

User set **4**, not usually defined.

The character sets may be modified using the set−char−mask(2) command.

SEE ALSO

set−char−mask(2), forward−word(2).

$buffer−mode−line(5)

NAME

$buffer−mode−line − Buffer mode line string

SYNOPSIS

$buffer−mode−line "*string*"

DESCRIPTION

Sets the buffer mode line, unique to this buffer, see $mode−line(5) use, description and syntax. If this variable is NOT set for a buffer and **$mode−line** is changed, then the buffer's mode line will also change to the new value. If this variable is set, then then buffer's mode line will be unaffected by any setting of **$mode−line**.

SEE ALSO

$mode−line(5).

$buffer−names(5)

NAME

$buffer−names − Filtered buffer name list

SYNOPSIS

$buffer−names *BufferName*

DESCRIPTION

$buffer−names must first be set to the required filter string, if the variable is evaluated before it is initialized the value will be set to "*ABORT*" and the command will fail. The filter takes the form of a regex.

Once initialized, evaluating **$buffer−names** returns the name of the next buffer which matches the filter until no more buffers are found, in which case an empty string is returned.

EXAMPLE

The following example prints out the name of all buffers to the massage line one at a time. Note that &set(4) is used on the !while(4) statement to avoid evaluating **$buffer−names** twice per loop.

```
set-variable $buffer-names ".*"
!while &not &seq &set #l0 $buffer-names ""
    100 ml-write &cat "buffer: " #l0
!done
```

The following example is the same except it lists only the buffers which are not directory listings

```
set-variable $buffer-names ".*[^/]"
!while &not &seq &set #l0 $buffer-names ""
    100 ml-write &cat "buffer: " #l0
!done
```

NOTES

The list of buffers is evaluated when the variable is initialized, buffers created after the initialization will not be included in the list.

Deleting buffers which are in the list, before they are evaluated, will have undefined effects.

SEE ALSO

list−buffers(2), $buffer−bname(5), $file−names(5), $command−names(5), $mode−names(5), Regular Expressions.

$buffer−scheme(5)

NAME

$buffer−scheme − Buffer color scheme.

SYNOPSIS

$buffer−scheme *schemeNum*; Default is `0`

DESCRIPTION

$buffer−scheme sets the current buffer's color scheme to *schemeNum*, where *schemeNum* is a color scheme defined with add−color−scheme(2), which identifies the foreground and background color schemes of the buffer. The color scheme is initialized to the global color scheme settings (see $global−scheme(5)) when the buffer is created.

SEE ALSO

$buffer−hilight(5), $cursor−color(5), $trunc−scheme(5), $global−scheme(5), $ml−scheme(5), $mode−line−scheme(5), $scroll−bar−scheme(5), $system(5).

$c−brace(5)

NAME

$c−brace − C−mode; brace indentation

SYNOPSIS

$c−brace *integer*; Default is `−4`

−n <= integer <= n

DESCRIPTION

$c−brace is part of the cmode(2m) environment for C programmers.

Sets the indent of a '`{`' and a '`}`' on a new line, from the current indent. For example, using the default settings, if the current indent was 20 then a line starting with a '`{`' or a '`}`' would be indented to 16, i.e.

```
    xxxxxxxxxxx
    xxxxxxxxxxx
{ xxxxxxxxxxx
    xxxxxxxxxxx
} xxxxxxxxxxx
    xxxxxxxxxxx
```

This may seem strange, but the current indent is the indent of the last '`{`' (or "`if`", "`else`" etc.) plus $c−statement(5) which is 4, so this brings it back into line with '`{`"s, "`if`"'s and "`else`"'s etc., e.g.

```
if(xxxxxx)
{
    xxxxxxxxxx
    xxxxxxxxxx
}
```

With a setting of −2, this would become:−

```
if(xxxxxx)
  {
    xxxxxxxxxx
    xxxxxxxxxx
  }
```

This works in conjunction with $c−statement(5), a change to **$c−statement** will change the position of '`{`'s.

SEE ALSO

cmode(2m), $c−statement(5).

$c−case(5)

NAME

$c−case − C−mode; case indentation
$c−switch − C−mode; switch indentation

SYNOPSIS

$c−case *integer*; Default is `−4`
−n <= integer <= n

$c−switch *integer*; Default is `0`
−n <= integer <= n

DESCRIPTION

$c−case and **$c−switch** are part of the cmode(2m) environment for C programmers.

$c−switch sets the offset of a "`case`" entry statement from the opening brace left margin position. The default value is zero. e.g.

```
switch(xxxxxxxxx)
{
case 1:
    xxxxxxxxxx
    xxxxxxxxxx
case 2:
    xxxxxxxxxx
}
```

Setting the value to 4, increases the leading space on the "`case`" statement, e.g.

```
switch(xxxxxxxxx)
{
    case 1:
        xxxxxxxxxx
        xxxxxxxxxx
    case 2:
        xxxxxxxxxx
}
```

$c−case sets the offset of the lines following a "`case`" statement, from the current indent. For example, using the default settings, if the current indent was 20 then a line starting with a "`case`" would be indented to 16, i.e.

```
    xxxxxxxxxx
case xxxxxxxxxx
    xxxxxxxxxx
```

This is used inside "`switch`" statements, the default setting give the following lay−out:−

```
switch(xxxxxxxxxx)
{
case 1:
    xxxxxxxxxx
    xxxxxxxxxx
case 2:
```

This works in conjunction with the $c−statement(5), a change to **$c−statement** will change the position of '`{`'s.

SEE ALSO

cmode(2m), $c−statement(5).

$c−contcomm(5)

NAME

$c−case − C−mode; comment continuation string

SYNOPSIS

$c−contcomm "*string*"

DESCRIPTION

$c−contcomm is part of the cmode(2m) environment for C programmers.

This defines the string which is inserted when a new line is started while in a comment. The string is only inserted if the cursor is at the end of the line when the newline(2) command is given. For example, for the default settings, if a **newline** was entered at the end of the first line, the second line would initialize to:−

```
/* xxxxxxxxxx
   @
```

where '@' is the current cursor position. With a setting of " * ", then:−

```
/* xxxxxxxxxx
 * @
```

SEE ALSO

cmode(2m).

$c−continue(5)

NAME

$c−continue − C−mode; line continuation indent
$c−contmax − C−mode; line continuation maximum indent

SYNOPSIS

$c−continue *integer*; Default is `10`
−n <= *integer* <= *n*

$c−contmax *integer*; Default is `16`
−n <= *integer* <= *n*

DESCRIPTION

$c−continue and **$c−contmax** are part of the cmode(2m) environment for C programmers.

$c−continue sets the indent to be added to a split line, i.e. for an indent of 20, a continued statement would be indented to 30. A continued statement is a single c statement which is spread over 2 or more lines, the 2nd and any following lines would be indented to 30. For example

```
          thisIsAVeryLongVariableWhichMeansAssignmentsAreSplit =
                    ThisIsTheFirstContinuedStatementLine +
                    ThisIsTheSecondContinuedStatementLine + etc ;
```

The indent is changed if there is an open bracket, continued statements are indented to the depth of the open bracket plus one, e.g.

```
          func(firstFuncArg,
               secondFuncArg,
               anotherBracketForFun(firstAnotherBracketForFunArg,
                                    secondAnotherBracketForFunArg),
               thirdFuncArg) ;
```

$c−contmax sets an upper limit of the indentation where an open bracket is encountered, in the case where the leading indent of the function name and open bracket exceeds **$c−contmax**, then the continuation is reduced to the continuation indent.

The effect of **$c−contmax** is described as follows; if **$c−contmax** is set to a large value then the default open brace offset appearence is:−

```
    longVariable = LongFunctionNameWhichMeans(isSoFar,
                                              OverAndYouRunOutOfRoom) ;
```

Setting **$c−contmax** to 16 gives:

```
        longVariable = LongFunctionNameWhichMeans(isSoFar,
                       overAndYouRunOutOfRoom) ;
```

Where by the second argument indent has been artificially reduced because of it's length.

SEE ALSO

cmode(2m).

$c−margin(5)

NAME

$c−margin − C−mode; trailing comment margin

SYNOPSIS

$c−margin *integer*; Default is `−1`

−1 <= *integer* <= *n*

DESCRIPTION

$c−margin is part of the cmode(2m) environment for C programmers.

If inserting a comment at the end of a C line, it is tedious typing *x* number of spaces to the comment column (by default tab doesn't insert a tab when cmode(2m) is enabled, it reformats the indentation of the line regardless of the cursor position). This variable sets the indent column of these comments. So with the default settings and the following line,

```
xxxxxx ;/
```

when a '*' is type the line becomes

```
xxxxxx ;                /*
```

The indenting of the "`/*`" occurs only if there is text on the line before it, and none after it. If the current column is already past **$c−margin** then it is indented to the next tab stop.

A value of −1 disables this feature.

SEE ALSO

cmode(2m).

$c−statement(5)

NAME

$c−statement − C−mode; statement indentation

SYNOPSIS

$c−statement *integer*; Default is 4

−n <= integer <= n

DESCRIPTION

$c−statement is part of the cmode(2m) environment for C programmers.

The indent of the current line is derived from **$c−statement** plus the indent of the last c token (*if*, *else*, *while* etc.) or the last '`{`' (which ever was found first). i.e. if the last '`{`' was found at column 16 then the current line will be indented to 20:−

```
                {
                    xxxxxxxxxx
                    xxxxxxxxxx
```

or

```
                if(xxxxx)
                    xxxxxxxxxx
```

C tokens are only used to indent the next line, whereas '`{`' are used in indenting every line to it's partnering '`}`'.

SEE ALSO

cmode(2m).

$command−names(5)

NAME

$command−names − Filtered command name list

SYNOPSIS

$command−names *CommandName*

DESCRIPTION

$command−names must first be initialized to the required filter string, if the variable is evaluated before it is initialized the value will be set to "*ABORT*" and the command will fail. The filter takes the form of a regex.

Once initialized, evaluating **$command−names** returns the name of the next command which matches the filter until no more commands are found, in which case an empty string is returned.

EXAMPLE

The following example prints out the name of all commands to the massage line one at a time. Note that &set(4) is used on the !while(4) statement to avoid evaluating **$command−names** twice per loop.

```
set-variable $command-names ".*"
!while &not &seq &set #l0 $command-names ""
    100 ml-write &cat "command: " #l0
!done
```

The following example is an alternative implementation of command−apropos(2).

```
define-macro alt-commad-apropos
    set-variable #l1 @ml "Apropos string"
    set-variable $command-names &cat &cat ".*" #l1 ".*"
    !force 0 delete-buffer "*commands*"
    1 popup-window "*commands*"
    !while &not &seq &set #l0 $command-names ""
        insert-string &spr "    %s\n" #l0
    !done
    beginning-of-buffer
    -1 buffer-mode "edit"
    1 buffer-mode "view"
!emacro
```

NOTES

$command−names does not differentiate between built in commands and macros.

The list of commands is evaluated when the variable is initialized, macros created after the initialization will not be included in the list.

SEE ALSO

list−commands(2), command−apropos(2), $buffer−names(5), $file−names(5), $mode−names(5), $variable−names(5), Regular Expressions.

$cursor−blink(5)

NAME

$cursor−blink − Cursor blink rate $cursor−color − Cursor foreground color

SYNOPSIS

$cursor−blink integer; Default is `0`

$cursor−color colorNum; Default is `0`

0 <= *colorNum* <= *n*

DESCRIPTION

$cursor−blink sets the cursor's flash rate, i.e. the period in which the cursor is drawn, hidden and then redrawn. The default setting of 0 disables cursor blinking. When set to a none zero value the variable is split into two componants, the first 16 bits, or lower short, sets the cursor visible time in milliseconds, and the higher short sets the hidden time. If the hidden time is set to 0 then the cursor will be hidden for the same length of time it is visible.

The cursor blink rate can be setup in the platform section of user−setup(3).

$cursor−color sets the cursor's fore−ground color, and can greatly improve cursor visibility. *colorNum* is a integer palette number created using add−color(2), the default is 0.

PLATFORM

UNIX termcap interface does not support **$cursor−color**.

EXAMPLE

The following example sets the cursor visible time to 600 ms (0x258) and a hidden time to 200 ms (0xc8):

```
set-variable $cusror-blink 0x00c80258
```

SEE ALSO

user−setup(3), add−color(2), $global−scheme(5), $ml−scheme(5), $mode−line−scheme(5), $system(5).

$cursor−x(5)

NAME

$cursor−x − Cursor X (horizontal) position
$cursor−y − Cursor Y (vertical) position

SYNOPSIS

$cursor−x *integer*

0 <= *integer* <= $frame−width − 1

$cursor−y *integer*

0 <= *integer* <= $frame−depth − 1

DESCRIPTION

$cursor−x and **$cursor−y** are automatically set to the position of the cursor at the last screen update (i.e. the variables are not updated between screen updates). The top left character of the screen is coordinate 0,0 bottom right is $frame−width, $frame−depth.

NOTES

These variables can not be set. Any attempt to set them will result in an error.

SEE ALSO

$mouse−x(5), $frame−depth(5), $frame−width(5).

$debug(5)

NAME

$debug − Macro debugging flag

SYNOPSIS

$debug *debugLevel*; Default is `0`

−2 <= *debugLevel* <= 2

DESCRIPTION

$debug is a flag to trigger macro debugging. A setting of 1 or 2 enables debugging, 0 disables debugging (default). A **$debug** setting of 2 debugs all macro lines encountered, whereas a setting of 1 debugs only the lines executed, i.e. if a false **!if** was encountered the lines within the **!if** would not be printed. Problems arise with **!elif** and **!else** and a *debugLevel* setting of 1 as the **!elif** and **!else** lines are never printed.

A −ve setting disables debugging and has no immediate effect. However as soon as the bell is rung the value is inverted (−1 to 1, −2 to 2) enabling debugging. This can be invaluable when tracing problems, for example the following macro code will loop infinitely:−

```
!repeat
    beginning-of-line
    backward-char
    !force forward-line
!until &not $status
```

This is a fairly obvious bug, but if buried in a thousand lines of macro code it could be very difficult to spot and to find it during execution would be very tedious if not impossible. But by setting **$debug** to −1 the macro can be executed as normal and as soon as the macro is stuck the user can simply press "`C-g`" (**abort−command**) which rings the bell and starts macro debugging at the current execution point.

SEE ALSO

execute−file(2).

$delay−time(5)

NAME

$delay−time − Mouse time event delay time
$repeat−time − Mouse time event repeat time

SYNOPSIS

$delay−time *milliseconds*; Default is `500`
$repeat−time *milliseconds*; Default is `25`

10 <= *milliseconds* <= t

DESCRIPTION

$delay−time sets the time waited between the user picking a mouse button and the generation of a `mouse−time−?` key event.

When user presses the left button (say) a `mouse−pick−1` key event is generated, If this key is bound then the command it is bound to is executed. If the user then holds down the button for **$delay−time** or more milliseconds then MicroEmacs checks the binding of the special `mouse−time−1` key, if this pseudo key is bound then the command it is bound to will be executed.

If the user continues to hold down the button for a further **$repeat−time** milliseconds another **mouse−time−1** key event will be generated. A **mouse−time−1** key event will be generated after every **$repeat−time** milliseconds until the user releases the button, at which point a `mouse−drop−1` key event is generated.

EXAMPLE

The following example implements the vertical scroll−bar up and down scrolling arrows for a buffer window:−

```
define−macro mouse−pick−command
    set−cursor−to−mouse
    !if &equ &band $mouse−pos 15 5
        ml−write "Mouse on up−arrow"
        1 scroll−up
        1 global−bind−key scroll−up "mouse−time−1"
    !elif &equ &band $mouse−pos 15 9
        ml−write "Mouse on down−arrow"
        1 scroll−down
        1 global−bind−key scroll−down "mouse−time−1"
    !endif
!emacro
```

```
define-macro mouse-drop-command
    !force global-unbind-key "mouse-time-1"
!emacro

global-bind-key mouse-pick-command "mouse-pick-1"
global-bind-key mouse-drop-command "mouse-drop-1"
```

SEE ALSO

$idle−time(5), set−cursor−to−mouse(2), $mouse−pos(5).

$file−ignore(5)

NAME

$file−ignore − File extensions to ignore

SYNOPSIS

$file−ignore "*string*"; Default is ""

DESCRIPTION

$file−ignore specifies a space separated list of file endings which the file completion is to ignore. This is used by any function which prompts the user for a file name, such as find−file(2). A file ending in this case is NOT the extension but the last *n* characters where *n* is the number of characters in the specified ignore file.

EXAMPLE

To ignore all files which have the extension "`o`", using:

```
set-variable $file-ignore "o"
```

would not only ignore "`foo.o`", but also "`foo.oo`", "`foo.po`" and "`foo`" as well as any file that ends in an "`o`". What is really required is

```
set-variable $file-ignore ".o"
```

It is useful to ignore the "`./`" and "`../`" directories so that a directory containing one file will auto−complete to that one file. This is achieved by using:

```
set-variable $file-ignore "./"
```

To ignore MicroEmacs '02 backup files ("~"), C object files ("`.o`"), "`./`" and "`../`" directories try using:

```
set-variable $file-ignore "~ .o ./"
```

NOTES

The file completion only completes further than the first non−unique point in the current list of possibles if and only if it can ignore all but one file, so if the current directory contains:

```
./ ../ foo foo.c foo.c~ foo.o
```

using the above ignore list, completing with "" has no effect as "`foo`" and "`foo.c`" cannot be ignored; completing with "`foo.`" will however complete to "`foo.c`".

SEE ALSO

find−file(2).

$file−names(5)

NAME

$file−names − Filtered file name list

SYNOPSIS

$file−names *FileName*

DESCRIPTION

$file−names must first be initialized to the required filter string, if the variable is evaluated before it is initialized the value will be set to "*ABORT*" and the command will fail.

The filter takes the form of a regex. The filter string should also contain the path to the required directory, the path many not contain wild−cards. If no path is specified the the path of the current buffers file name is taken, if the current buffer has no file name then the current working directory is used.

On initialization, $result(5) is set to the absolute path of the directory being evaluated.

Once initialized, evaluating **$file−names** returns the name of the next buffer which matches the filter until no more buffers are found, in which case an empty string is returned.

EXAMPLE

The following example creates a list of all files in the current directory to a fixed buffer "**files**". Note that &set(4) is used on the !while(4) statement to avoid evaluating **$file−names** twice per loop.

```
set-variable $file-names ".*"
!force 0 delete-buffer "*files*"
1 popup-window "*files*"
insert-string &spr "Directory listing of %s\n\n" $result
!while &not &seq &set #l0 $file-names ""
    insert-string &spr "    %s\n" #l0
!done
beginning-of-buffer
-1 buffer-mode "edit"
1 buffer-mode "view"
```

NOTES

Unlike MS−DOS and Windows systems, to match every file a filter of just "*" is required. A filter of "*.*" only matches file names with a '.' in them.

The list of files is evaluated when the variable is initialized, files created after the initialization will not be included in the list.

SEE ALSO

$result(5), find−file(2), $buffer−fname(5), $buffer−names(5), $command−names(5), $mode−names(5), Regular Expressions.

$file−template(5)

NAME

$file−template − Regular expression file search string

SYNOPSIS

$file−template "*string*"; Default is ""

DESCRIPTION

$file−template defines a regular expression search string used to identify a file in the grep(3) and compile(3) buffers. The format of the string is the same as magic mode search strings (see search−forward(2)).

EXAMPLE

A UNIX file name may be considered to contain any ASCII character except a space or a '`:`' (used as a divider in many programs). Thus **$file−template** should be:

```
set-variable $file-template "[!-9;-z]+"
```

This will correctly identify "`foo.c`" in the following example.

```
foo.c: 45:      printf("hello world\n") ;
```

SEE ALSO

$line−template(5), compile(3), get−next−line(2), grep(3), search−forward(2).

$fill−bullet(5)

NAME

$fill−bullet − Paragraph filling bullet character set
$fill−bullet−len − Paragraph filling bullet search depth

SYNOPSIS

$fill−bullet "*string*"; Default is "`*)].-`"
$fill−bullet−len *length*; Default is `5`

0 <= *length* <= $fill−col

DESCRIPTION

$fill−bullet contains the set of characters which are classified as bullet markers for fill−paragraph(2). If these characters are encountered in the first **$fill−bullet−len** characters of the paragraph AND the character is followed by a `SPACE` or a `tab` character then the user is given the option to indent to the right of the bullet.

$fill−bullet−len determines the maximum depth into the paragraph (in characters) the filling routines should search for a bullet character. The default value is 15. Note that the paragraph starts at the first non−white space character. e.g. to detect "`xviii) `" as a bullet then the bullet length must be set to at least 6 to detect the bullet character "`)`".

EXAMPLE

Examples of filled bullet paragraphs are shown as follows, based on the default **$fill−bullet** character set.

```
a) This is an  example of a  fill-paragraph.  The  closing
   bracket is classified as a bullet character and filling
   optionally takes place to the right of the bullet.

a] Another paragraph

*  A bullet paragraph

1. A numbered paragraph.

item - A dashed bullet.
```

SEE ALSO

$fill−col(5), $fill−ignore(5), $fill−mode(5), fill−paragraph(2), justify(2m).

$fill−col(5)

NAME

$fill−col − Paragraph Mode; right fill column

SYNOPSIS

$fill−col *columnNumber*; Default is `78`

−1 <= *columnNumber* <= 32767

DESCRIPTION

$fill−col defines the current fill column number. *columnNumber* defaults to 78 when undefined. This value is used in conjunction with justify(2m) and wrap(2m) modes.

SEE ALSO

buffer−mode(2), fill−paragraph(2), justify(2m), wrap(2m).

$fill−eos(5)

NAME

$fill−eos − Paragraph filling; end of sentence fill characters
$fill−eos−len − Paragraph filling; end of sentence padding length

SYNOPSIS

$fill−eos "*string*"; Default is "`.!?`"

$fill−eos−len *integer*; Default is `1`
0 <= *integer* <= *n*

DESCRIPTION

$fill−eos defines the end of sentence character set. Sentences ending in these characters are padded with additional *end−of−sentence* spaces, as defined by **$fill−eos−len**.

$fill−eos−len sets the number of spaces inserted after a full stop during paragraph filling. The default is 1 space.

SEE ALSO

fill−paragraph(2).

$fill−ignore(5)

NAME

$fill−ignore − Ignore paragraph filling character(s)

SYNOPSIS

$fill−ignore "*string*"; Default is ">_@"

DESCRIPTION

$fill−ignore describes a set of characters used by fill−paragraph(2) which disable paragraph filling when they appear at the start of a paragraph. An obvious example is an inserted mail message which is usually quoted with ">" characters. Any attempt to fill the paragraph causes **fill−paragraph** to skip to the end of it.

EXAMPLE

This is an example of an ignored paragraph when encountered by **fill−paragraph** with the default ignore character set.

```
> This is an example of a paragraph that
> is ignored.
```

SEE ALSO

$fill−col(5), $fill−bullet(5), $fill−mode(5), fill−paragraph(2), justify(2m).

$fill−mode(5)

NAME

$fill−mode − Paragraph mode; justification method

SYNOPSIS

$fill−mode *justification*; Default is `N`

justification `b` | `c` | `l` | `n` | `o` | `r` | `B` | `C` | `L` | `N` | `R`

DESCRIPTION

$fill−mode defines the justification mode i.e. *left*/*right*/*both*/... The default value is none automatic (**N**). The modes available are:−

b Both

Enables left and right margin justification.

c Center

Enables center justification.

l Left

Enables left justification.

n None

No filling is performed, adjacent lines are not merged into a single line. This subtly different from *left* justification which fills lines to the $fill−col(5).

o One Line

Enables the filling of the paragraph to a single line. Typically used to prepare a file for transfer to a word processing package.

r Right

Enables right justification.

B Both (automatic)

Automatically determines the mode, defaulting to left and right (both) justification.

C Center (automatic)

Automatically determines the mode, defaulting to center justification.

L Left (automatic)

Automatically determines the mode, defaulting to left justification.

N None (automatic)

Automatically determines the mode, defaults to *both* and not *none*.

R Right (automatic)

Automatically determines the mode, defaulting to right justification.

The lines are automatically justified only when the justification mode justify(2m) is enabled. Justification is performed between the left and right margins, defined as 0 and $fill−col(5) respectively.

Automatic Filling

Automatic filling is performed when the mode **$fill−mode** is specified in upper case. The format of the line (and adjacent lines) is interrogated and an *informed* guess is made as to the expected formating which is then adopted. The criteria for automatic formatting is defined as follows:−

center

If the left and right margins contain approximately the same amount of white space +/−1 character then the paragraph is centered.

right

If the text commences past half of the $fill−col(5) (i.e. first half of the line comprises white space) AND the line extends to, or past, the `$fill-col` then the paragraph is assumed to be right justified.

none

If the text commences in column 0 and occupies less than half of the line then the paragraph is assumed to be not justified. (i.e. left justified, but consecutive lines of the paragraph are not filled)

default

If none of the above criteria are met then the default mode is adopted, as determined by the lower−case value of the **$fill−mode** value. **SEE ALSO**

$fill−col(5), buffer−mode(2), fill−paragraph(2), justify(2m).

$find−words(5)

NAME

$find−words − Filtered word list

SYNOPSIS

$find−words *word*

DESCRIPTION

$find−words must first be initialized to the required filter string, if the variable is evaluated before it is initialized the value will be set to "*ABORT*" and the command will fail.

The filter string can contain wild−card characters compatible with most file systems, namely:−

?

Match any character.

[abc]

Match character only if it is *a*, *b* or *c*.

[a−d]

Match character only if it is *a*, *b*, *c* or *d*.

[^abc]

Match character only if it is not *a*, *b* or *c*.

*****

Match any number of characters.

Note that these are not the same characters used by exact(2m) mode.

Once initialized, evaluating **$find−words** returns the next word found in the main spell dictionaries which matches the filter until no more words are found, in which case an empty string is returned.

EXAMPLE

The following example finds all the words with "*foo*" in it (e.g. "*footnote*"), printing them to the massage line one at a time. Note that &set(4) is used on the !while(4) statement to avoid evaluating **$find−words** twice per loop.

```
set-variable $find-words "*foo*"
!while &not &seq &set #l0 $find-words ""
    100 ml-write &cat "Word: " #l0
!done
```

NOTES

The order of the words is undefined.

Due to the way words are derived, it is possible to have two or more copies of a word in the dictionary. If this is a matching word **$find−words** will return the word two or more times.

SEE ALSO

spell(2).

$fmatchdelay(5)

NAME

$fmatchdelay− Fence matching delay time

SYNOPSIS

$fmatchdelay *delayTime*; Default is `2000`

0 <= *delayTime* <= *n*

DESCRIPTION

The number of milliseconds to wait in a fence match operation. When a closing fence `')'` `']'` or `'}'` is added the opening fence is searched for, scrolling the screen up where necessary, this is the time that the opening fence is displayed, interruptible by typing any key.

When cmode(2m) is enable the search algorithm used is '*C*' aware and if a matching fence is not found then the bell is rung as a warning. The automatic matching of fences can be enabled/disabled via the fence(2m) mode.

A cursor can be moved to the matching fence using the goto−matching−fence(2) command.

SEE ALSO

fence(2m), cmode(2m), goto−matching−fence(2).

$frame−depth(5)

NAME

$frame−depth − Number of lines on the current frame canvas
$frame−width − Number of columns on the current frame canvas

SYNOPSIS

$frame−depth *integer*

3 <= *integer* <= 400

$frame−width *integer*

8 <= *integer* <= 400

DESCRIPTION

These variables allow the viewable size of the current frame canvas to be determined.

$frame−depth identifies depth of the current frame given as the number of character lines. This is the whole frame width, not just what is currently visible. The value returned is in the range 3 − *n*, *n* is system dependent but no greater than 400.

$frame−width identifies the width of the current frame as the number of character columns. The value returned is in the range 8 − *n*, *n* is system dependent but no greater than 400.

NOTES

The name of these variables changed from **$screen−depth** and **$screen−width** due to the support for multiple frames introduced in April 2002.

SEE ALSO

change−frame−depth(2), change−frame−width(2).

$global−scheme(5)

NAME

$global−scheme − Default global buffer color scheme.

SYNOPSIS

`$global−scheme` `schemeNum`; Default is `0`

DESCRIPTION

$global−scheme defines the default buffer color scheme to *schemeNum*, a color scheme defined by add−color−scheme(2).

SEE ALSO

add−color(2), add−color−scheme(2), $buffer−hilight(5), $buffer−scheme(5), $cursor−color(5), $trunc−scheme(5), $ml−scheme(5), $osd−scheme(5), $mode−line−scheme(5), $scroll−bar−scheme(5), $system(5).

$home(5)

NAME

$home − Users `home' directory location

SYNOPSIS

$home *directory*

DESCRIPTION

The file naming convention utilizes tilde ('~') to identify the users home directory (`$HOME`). When entering a file name:

```
~/xxx    -> $home/xxx
~yyy/xxx -> $home/../yyy/xxx
```

On most systems this is automatically set to the environment variable "HOME" if it is defined or may be defined explicitly in the start−up file. '~' may be used in the `me.emf` files but must be specified as '~'. It may be picked up in command files as **$home**.

$idle−time(5)

NAME

$idle−time − System idle event delay time

SYNOPSIS

$idle−time *milliseconds*; Default is `1000`

10 <= *milliseconds* <= t

DESCRIPTION

$idle−time sets the time waited between the last user event and the generation of a `idle−pick` key event. When user input stops for **$idle−time** milliseconds MicroEmacs checks the binding of the special `idle−pick` key, if this pseudo key is bound then the command it is bound to will be executed. MicroEmacs will then cycle, generating a `idle−pick` every **$idle−time** milliseconds until user activity starts. At this point a `idle−drop` key event is generated, if this pseudo key is bound then the command it is bound to will be executed.

This system is useful for things which can be done in the background.

EXAMPLE

The following example is taken from `ssaver.emf` and implements a simple screen saver:−

```
set-variable %screen-saver 0
define-macro screen-saver
    !if &not &pinc %screen-saver 1
        !if &seq @cck "idle-pick"
            ; default is to switch on in 5 minutes time
            &cond @? @# 300000 create-callback screen-saver
        !else
            !if &seq @cck "callback"
                @# create-callback screen-saver
            !elif @?
                ; user has suppled argument, install or remove
                !if &gre @# 0
                    &mul @# 60000 global-bind-key screen-saver "idle-pick"
                !else
                    !force global-unbind-key "idle-pick"
                !endif
                set-variable %screen-saver &sub %screen-saver 1
                !return
            !endif
            set-variable @# $frame-depth
            !while &dec @# 1
```

```
                            2 screen-poke @# 0 $global-scheme &spr "%n" $frame-width " "
                        !done
                        0 screen-poke 0 0 $global-scheme &spr "%n" $frame-width " "
                        -1 show-cursor
                        ; must set this to stop recursion when waiting for a key!
                        set-variable %screen-saver 0
                        set-variable @# @cg
                        set-variable %screen-saver 1
                        1 show-cursor
                        screen-update
                        ml-clear
                    !endif
                !endif
                set-variable %screen-saver &sub %screen-saver 1
            !emacro
```

NOTES

Care must be taken to ensure that a recursive loop is not created, consider the following example:−

```
define-macro bored
    !if &iseq @mc1 "Are you bored (y/n)? " "nNyY" "y"
        ml-write "Play a silly game!"
    !endif
!emacro
global-bind-key bored idle-pick
```

If this was executed MicroEmacs would very quickly crash! As soon as a second past **bored** would execute, which will prompt the user and wait for input. If a second passes without input **bored** will be executed again and again and again until stack memory runs out! To avoid this `idle-pick` should be unbound before waiting for user input, i.e.:−

```
define-macro bored
    global-unbind-key idle-pick
    !if &iseq @mc1 "Are you bored (y/n)? " "nNyY" "y"
        ml-write "Play a silly game!"
    !endif
    global-bind-key bored idle-pick
!emacro
global-bind-key bored idle-pick
```

SEE ALSO

$delay−time(5).

$kept−versions(5)

NAME

$kept−versions − Number of backups to be kept

SYNOPSIS

$kept−versions *integer*; Default is 0

0 <= *integer* <= n

DESCRIPTION

$kept−versions allows the user to specify the number of backup versions that are required for each file. For file "`XXXX`", each backup version is renamed to "`XXXX.~?~`", where `?` is the backup number. If **$kept−versions** is set to 0 this feature is disabled and the default single backup file is created.

The most recent backup will always be `.~0~` and the last version will be `.~n~` where `n` is **$kept−versions** − 1. when the file is next saved the `.~0~` backup file is moved to `.~1~`, `.~1~` to `.~2~` etc, backup `.~n~` is removed. Evidently if **$kept−versions** it set to a large number this can effect performance.

RESTRICTIONS

$kept−versions may only be used when DOS file name restrictions are not enabled. This means that some systems (such as DOS) cannot use this functionality, see $system(5) for more information. Backup files are only created when buffer mode backup(2m) is enabled.

NOTES

This feature is not supported when writing ftp files, a single backup file is created when backup files are enabled.

SEE ALSO

$system(5), autosv(2m), backup(2m), ftp(3), save−buffer(2).

$line−scheme(5)

NAME

$line−scheme − Set the current line color scheme

SYNOPSIS

$line−scheme *schemeNum*; Default is `−1`

DESCRIPTION

$line−scheme sets the color scheme to be used for the current line of the current window. The given *schemeNum* can be any scheme number previously defined by the function add−color−scheme(2).

A line's $line−scheme setting is removed by setting the variable to −1.

A $line−scheme setting takes precedence over the buffer's color scheme ($buffer−scheme(5)) and the buffer's hilighting scheme ($buffer−hilight(5)).

EXAMPLE

c−hash−eval(3) greys out lines of text by doing:

```
set-variable $line-scheme %lblack
```

The lines are rest by doing

```
set-variable $line-scheme -1
```

The gdb(3) interface hilights the current line of source by doing:

```
set-variable $line-scheme %yellow-lblack
```

NOTES

Due to line storage restrictions, only 15 different color schemes can be used in a buffer at any one time. When the 16th color scheme is used it replaces the first color scheme, all lines using the first color scheme will be colored using the new color scheme.

SEE ALSO

add−color−scheme(2), c−hash−eval(3), $buffer−scheme(5), $buffer−hilight(5), $mode−line−scheme(5), $scroll−bar−scheme(5), $system(5).

$line−template(5)

NAME

$line−template − Command line regular expression search string

SYNOPSIS

$line−template "*string*"; Default is ""

DESCRIPTION

$line−template defines a regular expression search string used to identify a line number in the grep(3) and compile(3) buffers. The format of the string is the same as magic mode search strings (see search−forward(2)).

EXAMPLE

The line number may be considered to contain any numeric number, thus **$line−template** is defined as:

```
set-variable $line-template "[0-9]+"
```

This correctly identifies "45" in the following ***grep*** output example:

```
foo.c: 45:      printf("hello world\n") ;
```

SEE ALSO

$file−template(5), compile(3), get−next−line(2), grep(3), search−forward(2).

$ml−scheme(5)

NAME

$ml−scheme − Message line color scheme

SYNOPSIS

$ml−scheme *schemeNum*; Default is `0`

DESCRIPTION

$ml−scheme defines the color scheme to be used on the message line, the color scheme *schemeNum* identifies the foreground and background color and is defined by an invocation to add−color−scheme(2).

The background color is always defined by $global−scheme(5).

SEE ALSO

$global−scheme(5), $osd−scheme(5), $mode−line−scheme(5), $scroll−bar−scheme(5), $system(5), add−color−scheme(2).

$mode−line(5)

NAME

$mode−line − Mode line format

SYNOPSIS

$mode−line "*string*"; Default is "`%s%r%u me (%e) - %l %b (%f)` "

DESCRIPTION

$mode−line defines the format of the mode line printed for every window, where the character following a percent ('%') has the following effect:−

`D`	Prints the current day.
`M`	Prints the current month.
`Y`	Prints the current year (2 digits).
`y`	Prints the current year (4 digits).
`b`	Prints the current buffer's name.
`c`	Prints the current buffer's column number.
`e`	Prints the current buffer's editing modes.
`f`	Prints the current buffer's file name.
`h`	Prints the current hour of the day.
`k`	Prints the current keyboard macro status.
`l`	Prints the current buffer's line number.
`m`	Prints the current minute of the hour.
`n`	Prints the current buffer's total number of lines.
`r`	Prints the current root user status (UNIX only).
`s`	Prints the horizontal window split character.
`u`	Prints the current buffer's (un)changed or view mode flag.
`%`	Prints a percentage escape character.
`−`	Prints a literal minus character ('`−`') − see NOTES.
`*`	All other characters are printed literally.

NOTES

- Refer to $window−chars(5) for the characters utilized in the mode line. Typically a the '`−`' character is changed to a '`=`' if it is the current window. If a '`−`' is always required, use "`%−`".
- A buffer can have its own mode−line, and be uneffected be the global mode line, see $buffer−mode−line(5).

SEE ALSO

$buffer−mode−line(5), $mode−line−scheme(5), $window−chars(5).

$mode−line−scheme(5)

NAME

$mode−line−scheme − Mode line color scheme

SYNOPSIS

$mode−line−scheme *schemeNum*; Default is `1`

DESCRIPTION

Sets the window mode−line color scheme, defining the foreground and background colors. The *schemeNum* is defined by a previous invocation to add−color−scheme(2).

SEE ALSO

add−color−scheme(2), $global−scheme(5), $ml−scheme(5), $scroll−bar−scheme(5), $system(5).

$mode−names(5)

NAME

$mode−names − Filtered mode name list

SYNOPSIS

$mode−names *ModeName*

DESCRIPTION

$mode−names must first be initialized to the required filter string, if the variable is evaluated before it is initialized the value will be set to "*ABORT*" and the command will fail. The filter takes the form of a regex.

Once initialized, evaluating **$mode−names** returns the name of the next mode which matches the filter until no more modes are found, in which case an empty string is returned.

EXAMPLE

The following example prints out the name of all modes to the massage line one at a time. Note that &set(4) is used on the !while(4) statement to avoid evaluating **$mode−names** twice per loop.

```
set−variable $mode−names "*"
!while &not &seq &set #l0 $mode−names ""
    100 ml−write &cat "mode: " #l0
!done
```

SEE ALSO

buffer−mode(2), &bmode(4), $buffer−names(5), $command−names(5), Regular Expressions.

$mouse(5)

NAME

$mouse − Mouse configuration variable

SYNOPSIS

$mouse *bitmask*; Default is system dependent

DESCRIPTION

The **$mouse** is used to define and configure the MicroEmacs mouse support, it is a bit based flag where:−

0x00f

Defines the number of button the mouse has, only values 1, 2 & 3 are useful. By default MicroEmacs uses the system information to determine the number of buttons on the mouse, this is not fool proof so the user can set these bits to the appropriate number if the initial value is incorrect.

0x010

If set the mouse is enabled, if clear the mouse will not function. On systems which do not support mice (such as UNIX Termcap) this bit will be clear and can not be altered.

0x020

If set the buttons are reversed, i.e. the left button becomes the right and vice versa. By default this bit is clear.

0xf0000

Defines the current mouse icon to used, valid values are as follows:

0x00000 − Set mouse to default icon.
0x10000 − Set mouse to arrow icon.
0x20000 − Set mouse to text I−beam icon.
0x30000 − Set mouse to crosshair icon.
0x40000 − Set mouse to the grab icon.
0x50000 − Set mouse to the wait icon.
0x60000 − Set mouse to the stop icon.

This feature is not supported on some systems and on others some icons are not obvious due to platform limitations.

EXAMPLE

The following example checks that the mouse is currently available, if not, it aborts.

```
!if &not &band $mouse 0x10
    ml-write "[Mouse support is not currently available]"
    !abort
!endif
```

NOTES

The mouse can be easily configured using user−setup(3).

SEE ALSO

user−setup(3), $system(5), $platform(5).

$mouse−pos(5)

NAME

$mouse−pos − Mouse position information

SYNOPSIS

$mouse−pos *integer*

DESCRIPTION

$mouse−pos is generated by invocation of the command set−cursor−to−mouse(2). The variable is set to a value that indicates the position of the mouse within a window. The values to the mouse intersection are interpreted as follows:−

0 − Text area

Intersection with the window text area.

1 − Message Line

Intersection with the message line.

2 − Mode Line

Intersection with the mode line.

3 − Horizontal Separator

Intersection with the horizontal window separator. This value is only set if a scroll bar is not present.

4 − Up Arrow

Intersection with the scroll bar up−arrow character.

5 − Upper Shaft

Intersection with the scroll bar upper shaft (above the scroll box).

6 − Scroll Box

Intersection with the scroll bar scroll box.

7 − Lower Shaft

Intersection with the scroll bar lower shaft (below the scroll box).

8 − Down Arrow

Intersection with the scroll bar down−arrow character.

9 − Corner

Intersection with the window corner, that is the character at the intersection of the scroll bar (or separator) and the mode line.

10 − Menu Line

Intersection with the menu line.

255 − Error

The position of the mouse could not be determined. This value should not arise, if it does then it is an indication that the window structure is probably corrupted. A delete−other−windows(2) is suggested or rapid exit from the editor after a save−some−buffers(2) command to save any edits (latter option is preferred).

Bit 4 − 2nd Column

Bit 4 (16) is set if 2 character column scroll bar or vertical window separator is in effect and the cursor exists in the second column This value is bitwise OR'ed with the aforementioned intersection values. **EXAMPLE**

The following macro can be used to print out the current position of the mouse, try binding the macro to the "`mouse−move`" key:

```
define−macro print−mouse−position
    !force set−cursor−to−mouse
    set−variable #l0 &band $mouse−pos 15
    !if &equ #l0 0
        ml−write "Mouse in text window"
    !elif &equ #l0 1
        ml−write "Mouse on message line"
    !elif &equ #l0 2
        ml−write "Mouse on Mode line"
    !elif &and &gre #l0 2 &les #l0 10
        ml−write "Mouse on scroll bar"
    !elif &equ #l0 10
        ml−write "Mouse on corner"
    !elif &equ #l0 11
        ml−write "Mouse on menu line"
    !endif
!emacro

global−bind−key print−mouse−position mouse−move
```

$mouse−pos is utilized by the mouse picking code, found in macro file `mouse.emf`.

SEE ALSO

$mouse−x(5), $mouse−y(5), set−cursor−to−mouse(2), set−scroll−with−mouse(2).

$mouse−x(5)

NAME

$mouse−x − Mouse X (horizontal) position
$mouse−y − Mouse Y (vertical) position

SYNOPSIS

$mouse−x *integer*

0 <= *integer* <= $frame−width − 1

$mouse−y *integer*

0 <= *integer* <= $frame−depth − 1

DESCRIPTION

$mouse−x and **$mouse−y** are automatically set to the position of the mouse at the last mouse event, where an event is a button press or release. Initialized to 0,0. The top left character of the screen is coordinate 0,0 bottom right is $frame−width, $frame−depth.

NOTES

These variables can not be set. Any attempt to set them will result in an error.

SEE ALSO

set−cursor−to−mouse(2), $mouse−pos(5), $cursor−x(5), $frame−depth(5), $frame−width(5).

$osd−scheme(5)

NAME

$osd−scheme − OSD color scheme

SYNOPSIS

$osd−scheme *schemeNum*; Default is `1`

DESCRIPTION

$ml−scheme defines the color scheme by default on an osd(2) dialog, the color scheme *schemeNum* identifies the foreground and background color and is defined by an invocation to add−color−scheme(2). Every osd dialog can over−ride this value by using the '*S*' flag.

SEE ALSO

osd(2), add−color−scheme(2), $global−scheme(5), $ml−scheme(5), $mode−line−scheme(5), $scroll−bar−scheme(5), $system(5).

$platform(5)

NAME

$platform − MicroEmacs host platform identifier
%platform − MicroEmacs host platform type identifier

SYNOPSIS

$platform "*string*"; Default is platform specific
%platform "*string*"; Default is platform specific

DESCRIPTION

The **$platform** variable is a fixed ASCII string used to identify the current working platform, attempts to set this variable result in an error returned from set−variable(2).

Possible values are:

"**aix**"

All IBM AIX O/S.

"**dos**"

All IBM−PCs and compatibles running MS−DOS.

"**freebsd**"

All FreeBSD O/S.

"**hpux**"

All Hewlett Packard's with HP−UX O/S.

"**irix**"

All Silicon Graphics (SGI) IRIX platforms 4.x, 5.x, 6.x.

"**linux**"

All LINUX O/S.

"**sunos**"

All Sun's with SUNOS O/S.

"**unixwr1**"

PC based UNIX platform (Consensus and Unixware).

"**win32**"

Microsoft Windows based systems including Windows 3.x (with Win32s), Windows '95 and NT.

$platform is often used in **.emf** files to allow portability of macro files across platforms, allowing macro files to perform platform specific operations. $system(5) is also often used for this purpose as its value is easier to assess.

%platform is created at start−up when `me.emf` is executed, its value is identical to **$platform** except when the platform is a console in which case a '`c`' is appended to the $platform value, e.g. for MicroEmacs running a termcap version on LINUX the value will be "`linuxc`". The variable is used when the console and window based versions need to be distinguish, e.g. some of the user−setup settings.

EXAMPLE

The following example is taken from the **me.emf** file which uses the **$platform** variable to load the platform specific initialization files.

```
;
; load in the platform specific stuff
execute-file $platform
```

This could be more explicitly done by:

```
;
; load in the platform specific stuff
!if   &seq $platform "dos"            ; is it an IBM-PC running dos ?
    execute-file "dos"
!elif &seq $platform "irix"           ; is it an sgi ?
    execute-file "irix"
!elif &seq $platform "hpux"           ; is it an hp ?
    execute-file "hpux"
    .
    .
!endif
```

NOTES

The **$platform** variable can not be set. Any attempt to set it will result in an error.

SEE ALSO

$system(5), set−variable(2).

$progname(5)

NAME

$progname − Program file name

SYNOPSIS

$progname *string*

DESCRIPTION

$progname is set the the MicroEmacs '02 program file name currently being run. This can be used by macros for many purposes, from spawning another MicroEmacs '02 session to working out where MicroEmacs '02 is running from.

EXAMPLE

The following example is used to spawn of another MicroEmacs '02 command to create a C tags file:−

```
shell−command &cat $progname " \"@ctags\" *.c *.h"
```

SEE ALSO

me(1).

$random(5)

NAME

$random − Generate a random number

SYNOPSIS

`$random` *integer*

0 <= *integer* <= 65535

DESCRIPTION

The **$random** variable returns a unique random number in the range 0 − *n* on reference to the variable.

The random number is derived from the system's random number generator (the quality of which is often dubious so try to avoid using the bottom bits). Setting this variable with any value resets the random sequence using the system time as the seed.

The range of the random number generator is system dependent. The value is typically capped using the &mod(4) arithmetic operator.

EXAMPLE

The variable may be assigned to generate a new seed as follows:−

```
set-variable $random 0        ; Set it so we get a new seed
```

The returned value is used with the **&mod** operator to limit the value to a desired range:−

```
set-variable %random0to9 &mod $random 10
```

SEE ALSO

&mod(4).

$rcs−file(5)

NAME

$rcs−file − RCS (and SCCS) file name
$rcs−ci−com − RCS (and SCCS) check in command
$rcs−cif−com − RCS (and SCCS) check in first command
$rcs−co−com − RCS (and SCCS) check out command
$rcs−cou−com − RCS (and SCCS) check out unlock command
$rcs−ue−com − RCS (and SCCS) unedit file command

SYNOPSIS

$rcs−file "*string*"; Default is ""
$rcs−ci−com "*string*"; Default is ""
$rcs−cif−com "*string*"; Default is ""
$rcs−co−com "*string*"; Default is ""
$rcs−cou−com "*string*"; Default is ""
$rcs−ue−com "*string*"; Default is ""

DESCRIPTION

RCS (Revision Control System) and SCCS (Source Code Control System) are programmers source code history data−bases. RCS introduces a system in which only one programmer can edit a source file at any one time, enforcing some form of stability in the global environment. The fact that this interface was developed for the RCS system is irrelevant, and should be usable under any other control systems such as SCCS.

When using RCS, finding a file (see find−file(2)) checks for the existence of the actual file. If this is not found then it checks for the existence of an RCS **$rcs−file** variable, and if present then it constructs the RCS file name and checks for its existence. If this file does not exist then it really is a new file and an new buffer is created. If the file does exist then the file is checked out using the **$rcs−co−com** which executes to create a file with the original file name, ready for loading.

$rcs−file is the name of the file when it is fully check in, as opposed to when it is ready to be viewed or edited. In RCS, this is usually in the `RCS` directory with an appended "`,v`", i.e. for the file `foo.c` in the `/test` directory, when fully checked in, the file will not be found at "`/test/foo.c`", but at "`/test/RCS/foo.c,v`". When testing for an RCS file, the file name is split into two parts, the path name and the file name, the path is always inserted at the start, and the file name can inserted in the rcs string by using the special "`%f`" token, thus if **$rcs−file** is set to "`RCS/%f,v`", the RCS file name is constructed from "`/test/`" + "`RCS/`" + "`foo.c`" + "`,v`".

If the RCS file is found then the **$rcs−co−com** (RCS **C**heck **O**ut **COM**mand) which is a simple system command line with the exception for `%f` which is replaced by the file name, is executed. This is expected to create the file (with the correct file name) ready for viewing.

Once a file is loaded, then the rcs−file(2) command has one of two effects:−

> If the file is in view mode then the **$rcs−cou−com** (RCS **C**heck **O**ut **U**nlock **COM**mand) is executed (system command line using the "%f" as the file name). If the RCS file does not exist then is simply toggles the view mode, allowing editing.
>
> If the file is not in view mode MicroEmacs attempts to check the file back into RCS using either **$rcs−ci−com** (if the RCS file already exists) or the the **$rcs−cif−com** (RCS **C**heck **I**n **F**irst **COM**mand). The "`%f`" is again used for the file name, the "`%m`" can also be used to get a comment from the user at check in time which will be inserted (without quotes) into the **$rcs−ci−com** command line. For example, one possible **$rcs−ci−com** setting is "`ci −m\"%m\" %f`" which uses the **ci(1)** program with the **−m** option to give a check in message.

If **rcs−file** is given a −ve argument instead of checking in or out the current buffer's file it executes the command specified by **$rcs−ue−com** to unedit or abort any changes made to the file. After the command has been executed the file is reloaded.

NOTES

The RCS variables are by default undefined and must be explicitly enabled in the start−up files.

EXAMPLE

The following are typical variable definitions for the RCS interface:−

```
set−variable $rcs−file     "RCS/%f,v"
set−variable $rcs−co−com   "co %f"
set−variable $rcs−cou−com  "co −l %f"
set−variable $rcs−ci−com   "ci −u −m\"%m\" %f"
```

Note that the **$rcs−cif−com** variable is usually left unassigned and **$rcs−ci−com** is used by default.

The following are typical variable definitions for the SCCS interface:−

```
set−variable $rcs−file     "SCCS/s.%f"
set−variable $rcs−co−com   "sccs get %f"
set−variable $rcs−cou−com  "sccs edit %f"
set−variable $rcs−ci−com   "sccs delget −y\"%m\" %f"
set−variable $rcs−ci−com   "sccs create %f"
set−variable $rcs−ue−com   "sccs unedit %f"
```

The following variable definitions can be used for MicroSoft's Visual Source Safe:−

```
set−variable $rcs−file    "%f"
set−variable $rcs−cou−com "ss.exe checkout %f"
set−variable $rcs−co−com  "ss.exe checkout %f"
set−variable $rcs−ci−com  "ss.exe checkin %f \"−c%m\""
```

The above definitions can check a file out for edit and commit changes back.

SEE ALSO

find−file(2), rcs−file(2).

$recent−keys(5)

NAME

$recent−keys − Recent key history.

SYNOPSIS

$recent−keys *string*

DESCRIPTION

$recent−keys is a system variable that displays the last 100 keys entered into the system in reverse order. This variable is typically used to solve keyboard mapping problems when keys are not bound etc. allowing a visual inspection of the input into the editor.

SEE ALSO

buffer−bind−key(2), global−bind−key(2), translate−key(2).

$result(5)

NAME

$result − Various command return values

SYNOPSIS

$result *returnValue*

DESCRIPTION

$result is used to return the results of several commands:

buffer−info(2) **$result** is set to the same output string as printed to the message−line by this command.

change−font(2)

$result is used to return the user select font when hte windows font selection dialog is used (Windows systems only).

count−words(2)

$result is set to the same output string as printed to the message−line by this command.

find−registry(2)

$result is used to return the name of a registry child node given the parent and index from the user.

get−registry(2)

$result is used to return the current value of a user supplied registry entry.

mark−registry(2)

$result is used to return the full name of the given registry node.

osd(2)

> **$result** is used to give and return information to osd item commands, information depends on the type of **osd** item.

osd−dialog(3)

osd−xdialog(3)

> **$result** is used to return the button pressed by the user.

shell−command(2)

> **$result** is set to the exit status of the **system** call. The combination of **shell−command** calls and return value checking can be used in a variety of ways, for example, to test the existence of a file:

```
set-variable %filename @ml"Enter file name"
shell-command &cat "test -f " %filename
!if &equ $result 0
    ml-write "file exists"
!else
    ml-write "file does not exists"
!endif
```

show−region(2)

> **$result** is set to the current status of the region when an argument of *0* is given to **show−region**.

spell(2)

> **$result** is used to return information on the current word, the information depends on the argument given to **spell**.

$file−names(5)

> **$result** is set to the absolute path of the **$file−names** query directory when the variable is set.

For more information see the help pages on referenced commands and variables.

NOTES

The current value of **$result** is lost on the next command call which uses it. As a call to create−callback(2) can cause the execution of a macro to interrupt another which is waiting for user input, the value of **$result** should be copied before getting user input.

SEE ALSO

buffer−info(2), change−font(2), count−words(2), find−registry(2), get−registry(2), mark−registry(2), osd(2), shell−command(2), show−region(2), spell(2), $file−names(5), create−callback(2), $status(5).

$scroll(5)

NAME

$scroll − Screen scroll control

SYNOPSIS

`$scroll` *scrollNum*; Default is `1`

0 <= *scrollNum* <= n

DESCRIPTION

$scroll controls the horizontal and vertical scrolling method used to display long lines and buffers. The variable is split into two componants, the first nibble (`0x0f`) sets the horizontal scroll, and the second nibble (`0xf0`) sets the vertical. For the purpose of documentation these parts are kept separate, but when setting the variable a single combined value must be given.

The horizontal settings are defined as follows:

`0x00`

Scroll method 0 will only scroll the current line, this is the fastest method in execution time.

`0x01`

Scroll method 1 (the default) will scroll the whole page horizontally when the scroll−left(2) and scroll−right(2) commands are used. However, when the current line must be scrolled to display the cursor due to a forward−char(2) type cursor movement, only the current line is scrolled and the rest are reset.

`0x02`

Scroll method 2 always scrolls the whole page horizontally, keeping the cursor in the current column range. If the cursor moves out of this range then all the page is scrolled to the new position. This is particularly useful when editing long lined tables.

`0x03`

Scroll method 3 fixes the scroll column using the **scroll−left** and **scroll−right** functions. If the current cursor position is not visible in the column range then only the current line is scrolled to the new position.

The vertical settings are defined as follows:

`0x00`

Scroll method 0 (the default) will scroll the current line to the middle of the current window whenver it is moved off screen, this is the fastest method in execution time.

`0x10`

Scroll method 1 will scroll the current line to the the top of the window whenver the current line is moved off the screen using backward−line(2) and to the bottom of the window when forward−line(2) is used. This creates the effect of a smooth scroll. **EXAMPLE**

The following example sets the scrolling method to be the default horizontally (`0x01`) and smooth method (`0x10`) vertically :

```
set−variable $scroll 0x11
```

SEE ALSO

scroll−left(2), forward−line(2), $window−x−scroll(5), $window−y−scroll(5).

$scroll−bar(5)

NAME

$scroll−bar − Scroll bar configuration

SYNOPSIS

$scroll−bar "*bitmask*"; Default is platform specific

DESCRIPTION

$scroll−bar defines the configuration of the scroll bar and/or the horizontal window separator for both main text windows and osd(2) dialogs. The variable is interpreted as a bit mask and defines which components of the scroll bar (or separator) should be rendered in a window. The characters used to render the scroll bar or separator are defined by $window−chars(5). The bit mask is defined as follows:−

0x001 − Vertical Scroll Bar Width

Bit 0 controls the width of the vertical scroll bar (or separator). A value of 0 corresponds to a single column width, a value of 1 is a double column width.

0x002 − Upper end cap

Bit 1 set indicates that the scroll bar has an upper end cap. This is the up arrow character at the top of a scroll bar.

0x004 − Lower end cap

Bit 2 set indicates that the scroll bar has a lower end cap. This is the down arrow character at the bottom of a scroll bar.

0x008 − Corner

Bit 3 set indicates that separate corner character is used at the intersection of the mode line and the separator.

0x010 − Scroll Box Enable

Bit 4 determines if the scroll bar has a scrolling box, when the bit is set each scroll bar will have a scroll box. When clear, scroll bars are rendered according to bits 0−3 & 7 only and the main area of the bar is left empty.

0x020 − Reverse Video Box

Bit 5 when set enables the scroll box to be rendered in reverse video, that is the background and foreground/hilight scroll colors are interchanged. This bit is typically set on X−Window platforms allowing the scroll box to comprise of `SPACE` characters allowing a solid box to be rendered in the foreground color.

Bit 5 is only enacted if scroll boxes are enabled.

0x040 − Horizontal Scroll Bar Width

Bit 6 controls the width of the horizontal scroll bar, used only by osd(2). A value of 0 corresponds to a single column width, a value of 1 is a double column width.

0x080 − Splitter

Bit 7 set indicates that the scroll bar has a splitter. This is the split bar character at the top of a scroll bar.

0x100 − Enable window Scroll Bars

When Bit 8 is clear, scroll bars are not present on windows. If a horizontal split has been performed then the window separator is rendered plain. This is useful when performance is important, as scroll bars require constant up−date.

0x200 − Horizontal Scroll Bar Width

Bit 9 enables scroll bars, when the bit is set each window is assigned a scroll bar in the right−hand column(s) of the window with a scroll box. **SEE ALSO**

$mouse−pos(5), $scroll−bar−scheme(5), set−scroll−with−mouse(2), $window−chars(5).

$scroll−bar−scheme(5)

NAME

$scroll−bar−scheme − Scroll bar color scheme

SYNOPSIS

$scroll−bar−scheme *schemeNum*; Default is `1`

DESCRIPTION

Sets the horizontal window scroll bar color scheme, assigning the foreground, background and selection colors which are used to render the vertical separator / scroll bars (see add−color−scheme(2). The separator is rendered in reverse video, i.e. the foreground color of the color scheme is used as the background color, and vice versa.

The separator is rendered in the standard colors when the associated buffer is not active, and in the current color when the buffer is active.

The scroll−bar is the window separator constructed by split−window−horizontally(2) or when the scroll bars are enabled via $scroll−bar(5).

SEE ALSO

$global−scheme(5), $ml−scheme(5), $mode−line−scheme(5), $scroll−bar(5), $system(5), $window−chars(5), split−window−horizontally(2).

$show−modes(5)

NAME

$mode−line − Select buffer modes to display

SYNOPSIS

$show−modes "*bit−string*"; Default is ""

DESCRIPTION

$show−modes defines which buffer modes are displayed on the mode−line.

SEE ALSO

$user−setup(3), $mode−line(5).

$show−region(5)

NAME

$show−region − Enable the hilighting of regions

SYNOPSIS

$show−region *flag*; Default is `1`

DESCRIPTION

$show−region enables or disables the current region hilighting, normally associated with mouse interaction in a buffer. Region hilighting occurs between the *mark* (see set−mark(2)) and *point* (current cursor) positions within the current buffer. An argument *n* of 0 disables region hilighting, an argument of 1 enables region hilighting between the two positions. If it is set to 3 then region hilighting will be enabled and a defined region (created using copy−region(2)or yank(2)) will continue to be hilighted until the region is changed.

A defined region can be redisplayed (if still valid) using the command show−region(2). The color of the region hilighting is defined by add−color−scheme(2) and is determined by $buffer−scheme(5), $global−scheme(5) or $buffer−hilight(5).

SEE ALSO

show−region(2), $buffer−hilight(5), $buffer−scheme(5), $global−scheme(5), $buffer−scheme(5), add−color−scheme(2), set−mark(2).

$status(5)

NAME

$status − Macro command execution status

SYNOPSIS

$status *boolean*

boolean TRUE (1) | FALSE (0)

DESCRIPTION

$status contains the return status of the last command executed (TRUE or FALSE). **$status** is generally used with the !force directives in macros.

NOTES

This variable can not be set, any attempt to set it will result in an error.

EXAMPLE

The following example shows how the variable is used within a macro construct, it converts all tab characters to their SPACE equivalent.

```
;
; tabs−to−spaces.
; Convert all of the tabs to spaces.
define−macro tabs−to−spaces
    ; Remember line
    set−variable #l0 $window−line
    beginning−of−buffer
    !force search−forward "\t"
    !while $status
        set−variable #l1 $window−acol
        backward−delete−char
        &sub #l1 $window−acol insert−space
        !force search−forward "\t"
    !done
    goto−line #l0
    screen−update
    ml−write "[Converted tabs]"
!emacro
```

In this case **$status** monitors the search−forward command which is searching for a tab character. The command returns a status value of TRUE if a tab is found, otherwise FALSE.

The **!force** statement prevents the macro from terminating when a FALSE condition is detected, if omitted the macro would terminate with an error as soon as the FALSE status is encountered. The definition of tabs−to−spaces(3) can be found in format.emf.

SEE ALSO

execute−file(2), !force(4), $result(5), tabs−to−spaces(3).

$system(5)

NAME

$system − System configuration variable

SYNOPSIS

$system *bitmask*; Default is system dependent

DESCRIPTION

The **$system** is used to define and configure the MicroEmacs environment, it is a bit based flag where:−

0x001

This bit is set if MicroEmacs is running in Console mode. On UNIX systems the default is to use X whenever possible, in which case this bit will be clear. If X is not used then a TERMCAP base interface is used instead and this bit will be set (see notes below on how to set which interface to use). On all other systems this bit will be clear.

0x002

If this bit is set then the current system supports definable RGB colors allowing any color to be created and used in a color scheme. This bit cannot be set, typically Windows and UNIX X−Windows systems support this.

0x004

If this bit is set then the current system supports ANSI colors (8 colors, black, red, green, yellow, blue, magenta, cyan & white), bits 0x002 and 0x004 are mutually exclusive. On UNIX systems if the TERMCAP interface is being used then this bit can be changed to (de)select the used of color. Many unix terminals do not support color so this should be set appropriately. On all other systems this bit cannot be changed and MS−DOS is currently the only other system to use ANSI colors.

0x008

If this bit is set then the current system supports Extended ANSI colors, brighter versions of the 8 ANSI colors doubling the number of colors available to 16. On UNIX systems if the TERMCAP interface is being used then this bit can be changed to (de)select the used of bold with color to create this extended color set for foreground colors. But many UNIX terminals do not support this use of color with the bold font so this should be set appropriately. On all other systems this bit cannot be changed and MS−DOS is currently the only other system to support this.

0x010

If this bit is set then the current system supports the use of fonts (bold, italic, light and underline). Whether these fonts can be successfully utilized depends upon the platform and the system font being used, for UNIX TERMCAP systems it will also depend on the terminal being used. This option is not supported on MS_DOS.

0x080

This bit is set if the current system is a UNIX based system such as LINUX or HPUX. This bit cannot be altered, its use is within macros.

0x100

This bit is set if the current system is a Microsoft based system such as DOS or Windows '95. This bit cannot be altered, its use is within macros.

0x200

If this bit is set then the current system uses the concept of drives (i.e. `c:/` on DOS systems). This bit cannot be altered, its use is within macros.

0x400

If this bit is set then a DOS style `8.3` file naming system should be used (i.e. "`BBBBBBBB.XXX`"), otherwise an unlimited file name length is used. This effects the backup and auto−save file names generated by MicroEmacs, the bit can be altered on systems that support unlimited file name length.

0x800

If this bit is set then the current system supports and uses ipipe−shell−command(2) when required. For systems such as DOS which cannot support ipipes, this bit will be clear and cannot be altered. For systems which do support ipipes, this bit can be cleared to disable their use.

0x1000

If this bit set, the then execution of the tab(2) command (bound to `tab`) always checks and adjusts the indentation of the current line when the current buffer is in cmode(2m) or has an indentation method. If the bit is clear then the `tab` may only checks the indentation when the cursor is in column zero depending on the setting of bit **0x200000**.

0x2000

If this bit is set the main menu Alt hot−key bindings are enabled. These are dynamic bindings automatically generated from the main menu. Typically the first item in the main menu is "`File`" with a hot key of '**F**', with this bit set '`A−f`' will open this menu item. Note that global and local key bindings override these. Also see bit **0x4000**.

0x4000

If this bit is set the Alt key acts as a prefix 1 modifier key. By default '`A-n`' is not bound, with this bit set the key is inferred to '`esc n`' which is bound to **forward−paragraph**. Note that global, local and menu hot−key bindings override these. Also see bit 0x2000.

0x8000

If this bit is set the undo history is kept after a save allowing the undo(2) command to back−up changes beyond the last save. When clear the undo history is discarded after the buffer is saved.

0x10000

Enable box character rendering fix, supported on Win32 and XTerm interfaces only. Windows ANSI fonts and many XTerm ISO−8859−1 fonts do not have well formed box characters which are used by osd(2) and other commands to create a better looking interface. When this bit is enabled MicroEmacs traps the printing of characters with an ASCII value of less than 32 and renders them directly. Following is a table of supported characters, other characters in the range of `0x00` to `0x1f` not listed are rendered as a space:

`0x08`

Special Character; Backspace

`0x09`

Special Character; Tab

`0x0b`

Box Character; Bottom right

`0x0c`

Box Character; Top right

`0x0d`

Box Character; Top left

`0x0e`

Box Character; Bottom left

`0x0f`

Box Character; Center cross

`0x10`

Arrows; Right

`0x11`

Arrows; Left

`0x12`

Box Character; Horizontal line

`0x15`

Box Character; Left Tee

`0x16`

Box Character; Right Tee

`0x17`

Box Character; Bottom Tee

`0x18`

Box Character; Top Tee

`0x19`

Box Character; Vertical Line

`0x1e`

Arrows; Up

`0x1f`

Arrows; Down

0x20000

Enables the client server, default is disabled (UNIX and Win32 NT or Win95+ platforms only). When enabled a hidden "`*server*`" buffer is created which monitors commands written to the server, the socket "`/tmp/mesrv`**uid**" on UNIX systems and the command input file "**$TEMP**`/me`**$MENAME**`.cmd`" on Win32 systems. Commands can be written out using the command ipipe−write(2) while in the "*server*" buffer, the command is written to the same socket on UNIX systems and to the response file and response file "**$TEMP**`/me`**$MENAME**`.rsp`" on Win32 systems. This functionality is used by the **−m** and **−o** command−line options and by the MicroSoft DevStudio interface.

0x40000

Enables the capture of the Alt space key ("`A−space`"), default is enabled (Win32 platform only). In the Windows environment the Alt Space key is used to activate the main window's pull down menu at the top left. if this bit is set MicroEmacs captures this key and executes it as normal, thereby disabling this standard windows binding.

0x80000

Enables the drawing of visible white spaces, i.e. space, tab and new−line characters. When disabled (default) white spaces are drawn using spaces (' ') which means the user cannot distinguish between a tab and spaces or determine the last character of the line by merely looking at the display. When enabled MicroEmacs uses visible characters to draw the white spaces, the characters used are set with the variable $window−chars(5).

0x100000

Enables hiding MicroEmacs generated backup files. On Windows and Dos platforms the Hidden file attribute is used to hide the file, whereas on UNIX the backup file name is prepended with a '`.`'.

0x200000

If this bit set, the then execution of the tab(2) command (bound to `tab`) checks and adjusts the indentation of the current line when the cursor is in column zero and current buffer is in cmode(2m) or has an indentation method. The setting of this bit has no effect if bit **0x1000** is set. If this and bit **0x1000** are clear then the `tab` will not check the indentation.

0x400000

When this bit is set the external clipboard (Windows & XTerm platforms) will never be set to empty, if the current yank buffer is the empty string the cut buffer will be set to a space (i.e. " "). This feature has been added to avoid problems with other software (e.g. **exceed(1)** which can crash if given an empty cut buffer).

0x800000

When this bit is set all use of the external clipboard (Windows & XTerm platforms) is disabled, this means that MicroEmacs will not attempt to retrieve or set the content of the system clipboard. **EXAMPLE**

The follow example works out the current buffer's backup file name using **$system** to determine the naming system being used by MicroEmacs:−

```
set−variable #l0 &stat "a" $buffer−fname
; Is an 8.3 dos style naming system being used?
!if &band $system 0x400
    !if &not &set #l1 &sin "." #l0
        set−variable #l1 &cat #l0 ".~~~"
    !elif &gre &set #l1 &sub &len #l0 #l1 2
        set−variable #l1 &cat &lef #l0 &sub &len #l0 1 "~"
    !else
        set−variable #l1 &spr "%s%n" #l0 &sub 3 #l1 "~"
    !endif
```

```
!elif $kept-versions
    set-variable #l1 &cat #l0 ".~0~"
!else
    set-variable #l1 &cat #l0 "~"
!endif
```

The following macro can be used to toggle the visible drawing of white spaces:

```
define-macro toggle-visible-white-spaces
    set-variable $system &bxor $system 0x80000
    screen-update
!emacro
```

NOTES

Most of the **$system** functionality can be set using the $user-setup(3) dialog.

UNIX X verses Termcap

By default, on X supporting systems MicroEmacs creates a new X window. This feature may be disabled in one of two ways:

- The environment variable `$TERM` is set to "`vt...`", in this case it is assumed that the machine is a server, and the host cannot support X.
- The `-n` option is used on the command line (see me(1)) to disable the windowing interface.

If X is disabled then the **termcap** interface is used instead, still allowing the use of colors through the ANSI standard, or the use of fonts (see bits **0x004** and **0x008**).

X provides the following features over and above a **termcap** based version of MicroEmacs '02:

- R,G,B style color creator giving access to up to 256 different colors for the ultimate hilighting schemes (see bit **0x002** and add-color(2)).
- Full mouse support, allowing user definable bindings to every mouse event (see global-bind-key(2)).
- Copy from and pasting to X's selection buffer (see yank(2)).

SEE ALSO

user-setup(3), $mouse(5), $platform(5), add-color(2), add-color-scheme(2), ipipe-shell-command(2), $global-scheme(5).

$tabsize(5)

NAME

$tabsize − Tab character width

SYNOPSIS

$tabsize *integer*; Default is 4

−0 < *integer* <= *n*

DESCRIPTION

$tabsize defines the width of a tab character.

Setting tabs to arbitrary widths is possible in MicroEmacs '02 but you must be aware of a subtle difference that it makes to your file and hence to your editing. When you start MicroEmacs '02, the tab width is set to the default (usually every 8th column) for the tab character (`CTRL−I`). As long as you stay with the default, every time you insert the tab character, a CTRL−I get inserted. Hence, you logically have a single character which might appear to be several spaces on the screen (or the output) depending upon the column location of the tab character. This means that to remove the spacing you have to delete a *single* character −− the tab character.

On the other hand, the moment you explicitly set the tab interval (even if it is to the default value), MicroEmacs '02 handles the tab character by expanding the character into the required number of spaces to move you to the appropriate column. In this case, to remove the spacing you have to delete the appropriate number of spaces inserted by M−e to get you to the right column.

The operating mode of the tab expansion is controlled by the tab(2m)mode.

SEE ALSO

buffer−mode(2) tab(2m), $tabwidth(5).

$tabwidth(5)

NAME

$tabwidth − Tab character interval

SYNOPSIS

$tabwidth `integer`; Default is `8`

−0 < *integer* <= *n*

DESCRIPTION

$tabwidth defines the interval of a tab character.

The tab interval is set to the given numeric argument. As always, the numeric argument precedes the command. Hence to get tabs every 4 spaces you would set the **$tabwidth** to 4.

SEE ALSO

buffer−mode(2) tab(2m), $tabsize(5), tabs−to−spaces(3).

$temp−name(5)

NAME

$temp−name − Temporary file name

SYNOPSIS

$temp−name *FileName*

DESCRIPTION

$temp−names is automatically set to a nonexistent file name in the systems temporary file directory. On UNIX systems the temporary directory is fixed to "`/tmp/`", on other systems the temporary directory is set by the **$TEMP** environment variable.

EXAMPLE

The following example uuencodes a given file into a temporary file and then inserts this file into the current buffer.

```
set−variable #l0 @ml04 "Uuencode and insert file"
set−variable #l1 $temp−name
!force shell−command &spr "uuencode %s < %s > %s" #l0 #l0 #l1
insert−file #l1
!force shell−command &cat "rm " #l1
```

NOTES

This variable can not be set, any attempt to set it will result in an error.

The returned file name is not guaranteed to be unique between calls, only that the file does not currently exist.

SEE ALSO

shell−command(2), file−op(2).

$time(5)

NAME

$time − The current system time

SYNOPSIS

$time "*string*"

DESCRIPTION

$time is a constantly changing variable which is set to the current system time. The format of **$time** is "`YYYYCCCMMDDWhhmmssSSS`", where:−

YYYY

The current year (full 4 digits so should be millennium bug free).

CCC

Day of the year (0−366).

MM

The month of the year (1−12).

DD

The day of the month (1−31).

W

The day of the week (0−6 Sunday=0).

hh

The hour (0−23).

mm

The minute (0−59).

ss

The second (0−59).

SSS

The millisecond (0−999).

$time can be set to an integer value which is a time offset in seconds, for example if the following was executed;−

```
set−variable $time "3600"
ml−write &cat "$time is " $time
set−variable $time "0"
```

The written time would one hour ahead of the system time.

EXAMPLE

The following macro times the time taken to execute a user command:−

```
define−macro time
    !force set−variable #l2 @1
    !if &not $status
        set−variable #l2 @ml00 "Time command"
    !endif
    set−variable #l0 $time
    !force execute−line #l2
    set−variable #l1 $time
    set−variable #l2 &add &mid #l0 16 2 &mul 60 &add &mid #l0 14 2 &mul 60 &mid #l
    set−variable #l3 &add &mid #l1 16 2 &mul 60 &add &mid #l1 14 2 &mul 60 &mid #l
    !if &les &set #l4 &sub &rig #l1 18 &rig #l0 18 0
        set−variable #l2 &add #l2 1
        set−variable #l4 &add 1000 #l4
    !endif
    ml−write &spr "Command took %d sec %d msec" &sub #l3 #l2 #l4
!emacro
```

time(3) is a macro defined in misc.emf.

organizer(3) uses **$time** to work out the current month.

SEE ALSO

time(3), organizer(3).

$timestamp(5)

NAME

$timestamp − Time stamp string

SYNOPSIS

$timestamp "*string*"; Default is "`<%Y%M%D.%h%m>`"

DESCRIPTION

$timestamp defines the file time−stamping string. MicroEmacs '02 searches for, and modifies, the string to the current time and date whenever the file is saved (written to disk) and time(2m) mode is enabled.

Time stamp string is defined, by default, as "`<%Y%M%D.%h%m>`". The first occurrence of the string in the file is up−dated with the time and date information when the buffer is written. The **$timestamp** string may contain any text, and includes the following, magic characters escaped by a percentage (`` `%' ``) character:−

`D` − Day.
`M` − Month.
`Y` − Year.
`h` − Hour.
`m` − Minute.
`s` − Second.

The format string may be redefined into any format. The '`%`' character has to be delimited by another '`%`' if it is to be used in the text (i.e. "`%%`").

The year component (`%Y`) may be a 2 or 4 digit string, depending whether it includes the century. When the time stamping searches for the `%Y` component it searches for either variant and replaces appropriately.

EXAMPLE

The startup file may define the time stamp required as follows:−

```
set−variable $timestamp "Last Modified : %Y/%M/%D %h:%m:%s"
```

Time stamping is performed on the string :−

```
Last Modified : 90/11/23 10:12:01
```

Where the time stamp is modified according to the file (buffer) type then the time stamp string may be modified within the buffer hooks. This allows different files to utilize different time stamping strings. The following example shows how the entry and exit buffer hooks are defined to modify the string:

```
0 define−macro bhook−nroff
    set−variable .timestamp $timestamp
    ; Buffer specific time stamp string.
    set−variable $timestamp "[%Y/%M/%D %h:%m:%s]"
!emacro
0 define−macro ehook−nroff
    ; Restore the existing time stamp.
    set−variable $timestamp .bhook−nroff.timestamp
!emacro
```

On entry to the buffer (buffer becomes current) the buffer hook **bhook−nroff** is executed which stores the current setting and then modifies the time stamp string. On exit from the buffer the buffer hook **ehook−nroff** is executed restoring the time stamp string.

SEE ALSO

buffer−mode(2) time(2m).

$trunc−scheme(5)

NAME

$trunc−scheme − Truncation color scheme.

SYNOPSIS

$trunc−scheme *schemeNum*; Default is `0`

DESCRIPTION

$trunc−scheme sets the color scheme used when drawing a line truncation indicator. The left truncation character (usually a '`$`' char) drawn at the start of the line indicates that the line has been scrolled to the right and therefore the start of the line has been truncated. A right truncation char (also usually a '`$`') drawn at the end of the line indicates the remainder of the line is too long to fit onto the width of the window so the end has been truncated and the indicator drawn.

The *schemeNum* selected must be a color scheme defined with add−color−scheme(2), which identifies the foreground and background color schemes. A hilight scheme can define its own truncation color scheme, see hilight(2) for more information.

NOTES

The truncation characters used are set by the $window−chars(5) variable.

SEE ALSO

$buffer−scheme(5), $global−scheme(5), add−color−scheme(2), hilight(2), $window−chars(5).

$variable−names(5)

NAME

$variable−names − Filtered variable name list

SYNOPSIS

$variable−names *VariableName*

DESCRIPTION

$variable−names must first be initialized to the required filter string, if the variable is evaluated before it is initialized the value will be set to "*ABORT*" and the command will fail.

The filter string can contain wild−card characters compatible with most file systems, namely:−

?

Match any character.

[abc]

Match character only if it is *a*, *b* or *c*.

[a−d]

Match character only if it is *a*, *b*, *c* or *d*.

[^abc]

Match character only if it is not *a*, *b* or *c*.

*****

Match any number of characters.

Note that these are not the same characters used by exact(2m) mode.

Once initialized, evaluating **$variable−names** returns the name of the next variable which matches the filter until no more variables are found, in which case an empty string is returned.

EXAMPLE

The following example prints out the name of all variables to the massage line one at a time. Note that &set(4) is used on the !while(4) statement to avoid evaluating **$variable−names** twice per loop.

```
set-variable $variable-names "*"
!while &not &seq &set #l0 $variable-names ""
    100 ml-write &cat "variable: " #l0
!done
```

NOTES

The list of variables is evaluated when the variable is initialized, variables defined after the initialization will not be included in the list. The list can contain the current buffer's buffer variables (See Variables(4) for more information on the different types of variables).

Using unset−variable(2) to delete a variable which are in the list, before it has be evaluated, will have undefined effects.

SEE ALSO

list−variables(2), $command−names(5).

$version(5)

NAME

$version − MicroEmacs version date−code

SYNOPSIS

$version "*YYYYMMDD*"

DESCRIPTION

$version is a system variable which is defined as the MicroEmacs build date code. This value is fixed at compile time and cannot be changed. The variable may be used in macros to identify incompatibility issues.

EXAMPLE

Given a macro that only operates with a MicroEmacs executable built on or after 1st August 2001 then this macro should check that `$version` is not less than `20010801`. The check may be performed as follows:

```
!if &les $version "20010801"
    ml-write "[Error: MicroEmacs executable is incompatible]"
    !abort
!endif
```

NOTES

This variable was introduced in 2001−08−01, evaluating this variable on an earlier version of MicroEmacs would return the string "`ERROR`" unless an environment variable $version has been defined. "`ERROR`" evaluates to `0` hence the test still operates correctly.

This variable is used in the macro file `me.emf` to check for any macro − executable incompatibility issues.

$window−col(5)

NAME

$window−col − Window cursor column (no expansion)
$window−line − Window cursor line (with narrows)
$window−acol − Window cursor actual column (expansion)
$window−aline − Window cursor actual line (ignore narrows)

SYNOPSIS

$window−col *integer*

0 <= *integer* <= 65535

$window−line *integer*

1 <= *integer* <= n

$window−acol *integer*

0 <= *integer* <= n

$window−aline *integer*

1 <= *integer* <= n

DESCRIPTION

$window−col is defined as the current position of the cursor in the current line in the current window. Column zero is the left hand edge. This differs from **$window−acol** in that tab and special characters only count for 1 character. **$window−col** is valid in the range 0 − *n*.

$window−line is defined as the current buffer line number the cursor is on in the current window. Line numbering starts from 1. **$window−line** is valid in the range 1 − *n*.

$window−aline is identical to **$window−line** except when the current buffer contains narrowed out sections before the current line. In this case **$window−line** will be set to the line number without counting the number of lines in the narrow, whereas **$window−aline** will return the current line number including all lines narrowed out before it. When this variable is set, the line required may lie in a narrowed out section in which case the narrow is automatically removed. See narrow−buffer(2) for more information on narrowing.

$window−acol is defined as the current column of the cursor in the current window. Column zero is the left hand edge. This differs from **$window−col** in that tab and special characters may not count for 1 character.

NOTES

Variable **$window−wcol** was renamed to **$window−acol** in June 2000. Variable **$window−wline** was also removed and a new variable **$window−y−scroll** introduced at this time. The following macro code can be used to calculate the value of the original **$window−wline** variable:

```
&sub &sub $window−line $window−y−scroll 1
```

SEE ALSO

$frame−depth(5), $window−depth(5), $window−width(5), $window−y−scroll(5), narrow−buffer(2).

$window−chars(5)

NAME

$window−chars − Character set used to render the windows

SYNOPSIS

$window−chars "*sting*"; Default is
`"=−#*%=^|#|v*==^^||##||vv**|<−#−>*||<<−−##−−>>** x*[]>\.$$\"`

DESCRIPTION

$window−chars is a fixed length string that defines the set of characters used to render the windows. The characters have fixed indices defined as follows:−

Index 0

The active window mode line separator character, This replaces all *Index 1* characters when the window is current. Default is '='.

Index 1

The inactive window mode line separator character. This character is replaced by *Index 0* characters when the window becomes current. Default is '**−**'.

Index 2

UNIX based platforms only. The **root** or **superuser** indicator character that appears on the mode line. Default is '**#**'.

Index 3

The buffer changed indicator character that appears on the mode line. Default is '*****'.

Index 4

The buffer in view(2m) mode indicator character that appears in the mode line. Default is '**%**'.

Index 5

Single column vertical scroll bar split window horizontally character. Default is '='.

Index 6

Single column vertical scroll bar up−arrow character. Default is '^'.

Index 7

Single column vertical scroll bar upper−shaft character. Default is '**|**'.

Index 8

Single column vertical scroll box character. Default is '**#**'.

Index 9

Single column vertical scroll bar lower−shaft character. Default is '**|**'.

Index 10

Single column vertical scroll bar down−arrow character. Default is '**v**'.

Index 11

Single column vertical scroll bar corner character. Default is '*****'.

Index 12−13

Double column vertical scroll bar split window horizontally character. Default is '**==**'.

Index 14−15

Double column vertical scroll bar up−arrow characters. Default is "^".

Index 16−17

Double column vertical scroll bar upper−shaft characters. Default is "**||**".

Index 18−19

Double column vertical scroll box characters. Default is "**##**".

Index 20−21

Double column vertical scroll bar lower−shaft characters. Default is "**||**".

Index 22−23

Double column vertical scroll bar down−arrow characters. Default is "**vv**".

Index 24−25

Double column vertical scroll bar corner characters. Default is "******".

Index 26−32

Single column horizontal scroll bar. Default is "**|<−#−>***".

Index 33−46

Double column horizontal scroll bar. Default is "**||<<−−##−−>>****".

Index 47

Osd title bar blank character. Default is ' '.

Index 48

Osd title bar right corner kill character. Default is '**x**'.

Index 49

Osd dialog bottom right corner resize character. Default is '*****'.

Index 50

Osd open button character. Default is ' '.

Index 51

Osd close button character. Default is ' '.

Index 52

Displayed tab character (used when $system(5) bit 0x80000 is set). Default is '>'.

Index 53

Displayed new−line character (used when $system(5) bit 0x80000 is set). Default is '\'.

Index 54

Displayed space character (used when $system(5) bit 0x80000 is set). Default is '**.**'.

Index 55

Displayed truncated text to left character (used when the current line is scrolled to the right). Default is '**$**'.

Index 56

Displayed truncated text to right character (used when the current line is longer than the window width). Default is '**$**'.

Index 57

Inserted end of wrapped line character in an ipipe−shell−command(2) buffer. Default is '\'. **EXAMPLE**

The **$window−chars** is typically platform dependent, it's setting is determined by the characters available in character set of the hosting platform. MS−DOS and Microsoft Windows use an OEM font might use the following value:

```
"=-#*%=\C^\xB1 \xB1\C_\CD==\C^\C^\xB1\xB1  \xB1\xB1\C_\C_\C[
\CZ|\CQ\xB1 \xB1\CP\CD||\CQ\CQ\xB1\xB1  \xB1\xB1\CP\CP\C[
\CZ x*  >\\.$$\\"
```

This utilizes character−set specific characters to render some of the window components.

NOTES

- $scroll−bar(5) allows the scroll box to be rendered in reverse video allowing a space to be used for the scroll box.
- Use symbol(3) to determine the displayable characters on the host platform.
- The use of MicroEmacs's extended character set on Windows and XTerm platforms can greatly improve the look and usability of MicroEmacs, see the Extend Char Set option in the Platform page of user−setup(3) and bit `0x10000` of variable $system(5).

SEE ALSO

split−window−horizontally(2), symbol(3), $box−chars(5), $global−scheme(5), $mode−line(5), $mode−line−scheme(5), $scroll−bar(5), $scroll−bar−scheme(5), $system(5).

$window−depth(5)

NAME

$window−depth − Number of text lines in a window
$window−width − Number of character columns in a window

SYNOPSIS

$window−depth *integer*

1 <= *integer* <= $frame−depth

$window−width *integer*

0 <= *integer* <= $frame−width − 1

DESCRIPTION

$window−depth returns the depth (height) of the current window, excluding the mode line, specified in text lines. (i.e. the number of lines of text in the window). The returned value is an integer in the range:

0 − ($frame−depth − **3**)

$window−width returns the width, in characters, of the current window. The returned value is an integer in the range:

0 − $frame−width.

NOTES

These variables can not be set, any attempt to set them results in an error.

SEE ALSO

$frame−depth(5), $frame−width(5), $window−scroll−bar(5), $window−mode−line(5),

$window−flags(5)

NAME

$window−flags − Current window setup flags

SYNOPSIS

$window−flags *bitmask*; Default is `0`

DESCRIPTION

The **$window−flags** variable is used to set or get various behavioural characteristic settings of the current window, it is a bit based flag where:

0x001

If set the width of the window is locked, calls to resize−all−windows(2) will maintained the width of this window whenever possible.

0x002

If set the depth of the window is locked, calls to resize−all−windows(2) will maintained the depth of this window whenever possible.

0x004

If set the buffer being displayed by the window is locked, the user can still manually change the buffer being displayed (by using commands like find−buffer(2)) but commands that pop−up buffers (such as help(2) or find−tag(2)) will not use this window.

0x008

When set the command compare−windows(2) will ignore this window.

0x010

When set the commands like previous−window(2) and next−window(2) will skip this window unless the numeric argument given to the command is used to override the flag setting.

0x020

When set the command delete−other−windows(2) will not delete this window unless the numeric argument given to the command is used to override the flag setting.

0x040

When set the command delete−window(2) will not delete this window unless the numeric argument given to the command is used to override the flag setting.

0x080

When set the window cannot be split using either the split−window−horizontally(2) or split−window−vertically(2) commands.

0x100

If not set the window cannot be deleted if it is the only window without this bit set. This more esoteric feature is utilized by the toolbar, all toolbar windows have this bit set which means that the main user window cannot be delete. **NOTES**

The $window−flags setting is not preserved during a window splitting operation (i.e. using a command like split−window−vertically(2)) as the persistence of these settings can lead to unexpected behaviour.

The toolbar uses bit 0x1000 to indicate that the window is displaying a toolbar tool, this bit should not be used by users and its value should be maintained.

SEE ALSO

next−window(2), delete−other−windows(2), compare−windows(2).

$window−mode−line(5)

NAME

$window−mode−line − Window mode line position
$window−scroll−bar − Window scroll bar (or separator) position

SYNOPSIS

$window−mode−line *integer*

1 <= *integer* <= $frame−depth − 2

$window−scroll−bar *integer*

0 <= *integer* <= $frame−width − 1

DESCRIPTION

$window−mode−line stores the screen line of the current windows mode−line, where screen lines are counted from 0 at the top of the screen. Often used in conjunction with set−cursor−to−mouse(2) and $mouse−y(5) to add more complex mouse functionality.

$window−scroll−bar stores the screen position of the right−hand horizontal window separator line or scroll−bar (see split−window−horizontally(2) and $scroll−bar(5)). A value of greater than $frame−width(5) indicates that there is no right−hand separator column or scroll bar present. Often used in conjunction with $mouse−x(5).

EXAMPLE

In the following example the position of the mouse is checked to see if it is on the mode line of the window, if so then a different action is taken.

```
set-cursor-to-mouse
;   If we are on the mode line then interpret position of
;   the cursor on line to control the screen.
!if &equal $window-mode-line $mouse-y
    !if &less $mouse-x "2"
        menu-main     ; Inform buffer to pop up menu.
    !elif &equal $mouse-x "2"
        delete-window
    !elif &equal $mouse-x "3"
        delete-other-windows
    !elif &equal $mouse-x "4"
        backward-page
    !elif &equal $mouse-x "5"
        forward-page
```

```
            !elif &equal $mouse-x "6"
                recenter
            !elif &equal $mouse-x "7"
                undo
            !endif
        !else
            .....
        !endif
```

SEE ALSO

$mode−line(5), $mouse−x(5), $mouse−y(5), $scroll−bar(5), $mouse−pos(5), set−cursor−to−mouse(2), split−window−horizontally(2).

$window−x−scroll(5)

NAME

$window−x−sroll − Current window X scroll
$window−xcl−sroll − Current window current line X scroll
$window−y−sroll − Current window Y scroll

SYNOPSIS

$window−x−sroll *integer*
$window−xcl−sroll *integer*

0 <= *integer* <= 65535

$window−y−sroll *integer*

0 <= *integer* <= n

DESCRIPTION

$window−x−sroll defines the horizontal scroll position in the current window for all lines except the current line, **$window−xcl−sroll** defines the scroll position for the current line. The variables set how many characters are scrolled off the left hand edge of the current window, the variables are indirectly set by commands such as scroll−left(2), forward−char(2) etc.

$window−y−sroll defines the vertical scroll position in the current window. It sets the number of lines are scroll up off the top of the current window, it is indirectly set by commands such as scroll−up(2), forward−line(2) etc.

EXAMPLE

The following example first stores the current window's buffer position and the window layout. The middle `'...'` section could be replaced with macro code performing any number of operations before the last section which restores the initial position:

```
set-variable #l0 $window-line
set-variable #l1 $window-col
set-variable #l2 $window-xcl-scroll
set-variable #l3 $window-x-scroll
set-variable #l4 $window-y-scroll
    .
    .
    .
set-variable $window-line #l0
set-variable $window-col #l1
```

```
set-variable $window-xcl-scroll #l2
set-variable $window-x-scroll #l3
set-variable $window-y-scroll #l4
```

NOTES

If these variables are set by the user or a macro the value is validated against the $scroll(5) method and the current cursor position which may lead to the variable being reset if found to be invalid. For example, if the current line is 10 when the **$window−y−scroll** is set to 20 the variable will be reset to 0 as a value of 20 will mean the current line is not displayed in the current window.

SEE ALSO

scroll−left(2), scroll−up(2), $scroll(5), $window−line(5), $window−col(5), $window−acol(5).

%compile−com(5)

NAME

%compile−com − Default system compile command line

SYNOPSIS

%compile−com "*string*"; Default is "`make`"

DESCRIPTION

Sets the default command−line inserted into the message line when the compile(3) command is executed. **%compile−com** does not need to be defined to run the **compile** command.

SEE ALSO

compile(3), %grep−com(5).

cygnus(3)

NAME

cygnus − Open a Cygwin BASH window
%cygnus−bin−path − Cygwin BASH directory
%cygnus−hilight − Cygwin shell hilight enable flag
%cygnus−prompt − Cygwin shell prompt

PLATFORM

Windows '95/'98/NT − win32 ONLY

SYNOPSIS

cygnus

%cygnus−bin−path "*path*"
%cygnus−hilight [`0`|`1`]
%cygnus−prompt "*hilightString*"

DESCRIPTION

cygnus creates an interactive BASH shell window within a MicroEmacs buffer window, providing a UNIX command line facility within the Microsoft Windows environment. This is a preferable environment to the MS−DOS shell and is certainly far more comfortable for those people familiar with UNIX.

Within the window BASH commands may be entered and executed, the results are shown in the window. Within the context of the BASH shell window then directory naming conforms to the **cygwin** standard conventions (as opposed to the Microsoft directory naming).

On running **cygnus** a new buffer is created called `*cygnus*` which contains the shell. Executing the command again creates a new shell window called `*cygnus1*`, and so on. If a cygwin window is killed off then the available window is used next time the command is run.

Additional controls are available within the shell window to control the editors interaction with the window. The operating mode is shown as a digit on the buffer mode line, this should typically show "3", which corresponds to *F3*. The operating modes are mapped to keys as follows:−

F2

Locks the window and allows local editing to be performed. All commands entered into the window are interpreted by the editors. **F2** mode is typically entered to cut and paste from the window, search

for text strings etc. In mode 2, a **2** is shown on the mode line.

F3

The normal operating mode, text typed into the window is presented to the shell window. Translation of MicroEmacs commands (i.e. beginning−of−word) are translated and passed to the shell. For interactive use this is the default mode. In mode 3, a **3** is shown on the mode line.

F4

All input is passed to the shell, no MicroEmacs commands are interpreted and keys are passed straight to the shell window. This mode is used where none of the keys to be entered are to be interpreted by MicroEmacs. Note that you have to un−toggle the F4 mode before you can swap buffers as this effectively locks the editor into the window.

F5

Clears the buffer contents. This simply erases all of the historical information in the buffer. The operation of the shell is unaffected.

To exit the shell then end the shell session using "`exit`" or "`C−d`" as normal and then close the buffer. A short cut "`C−c C−k`" is available to kill off the pipe. However, it is not recommended that this method is used as it effectively performs a hard kill of the buffer and attached process

%cygnus−bin−path is a user defined variable that defines the file system location of the *cygwin* directory. This variable MUST be defined within the user start up script in order for the **cygnus** command to start the shell. With a default installation of *cygwin* then the settings are typically defined as:−

Release B19

```
set−variable %cygnus−bin−path "C:/Cygnus/B19/h−i386~1/bin"
```

Release B20

```
set−variable %cygnus−bin−path "c:/cygnus/cygwin−b20/H−i586−cygwin32/bin"
```

%cygnus−hilight is a boolean flag which controls how the cygnus command shell window is hilighted. This value MUST be defined within the user start up script prior to executing cygnus if hilighting is to be enabled; by default hilighting is disabled. A value of 1 enables shell hilighting i.e.

```
set−variable %cygnus−hilight 1
```

%cygnus−prompt is an optional variable that is used in conjunction with **%cygnus−hilight**, it defines the hilighting string identifying the prompt. This allows the prompt to be rendered with a different color. The default prompt is `bash−2.01$` and may be hilighted using a definition:−

```
set−variable %cygnus−prompt "bash−2.01$"
```

The user typically overrides the prompt definition within the BASH startup file, a more appropriate definition of the prompt may be:−

```
        set-variable %cygnus-prompt "^[a-z]*@[^>]*>"
```

NOTES

The **cygnus** command uses the ipipe−shell−command(2) to manage the pipe between the editor and the **bash** shell. The window is controlled by the macro file `hkcygnus.emf` which controls the interaction with the shell.

The macro **cygnus** in `hkcygnus.emf` defines the parameter setup to connect to the cygwin bash shell (Version 19), installed in the default location `c:/cygnus`. If your installation of cygnus is in a different location then correct the macro to match your install location, preferably correct by creating a *mycygnus.emf* file in your user directory simply containing a re−defined **cygnus** macro.

If you have exported some of the cygwin environment variables in your `autoexec.bat` then you will have to figure out for yourself what variables macro *cygnus* needs to export − the current configuration is for a vanilla install.

The **bash** shell is executed with options *i*, for interactive shell and *m* to enable job control.

TESTED CONFIGURATIONS

This configuration has only been tested on a Windows '98 installation, whether this works on NT and Windows '95 (OEM SR2) is unknown.

We have only been running "make" operations in the shell and do not know how the likes of "more", "man" or anything other terminal interaction works.

Tested Configurations

Windows '98 (Pentium 120MHz/Pentium Pro 200MHz/Cyrix 300MHz/Pentium II 450MHz)

> cygwin version B19.3 − this is the original "cygwin" distribution + the latest "coolview.tar.gz" patch.
> cygwin version B20 − the latest cygwin distribution.

BUGS

Break Key

A break in a bash shell is `C−c`, the macros define the key `C−c C−c` to perform the break. This sequence is sent to the process but is not enacted by the shell. This is a property of the Bash shell rather than MicroEmacs.

Slow Response

If you are getting a very slow response from the bash shell then check the directory where *bash* was started. Sometimes there are problems if the shell is started in "`c:/`" (which is typically "`/`") then the *bash* shell is very unresponsive and tends to '*ignore me*' for periods of time. If it is started in another location, i.e. "*c:/temp*" directory, then this problem does not occur.

You can see the start−up location in the top of the buffer when the shell is started.

Prompt at top of buffer

Very, very occasionally the ishell sticks at the top of the buffer with only a couple of lines showing. A swap of the buffers or a quick window resize sorts out the problem. A fix for this problem has been applied but still may occasionally occur.

WinOldAp

Winoldap is created by the Microsoft environment whenever a BASH shell is created. On occasions where processes have terminated badly the user may be prompted to kill these off; this is the normal behaviour of windows. It is strongly advised that all of the BASH processes are killed from within the Bash shell itself and the shell is always exited correctly (i.e. `exit`) before leaving the editor. The Windows operating system for '95/'98 is not particularly resilient to erroneous processes (for those of us familiar with UNIX) and can bring the whole system down. I believe that NT does not suffer from these problems (much).

Locked Input

There are occasions after killing a process the editor appears to lock up. This is typically a case that the old application has not shut down correctly. Kill off the erroneous task (`Alt−Ctrl−Del` − *End Task*) then bring the editor under control using a few `C−g` abort−command(2) sequences. **SEE ALSO**

ipipe−shell−command(2), ishell(3).
Cygnus Win32 home sites **www.cygnus.com** and **www.cygnus.co.uk**

diff(3)

NAME

diff − Difference files or directories
diff−changes − Find the differences from a previous edit session
%diff−com − Diff command line

SYNOPSIS

diff "*oldFile*" "*newFile*"
diff−changes
`%diff-com` "*string*"; Default is "`diff`"

DESCRIPTION

diff executes the **diff(1)** command with the command line set by the %diff−com(5) variable and the user supplied *oldFile* and *newFile*. The output of the command is piped into the ***diff*** buffer and is hilighted to show the changes (GNU diff only).

Your version of **diff(1)** will determine whether it is possible to difference directories.

diff−changes is a simple macro that differences the current buffer and the last backup of the associated file. It is a quick way to determine what has been modified recently. This macro only works if a backup file exists.

%diff−com is the command line that is used to execute a **diff(1)** system command.

For GNU diff then the following command line setting is recommended:−

```
diff --context --minimal --ignore-space-change \
    --report-identical-files --recursive
```

which should be defined in your personal user configuration. This is the default for Linux.

NOTES

diff and **dif−changes** are macros defined in `tools.emf`.

diff(1) must be executable on the system before diff or diff−changes can function.

diff(1) is a standard utility on UNIX systems. For Windows 95/NT a version of GNU **diff** may be found at:

<ftp.winsite.com/ftp/pub/pc/winnt/misc/gnudiff.zip>

For MS−DOS users, a DJGPP port of **diff** is also available on the net. A commercial version of **diff** is also available from MKS.

SEE ALSO

compare−windows(2), compile(3), gdiff(3), grep(3), %grep−com(5).

%ftp−flags(5)

NAME

%ftp−flags − "Configure the FTP console"
%http−flags − "Configure the HTTP console"

SYNOPSIS

%ftp−flags "[c|s|p]" ; Default is undefined.
%http−flags "[c|s|p]" ; Default is undefined.

DESCRIPTION

The **%ftp−flags** and **%http−flags** modify the behavior of the editor during FTP and HTTP file transfers, respectively. (see ftp(3) and find−file(2)).

By default, the flags are disabled, the facilities outlined below are enabled by setting the variable in the user configuration. The flag values for both flags are defined as follows:−

c

Create a console buffer (`*ftp−console*` for ftp, `*http−console*` for http) into which the FTP/HTTP command interactions with the remote server are logged.

s

Show the console whenever a FTP/HTTP operation is performed. The console is popped into the display pane and shows the current interaction status.

p

Show the download progress within the console window ('#' for every 2Kb downloaded)

Typically the following flags are enabled in the *user*.emf file:−

```
set−variable %ftp−flags "csp"
set−variable %http−flags "csp"
```

Once familiar with this facility the console pop−up becomes inconvenient and the flags are typically reduced to:−

```
set−variable %ftp−flags "cp"
set−variable %http−flags "cp"
```

This disables the pop−up feature of the console. Enabling the limited flag set allows some post mortem debugging to be performed if anything goes wrong. The console buffers are manually selected when these flags are set.

NOTES

Note that ftp and http facilities are available on UNIX by default, but must be compiled in for Windows versions.

SEE ALSO

%http−proxy−addr(5), find−file(2), ftp(3).

gdiff(3)

NAME

gdiff − Graphical file difference
%gdiff−com − Gdiff diff(1) command line

SYNOPSIS

gdiff "*version1*" "*version2*"

`%gdiff-com` "*string*"; Default is "`diff -c -w`"

DESCRIPTION

gdiff is a macro utility that facilitates the merging of two files (typically with different modification revisions). The changes between the revisions are hilighted with color, allowing modification regions and lines to be selected for the generation of a newer revision file, which might encompass selected modifications from each of the base revisions.

gdiff executes the **diff(1)** command with the command line set by the %gdiff−com(5) variable and the user supplied *version1* and *version2*. The output is displayed in two buffer windows, side by side, and the differences in the lines are hilighted to show the changes. In addition the content of the two buffers is *normalized* such that both windows are aligned at the same line position, allowing the changes in the text to be viewed in both windows at the same time.

Whilst in **gdiff** view mode then both scroll bars (if visible) are *locked*, such that either scrolls BOTH windows at the same time. Other key commands are disabled, as are the menu interactions. The short cut keys are defined as follows:−

`esc h/A-h` − View the help page.

Invokes the display of a OSD help box, summarizing the interaction commands

`C-up` − Move to previous difference

Moves to the previous changed region above the current cursor position.

`C-down` − Move to next difference

Moves to the next changed region below the current cursor position.

`left mouse button`
`space`
`enter`

`r` − Select difference version

Selects the difference version of the currently selected window. The region is hilighted as the required region to be incorporated into the new revision.

`R` − Select neither version.

Marks both regions as not required.

`l` − Line select current version

Selects the current line from the region as being included, without including ALL of the region modifications.

`L` − Line select neither version

Discards lines from both revisions of the file.

`g` − Globally selects the current version.

Shortcut allows ALL modifications to the current side to be accepted. This is typically the fastest method to select all changes, minor region adjustment may then be performed on those regions which are inappropriately included by the selection.

`G` − Globally selects neither version.

Marks all regions as not being acceptable.

`C−x C−s` − Save current side

Saves the current window to the specified file, merging the selected changes between the two revisions. Note that the save only operates iff all hilighted changes have been selected.

`C−x C−w` − Save current side as

Same as **Save current side** except the user is prompted to enter a new filename to which the modifications are written.

`C−x k` − Exit graphical diff

Exits the **gdiff** utility. **Hilighting**

The hilighting within the windows is dependent upon the color scheme selected, in general the following hilights apply:−

normal text

No change

cyan/grey

Addition/removal of line(s)/region(s) between files.

yellow

Modification in line(s)/region(s).

green/red

Selected region, red or green is attributed to a selection for each window. **NOTES**

gdiff is a macro defined in `gdiff.emf`, inspired by the GNU utility of the same name **gdiff(1)**

diff(1) must be executable on the system before **gdiff** can function. The **diff(1)** invocation must include the *context* difference, which annotates the differences with a `+`, **−** or `!` markers. **diff(1)** is typically invoked with the options **−c −w**.

diff(1) is a standard utility on UNIX systems. For Windows 95/NT a version of GNU **diff** may be found at:

> <*ftp.winsite.com/ftp/pub/pc/winnt/misc/gnudiff.zip*>

For MS−DOS users, a DJGPP port of GNU **diff** is also available on the net. A commercial version of **diff** is also available from MKS.

SEE ALSO

compare−windows(2), compile(3), **diff(1)**, gdiff(3f), grep(3), %grep−com(5).

%grep−com(5)

NAME

%grep−com − Grep command line

SYNOPSIS

`%grep−com` "*string*"; Default is "`grep` "

DESCRIPTION

Sets the command line used to execute a **grep(1)** system command. The output of the grep(3) execution should include both file and line number information so that the command get−next−line(2) can be used properly. This is not defined by default and the **grep** command will not execute until it is defined.

grep(1) is typically used with the **−n** option which produced line numbering information which drives the get−next−line(2) command.

EXAMPLE

The following example shows how the **grep** strings are defined.

```
set−variable %grep−com "grep −n "
0 add−next−line "*grep*"
add−next−line "*grep*" "%f:%l:"
```

This definition corresponds to a **grep** output such as:−

```
m5var000.5:13:Sets the  number of seconds to wait
m5var000.5:14:temporary file to t seconds. A
m5var000.5:15:Note than the  temporary
m5var000.5:17:saving a buffer.  Backup  files are
m5var000.5:24:On unlimited  length  file  name  systems
```

where **grep** produces file and line number information for every match.

Use add−next−line(2) to define the line pattern produced by **grep**. Some versions of **grep** place the file name on a single line matches within the file occur on subsequent lines. In this case additional *add−next−line* patterns may be defined to cater for the **grep** output as follows:

```
set−variable %grep−com "grep /n "
0 add−next−line "*grep*"
add−next−line "*grep*" "File: %f:"
add−next−line "*grep*" "%l:"
```

This definition would be used with a **grep** output such as:−

```
File:m5var000.5:
13:Sets the  number of seconds to wait
14:temporary file to t seconds. A
15:Note than the  temporary
17:saving a buffer.  Backup  files are
24:On unlimited  length  file  name  systems
File:m5var001.5:
```

NOTES

grep(1) is a standard utility on UNIX systems. For Windows 95/NT a version of GNU **grep** may be found at:

<ftp.winsite.com/ftp/pub/pc/winnt/misc/gnugrep.zip>

For MS−DOS users, a DJGPP port of **grep** is also available on the net. A commercial version of **grep** is also available from MKS.

SEE ALSO

add−next−line(2), **grep(1)**, grep(3), add−next−line(2).

%http−proxy−addr(5)

NAME

%http−proxy−addr − Set HTTP proxy server address
%http−proxy−port − Set HTTP proxy server port

SYNOPSIS

%http−proxy−addr "*proxy−addr*"
%http−proxy−port "*port−number*"; Default is `80`

DESCRIPTION

If the **%http−proxy−addr** variable is set all HTTP file loading requests, using commands like find−file(2), are sent via the given proxy server. **%http−proxy−port** should be set to the proxy servers port number, defaulting to `80` if not set. These variables are typically set in your `<user>.emf` setup file, e.g.:

```
set-variable %http-proxy-addr "proxy.foobar.com"
set-variable %http-proxy-port "8080"
```

NOTES

Note that http is available on UNIX by default, but must be compiled in for win32 versions.

SEE ALSO

%http−flags(5), find−file(2), ftp(3).

%tag−file(5)

NAME

%tag−file − Tags file name
%tag−template − Tag file search string
%tag−option − Tag file search option

SYNOPSIS

%tag−file "*fileName*"
%tag−template "*string*"
%tag−option "*string*"

DESCRIPTION

The **%tag−file** and **%tag−template** variables must be defined for find−tag(2) to work, they define the information required to locate tag references.

%tag−file is the name of the tag file to be used, usually set to "**tags**". **%tag−template** is a regular expression search string used to identify tags in a tag file. For example, a tag usually consists of a name "`%[^\t]`" followed by a tab "`\t`" followed by the file name that contains the function "`%[^\t]`" followed by another tab, followed by the search string and end of line "`%[^\n]\n`", i.e.

```
set-variable %tag-template  "%[^\t]\t%[^\t]\t%[^\n]\n"
```

This would match a **vi(1)** tag string definition, as created by the UNIX utility **ctags(1)**. The tags file typically contains entries such as:−

```
$auto-time    m5var000.5 /^.XI $auto-time - "Automatic buffer"$/
$buffer-bhook m5var002.5 /^.XI $buffer-bhook - "Buffer macro"$/
$buffer-ehook m5var002.5 /^.XI $buffer-ehook - "Buffer macro"$/
```

The **tag−template** definition is modified to match the output of the **ctags(1)** utility. The format of the tags file may differ from platform to platform, typically the differences are encountered in the line contents field which is usually defined as `/ .... /` for a forward search tag and `? .... ?` for a reverse search tag. Note that a tag's search string typically starts with the character '`^`' and ends with '`$`' which indicate the start and end of the line. The variable fields are expected to be in conventional order of *label*, *filename* and *lineText*.

%tag−option is a user defined variable that modifies the behavior of find−tag(2). This is defined as a string, where each character identifies an option, when undefined then default behavior is assumed. The options are defined as:−

m − Enable multiple tags support

Allows a single tag to be present multiple times in the tag file, typically used when a function is defined multiple times. When enabled **find−tag** can be used to loop through all definitions of a given tag.

r − recursive tags file

By default, the **tags** file is assumed to reside in the current directory location. The **r** option enables an ascending search up the directory hierarchy from the current directory position in search of a recursively generated tags file.

c − Continue recursive tag search

Used in conjunction with flag **r**; when not specified, the recursive searching of a tag stops at the first tag file found, regardless of whether the given tag was located in the found tag file. If this flag is given and the tag was not found in the first tag file, the recursive search continues. This allows local tag files to be created and regularly maintained, yet still being able to access a higher level tag file when required.

Modifications to this variable should be made in the *user*.emf file, e.g. To enable multi recursive ascent tag searching define:−

```
        set−variable %tag−option  "mrc"
```

NOTES

Note that GNU Emacs uses it's own tag file format generated by **etags(1)** which does not contain the appropriate information to drive the MicroEmacs '02 **find−tag** command.

The above settings should support the extended version 2 tag file format which has an extra tag type field at the end of each line.

SEE ALSO

ctags(1), ctags(3f), find−tag(2), **vi(1)**.

.calc.result(5)

NAME

.calc.result − Last calc calculation result

SYNOPSIS

.calc.result *integer*

DESCRIPTION

.calc.result is used to store the result of the last calculation made by calc(3).

The "*LR*" (Last Result) in the next calculation is substituted with this value.

SEE ALSO

calc(3).

Macro Language Glossary

MACRO LANGUAGE GLOSSARY

The following is a list of all of the macro language commands available in **MicroEmacs '02**.

Functions

All functions are denoted by a **&** prefix as follows:−

&abs(4)	Absolute value of a number
&add(4)	Add two numbers
&and(4)	Logical AND operator
&atoi(4)	ASCII to integer conversion
&band(4)	Bitwise AND operator
&bmode(4)	Determine buffer mode
&bnot(4)	Bitwise NOT operator
&bor(4)	Bitwise OR operator
&bxor(4)	Bitwise XOR operator
&cat(4)	Concatenate two strings together
&cbind(4)	Return the command a key is bound to
&cond(4)	Conditional expression operator
&dec(4)	Pre−decrement variable
&divide(4)	Division of two numbers
&equal(4)	Numerical equivalence operator
&exist(4)	Test if a variable or command exists
&find(4)	Find a file on the search path
&gmode(4)	Determine global mode
&great(4)	Numerical greater than operator
&inc(4)	Pre−increment variable
&indirect(4)	Evaluate a string as a variable
&inword(4)	Test for a word character
&irep(4)	Case insensitive replace string in string
&isequal(4)	Case insensitive String equivalence operator
&isin(4)	Case insensitive test for string in string
&itoa(4)	Integer to ASCII conversion
&kbind(4)	Return the key a command is bound to
&ldel(4)	Delete list item
&left(4)	Return the left most characters from a string
&len(4)	Return the length of a string
&less(4)	Numerical less than operator
&lfind(4)	Find list item
&lget(4)	Get list item
&linsert(4)	Insert list item
&lset(4)	Set list item
&mid(4)	Return a portion (middle) of a string

&mod(4) Modulus of two numbers
&multiply(4) Multiply two numbers
&nbind(4) Return the numerial argument of a binding
&nbmode(4) Determine named buffer mode
&negate(4) Negation of two numbers
&not(4) Logical NOT operator
&opt(4) MicroEmacs optional feature test
&or(4) Logical OR operator
&pdec(4) Post−decrement variable
&pinc(4) Post−increment variable
&reg(4) Retrieve a registry value (with default)
&rep(4) Replace string in string
&right(4) Return the right most characters from a string
&risin(4) Recursive case insensitive test for string in string
&rsin(4) Recursively test for string in string
&sequal(4) String equivalence operator
&set(4) In−line macro variable assignment
&sgreat(4) String greater than operator
&sin(4) Test for string in string
&sless(4) String less than operator
&slower(4) Return the string converted to lower case
&sprintf(4) Formatted string construction
&stat(4) Retrieve a file statistic
&sub(4) Subtract two numbers
&supper(4) Return the string converted to upper case
&trboth(4) Return string trimmed of white chars on both sides
&trleft(4) Return string trimmed of white chars on left side
&trright(4) Return string trimmed of white chars on right side
&which(4) Find a program on the path
&xirep(4) Regex case insensitive Replace string in string
&xisequal(4) Case insensitive regex String equivalence operator
&xrep(4) Regex replace string in string
&xsequal(4) Regex string equivalence operator

Directives

The macro directives are denoted by a **!** prefix as follows:−

!abort(4) Exit macro with a FALSE status
!bell(4) Sound audio alarm
!continue(4) Restart a conditional loop
!done(4) End a conditional loop
!ehelp(4) Terminate a help definition
!elif(4) Conditional test statement, continuation
!else(4) Conditional alternative statement
!emacro(4) Terminate a macro definition
!endif(4) Conditional test termination
!force(4) Ignore command or macro status

!goto(4) Unconditional labeled jump
!if(4) Conditional test statement
!jump(4) Unconditional jump
!nmacro(4) Ignore command or macro status
!repeat(4) Conditional loop (post testing)
!return(4) Exit macro with a TRUE status
!tgoto(4) Conditional labeled jump
!tjump(4) Unconditional relative branch
!until(4) Test a conditional loop
!while(4) Conditional loop

Variables

The macro variables are denoted by a **%** for user variables; **#** for a register variable and **@** for an interactive variable as follows:−

@0(4) Macro arguments (macro name)
@1(4) Macro arguments (first argument)
@2(4) Macro arguments (second argument)
@?(4) Macro arguments (numeric argument given)
@cc(4) Current command name
@cck(4) Current command key
@cg(4) Get a command name from the user
@cgk(4) Get a key from the user
@cl(4) Last command name
@clk(4) Last command key
@cq(4) Get a quoted command name from the user
@cqk(4) Get a quoted key from the user
@fs(4) Frame store variable
@hash(4) Macro arguments (numeric argument value)
@mc(4) Message line character input request
@ml(4) Message line input request
@mn(4) Message line input as normal request
@mna(4) All input from Message line as normal
@mx(4) Message line input by executing command
@mxa(4) All input from Message line by executing command
@p(4) Macro arguments (calling macro name)
@s0(4) Last search's whole match string
@s1(4) Last search's first group value
@s2(4) Last search's second group value
@wc(4) Extract character from the current buffer
@wl(4) Extract a line from the current buffer
@y(4) Yank buffer variable
BufferVariables(4) Buffer variables
CmdVariables(4) Command variables
CommandVariables(4) Last, current and get a command key/name
CurrentBufferVariables(4) Extract information from the current buffer
MacroArguments(4) Arguments to macros

MacroNumericArguments(4) Numeric arguments to macros
MessageLineVaraibles(4) Prompt the user for input on message line
RegisterVariables(4) Register variables
SearchGroups(4) Last search group values
Variables(4) User defined macro variables

&abs(4)

NAME

&abs, &add, &sub, &mul, &div, &mod, &neg, &inc, &dec, &pinc, &pdec − Numeric macro operators

SYNOPSIS

&abs *num1*
&add *num1 num2*
&sub *num1 num2*
&multiply *num1 num2*
&divide *num1 num2*
&mod *num1 num2*
&negate *num*

&inc *variable increment*
&dec *variable decrement*
&pinc *variable increment*
&pdec *variable decrement*

DESCRIPTION

The numeric operators operate on variables or integers to perform integer computations, returning the integer result of the operation. The contents of the variables are interpreted as signed integers typically with a dynamic range of 2^31 <= *num* <= 2^31−1.

The operators may all be abbreviated to their three letter abbreviation (i.e. **&multiply** may be expressed as **&mul**). In all cases the first argument is completely evaluated before the second argument.

&abs *num1*

Returns the absolute value of *num1* i.e. if *num1* is positive it returns *num1*, else −*num1*

&add *num1 num2*

Addition of two numbers *num1* and *num2*. i.e. *num1* + *num2*

&sub *num1 num2*

Subtract the second number *num2* from the first *num1* i.e. *num1* − *num2*.

&multiply *num1 num2*

(Signed) Multiply *num1* by *num2*. i.e. *num1* * *num2*. **&mul** is the three letter abbreviation.

&div *num1 num2*

Divide the first number *num1* by the second *num2*, returning the integer result. i.e. *num1* / *num2*. **&div** is the three letter abbreviation.

&mod *num1 num2*

Divide the first number *num1* by the second *num2*, returning the integer remainder. i.e. *num1* % *num2*.

&negate *num*

Negate the integer (multiply by −1) i.e. −*num*. **&neg** is the three letter abbreviation.

Expression evaluation is prefix. Operators may be nested using a pre−fix ordering, there is no concept of brackets (in−fix notation). The expression `(2 * 3) + 4` is expressed as:−

```
&add &mul 2 3 4
```

conversely `2 * (3 + 4)` is expressed as:−

```
&mul 2 &add 3 4
```

The pre/post incrementing and decrementing operators provide a mechanism for stepping through indexed information without incurring the overhead of providing multiple statements to perform assignment operations. The *variable* argument MUST be the name of a variable, it cannot be an expression or an indirection. The *increment* may be any integer expression (including another auto (dec)increment). Note that *variable* is re−assigned with it's new value within the operator, therefore use with care when performing multiple (dec)increments within the same statement line. The four operators are defined as follows:

&inc *variable increment*

Pre−increment the *variable* by *increment*, returning the incremented value i.e. *variable* += *increment*.

&dec *variable decrement*

Pre−decrement the *variable* by *decrement*, returning the decrement value i.e. *variable* −= *decrement*.

&pinc *variable increment*

Post−increment the *variable* by *increment*, returning the pre−increment value i.e. *variable*++., where the ++ value is determined by *increment*. The return value is the value of *variable* as passed by the caller, the next reference to *variable* uses the *variable*+*increment* value.

&pdec *variable decrement*

Post−decrement the *variable* by *decrement*, returning the pre−decrement value i.e. *variable*−−, where the −− value is determined by *decrement*. **EXAMPLE**

Add two numbers together and assign to a variable:−

```
set-variable %result &add %num1 %num2
```

Increment `%result` by 1 and add to `%result2`

```
set-variable %result  &add %result 1
set-variable %result2 &add %result2 %result
```

The previous example could have used the increment operators to achieve the same result in a single operation e.g.

```
set-variable %result2 &add %result2 &inc %result 1
```

SEE ALSO

Variable Functions, &great(4).

&and(4)

NAME

&and, &or, &not, &equal, &sequal − Logical macro operators

SYNOPSIS

&and *log1 log2*
&or *log1 log2*
&not *log*

&equal *num1 num2*
&great *num1 num2*
&less *num1 num2*

DESCRIPTION

The logical testing operators perform comparison tests, returning a boolean value of TRUE (1) or FALSE (0).

The functions may all be abbreviated to their three letter abbreviation (i.e. **&great** may be expressed as **&gre**). In all cases the first argument is completely evaluated before the second argument. Logical operators include:−

&and *log1 log2*

TRUE if the logical arguments *log1* and *log2* are both TRUE.

&or *log1 log2*

TRUE if either one of the logical arguments *log1* and *log2* are TRUE.

&not *log*

Logical NOT. Returns the opposite logical value to *log*.

The numerical logical functions operate with integer arguments:

&equal *num1 num2*

TRUE. If numerical arguments *num1* and *num2* numerically equal. Abbreviated form of the function is **&equ**.

great *num1 num2*

TRUE. If numerical argument *num1* is greater than *num2*. Abbreviated form of the function is **&gre**.

&less *num1 num2*

TRUE. If numerical argument *num1* is less than *num2* Abbreviated form of the function is **&les**.

Evaluation of the logical operators are left to right, the leftmost argument is fully evaluated before the next argument. The operator ordering is prefix notation (see &add(4) for an example of prefix ordering).

EXAMPLE

Test for integers in the range greater than 12:

```
!if &great %i 12
    ...
```

Test for integers in the range 8−12, inclusive

```
!if &and &great 7 &less 13
    ...
```

NOTES

MicroEmacs always evaluates all arguments operators BEFORE the result is obtained, this differs from most programming languages. Consider the following example:

```
!if &and &bmod "edit" &iseq @mc1 "Save buffer first [y/n]? " "nNyY" "y"
    save−buffer
!endif
```

This would not not work as the user may expect, the user would be prompted to save every time regardless of whether the buffer has been changed. Instead the following should be used:

```
!if &bmod "edit"
    !if &iseq @mc1 "Save buffer first [y/n]? " "nNyY" "y"
        save−buffer
    !endif
!endif
```

SEE ALSO

Variable Functions, &add(4), &sequal(4), &sin(4), &cond(4).

&atoi(4)

NAME

&ato, &gmod, &bmo, &ind, &inw, &exi − Miscellaneous functions

SYNOPSIS

&atoi *char*
&itoa *num*

&gmode *mode*
&bmode *mode*
&nbmode *buffer mode*
&inword *char*

&indirect *str*

&exist *str*

DESCRIPTION

These are a selection of miscellaneous functions providing tests and exchanging of information.

The functions may all be abbreviated to their three letter abbreviation (i.e. **&indirect** may be expressed as **&ind**). In all cases the first argument is completely evaluated before the second argument.

&atoi *char*

Converts the given character *char* to it's ASCII number which is returned. (see **&itoa**). Abbreviated command is **&ato**.

&itoa *num*

Converts an integer *num* to it's ASCII character representation which is returned to the caller. Abbreviated command is **&ito**.

&gmode *mode*

Returns `1` if the given mode *mode* is globally enabled. Allows macros to test the global mode state (see Operating Modes). Abbreviated command is **&gmo**.

&bmode *mode*

Returns `1` if the mode *mode* is enabled in the current buffer. Allows macros to test the state of the buffer mode. Abbreviated command is **&bmo**.

&nbmode *buffer mode*

Returns `1` if the mode *mode* is enabled in buffer *buffer* . Allows macros to test the state of a buffer mode other than the current. Abbreviated command is **&nbm**.

&inword *char*

`TRUE`. If the given character *char* is a 'word' character, see forward−word(2) for a description of a 'word' character. Abbreviated command is **&inw**.

&indirect *str*

Evaluate *str* as a variable. The *str* argument is evaluated and takes the resulting string, and then uses it as a variable name. i.e. a variable may reference another variable which contains the data to be referenced. Abbreviated command is **&ind**.

&exist *str*

Tests for the existance of *str* which may be a variable or a command/macro name, returning `TRUE` if the variable or command does currently exist. Abbreviated command is **&exi**. **EXAMPLE**

The **&exi** function is etremely useful in initializing, for example:

```
!if &not &exi %my-init
    ; %my-init is not yet defined so this is the first call
    set-variable %my-init 1
    .
    .
```

Or in all the file hooks a user defined extension is checked for and executed if defined:

```
define-macro fhook-c
    .
    .
    ; execute user extensions if macro is defined
    !if &exi my-fhook-c
        my-fhook-c
    !endif
!emacro
```

The **&ind** function deserves more explanation. **&ind** evaluates its string argument *str*, takes the resulting string and then uses it as a variable name. For example, given the following code sequence:

```
; set up reference table

set-variable  %one      "elephant"
set-variable  %two      "giraffe"
set-variable  %three    "donkey"
```

```
        set-variable  %index "%two"
        insert-string &ind %index
```

the string "`giraffe`" would have been inserted at the point in the current buffer.

The **&bmode** invocation allows a calling macro to determine the buffer mode state (see Operating Modes). Consider the following example which is a macro to perform a case insensitive alphabetic sort using the sort−lines(2) function. **sort−list** sorts according to the state of the exact(2m) mode, hence the macro has to determine the buffer state in order to be able to do the sort.

```
        define-macro sort-lines-ignore-case
            set-variable #l0 &bmod exact
            -1 buffer-mode "exact"
            !if @?
                @# sort-lines
            !else
                sort-lines
            !endif
            &cond #l0 1 -1 buffer-mode "exact"
        !emacro
```

The **&inword** function is shown in the following example. In this case the mouse is positioned over a word. The **&inword** function is used to determine if the cursor is on a valid word character, if so the cursor is placed at the start of the word.

```
        define-macro mouse-control-drop-left
            set-cursor-to-mouse
            !if &inword @wc
                backward-word
                set-mark
                forward-word
            !else
                ...
            !endif
            copy-region
            set-cursor-to-mouse
        !emacro
```

SEE ALSO

Operating Modes, Variable Functions, &sprintf(4), &equal(4).

&band(4)

NAME

&band, &bor, &bnot, &bxor − Bitwise macro operators

SYNOPSIS

&band *num1 num2*
&bor *num1 num2*
&bxor *num1 num2*
&bnot *num*

DESCRIPTION

The bitwise operators perform bit operations on numeric values returning a numerical result of the operation.

The functions may all be abbreviated to their three letter abbreviation (i.e. **&band** may be expressed as **&ban**). In all cases the first argument is completely evaluated before the second argument.

&band *num1 num2*

Bitwise AND of *num1* and *num2* i.e. *num1* & *num2*.

&bor *num1 num2*

Bitwise (inclusive) OR of *num1* and *num2* i.e. *num1* | *num2*.

&bxor *num1 num2*

Bitwise (exclusive OR) XOR of *num1* and *num2* i.e. *num1* ^ *num2*.

&not *num*

Bitwise NOT operator of *num*, inverts the state of all bits i.e. ~*num*.

Evaluation of the bitwise operators are left to right, the leftmost argument is fully evaluated before the next argument. The operator ordering is prefix notation (see &add(4) for an example of prefix ordering).

SEE ALSO

Variable Functions, &add(4), &and(4), &negate(4), &or(4).

&cat(4)

NAME

&cat, &lef, &rig, &mid, &len, &slo, &trb − String macro operators

SYNOPSIS

&cat *str1 str2*
&lef *str len*
&right *str index*
&mid *str index len*

&len *str*

&slower *str*
&supper *str*

&trboth *str*
&trleft *str*
&trright *str*

DESCRIPTION

The string operators operate on character strings (**%** or **$** variables), performing general string manipulation, returning a string result.

The operators may all be abbreviated to their three letter abbreviation (i.e. **&right** may be expressed as **&rig**). In all cases the first argument is completely evaluated before the second argument.

&cat *str1 str2*

Concatenate two string *str1* with *str2* to form a new string. i.e. *str1str2*

&lef *str len*

Return *len* leftmost characters from *str*. If *str* length is shorter than *len* then the string itself is returned. A *len* of zero returns the empty string.

&rig *str index*

Returns the rightmost characters of string *str* from index *index*. This function causes some confusion, consider **&lef** and **&rig** to be the string equivalents of their integer counterparts &div and &mod; **&rig** returns the remainder of the equivalent **&lef** function. Invocation with *index* set to zero returns *str*.

&mid *index len*

Extracts a sub−string from string *str*, starting at position *index* of length *len*.

&len *str*

Returns the integer length of the string (number of characters).

&slower *str*

Returns the given string with all upper case characters converted to lower case.

&supper *str*

Returns the given string with all lower case characters converted to upper case.

&trboth *str*

Returns the given string trimmed of white spaces (i.e. ' ', '`\t`', '`\r`', '`\n`', '`\Cl`' and '`\Ck`') from both sides of the string.

&trleft *str*

Returns the given string trimmed of white spaces from the left side of the string only.

&trright *str*

Returns the given string trimmed of white spaces from the right side, or end, of the string only.

Evaluation of the strings is left to right, the leftmost argument is fully evaluated before the next argument. The operator ordering is prefix notation (see &add(4) for an example of prefix ordering).

EXAMPLE

Concatenate two strings `abc` and `def` together:−

```
set−variable %result &cat "abc" "def"
```

To concatenate three strings `abc`, `def` `ghi` together:

```
set−variable %result &cat "abc" &cat "def" "ghi"
```

or, a slightly different ordering:

```
set−variable %result &cat &cat "abc" "def" "ghi"
```

Retrieve the leftmost character of a string variable, modify the variable to contain the remainder.

```
set−variable %foo "abcdef"
```

```
set-variable %c   &lef %foo 1
set-variable %foo &rig %foo 1
```

Where `%c` = "`a`"; `%foo` = "`bcdef`" following evaluation.

To retrieve the characters `cde` into variable `%result` from the string "`abcdef`" use:

```
set-variable %result &mid "abcdef" 2 3
```

To retrieve the rightmost character from the string:

```
set-variable %foo "abcdef"
set-variable %result &rig %foo &sub &len %foo 1
```

or the same result could be achieved using **&mid**:

```
set-variable %result &mid %foo &sub &len %foo 1 1
```

To get an input string from the user which is free of spaces at the start and end:

```
set-variable %result &trb @ml "Enter string"
```

NOTES

The original **MicroEMACS** "**&rig** *str n*" function returns the last *n* characters from the string *str* this differs from the definition of **&rig** in this implementation. As most string decomposition is performed left to right, and to make **&lef** and **&rig** complement each other, the indexing of the function has been modified.

SEE ALSO

Variable Functions, &sin(4), &sequal(4), &lget(4), &sprintf(4).

&cbind(4)

NAME

&cbind, &kbind, &nkind − Command/key binding operators

SYNOPSIS

&cbind *key*
&kbind *n command*
&nbind *key*

DESCRIPTION

&cbind returns the command bound to the given key sequence, **&kbind** can be abbreviated to **&kbi**. If the key is not bound then **&kbind** returns the string "*ERROR*".

&nbind returns the numerical argument associated with the given key binding, **&nbind** can be abbreviated to **&nbi**. If the key is not bound then **&nbind** returns the string "*ERROR*", if the binding has no argument then an empty string (`""`) is returned.

&kbind returns a key sequence bound to the given *command* with the given numerical argument *n*. If no binding can be found then **&kbind** returns an empty string (`""`).

EXAMPLE

The following example waits for the user to press a key, then prints what command the key is bound to.

```
ml-write "Enter key: "
set-variable #l0 @cgk
ml-write &spr "%s is bound to %s" #l0 &cbin #l0
```

NOTES

In March 2001 **&kbind** was renamed **&ckind** and a new **&nkind** and **&kbind** added.

SEE ALSO

Variable Functions, global−bind−key(2).

&cond(4)

NAME

&cond − Conditional expression operator

SYNOPSIS

&cond *log expr1 expr2*

DESCRIPTION

The conditional expression **&cond** provides an alternative way to write !if−!else−!endif constructs, e.g.:−

```
!if &gre %a %b
    set−variable %z %a
!else
    set−variable %z %b
!endif
```

may be replaced with a conditional expression, breaking down the components then

log is **&gre %a %b**
expr1 is **%a**
expr2 is **%b**

rewriting the expression we get:

```
set−variable %z &cond &gre %a %b %a %b
```

This is far more concise, albeit a little less readable, but does improve the performance of macros as there is less information to interpret.

The **&cond** operator accepts three fields, ALL fields are evaluated although only one of the results *expr1* or *expr2* is used. The *log* field is a logical value, if it is non−zero (`TRUE`) then the result of the *expr1* evaluation is used, otherwise the result of *expr2* is used.

It should be noted that the conditional expression may be used in any construct i.e. &add(4), &cat(4), etc. the *expr* arguments may be strings, numbers or booleans the resultant value of the *expr* arguments is simply returned to the calling expression.

SEE ALSO

Variable Functions, &add(4), &great(4).

&find(4)

NAME

&find − Find a file on the search path
&which − Find a program on the path

SYNOPSIS

&find *<basename> <extension>*
&which *<progname>*

DESCRIPTION

&find searches for a named file *<basename><extension>* on the MicroEmacs '02 search path defined by the variable $search−path(5) (initialized from the environment variable $MEPATH(5)). Each path component defined in **$search−path** is prepended to the constructed file name and it's existence is tested. If the file exists, then the FULL path name of the file is returned to the caller, otherwise `ERROR`.

<basename>

The base name of the file, excluding any extension.

<extension>

The extension of the file name, this must be specified with the extension delimiter, typically dot ('`.`'). A `NULL` string (e.g. '`""`') may be specified if no extension is required.

&which searches for the given executable program *<progname>* on the system program search path defined the the environment variable **$PATH**.

USAGE

&find is typically used with insert−file(2) and find−file(2) within macro scripts, and is used to locate user specific files.

EXAMPLE

The following example uses **&find** to locate the uses 'C' template file. Given a **$search−path** setting of `/usr/bob/emacs:/usr/local/microemacs`:−

```
insert-file &find "c" ".etf"
```

Would insert the file `/usr/bob/emacs/c.etf` if it existed, else the file `/usr/local/microemacs/c.etf` if it exists.

SEE ALSO

Variable Functions, find−file(2), $search−path(5), insert−file(2).

&rep(4)

NAME

&rep, &irep, &xrep, &xirep − Replace string in string functions

SYNOPSIS

&rep *str1 str2 str3*
&irep *str1 str2 str3*
&xrep *str1 str2 str3*
&xirep *str1 str2 str3*

DESCRIPTION

These functions search for *str2* in *str1*, replacing it with *str3*, returning the resultant string.

The functions may all be abbreviated to their three letter abbreviation (i.e. **&xirep** may be expressed as **&xir**). In all cases the first argument is completely evaluated before the second and third arguments.

&rep *string search replace*

Searches for the *search* string in the given *string* using a simple case sensitive exact match algorithm. Any occurrences are removed from *string* and *replace* is inserted in its place. Either of the 3 input strings can be the empty string (`""`).

&irep *string search replace*

&irep is identical to **&rep** except a case insensitive search algorithm is used.

&xrep *string regex−search regex−replace*

&xrep can be used to access the more powerful regular expression searching capabilities. The function is similar to **&rep** except it takes a regex search string and the replacement string may also refer to all or part of the matched string. See Regular Expressions for information on the *regex* format.

&xirep *string regex−search regex−replace*

&xirep is identical to **&xrep** except a case insensitive regex search is used. **EXAMPLE**

The following example turns a UNIX format file name (using a '`/`' to divide directories − like MicroEmacs) into an windows format name (using a '`\`'):

```
set-variable #l0 &rep #l0 "/" "\\"
```

The following example replaces one or more white spaces in the variable with a single space, this is an easy way to remove unnecessary spaces:

```
set-variable #l0 "This   is   not   so    spacey    after   xrep"
set-variable #l0 &xrep #l0 "\\s +" " "
ml-write #l0
```

SEE ALSO

Operating Modes, Variable Functions, &sequal(4), &sin(4).

&sequal(4)

NAME

&seq, &iseq, &sle, &sgre, &xseq, &xiseq − String logical macro operators

SYNOPSIS

&sequal *str1 str2*
&isequal *str1 str2*
&sless *str1 str2*
&sgreat *str1 str2*

&xsequal *str1 regex*
&xisequal *str1 regex*

DESCRIPTION

The string logical testing operators perform string comparison tests, returning a boolean value of TRUE (1) or FALSE (0).

The functions may all be shortened to their three letter abbreviation (i.e. **&sequal** may be expressed as **&seq**). In all cases the first argument is completely evaluated before the second argument. String logical operators include:−

&sequal *str1 str2*

TRUE if the two strings *str1* and *str2* are the same. Abbreviated form of the function is **&seq**.

&sless *str1 str2*

TRUE if string *str1* alphabetically less than *str2*. Abbreviated form of the function is **&sle**.

&sgreat *str1 str2*

TRUE if string *str1* alphabetically larger than *str2*. Abbreviated form of the function is **&sgr**.

&isequal *str1 str2*

TRUE if the two strings *str1* and *str2* are the same ignoring letter case. Abbreviated form of the function is **&ise**.

&xsequal *str1 regex*

TRUE if the string *str1* matches the *regex* (case sensitive). Abbreviated form of the function is **&xse**.

See Regular Expressions for information on the *regex* format.

&xisequal *str1 regex*

TRUE if the string *str1* matches the *regex* (case insensitive). Abbreviated form of the function is **&xis**. See Regular Expressions for information on the *regex* format.

Evaluation of the string logical operators are left to right, the leftmost argument is fully evaluated before the next argument. The operator ordering is prefix notation (see &add(4) for an example of prefix ordering).

EXAMPLE

Test for variable $buffer−bname(5) is equal to `*scratch*`:

```
!if &seq $buffer-bname "*scratch*"
    ...
```

The following example tests a character is in the range `a−z`:

```
!if &not &and &sle %c "a" &sgr %c "z"
    ...
```

The following example inserts the string "`c`" into the alphabetically order string list **%test−list**:

```
set-variable %test-list "|a|b|d|e|"
set-variable %test-insert "c"

set-variable #l0 1
!while &and &not &seq &lget %test-list #l0 "" ...
        ... &sle &lget %test-list #l0 %test-insert
    set-variable #l0 &add #l0 1
!done
set-variable %test-list &lins %test-list #l0 %test-insert
```

The first test on the **!while &and** conditional checks that the current item in the list is not an empty string (""). If it is the end of the list has been reached.

The following example tests the current buffers file name for a ".c" extension:

```
!if &xse $buffer-fname ".*\\.c"
    ...
```

Note the '\' character is needed to protect the second '`.`', i.e. so that it does not match any character and the second '\' is required as the string is first parsed by the macro interpreter which changes it to "`.*\.c`" which is then interpreted as a regex.

SEE ALSO

Variable Functions, &sin(4), &slower(4), &rep(4), &add(4), &equal(4), &cond(4), Regular Expressions.

&sin(4)

NAME

&sin, &isin, &rsin, &risin − String in string test functions

SYNOPSIS

&sin *str1 str2*
&isin *str1 str2*
&rsin *str1 str2*
&risin *str1 str2*

DESCRIPTION

These functions test for the existence of *str1* in *str2*, returning the position of the string in *str2* or 0 if not found.

The functions may all be abbreviated to their three letter abbreviation (i.e. **&risin** may be expressed as **&ris**). In all cases the first argument is completely evaluated before the second argument.

&sin *str1 str2*

Returns `0` if string *str1* does not exists in string *str2*. Otherwise the function returns the character position + 1 of the location of the first character of the first occurrence of *str1*.

&isin *str1 str2*

Returns `0` if case insensitive string *str1* does not exists in string *str2*. Otherwise the function returns the character position + 1 of the location of the first character of the first occurrence of *str1*.

&rsin *str1 str2*

Returns `0` if string *str1* does not exists in string *str2*. Otherwise the function returns the character position + 1 of the location of the first character of the last occurrence of *str1*.

&risin *str1 str2*

Returns `0` if case insensitive string *str1* does not exists in string *str2*. Otherwise the function returns the character position + 1 of the location of the first character of the last occurrence of *str1*. **EXAMPLE**

The **&sin** and similar functions are useful for two different purposes. Consider the following example, this utilizes **&sin** in two different contexts. `!while &not &sin @wc " \t\n"` is a test for the end of the number, i.e. a white space character (`<tab>`, `<SPACE>` or `<NL>`).

The invocation `set-variable #l1 &isin @wc "0123456789abcdef"` is subtly different. In this case the return value is used to convert the character to it's integer hex value by using the value returned by **&isin**.

```
;
; calc-hexnum
; Convert the sting from the current position in the buffer
; to a hexadecimal number.
define-macro calc-hexnum
    forward-delete-char
    forward-delete-char
    set-variable #l0 0
    !while &not &sin @wc " \t\n"
        set-variable #l1 &isin @wc "0123456789abcdef"
        !if &not #l1
            ml-write "Bad Hex number found"
            !abort
        !endif
        set-variable #l0 &mul #l0 16
        set-variable #l0 &add #l0 &sub #l1 1
        forward-delete-char
    !done
    insert-string #l0
!emacro
```

The **&rsin** function is very similar to sin except the value return is the position of the last occurrence of the string in the given string instead of the first. This is particularly useful when extracting the path or file name from a complete file name. For example, given a UNIX style file name such as `"/usr/local/bin/me"` the path can be obtained using `set-variable %path &lef %pathfile &rsin "/" %pathfile` and the file name by using `set-variable %file &rig %pathfile &rsin "/" %pathfile`

SEE ALSO

Operating Modes, Variable Functions, &sequal(4), &rep(4).

&ldel(4)

NAME

&ldel, &lfind, &lget, &linsert, &lset − List manipulation functions

SYNOPSIS

&ldel *list index*
&lfind *list value*
&lget *list index*
&linsert *list index value*
&lset *list index value*

DESCRIPTION

The list manipulation functions perform operations on specially formatted strings called lists. A list is defined as:

```
"|value1|value2|.....|valueN|"
```

Where '`|`' is the dividing character, this is not fixed to a '`|`', but is defined by the first character of the string. Following are all valid lists.

```
"|1|2|3|4|5|"
"X1X2X3X4X5X"
"\CAHello\CAWorld\CA"
"??"
```

The functions may all be abbreviated to their three letter abbreviation (i.e. **&linsert** may be expressed as **&lin**). In all cases the first argument is completely evaluated before the second or third argument.

&ldel *list index*

Creates a new list from deleting item *index* from *list*. If *index* is out of *list*'s range (0 < index <= # items in list) then *list* is returned unchanged.

&lfind *list value*

Returns the index whose item is the same as *value* in *list*. If *value* is not found in *list* then "`0`" is returned.

&lget *list index*

Returns the value of item *index* in *list*. If *index* is out of *list*'s range (0 < index <= # items in list) then an empty string is returned.

&linsert *list index value*

Creates a new list from inserting *value* into *list* at point *index*, thereby pushing item *index* to *index*+1 etc. If *index* is 0 the *value* is inserted at the beginning of the list, if *index* is less than 0 or greater that the number of items in *list* then *value* is inserted at the end of the list.

&lset *list index value*

Creates a new list from setting *index* of *list* to *value*. If *index* is out of *list*'s range (0 < index <= # items in list) then **&lset** behaves like **&linsert**. **EXAMPLE**

The following example moves item 4 in a list to position 2:

```
set-variable #l0 &lget %list 4
set-variable #l1 &ldel %list 4
set-variable %list &lins #l1 2 #l0
```

The following example is taken from vm.emf, it firstly checks where the user has entered a vm command, if not then the key is execute as normal, otherwise the appropriate vm command is executed.

```
define-macro vm-input
    set-variable #l2 @cck
    set-variable #l3 @cc
    !if &not &set #l0 &lfi "|esc h|delete|space|return|A|a|C|c|....|z|" #l2
        !if &not &seq #l3 "ERROR"
            execute-line &spr "!nma %s %s" &cond @? @# "" #l3
            !return
        !endif
        ml-write &spr "[Key \"%s\" not bound - \"esc h\" to view help]" #l2
        !abort
    !endif
    set-variable #l1 &lget "|%osd-vm-help osd|vm-del-windows|scroll-down|....
                       ....vm-goto-list|vm-Archive-box|vm-archive-box|....
                       vm-cut-all-data|0 vm-extract-data|...|vm-forward|" #l0
    execute-line #l1
!emacro
```

SEE ALSO

Variable Functions, &mid(4), &cat(4).

&opt(4)

NAME

&opt − MicroEmacs optional feature test

SYNOPSIS

&opt *str*

DESCRIPTION

This function can be used to test the availability of optional features in the current session of MicroEmacs. Some features, like spelling checker support, are a compilation option, other options like mouse support may also be unavailable on some platforms. The **&opt** function can be used by macros to check that required base functionality is available.

The function returns `1` in the given feature "*str*" is supported, otherwise it returns `0` if the feature is unknown or not supported in the running version.

NOTES

Optional components of MicroEmacs '02 are enabled/disabled at compile time, most options are configured by `MEOPT_`<*NAME*> `#define`'s within the source file `emain.h`. Following is a complete list of options, giving the **opt** string and `#define` label:

abb − MEOPT_ABBREV

Abbreviation functionality (see expand−abbrev(2)).

cal − MEOPT_CALLBACK

Callback and idle event handling (see create−callback(2)).

cfe − MEOPT_CFENCE

Fence matching (see $fmatchdelay(5)).

cli − MEOPT_CLIENTSERVER

Client/server support (see Client−Server).

col − MEOPT_COLOR

All color support (making hilighting redundent etc, see add−color(2)).

cry − MEOPT_CRYPT

File encryption (see crypt(2m) mode).

deb − MEOPT_DEBUGM

Macro debugging (see $debug(5)).

dir − MEOPT_DIRLIST

Directory listing when loading a directory (see file−browser(3) and dir(2m) mode).

ext − MEOPT_EXTENDED

Miscellaneous more advanced commands and features such as append−buffer(2).

fho − MEOPT_FILEHOOK

File type auto−detection and configuration (see add−file−hook(2)).

fra − MEOPT_FRAME

Multiple frames (Internal or external, see opt "**mwf**" and command create−frame(2)).

has − MEOPT_CMDHASH

Use a hash table for rapid command name lookup.

hil − MEOPT_HILIGHT

Hilight and user definable indentation rules (see hilight(2) and indent(2)).

hsp − MEOPT_HSPLIT

Horizontal window splitting (see split−window−horizontally(2)).

ipi − MEOPT_IPIPES

Interactive pipes (see ipipe−shell−command(2)).

ise − MEOPT_ISEARCH

Incremental search (see isearch−forward(2)).

lbi − MEOPT_LOCALBIND

Buffer, message−line and OSD local binding overrides (see buffer−bind−key(2)).

mag − MEOPT_MAGIC

Regular expression search engine (see magic(2m) mode).

mou − MEOPT_MOUSE

Mouse support (see $mouse(5)).

mwf − MEOPT_MWFRAME

Multiple window frame support (see opt "**fra**").

nar − MEOPT_NARROW

Buffer narrowing (see narrow−buffer(2)).

nex − MEOPT_FILENEXT

Location list stepping (see get−next−line(2)).

osd − MEOPT_OSD

On Screen Display GUI support (see osd(2)).

pok − MEOPT_POKE

Direct screen poking (see screen−poke(2)).

pos − MEOPT_POSITION

Position storing and returning (see set−position(2)).

pri − MEOPT_PRINT

Printing support (see print−buffer(2)).

rcs − MEOPT_RCS

File Revision Control Support (see $rcs−co−com(5)).

reg − MEOPT_REGISTRY

Internal registry and history support (see read−registry(2) and read−history(2)).

scr − MEOPT_SCROLL

Window scroll−bar support.

soc − MEOPT_SOCKET

URL support, FTP and HTTP via sockets (see find−file(2)).

spa − MEOPT_SPAWN

External process launching (see shell−command(2)).

spe − MEOPT_SPELL

Spelling checker support (see spell(2)).

tag − MEOPT_TAGS

Tags support (see find−tag(2)).

tim − MEOPT_TIMSTMP

File timestamping on save (see time(2m) mode).

typ − MEOPT_TYPEAH

Input detect or 'type−ahead' for background processing support.

und − MEOPT_UNDO

Undo support (see undo(2)).

wor − MEOPT_WORDPRO

Word−processor style commands like fill−paragraph(2) (see forward−paragraph(2)). **EXAMPLE**

The following example checks for URL support and if not available it pops up an error:

```
!if &not &opt "soc"
    osd−dialog "Opt Test" "Error: No URL support!" "  &OK  "
!endif
```

SEE ALSO

Building MicroEmacs.

&reg(4)

NAME

&reg − Retrieve a registry value (with default)

SYNOPSIS

&reg *root subkey default*

DESCRIPTION

&reg retrieves the value of a node defined by *root*/*subkey* from the registry. The node name is specified in two components, typically required when iterating over a registry tree, where the *root* component is static and the *subkey* is dynamic, *subkey* may be specified as the null string (`""`) if an absolute registry path is specified.

The *default* value is the value of the node to return if the registry node does not exist.

EXAMPLE

The following example is taken from `me.emf` and uses the registry to retrieve some of the default configuration files:

```
; Load in the color setup
!force execute-file &reg "/history" &cat $platform "/color" "color"
; execute company setup
!if &not &seq &set #l0 &reg "/history" "company" "" ""
    !force execute-file #l0
!endif
```

SEE ALSO

get−registry(2), set−registry(2).

&set(4)

NAME

&set − In−line macro variable assignment

SYNOPSIS

&set *<var>* *<expr>*

DESCRIPTION

&set performs an in−line macro variable assignment assigning a variable *<var>* the value of the expression *<expr>*, returning the evaluated result to the caller. *<expr>* may be numeric, boolean or a string expression.

&set is typically used for defining (and simultaneously using) indices e.g. as with add−color(2) or add−color−scheme(2). This is a short−hand of set−variable(2).

EXAMPLE

The following example uses **&set** to define new colors (see `color.emf`):

```
; Standard colors
add-color &set %white     0 200 200 200
add-color &set %black     1 0   0   0
add-color &set %red       2 200 0   0
add-color &set %green     3 0   200 0
add-color &set %yellow    4 200 200 0
add-color &set %blue      5 0   0   200
add-color &set %magenta   6 200 0   200
add-color &set %cyan      7 0   200 200
```

SEE ALSO

Variable Functions, &inc(4), set−variable(2).

&sprintf(4)

NAME

&sprintf − Formatted string construction

SYNOPSIS

&sprintf *format args*

DESCRIPTION

The **&sprintf** function (or **&spr** in it's abbreviated form) provides a mechanism to generated a formatted string, similar to the 'C' programming language **sprintf(2)** function.

The **&sprintf** function is generally used where a number of different sources of information have to be converted and joined together to form a new string. It is possible to do this using &cat(4), but it does become complicated if the number of strings to be spliced together is greater than about 4, **sprintf** alleviates these problems and results in faster execution. Where only two, or three strings are to be concatenated **&cat** provides better execution times.

The **&sprintf** function produces a string construct for the *format* and a caller determined number of arguments *args* (variable arguments). The *format* string may contain special '**%**' formatting commands to insert strings and numbers into the base *format* string. The format for the '**%**' commands is "**%nc**" where:−

n

An optional numerical argument, the interpretation of the numeric value is determined by the following command (**c**).

c

The command determines the interpretation of the next argument *arg* which are specified as follows:

d (Decimal integer)

Expects a single numeric argument *arg* which is inserted into the *format* string as decimal text string. If *n* is specified then the inserted text string is fixed to *n* character in length.

n (Repeat String)

Expects two arguments *arg*, the first is a numeric argument giving the number of times to insert the given string (the second argument). If *n* is specified then the string is inserted *n* * *numeric−argument* times.

s (String)

Expects a single argument *arg* which is a string to be inserted into the key. If *n* is given then it is inserted*n* times.

x (Hexadecimal integer)

Expects a single numeric argument *arg* which is inserted into the format string as hexadecimal text string. If *n* is given then the inserted text string will be fixed to *n* character in length.

%

Inserts a single '%', *n* has no effect.

The **&sprintf** function may be nested (i.e. a string argument to **&sprintf** may be the result of another **&sprintf** invocation). Although this type of construct is not generally required !!

EXAMPLE

The following examples show how the command may be used:−

```
set−variable %result &sprintf "Foo [%s%s]" "a" "b"
```

generates "`Foo [ab]`"

```
set−variable %result &sprintf "Foo [%n%s]" 10 "a" "b"
```

generates "`Foo [aaaaaaaaaab]`".

```
set−variable %result &sprintf "[%d] [%3d] [%x] [%3x]" 10 11 12 13
```

generates "`[10] [ 11] [c] [  d]`"

NOTES

It is the callers responsibility to ensure that the correct number of arguments is supplied to match the requested formatting string. The results are undefined if an incorrect number of arguments are supplied.

SEE ALSO

Variable Functions, &cat(4).

&stat(4)

NAME

&stat − Retrieve a file statistic

SYNOPSIS

&stat <*stat*> <*filename*>

DESCRIPTION

&stat returns the specified <*stat*> on the given <*filename*>. Valid <*stat*> values are:−

a

Returns the absolute file name, corrects relative paths and symbolic links, i.e. on unix if the filename is a symbolic link it returns the file name the link points to (recursive), otherwise returns the file name.

d

Returns the file's modification time stamp. The value returned is an integer, larger values indicate a later time.

r

Returns a non−zero value if the user has permission to read the given file.

s

Returns the size of the file in bytes.

t

Returns the type of the file, where values returned are

```
X   File does not exist.
R   File is a regular file.
D   File is a directory.
H   File is a http URL link (see note).
F   File is an ftp URL file (see note).
N   File is an untouchable system file.
```

Note that a URL type is determined from the file name, e.g. http://..., and its existence is not verified.

w

Returns a non−zero value if the user has permission to write to the given file.

x

Returns a non−zero value if the user has permission to execute the given file. **EXAMPLE**

The following example is a macro which, given a file name, uses **&stat** to check that file file is regular:

```
define-macro test-file
    !force set-variable #l0 @1
    !if &not $status
        set-variable #l0 @ml04 "File name"
    !endif
    !if &not &equ &stat "t" #l4 1
        ml-write &spr "[%s is not a regular file]" #l0
        !abort
    !endif
!emacro

test-file "foobar"
```

The macro can be passed a file name and aborts if the file is not regular, there by returning the state.

The follow example checks that a file is not empty, this is used by **mail−check** to test for any incoming mail.

```
    !if &gre &stat "s" %incoming-mail-box
        ml-write "[You have new mail]"
    !endif
```

SEE ALSO

Variable Functions, find−file(2).

!return(4)

NAME

!return, !abort − Exit macro

SYNOPSIS

!return [*n*]
!abort [*n*]

DESCRIPTION

The **!return** directive causes the current macro to exit with a `TRUE` status, either returning to the caller (if any) or to interactive mode. If an argument *n* is specified then the return status is determined by the value of *n*.

!abort has the same effect as **!return** only always returning a `FALSE` status to halt the execution of any calling macro. If an argument *n* is given to **!abort** the bell is also rung, the valid values of `n` are the same as for the !bell(4) directive.

EXAMPLE

The following example checks the current language and warns if it has not be set, i.e. `Default`.

```
;       Check the current language

!if &not &seq %language "Default"
    !return
!endif
ml-write "Warning - you have not setup the Language - use user-setup"
```

The following example is shows the logic of the **!return** directive:−

```
; !return example
define-macro i-will-return
    ml-write "you will see me"
    !return
    ml-write "you wont see me"
!emacro

define-macro test-return
    ml-write "you will see me"
    i-will-return
    ml-write "you will see me"
!emacro
```

Similarly, for the **!abort** directive

```
; !abort example
define-macro i-will-abort
    ml-write "you will see me"
    !abort
    ml-write "you wont see me"
!emacro

define-macro test-abort
    ml-write "you will see me"
    i-will-abort
    ml-write "you wont see me"
!emacro
```

For the last two examples above, all the "**will**"s are displayed and none of the "**wont**"s are.

SEE ALSO

define−macro(2), !bell(4), !if(4), !goto(4).

!bell(4)

NAME

!bell − Sound audio alarm

SYNOPSIS

!bell [*n*]

DESCRIPTION

!bell gives a warning (audible or visual) to alert the user of a problem. **!bell** honors the quiet(2m) mode, as such if **quiet** mode is disabled an audible warning is given, otherwise a visual warning is given to the user (usually the message "*[BELL]*" in the bottom left hand corner).

The optional numerical argument *n* can be used to over−ride the current setting of the **quite**, a value of `0` specifies a quite bell, `2` an audible one, when omitted the default is `1` for honoring the quite mode.

!bell is generally used in conjunction with !abort, the !bell function warning the user and the !abort function to quit the macro.

EXAMPLE

The following example checks for incoming mail and is taken from mail.emf. If any mail has arrived an audible warning is assured by toggling the **quiet** mode.

```
;
; Mail checker
define−macro mail−check
    !if &seq &set %vm−mail−src &reg "/history" &cat $platform "/mail−src" "" ""
        ml−write "[Incoming mail file not setup! Use Help/User setup]"
        !abort
    !endif
    600000 create−callback mail−check
    ml−write &spr "Checking for mail in %s..." %vm−mail−src
    set−variable #l0 &cond &gre &stat "s" %vm−mail−src 0 "M" "−"
    !if &not &seq &mid $mode−line 2 1 #l0
        set−variable #l1 &rig $mode−line &cond &seq &mid $mode−line 2 1 "%" 4 3
        set−variable $mode−line &cat &cat &lef $mode−line 2 #l0 #l1
        screen−update
        !if &seq #l0 "M"
            ; use no argument to the global−mode so it toggles it back to its orig
            !bell
            global−mode "quiet"
            !bell
            global−mode "quiet"
```

```
                !endif
            !endif
            ml-clear
        !emacro
```

SEE ALSO

!abort(4), abort−command(2), quiet(2m).

!while(4)

NAME

!while, !continue, !done − Conditional loop

SYNOPSIS

!while *condition*

... loop body ...
[**!continue**]
!done **DESCRIPTION**

The **!while** directive allows statements only to be executed if a *condition* specified in the directive is met. Every line following the **!while** directive, until the first **!done** directive, is only executed if the expression following the **!while** directive evaluates to a TRUE value.

A **!continue** may be used in the loop, this immediately returns control to the **!while** statement and skips the rest of the section.

!while statement may not be nested. That is, only one **!while** statement may be outstanding at a time, a !repeat(4) statement may be used within the **!while** to create an inner loop if required. Alternatively the !goto(4) used in conjunction with the !if(4) statement may be used to construct loops.

EXAMPLE

For example, the following macro segment fills to the fill column with spaces.

```
!while &less $curcol $fill-col
    insert-string " "
    !if &equal %example "1"     ; Silly to show continue
        !continue               ; Goto !while
    !endif
    ml-write "You wont see me if %example = 1"
!done
```

SEE ALSO

!if(4), !goto(4), !repeat(4).

!emacro(4)

NAME

!emacro − Terminate a macro definition
!ehelp − Terminate a help definition

SYNOPSIS

define−macro *macro−name*

... macro body ...

!emacro
define−help *item−name*

... help body ...

!ehelp

DESCRIPTION

!emacro terminates the storage of an open macro, (opened with define−macro(2)). Only the lines between **define−macro** and the **!emacro** directive comprise the new macro *macro−name*.

Similarly **!ehelp** terminates the storage of an open help definition, (opened with define−help(2)). Only the lines between **define−help** and the **!ehelp** directive comprise the new help text for item *item−name*.

!emacro and **!ehelp** may not be used in any other context.

EXAMPLE

For example if a file is being executed contains the text:

```
;
; Read in a file in view mode, and make the window red
;
define-macro view-a-file
    find-file @ml"File to view: "
    1 buffer-mode "view"
    set-variable $buffer-bcol %red
!emacro

define-help view-a-file
    This is the help text for the macro view-a-file.
!ehelp
```

```
    ml-write "[view-a-file macro has been loaded]"
```

then only the lines between the **define−macro** command and the **!emacro** directive are stored in macro *view−a−file* and the lines between the **define−help** command and the **!ehelp** directive are stored as help for *view−a−file*. The ml−write line is executed when the file is loaded, and the message will appear on the message line, this does not however form part of the macro or help.

SEE ALSO

Operating Modes, define−macro(2), define−help(2).

!if(4)

NAME

!if, !elif, !else, !endif − Conditional statements

SYNOPSIS

!if *condition*

... condition body ...
[**!elif** *condition*

... condition body ...
]
[**!else**

... condition body ...
]
!endif **DESCRIPTION**

The conditional directives allow statements to be executed only if a condition specified in the directive is met, as follows:−

- Every line following the **!if** directive, until the first **!elif**, **!else** or **!endif** directive, is only executed if the expression following the **!if** directive evaluates to a TRUE value (non−zero).
- If the **!if** evaluates to FALSE and a **!elif** directive is next then the expression following the **!if** is evaluated and following statements are executed if TRUE.
- If no **!if** or **!elif** is found to be TRUE and a **!else** is found then the statements following it are executed.

The *condition* may be any logical condition as evaluated by the variable functions (e.g. &equal(4)) returning TRUE or FALSE. An integer value, non−zero evaluates TRUE, zero evaluates to FALSE. A non−numerical argument, such as a string is always FALSE.

The *conditional body* may be any **MicroEmacs '02** function, macro or directive with the exception of **define−macro** and **!emacro**. All directives that alter the execution of the macro are handled correctly within the **!if** statement (e.g. !goto, !return etc.

EXAMPLE

The following macro segment creates the portion of a text file automatically. (yes believe me, this will be easier to understand then that last explanation....)

```
!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
            ml-write "Contact U.N.I.T."
        !elif &sequal %time "pre 20th century"
            ml-write "start praying for a miracle"
        !else
            insert-string "Investigate the situation....\n"
            insert-string "(SAY 'stay here Sara')\n"
        !endif
    !else
        set-variable %conditions @ml"Atmosphere conditions outside? "
        !if &sequal %conditions "safe"
            insert-string &cat "Go outside......" "\n"
            insert-string "lock the door\n"
        !else
            insert-string "Dematerialize..try somewhere else"
            newline
        !endif
    !endif
```

SEE ALSO

Variable Fuctions, !goto(4), &equal(4), !return(4), $status(5).

!force(4)

NAME

!force − Ignore command or macro status

SYNOPSIS

!force [*n*] *command*

DESCRIPTION

!force ignores the return status of a *command* while executing a macro. When MicroEmacs '02 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. $status(5) may be used following **!force** to determine if the command failed or not.

A double **!force** can be used to catch a user termination (via the abort−command(2) bound to `C−g`). A macro command aborted by the user will be terminated even with a single !force directive, but not with two. See the example below.

When specifying a numerical argument with a *command*, it is placed after the *!force* directive and before the *command* i.e.

```
!force 1 forward−char
```

EXAMPLE

The following example shows how **!force** is used in conjunction with **$status**.

```
;       Merge the top two windows

push−position           ;remember where we are
1 next−window           ;go to the top window
delete−window           ;merge it with the second window
!force pop−position     ;This will continue regardless
!if $status
    ml−write "Call PASSED"
!else
    ml−write "Call FAILED"
!endif
```

The following example creates an infinite loop that can only be broken out of by a user abort. The calling macro catches this by using a double **!force** and continues. This concept is used by commands which take a considerable amount of time yet cannot be simply aborted by the user such as the spell−checker's best guess list generator.

```
define-macro infinite-loop
    set-variable #l0 1
    !while 1
        ml-write &cat "In loop, C-g to exit: " &pinc #l0 1
    !done
!emacro

define-macro catch-abort
    !force !force infinite-loop
    ml-write "You will see this"
!emacro
```

SEE ALSO

$status(5).

!goto(4)

NAME

!goto − Unconditional labeled jump
!tgoto − Conditional labeled jump

SYNOPSIS

!goto *label*

...
***label**

!tgoto *condition label*

...
***label** **DESCRIPTION**

Flow can be controlled within a MicroEmacs '02 macro using the **!goto** directive. It takes as an argument a *label*. 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, trying to jump to a non−existing label terminates execution of a macro. *labels* may be located at any position within the macro (forwards or backwards from the **!goto**).

A conditional jump may be implemented with a **!tgoto**, this takes an additional argument *condition*, which may be a literal numeric value, a variable or an evaluated expression (see Variable Functions). If the *condition* evaluates to TRUE (or non−zero) then the branch is taken and control continues from the *label*.

!tgoto is an ideal replacement for !while(4) and !repeat(4) where nested loops are required.

EXAMPLE

For example, create a block of DATA statements for a BASIC program:

```
        insert-string "1000 DATA "
        set-variable %linenum 1000
*nxtin
        screen-update           ;make sure we see the changes
        set-variable %data @ml"Next number: "
        !if &equal %data 0
            !goto finish
        !endif
        !if &greater $curcol 60
            2 backward-delete-char
```

```
                        newline
                        set-variable %linenum &add %linenum 10
                        insert-string &cat %linenum " DATA "
                    !endif
                    insert-string &cat %data ", "
                    !goto nxtin
            *finish
                    2 backward-delete-char
                    newline
```

Not that any of us are writing basic programs these days !!

NOTES

!goto and **!tgoto** are expensive operations because a symbolic name lookup is performed in the macro file. For time critical macros then the !jump(4) and !tjump(4) directives should be used as these do not perform a symbolic name search. The *jump* equivalents are source sensitive since a line displacement rather than a *label* is used − this makes them a little dangerous to use.

SEE ALSO

Variable Functions, !if(4), !jump(4), !repeat(4), !return(4), !tjump(4), !while(4).

!jump(4)

NAME

!jump − Unconditional relative branch
!tjump − conditional relative branch

SYNOPSIS

!jump *offset*
!tjump *condition offset*

DESCRIPTION

Flow can be controlled within a MicroEmacs '02 macro using the **!jump** directive. It takes as a numerical argument *offset*. The *offset* is a signed relative displacement, it may be a literal numeric value, a variable or an evaluated expression (see Variable Functions). The displacement to jump starts from the current **!jump** line. (i.e. **0 goto**) would loop forever as it jumps to itself). Negative *offset* branches backwards, positive *offset* forwards.

A conditional relative branch, with a numerical displacement is specified using **!tjump**. This has an additional argument *condition* which is evaluated and if TRUE (Non−zero) then the branch is taken. The *condition* may be a variable or an evaluated expression.

!jump and **!tjump** are fast equivalents of !goto(4) and !tgoto(4), respectively. **!jump** should be used with care as these calls are source sensitive and unexpected results may be obtained if the *offset*'s are specified incorrectly.

WARNING

Comments are not counted as valid lines within the relative displacement, these are stripped out when the macro is loaded. When using a relative branch ensure that ONLY the *code* lines are counted.

EXAMPLE

For some seriously dirty macro tricks then the **!jump** directive becomes very useful. The following example is taken from the **Metris** macro (which is packed with goodies if you can find time to work out what it does !!). The following example uses the random number generator $random(5) to generate a random number which scaled and used as a **!jump** offset, thereby creating a *switch* type statement.

```
0 define-macro met-select-piece
    !jump &mul 5 &add 1 &div &mod $random 71 10
    set-variable :met-np1 " X "  ; 1st 3 lines are dummies to get offset right
```

```
            set-variable :met-np1 " X "
            set-variable :met-np1 " X "
            set-variable :met-np1 " X "
            set-variable :met-np1 " X "
            set-variable :met-np2 "XX "
            set-variable :met-np3 " X "
            set-variable :met-ncol %lyellow
            !return
            set-variable :met-np1 "XX "
            set-variable :met-np2 "XX "
            set-variable :met-np3 "   "
            set-variable :met-ncol %yellow
            !return
            set-variable :met-np1 "X  "
            set-variable :met-np2 "XX "
            set-variable :met-np3 " X "
            set-variable :met-ncol %lmagenta
            !return
            set-variable :met-np1 "  X"
            set-variable :met-np2 " XX"
            set-variable :met-np3 " X "
            set-variable :met-ncol %lgreen
            !return
            set-variable :met-np1 " X "
            set-variable :met-np2 " X "
            set-variable :met-np3 " XX"
            set-variable :met-ncol %magenta
            !return
            set-variable :met-np1 " X "
            set-variable :met-np2 " X "
            set-variable :met-np3 "XX "
            set-variable :met-ncol %green
            !return
            set-variable :met-np1 " X "
            set-variable :met-np2 " X "
            set-variable :met-np3 " X "
            set-variable :met-ncol %lblue
            !return
            set-variable :met-np1 " X "
            set-variable :met-np2 " X "
            set-variable :met-np3 "X X"
            set-variable :met-ncol %lred
        !emacro
```

SEE ALSO

Variable Fuctions, !goto(4), !if(4), !repeat(4), !return(4), !tgoto(4), !while(4).

!nmacro(4)

NAME

!nmacro − Execute line as if not in a macro

SYNOPSIS

!nmacro *command*

DESCRIPTION

!nmacro causes *command* to be executed as if it were initiated from the command line by the user, rather than from the macro context. When MicroEmacs '02 executes a macro, by default any input the command requires is expected on the same line immediately following the command. If a line is preceded by a **!nmacro** (or **!nma**) directive, the command is executed as if it was invoked from the command line by the user, as such, the rest of the line is ignored and all input is obtained directly from the user, as per normal command interaction.

EXAMPLE

The following example is taken from macro file `meme3_8.emf` and shows how to add a buffer mode.

```
; Add a buffer mode
define-macro add-mode
    ; Has the require mode been given as an argument, if so add it
    !force 1 buffer-mode @1
    !if &not $status
        ; No - use 1 buffer-mode to add a mode
        !nma 1 buffer-mode
    !endif
!emacro
```

The first line checks that the mode to add has not already been given as a macro argument, e.g. by executing the following line

```
buffer-add-mode "view"
```

If this line fails then the argument was not specified and must be obtained from the user as normal.

NOTES

Individual arguments may be obtained from the user using the @mn(4) interactive macro variables.

SEE ALSO

@mn(4).

!repeat(4)

NAME

!repeat, !until − Conditional loop (post testing)

SYNOPSIS

!repeat

... loop body ...
!until *condition* **DESCRIPTION**

Th **!repeat** command operates in a similar fashion to !while/!done except the condition is tested at the end. Control finishes if the condition is met. As with the !while(4) there is no nesting of multiple **!repeat** statements.

EXAMPLE

For example, the following macro segment fills to the fill column with spaces.

```
!repeat
    insert-string " "
!until &equal $curcol $fill-col
```

SEE ALSO

!if(4), !goto(4), !repeat(4).

MacroArguments(4)

NAME

@?, @#, @0, @1, @2, @3, ... @p − Macro arguments

SYNOPSIS

@? − Boolean flagging if a numeric argument was supplied
@# − The value of the numeric argument

@0 − The name of the macro
@1 − The first argument of macro
@2 − The second argument of macro
@3 ... **@***n*

@p − The name of the calling (or parent) macro.

DESCRIPTION

Macros may be passed arguments, allowing a macro to be used by other macros. The **@?** and **@#** are used to determine the numeric argument given to the command. The **@***n* variable (where *n* is an integer) used in the context of a macro allows the macro body to determine it's arguments.

From a macro all commands are called in the following form

```
[num] <macro−name> "arg1" "arg2" ....
```

When executed macros do not have to be given an argument, in this case **@?** will be *0* and **@#** will be *1* (the default argument). If an argument is given then **@?** will be *1* and **@#** will be set to the numeric argument given.

The current macro command name <*macro−name*> can be obtain by using the **@0** variable, e.g.

```
define−macro Test−it
    ml−write @0
!emacro
```

When executed, writes the message "`Test−it`" which is the name of the macro.

Arguments may be passed into macro commands in the same way as standard commands are given arguments. The macro being called can access these by the **@1** to **@n** variables, where *n* is a positive integer. Any variables given as arguments are evaluated so if the variable name is required then enclose it in quotes, e.g.

```
set−variable %test−var "Hello World"
```

```
efine-macro Test-it
    ml-write &cat &cat &cat &cat @0 " " @1 " = " &ind @1
    set-variable  @1 @2
!emacro

Test-it "%test-var" "Goodbye World"
```

On execution the macro writes the message

```
"Test-it %test-var = Hello World"
```

and will set variable `%test-var` to "`Goodbye World`".

The **@p** variable can be used to obtain the name of the macro which is executing the current macro, i.e. the value of the parent's **@0** variable. If the macro was executed directly by the user then there is no parent macro and the value of **@p** is an empty string ("").

DIAGNOSTICS

If an attempt is made to access an argument which has not been given then a error occurs. This error can be trapped using the !force(4) directive, enabling the macro to take appropriate action, see example.

EXAMPLE

Consider the implementation of replace−all−string(3) macro defined in search.emf:

```
define-macro replace-all-string
    !force set-variable #l0 @3
    !if &not $status
        set-variable #l1 @ml05 "Replace all"
        set-variable #l2 @ml05 &spr "Replace [%s] with" #l1
        set-variable #l0 @ml00 "In files"
    !else
        set-variable #l1 @1
        set-variable #l2 @2
    !endif
    .
    .
    .
!emacro
```

In this example if the 3rd argument is not given then the macro gets all arguments from the user.

The **@p** variable having a value of "" when a macro is called directly by the user can be useful when determining the amount of information to feed−back to the user. For example, executing the clean macro is an easy way to remove surplus white characters, so it is often used by other macros as well as by the user. When called directly **clean** refreshes the display and prints a message of completion, but when called by other macros this would cause an unwanted screen−update and message, so clean only does this when executed by the user. This is done as follows:

```
define-macro clean
    ;
    ; Prepare to clean up file.
    .
    .
    .
    !if &seq @p ""
        screen-update
        ml-write "[Cleaned up buffer]"
    !endif
!emacro
```

NOTES

The parsing of arguments can be inefficient because of the way the arguments have to be parsed; to get the 4th argument the 1st, 2nd and 3rd arguments must be evaluated. This is because each argument is not guaranteed to be only one element, it could be an expression which needs to be evaluated. Consider the following invocation of our Test−it macro

```
Test-it &cat "%test" "-var" "Goodbye World"
```

The 2nd argument is not *"%test"* as this is part of the first argument, the 2nd argument is in fact the 4th element and the invocation will have the same effect except slower.

SEE ALSO

MacroNumericArguments, define−macro(2), replace−all−string(3), !force(4).

CommandVariables(4)

NAME

@clk, @cl − Last key or command name
@cck, @cc − Current key or command name
@cgk, @cg − Get a key or command name from the user
@cqk, @cq − Get a quoted key or command name from the user

SYNOPSIS

@clk
@cl
@cck
@cc
@cgk
@cg
@cqk
@cq

DESCRIPTION

The Command Variables allow macros to obtain MicroEmacs '02 input commands and keystrokes from the user. The general format of the command is:−

@c*i*[**k**]

Where,

i

Determines the source of the input as follows:−

l

The last input entered.

c

The current input entered.

q

Provides a low level character input mechanism, obtaining a single raw character input from the user. The input fetch does not interact with the message line and the user is NOT prompted for input (use ml−write(2) to create your own message). **@cq** is very low level, it is generally preferable to use **@cg** which provides a more intelligent binding.

g

Like **@cq**, **@cg**[**k**] gets a single character input, however if the input is bound to a function then the function name is returned instead of the character e.g. if `^F` or `<left-arrow>` is depressed then **forward−char** is returned. This has distinct advantages over **@cq** as the binding becomes device independent and executes on all platforms. In addition, it honors the users bindings, however bizarre.

k

When, omitted command input is returned to the caller (i.e. the name of the command, such as "`forward-char`"). When present, the raw keystroke is returned to the caller, i.e. "`^F` (control−F).

The **@cl**, **@clk**, **@cc** and **@cck** variables can also be set, this feature can be used by macros to change the command history. While setting the current command is limited in use, setting the last command can be immensely useful, consider the following macro code:−

```
kill-line
forward-line
set-variable @cl kill-line
kill-line
```

Without the setting of the **@cl** variable, the current kill buffer will contain only the last line. But the setting of **@cl** to kill−line fools MicroEmacs into thinking the last command was a kill command so the last kill line as appended to the current yank buffer, i.e. the kill buffer will have both lines in it.

This feature can be used for any command whose effect depends on the previous command. Such commands include forward−line(2), kill−region(2), reyank(2) and undo(2). This feature should not be abused as unexpected things may happen.

Summary

@cl

Get or set the last command.

@clk

Get or set the last key stroke.

@cc

Get or set the current command.

@cck

Get or set the current keystroke.

@cg

Get a command name from the user.

@cgk

Get a keystroke from the user.

@cq

Get a quoted command name from the user.

@cqk

Get a quoted keystroke from the user. **EXAMPLE**

The following example shows how the **@cc** and **@cl** commands are used:−

```
define-macro current-last-command
    insert-string &spr "Last key [%s] name [%s]\n" @clk @cl
    insert-string &spr "Current key [%s] name [%s]\n" @cck @cc
!emacro
```

Pressing the up key and then executing this macro using execute−named−command (esc x) will insert the lines:−

```
Last key [up] name [backward-line]
Current key [esc x] name [execute-named-command]
```

@cg like **@cq** gets a single character input, however if the keyboard input is bound to a function then the function name is returned instead of the character e.g. if `^F` or `<left-arrow>` is depressed then **forward−char** is returned. This has distinct advantages over **@cq** as the binding becomes device independent and executes on all platforms, additionally it honors the users bindings, however bizarre.

@cq provides a low level character input mechanism, obtaining a single raw character input from the user. This does not interact with the message line and the user is not prompted for input (use ml−write(2) to create your own message). **@cq** is very low level, it is generally preferable to use **@cg** which provides a more intelligent binding.

EXAMPLE

The following example is taken from `draw.emf` which uses **@cg** to obtain cursor movements from the user. Note how the input from **@cg** (stored in variable **%dw−comm**) is compared with the binding name rather than any keyboard characters.

```
!repeat
    0 screen-update
    !force set-variable #l0 @cg
    !if &seq #l0 "abort-command"
```

```
                !if &iseq @mc1 "Really quit [y/n]? " "nNyY" "y"
                    find-buffer :dw-buf
                    0 delete-buffer "*draw*"
                    -1 buffer-mode "view"
                    !abort
                !endif
            !elif &seq #l0 "newline"
                .
                .
            !elif &seq #l0 "forward-line"
                1 draw-vert
            !elif &seq #l0 "backward-line"
                -1 draw-vert
            !elif &seq #l0 "forward-char"
                1 draw-horz
            !elif &seq #l0 "backward-char"
                -1 draw-horz
            !elif &seq #l0 "osd"
                .osd.draw-help osd
            !elif &set #l1 &sin #l0 "mdeu-="
                !if &les #l1 5
                    set-variable :dw-mode &sub #l1 1
                    set-variable :dw-modes #l0
                    draw-setmode-line
                !elif &sin #l0 "-="
                    set-variable :dw-char #l0
                    draw-setmode-line
                !endif
            !else
                ml-write "[Invalid command]"
            !endif
        !until 0
```

SEE ALSO

@wc(4), &kbind(4), define−macro(2).

@fs(4)

NAME

@fs − Frame store variable

SYNOPSIS

@fs *row column*

DESCRIPTION

The frame store variable **@fs** gives macros a way of obtaining the character currently being drawn on the screen at the given location. If the given value of *row* or *column* is out range, i.e. less than zero or greater than or equal to the screen size (see $frame−width(5)) then the value returned is the empty string (i.e. "").

This variable cannot be set and is only updated during a screen update, this means that macros that change the cursor position will need to redraw the screen before using this variable.

EXAMPLE

The following example gets the word under the current mouse position, this may not be the current cursor position:

```
define-macro word-under-mouse
    set-variable #l0 $mouse-y
    set-variable #l1 $mouse-x
    !if &not &inw @fs #l0 #l1
        ml-write "[mouse not over a word]"
        !return
    !endif
    set-variable #l2 @fs #l0 #l1
    set-variable #l1 &sub #l1 1
    !if &inw @fs #l0 #l1
        set-variable #l2 &cat @fs #l0 #l1 #l2
        !jump -3
    !endif
    set-variable #l1 $mouse-x
    set-variable #l1 &add #l1 1
    !if &inw @fs #l0 #l1
        set-variable #l2 &cat #l2 @fs #l0 #l1
        !jump -3
    !endif
    ml-write &spr "[mouse is over the word \"%s\"]" #l2
!emacro
```

SEE ALSO

$frame−width(5), screen−update(2), MacroArguments, MacroNumericArguments, define−macro(2).

MessageLineVariables(4)

NAME

@mn, @mna, @ml, @mc, @mx, @mxa − Message line input

SYNOPSIS

@mn
@mna
@ml[*f*][*h*] "*prompt*" ["*default*"] ["*initial*"] ["*com−list*"] ["*buffer−name*"]
@mc[*f*] *prompt* [*valid−list*]
@mx "*command−line*"
@mxa "*command−line*"

DESCRIPTION

The **Message Line Variables** provide a method to prompt the user for an input returning the data to the caller. The **@mn** variable cause MicroEmacs to input data from the user in the default way for that command's argument, i.e. the normal prompt with the normal history and completion etc. Similarly **@mna** causes MicroEmacs to input the current argument and any subsequent arguments in the default way.

The **@ml** variable can be used to get a string (or Line) of text from the user using the message−line in a very flexible way. The first optional flag **f** is a bitwise flag where each bit has the following meaning

`0x01`

The *default* value will be specified and this will be returned by default.

`0x02`

The *initial* value will be specified and this will be initial value given on the input line.

`0x04`

Auto−complete using the initial value, usually used with bit `0x02`.

`0x08`

Hide the input string, the characters in the current input string are all displayed as `'*'`s.

If no value is specified then default value is 0 and **h** can not be specified. The *default* value is returned when the user enters an empty string. If the *initial* string is specified the the input buffer will be initialized to the given string instead of and empty one.

The flag **h** specifies what type of data is to be entered, this specifies the history to be used and the semantics allowed, **h** can have the following values

`0`	For a general string input using the general history.
`1`	For an absolute file name, with completion and history.
`2`	For a MicroEmacs '02 buffer name, with completion and history.
`3`	For a MicroEmacs '02 command name, with completion and history.
`4`	For a file name, with completion and history.
`5`	For a search string, with history.
`6`	For a MicroEmacs '02 mode name, with completion and history.
`7`	For a MicroEmacs '02 variable name, with completion and history.
`8`	For a general string using no history.
`9`	For a user supplied completion list (`com−list`).
`a`	For a user supplied completion list (`buffer−name`).

A default value of 0 is used if no value is specified. At first glance type 1 and 4 appear to be the same. They differ only when a non absolute file name is entered, such as "foobar". Type 1 will turn this into an absolute path, i.e. if the current directory is "/tmp" then it will return "/tmp/foobar". Type 4 however will return just "foobar", this is particularly useful with the &find(4) directive to then find the file "foobar".

When a value of `9` is used the argument *com−list* must be given which specifies a list of completion values in the form of a MicroEmacs list (see help on &lget(4) for further information on lists). The user may enter another value which is not in the list, which will be returned.

Alternatively a completion list may be given in the form of a buffer using a value of `a`. The argument *buffer−name* must be given to specify the buffer name from which to extract the completion list; each line of the buffer is taken as a completion value. This option is particularly useful for large completion lists as there is no size restrictions.

The **@mc** variable can be used to get a single character from the user using the message−line. The optional flag **f** is a bitwise flag where each bit has the following meaning

`0x01`

The *valid−list* specifies all valid letters.

`0x02`

Quote the typed character, this allows keys such as '`C−g`' which is bound to the abort command to be entered.

The default value for **f** is 0. When **@mc** is used, the user is prompted, with the given prompt, for a single character. If a *valid−list* is specified then only a specified valid character or an error can be returned. For example, a yes/no prompt can be implemented by the following

```
!if &iseq @mc1 "Are you bored [y/n]? " "yYnN" "y"
```

```
        save-buffers-exit-emacs
    !endif
```

By using the &isequal(4) operator a return of "Y" or "y" will match with "y".

When the **@mx** variable is used MicroEmacs sets the system variable $result(5) to the input prompt, it will then execute the given `command-line`. If this command aborts then so does the calling command, if it succeeds then the input value is taken from the **$result** variable. Similarly **@mxa** causes MicroEmacs to get the current and any subsequent arguments in this way.

These variables are useful when trying to use existing commands in a different way, such as trying to provide a GUI to an existing command. See the **delete-buffer** example below.

EXAMPLE

The following example can be used to prompt the user to save any buffer changes, the use of **@mna** ensures the user will be prompted as usual regardless of the number of buffers changed:

```
save-some-buffers @mna
```

The following example sets `%language` to a language supplied by the user from a given list, giving the current setting as a default

```
set-variable %languages "|American|British|French|Spanish|"
set-variable %language "American"

set-variable %language @ml19 "Language" %language %languages
```

The following example is taken from `diff-changes` in tools.emf, it uses **@mc** to prompt the user to save the buffer before continuing:−

```
define-macro diff-changes
    !if &seq $buffer-fname ""
        ml-write "[Current buffer has no file name]"
        !abort
    !endif
    !if &bmod "edit"
        !if &iseq @mc1 "Save buffer first [y/n]? " "nNyY" "y"
            save-buffer
        !endif
    !endif
        .
        .
```

Note that the input is case insensitive. The following version would not work as the user may expect when the buffer has not been edited:

```
        .
        .
    !if &and &bmod "edit" &iseq @mc1 "Save buffer first [y/n]? " "nNyY" "y"
        save-buffer
        .
```

.

Unlike **C** and other similar languages MicroEmacs macro language always evaluates both **&and** arguments. This means that the user will be prompted to save the buffer regardless of whether the buffer has been edited.

The **@mx** variables are useful when using existing commands in a new environment. For example, consider providing a GUI for the delete−buffer(2) command, when executed the calling GUI may not be aware that changes could be lost or a process may still be active. These variables can be used as a call back mechanism to handle this problem:

```
define-macro osd-delete-buffer-callback
    !if &sin "Discard changes" $result
        2 osd-xdialog "Delete Buffer" "  Dicard changes?  " 2 10 6 "&Yes" "&No"
        set-variable $result &cond &equ $result 1 "y" "n"
    !elif &sin "Kill active process" $result
        2 osd-xdialog "Delete Buffer" "  Kill active process?  " 2 10 6 "&Yes" "&N
        set-variable $result &cond &equ $result 1 "y" "n"
    !else
        1000 ml-write &spr "[Unknown prompt %s]" $result
        !abort
    !endif
!emacro

define-macro osd-delete-buffer
    .
    . set #l0 to buffer name to be deleted
    .
    delete-buffer #l0 @mxa osd-delete-buffer-callback
!emacro
```

SEE ALSO

define−macro(2).

SearchGroups(4)

NAME

@s0, @s1, @s2, ... @s9 − Last search group values

SYNOPSIS

@s0 − Last search's whole match string
@s1 − Last search's first group value
@s2 − Last search's second group value
...
@s9 − Last search's nineth group value

DESCRIPTION

The search group variables **@s***n* return the string matches of the last regular expression search i.e. search−forward(2) (in magic(2m) mode) or regex−forward(3).

@s0 returns the whole of the matched string, **@s***n*, *n* = 1..9, returns the bracket matches corresponding to the group demarkation points indicated by `\(` and `\)` in the search regular expression.

DIAGNOSTICS

An error is generated if an attempt is made to access these variables and the last search failed or the last search did not have the specified group.

The value returned for an unused group, e.g. @s2 for the regex string "`\(a\)\|\(b\)`" if "`a`" was matched, is an empty string ("").

EXAMPLE

The following macro code gives a simple example of their potential use:

```
forward−search "Token *{\\(Start\\|End\\)}"
!if $status
    ml−write "[found \"%s\"]" @s0
    if &seq @s1 "Start"
       .
       .
```

NOTES

Remember that the regular expression escape character '\' has to be duplicated within a macro file as '\' is also the macro file escape sequence.

SEE ALSO

magic(2m), search−forward(2), regex−forward(3).

CurrentBufferVariables(4)

NAME

@wc, @wl − Extract characters from the current buffer

SYNOPSIS

@wl
@wc

DESCRIPTION

Buffer variables are special in that they can only be queried and cannot be set. Buffer variables allow text to be taken from the current buffer and placed into a variable. Two types of extraction are provided **@wl** provides a line extraction method, **@wc** provides a character extraction method.

For example, if the current buffer contains the following text:

```
Richmond
Lafayette
<*>Bloomington                (where <*> is the current point)
Indianapolis
Gary
=* me (BE..) == rigel2 == (c:/data/rigel2.txt) ===================
```

The **@wl** variable allows text from the current buffer to be accessed, a command such as:−

```
set−variable %line @wl
```

would start at the current point in the current 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 set−variable command executes, the string "`Bloomington`" is placed in the variable **%line** and the buffer rigel2 now looks like this:

```
Richmond
Lafayette
Bloomington
<*>Indianapolis               (where <*> is the current point)
Gary
=* me (BE..) == rigel2 == (c:/data/rigel2.txt) ===================
```

The buffer command **@wc** gets the current character in the buffer, it does not change the buffer position. It is important to stress that the cursor position is not modified, in general a macro will interrogate the character under the cursor and then affect the buffer (i.e. by moving the cursor, deleting the character etc.) dependent upon the value of the character returned.

EXAMPLE

The **@wc** variable provides the most useful mechanism to modify the current buffer. The following example is a macro called **super−delete** which is bound to `<CTRL−del>`. The macro deletes characters under the cursor in blocks. If a white space character is under the cursor then all characters up until the next non−white space character are deleted. If a non−white space character is under the cursor then all non−white space characters up until the next white space character are deleted, then the white space is deleted. White space in this context is a `SPACE`, `tab` or `CR` character.

```
;
;---    Macro to delete the white space, or if an a word all of the
;       word until the next word is reached.
;
define-macro super-delete
    !while &not &sin @wc " \t\n"
        forward-delete-char
    !done
    !repeat
        forward-delete-char
    !until &or &seq @wc "" &not &sin @wc " \t\n"
    !return
!emacro

global-bind-key super-delete "C-delete"
```

SEE ALSO

define−macro(2).

@y(4)

NAME

@y − Yank buffer variable

SYNOPSIS

@y − Yank buffer variable

DESCRIPTION

The *Yank Buffer Variable* **@y** retrieves the current yank(2) string from the kill buffer and returns it to the caller.

EXAMPLE

The current contents of the yank buffer can be obtained using **@y**, so to set variable `#l1` to the current or last word if the cursor is not in a word, simply use:

```
forward-char
backward-word
set-mark
forward-word
copy-region
set-variable #l1 @y
```

SEE ALSO

yank(2), MacroArguments, MacroNumericArguments, define−macro(2).

Variables(4)

NAME

Variables − Macro variables

SYNOPSIS

#tn
$*variableName*
%*variableName*
.*variableName*
.*commandName***.***variableName*
:*variableName*
:*bufferName***:***variableName*

DESCRIPTION

Variables are part of MicroEmacs macro language and may be used wherever an argument is required. The variable space comprises:−

− Register Variable
$ − System Variable
% − Global Variable
. − Command Variable
: − Buffer Variable

All variables hold string information, the interpretation of the string (numeric, string or boolean) is determined when the variable is used within the context of the command. There are five types of variable, **Register Variables** (prefixed with a hash **#**), **System Variables** (prefixed with a dollar **$**), **Global Variables** (prefixed with a percentage **%**), **Buffer Variables** (prefixed with a colon **:**) and **Command Variables** (prefixed with a period **.**).

### Register Variables

Register Variables provide a set of 10 prefixed global (**#g0** .. **#g9**), parent (**#p0** .. **#p9**) and local (**#l0** .. **#l9**) register variables. The interpreted decode time of the register variables is significantly smaller than other variable types as no name space search is performed.

Register variables are assigned using set−variable(2), their value may be queried with describe−variable(2), unlike Global Buffer or Command variables they cannot be deleted.

Register variables are implemented like a stack, where the global registers are the top of the stack and every executing macro gets its own set of resister variables (**#l?**). The macro also has access to the global registers (**#g?**) and its calling, or parent macro (**#p?**). If the macro has no parent macro then the global registers are also the parent registers. Outside macros, i.e. using **set−variable** manually, the global parent and local registers are the same.

Register variables are typically used for retaining short term state, computation steps etc. As with the User Variables, the global register variables are global and care must be taken with nested macro invocations to ensure that the register usage does not conflict.

System Variables

MicroEmacs defines many System variables which are used to configure many aspects of the editors environment. The functionality of each system variable has been documented, they can be set and described but cannot be unset. If the user attempts to set or describe a non−existent MicroEmacs system variable (e.g. **$PATH**) the system environment is used instead, allowing the user to query and alter the system environment.

Global, Command and Buffer Variables

The Global variables are denoted by an initial **%** character followed by the name of the variable *variableName*. The *variableName* may be any ASCII character string up to 127 characters in length, all characters of the name are significant. Shorter names are preferred as this speeds up execution. Global Variables exist in a global context which all macros have access to.

Command variables exist within the scope of a command, they are denoted by the period (**.**) character. They can be accessed by one of two forms, either **.***variableName* or **.***commandName***.***variableName*. The first form, without the command name, assumes the scope to be the current command, as such may only be used to access internal variables. The second form qualifies the scope by specifying the command, this form is much more versatile and may be used to access any command variable from any other command, e.g.

```
define−macro foo
    set−variable .foo "Hello world"
    1000 ml−write &cat "foo1: " .foo
    1000 ml−write &cat "foo2: " .foo.foo
!emacro
define−macro bar
    foo
    1000 ml−write &cat "bar1: " .foo
    1000 ml−write &cat "bar2: " .foo.foo
!emacro

bar
```

When **bar** is executed the following messages may be observed:−

```
foo1: Hello World
foo2: Hello World
bar1: ERROR
bar2: Hello World
```

When a macro file or buffer is executed, they are executed within their own scope so local scope command variables (form 1) may be created and used in that scope. Any such variables created are automatically deleted at the end of execution. For example, the default color scheme generator macro file, `schemed.emf`, creates command variables for the created colors to aid readability:−

```
add−color &set .green     3 0   200 0
a0dd−color &set .lgreen   11 0   255 0

...

add−color−scheme .scheme.cardback   .lgreen   .green   .lgreen ...
```

The variables only exist as a file or buffer is being executed, they are not accessible by another command once the command or buffer execution has finished.

Buffer variables are similar to Command variable in function and behaviour except that their scope is of a buffer and are denoted by the colon (**:**) character. Access can be in one of two forms, either **:***variableName* where the scope is assumed to be the current buffer or **:***bufferName***:***variableName*, where the scope is explicitly given allowing access to any buffer variable, e.g.

```
find−buffer "foo"
set−variable :foo "Hello world"
find−buffer "bar"
set−variable :bar "Hello world"
1000 ml−write &cat ":foo     " :foo
1000 ml−write &cat ":foo:foo " :foo:foo
1000 ml−write &cat ":bar     " :bar
1000 ml−write &cat ":bar:bar " :bar:bar
```

When the above is executed the following messages may be observed:−

```
:foo     ERROR
:foo:foo Hello World
:bar     Hello World
:bar:bar Hello World
```

Global, Buffer and Command variables are automatically defined when they are used. A variable is assigned with set−variable(2) and may be subsequently deleted with unset−variable(2). The current assignment of a variable may be queried from the command line using describe−variable(2). e.g.

```
define−macro foo
!emacro
set−variable %foo "Some string"
set−variable :bar "Some string"
set−variable .foo.bar "Some string"

...

ml−write &spr "%s %s %s" %foo :bar .foo.bar

...

unset−variable :bar
unset−variable %foo
unset−variable .foo.bar
```

An undefined variable returns the string `ERROR`, this known state is used to advantage with the hilighting initialization, e.g.

```
!if &sequal .hilight.c "ERROR"
    set-variable .hilight.c &pinc .hilight.next 1
!endif
;
; Hi-light C Mode
;
0 hilight .hilight.c  2 50             $global-scheme
```

In this case the variable **.hilight.c** is explicitly tested for definition, if it is undefined then it is assigned a new value.

Conventionally, names are separated with a minus sign character (`-`) e.g. `foo-bar`. It is strongly advised that the name space is kept reasonably clean, since there are no restrictions on the number of macros that may be defined, problems will arise if different macros use the same variables in different contexts. Where possible, Command or Buffer Variables are preferable to Global Variables since they have no side effects on other macros or buffers. It is advised that all variable names associated with a particular macro set are prefixed with short identifier to make the variable name space unique. e.g. the **Metris** macro prefixes all variables with **:met-**; the **draw** macro uses **:dw-**, the **patience** macro **:pat-** etc.

Macro writers should endeavor to use the minimal number of variables, obviously the more variables that exist in the system, the greater the lookup time to find a variable. Use Register Variables in preference to Command, Global or Buffer variables for intimidate computation steps, temporary state etc.

Note that Buffer Variables are automatically deleted when the buffer is deleted.

EXAMPLE

The following example is the macro to convert tabs to spaces, it is shown in two forms, with User Variables and with Register Variables, the register variable implementation is obviously preferable since no new variables have been defined.

User Variable Implementation

```
;
; tabs-to-spaces.
; Convert all of the tabs to spaces.
define-macro tabs-to-spaces
    set-variable %curline $window-line        ; Remember line
    beginning-of-buffer
    !force search-forward "\t"
    !while $status
        3 drop-history
        set-variable %curcol $window-acol
        backward-delete-char
        &sub %curcol $window-acol insert-space
```

```
            !force search-forward "\t"
        !done
        3 drop-history
        goto-line %curline
        update-screen
        ml-write "Converted tabs!"
    !emacro
```

Register Variable Implementation

```
    ;
    ; tabs-to-spaces.
    ; Convert all of the tabs to spaces.
    define-macro tabs-to-spaces
        ; Remember line
        set-variable #l0 $window-line
        beginning-of-buffer
        !force search-forward "\t"
        !while $status
            set-variable #l1 $window-acol
            backward-delete-char
            &sub #l1 $window-acol insert-space
            !force search-forward "\t"
        !done
        goto-line #l0
        screen-update
        ml-write "[Converted tabs]"
    !emacro
```

SEE ALSO

@wc(4), define−macro(2), describe−variable(2), set−variable(2), unset−variable(2).

MacroNumericArguments(4)

NAME

@#, @? − Macro numeric arguments

SYNOPSIS

@# − The numerical argument to a macro
@? − The truth of the numerical argument to a macro

DESCRIPTION

All built−in commands and macros are invoked with a numerical argument. The argument is obtained from either the command line when the user invokes a command line such as:

esc 5 esc x forward−char

where the argument is entered after prefix 1 (**esc**). In this case, causing the cursor to be moved forward 5 characters. Within a macro file the same operation is defined as:−

5 forward−char

In both cases the numerical argument 5 is passed to the command requesting that the resultant operation is performed 5 times in succession before returning. The command itself is invoked once, it is the responsibility of the command to iterate if requested.

The command determines how the numerical argument is interpreted, in the case of spell−word the argument identifies the type of word that is being spelled and NOT the number of words to spell.

The invocation of named macros operate in the same way, the macro may use the variables **@?** and **@#** to determine the status of the numerical argument passed to it. The variables are interpreted as follows:

@?

A logical value defined as TRUE (1) if a numerical argument has been specified, otherwise FALSE (0).

@#

A signed integer value of the supplied numeric argument. If no argument is supplied (i.e. **@?**==FALSE) then **@#** is set to 1.

The **@?** and **@#** are only valid for the current macro invocation. Other macros or commands that are invoked have their own values of **@?** and **@#**.

EXAMPLE

Consider the following example, which sorts lines into alphabetical order using the sort−lines(2) function. A new command **sort−lines−ignore−case** is created using a macro to sort lines case insensitively regardless of the current buffer mode. The command **sort−lines** takes an optional argument which determines which column should be used to perform the sort.

```
;
; sort-lines-ignore-case
; Sort lines case insensitively regardless of the current 'exact' mode
; setting.
define-macro sort-lines-ignore-case
    set-variable #l0 &bmod exact
    -1 buffer-mode "exact"
    !if @?
        @# sort-lines
    !else
        sort-lines
    !endif
    &cond #l0 1 -1 buffer-mode "exact"
!emacro
```

@? is used to test the presence of the argument, if it is false **sort−lines** is invoked without an argument. When true the numeric argument is propagated e.g. **@# sort−lines**.

This particular macro highlights an important consideration when passing the numerical argument to other functions, had the macro been implemented as:

```
; INCORRECT IMPLEMENTATION
define-macro sort-lines-ignore-case
    set-variable #l0 &bmod exact
    -1 buffer-mode "exact"
    @# sort-lines
    &cond #l0 1 -1 buffer-mode "exact"
!emacro
```

then when **sort−lines−ignore−case** is invoked with no arguments **@#** is defined as 1, this is would be incorrectly propagated to **sort−lines** causing it to sort on column 1 rather than column 0 as expected.

SEE ALSO

MacroArguments, define−macro(2).

Global Glossary

GLOSSARY

The following is a list of all keywords associated with **MicroEmacs '02**:

!abort(4) Exit macro with a FALSE status
!bell(4) Sound audio alarm
!continue(4) Restart a conditional loop
!done(4) End a conditional loop
!ehelp(4) Terminate a help definition
!elif(4) Conditional test statement, continuation
!else(4) Conditional alternative statement
!emacro(4) Terminate a macro definition
!endif(4) Conditional test termination
!force(4) Ignore command or macro status
!goto(4) Unconditional labeled jump
!if(4) Conditional test statement
!jump(4) Unconditional jump
!nmacro(4) Ignore command or macro status
!repeat(4) Conditional loop (post testing)
!return(4) Exit macro with a TRUE status
!tgoto(4) Conditional labeled jump
!tjump(4) Unconditional relative branch
!until(4) Test a conditional loop
!while(4) Conditional loop
$INFOPATH(5) GNU info files base directory
$LOGNAME(5) System user name (UNIX)
$MEBACKUPPATH(5) Backup file location
$MEBACKUPSUB(5) Backup file name modifier
$MENAME(5) MicroEmacs user name
$MEPATH(5) MicroEmacs search path
$ME_ISHELL(5) Windows ishell command.com
$ME_PIPE_STDERR(5) Command line diversion to stderr symbol
$auto−time(5) Automatic buffer save time
$box−chars(5) Characters used to draw lines
$buffer−backup(5) Buffer backup file name
$buffer−bhook(5) Buffer macro hook command name (buffer current)
$buffer−bname(5) Name of the current buffer
$buffer−dhook(5) Buffer macro hook command name (buffer deletion)
$buffer−ehook(5) Buffer macro hook command name (buffer swapped)
$buffer−fhook(5) Buffer macro hook command name (buffer creation)
$buffer−fmod(5) Buffer file modes (or attributes)
$buffer−fname(5) Name of the current buffer's file name
$buffer−hilight(5) Define current buffer hilighting scheme
$buffer−indent(5) Current buffer indentation scheme
$buffer−input(5) Divert buffer input through macro

$buffer−ipipe(5) Divert buffer incremental pipe input through macro
$buffer−mask(5) Current buffer word class mask
$buffer−mode−line(5) Buffer mode line string
$buffer−names(5) Filtered buffer name list
$buffer−scheme(5) Buffer color scheme
$c−brace(5) C−mode; brace indentation
$c−case(5) C−mode; case indentation
$c−contcomm(5) C−mode; comment continuation string
$c−continue(5) C−mode; line continuation indent
$c−contmax(5) C−mode; line continuation maximum indent
$c−margin(5) C−mode; trailing comment margin
$c−statement(5) C−mode; statement indentation
$c−switch(5) C−mode; switch indentation
$command−names(5) Filtered command name list
$cursor−blink(5) Cursor blink rate
$cursor−color(5) Cursor foreground color
$cursor−x(5) Mouse X (horizontal) position
$cursor−y(5) Mouse Y (vertical) position
$debug(5) Macro debugging flag
$delay−time(5) Mouse time event delay time
$file−ignore(5) File extensions to ignore
$file−names(5) Filtered file name list
$file−template(5) Regular expression file search string
$fill−bullet(5) Paragraph filling bullet character set
$fill−bullet−len(5) Paragraph filling bullet search depth
$fill−col(5) Paragraph Mode; right fill column
$fill−eos(5) Paragraph filling; end of sentence fill characters
$fill−eos−len(5) Paragraph filling; end of sentence padding length
$fill−ignore(5) Ignore paragraph filling character(s)
$fill−mode(5) Paragraph mode; justification method
$find−words(5) Filtered word list
$fmatchdelay(5) Fence matching delay time
$frame−depth(5) Number of lines on the current frame canvas
$frame−width(5) Number of columns on the current frame canvas
$global−fmod(5) Global file modes (or attributes)
$global−scheme(5) Global buffer color scheme
$home(5) Users `home' directory location
$idle−time(5) System idle event delay time
$kept−versions(5) Number of backups to be kept
$line−scheme(5) Set the current line color scheme
$line−template(5) Command line regular expression search string
$ml−scheme(5) Message line color scheme
$mode−line(5) Mode line format
$mode−line−scheme(5) Mode line color scheme
$mode−names(5) Filtered mode name list
$mouse(5) Mouse configuration variable
$mouse−pos(5) Mouse position information
$mouse−x(5) Mouse X (horizontal) position
$mouse−y(5) Mouse Y (vertical) position

$osd−scheme(5) OSD color scheme
$platform(5) MicroEmacs host platform identifier
$progname(5) Program file name
$random(5) Generate a random number
$rcs−ci−com(5) RCS (and SCCS) check in command
$rcs−cif−com(5) RCS (and SCCS) check in first command
$rcs−co−com(5) RCS (and SCCS) check out command
$rcs−cou−com(5) RCS (and SCCS) check out unlock command
$rcs−file(5) RCS (and SCCS) file name
$rcs−ue−com(5) RCS (and SCCS) unedit file command
$recent−keys(5) Recent key history
$repeat−time(5) Mouse time event repeat time
$result(5) Various command return values
$screen−depth(5) Number of character lines on the screen canvas
$screen−width(5) Number of character columns on the screen canvas
$scroll(5) Screen scroll control
$scroll−bar(5) Scroll bar configuration
$scroll−bar−scheme(5) Scroll bar color scheme
$search−path(5) MicroEmacs search path
$show−modes(5) Select buffer modes to display
$show−region(5) Enable the hilighting of regions
$status(5) Macro command execution status
$system(5) System configuration variable
$tabsize(5) Tab character width
$tabwidth(5) Tab character interval
$temp−name(5) Temporary file name
$time(5) The current system time
$timestamp(5) Time stamp string
$trunc−scheme(5) Truncation color scheme
$variable−names(5) Filtered variable name list
$version(5) MicroEmacs version date−code
$window−acol(5) Window cursor actual column
$window−aline(5) Window cursor actual line
$window−chars(5) Character set used to render the windows
$window−col(5) Window cursor column (no expansion)
$window−depth(5) Number of text lines in a window
$window−flags(5) Current window setup flags
$window−line(5) Window cursor line
$window−mode−line(5) Window mode line position
$window−scroll−bar(5) Window scroll bar (or separator) position
$window−wcol(5) Window cursor column (historic)
$window−width(5) Number of character columns in a window
$window−wline(5) Window cursor line (historic)
$window−x−scroll(5) Current window X scroll
$window−xcl−scroll(5) Current window current line X scroll
$window−y−scroll(5) Current window Y scroll
%company−name(5) Name of company for template
%compile−com(5) Default system compile command line
%cygnus−bin−path(5) Cygwin BASH directory

%cygnus−hilight(5) Cygwin shell hilight enable flag
%cygnus−prompt(5) Cygwin shell prompt
%diff−com(5) Diff command line
%ftp−flags(5) Configure the FTP console
%gdiff−com(5) Gdiff command line
%grep−com(5) Grep command line
%http−flags(5) Configure the HTTP console
%http−proxy−addr(5) Set HTTP proxy server address
%http−proxy−port(5) Set HTTP proxy server port
%tag−file(5) Tag file name
%tag−option(5) Tag file search option
%tag−template(5) Tag file search string
&abs(4) Absolute value of a number
&add(4) Add two numbers
&and(4) Logical AND operator
&atoi(4) ASCII to integer conversion
&band(4) Bitwise AND operator
&bmode(4) Determine buffer mode
&bnot(4) Bitwise NOT operator
&bor(4) Bitwise OR operator
&bxor(4) Bitwise XOR operator
&cat(4) Concatenate two strings together
&cbind(4) Return the command a key is bound to
&cond(4) Conditional expression operator
&dec(4) Pre−decrement variable
&divide(4) Division of two numbers
&equal(4) Numerical equivalence operator
&exist(4) Test if a variable or command exists
&find(4) Find a file on the search path
&gmode(4) Determine global mode
&great(4) Numerical greater than operator
&inc(4) Pre−increment variable
&indirect(4) Evaluate a string as a variable
&inword(4) Test for a word character
&irep(4) Case insensitive replace string in string
&isequal(4) Case insensitive String equivalence operator
&isin(4) Case insensitive test for string in string
&itoa(4) Integer to ASCII conversion
&kbind(4) Return the key a command is bound to
&ldel(4) Delete list item
&left(4) Return the left most characters from a string
&len(4) Return the length of a string
&less(4) Numerical less than operator
&lfind(4) Find list item
&lget(4) Get list item
&linsert(4) Insert list item
&lset(4) Set list item
&mid(4) Return a portion (middle) of a string
&mod(4) Modulus of two numbers

&multiply(4) Multiply two numbers
&nbind(4) Return the numerial argument of a binding
&nbmode(4) Determine named buffer mode
&negate(4) Negation of two numbers
&not(4) Logical NOT operator
&opt(4) MicroEmacs optional feature test
&or(4) Logical OR operator
&pdec(4) Post−decrement variable
&pinc(4) Post−increment variable
&reg(4) Retrieve a registry value (with default)
&rep(4) Replace string in string
&right(4) Return the right most characters from a string
&risin(4) Recursive case insensitive test for string in string
&rsin(4) Recursively test for string in string
&sequal(4) String equivalence operator
&set(4) In−line macro variable assignment
&sgreat(4) String greater than operator
&sin(4) Test for string in string
&sless(4) String less than operator
&slower(4) Return the string converted to lower case
&sprintf(4) Formatted string construction
&stat(4) Retrieve a file statistic
&sub(4) Subtract two numbers
&supper(4) Return the string converted to upper case
&trboth(4) Return string trimmed of white chars on both sides
&trleft(4) Return string trimmed of white chars on left side
&trright(4) Return string trimmed of white chars on right side
&which(4) Find a program on the path
&xirep(4) Regex case insensitive Replace string in string
&xisequal(4) Case insensitive regex String equivalence operator
&xrep(4) Regex replace string in string
&xsequal(4) Regex string equivalence operator
.calc.result(5) Last calc calculation result
.which.result(5) Program path
0−9(9) UNIX t/nroff file
@0(4) Macro arguments (macro name)
@1(4) Macro arguments (first argument)
@2(4) Macro arguments (second argument)
@?(4) Macro arguments (numeric argument given)
@cc(4) Current command name
@cck(4) Current command key
@cg(4) Get a command name from the user
@cgk(4) Get a key from the user
@cl(4) Last command name
@clk(4) Last command key
@cq(4) Get a quoted command name from the user
@cqk(4) Get a quoted key from the user
@fs(4) Frame store variable
@hash(4) Macro arguments (numeric argument value)

@mc(4) Message line character input request
@ml(4) Message line input request
@mn(4) Message line input as normal request
@mna(4) All input from Message line as normal
@mx(4) Message line input by executing command
@mxa(4) All input from Message line by executing command
@p(4) Macro arguments (calling macro name)
@s0(4) Last search's whole match string
@s1(4) Last search's first group value
@s2(4) Last search's second group value
@wc(4) Extract character from the current buffer
@wl(4) Extract a line from the current buffer
@y(4) Yank buffer variable
abort−command(2) (**C−g**) Abort command
about(2) Information About MicroEmacs
add−color(2) Create a new color
add−color−scheme(2) Create a new color scheme
add−dictionary(2) Declare existence of a spelling dictionary
add−file−hook(2) Declare file name context dependent configuration
add−global−mode(3) Set a global buffer mode
add−mode(3) Set a local buffer mode
add−next−line(2) Define the searching behavior of command output
add−spell−rule(2) Add a new spelling rule to the dictionary
alarm(3) Set an alarm
aman(3) Compile an nroff file into a buffer (UNIX)
append−buffer(2) Write contents of buffer to end of named file
ascii−time(3) Return the current time as a string
asm(9) Assembler File
asn.1(9) ASN.1 file
auto(2m) Automatic source file line type detection
auto−spell(3) Auto−spell support
auto−spell−buffer(3) Auto−spell whole buffer
auto−spell−ignore(3) Auto−spell ignore current word
auto−spell−reset(3) Auto−spell hilight reset
autosv(2m) Automatic file save
awk(9) AWK File
Bindings(2) Default Key Bindings
BufferVariables(4) Buffer variables
Build(2) Build
backup(2m) Automatic file backup of last edit
backward−char(2) (**C−b**) Move the cursor left
backward−delete−char(2) (**backspace**) Delete the previous character at the cursor position
backward−delete−tab(2) (**S−tab**) Delete white space to previous tab−stop
backward−kill−word(2) (**esc backspace**) Delete the previous word at the cursor position
backward−line(2) (**C−p**) Move the cursor to the previous line
backward−paragraph(2) (**esc p**) Move the cursor to the previous paragraph
backward−word(2) (**esc b**) Move the cursor to the previous word
bas(9) Visual Basic
bat(9) MS−DOS Batch File

beginning−of−buffer(2) (**esc** <) Move to beginning of buffer/file
beginning−of−line(2) (**C−a**) Move to beginning of line
benchmrk(3f) Benchmark MicroEmacs macro processor speed
binary(2m) Binary editor mode
bnf(9) Backus−Naur Form
btm(9) 4−DOS Batch File
buffer−abbrev−file(2) Set buffers' abbreviation file
buffer−bind−key(2) Create local key binding for current buffer
buffer−help(3) Displays help page for current buffer
buffer−info(2) (**C−x =**) Status information on current buffer position
buffer−mode(2) (**C−x m**) Change a local buffer mode
buffer−setup(3) Configures the current buffer settings
buffer−unbind−key(2) Remove local key binding for current buffer
Client−Server(2) Client−Server Model
CmdVariables(4) Command variables
CommandVariables(4) Last, current and get a command key/name
CompanyProfiles(2) Defining a company profile
Compatibility(2) Compatibility with the original MicroEmacs
CurrentBufferVariables(4) Extract information from the current buffer
c(9) C programming language
c−hash−del(3) Remove C/C++ #define evaluation
c−hash−eval(3) Evaluate C/C++ #defines
c−hash−set−define(3) Set a C/C++ #define
c−hash−unset−define(3) Unset a C/C++ #define
calc(3) Integer calculator
capitalize−word(2) (**esc c**) Capitalize word
cbl(9) Cobol (85) File
cc(9) C++ programming language
change−buffer−name(2) (**esc C−n**) Change name of current buffer
change−directory(2) [**C−x C−d**] Change the current working directory
change−file−name(2) (**C−x n**) Change the file name of the current buffer
change−font(2) Change the screen font
change−frame−depth(2) Change the number of lines on the current frame
change−frame−width(2) Change the number of columns on the current frame
change−screen−depth(2) Change the number of lines on the screen
change−screen−width(2) Change the number of columns on the screen
change−window−depth(2) Change the depth of the current window
change−window−width(2) Change the width of the current window
charset−change(3) Convert buffer between two character sets
charset−iso−to−user(3) Convert buffer from ISO standard to user character set
charset−user−to−iso(3) Convert buffer from user to ISO standard character set
check−line−length(3) Check the length of text lines are valid
clean(3) Remove redundant white spaces from the current buffer
cls(9) Visual Basic
cmode(2m) C Programming language mode
command−apropos(2) (**C−h a**) List commands involving a concept
command−wait(2) Conditional wait command
comment−end(3) End the current comment
comment−line(3) Comment out the current line

comment−restyle(3) Reformat the current comment
comment−start(3) Start a new comment
comment−to−end−of−line(3) Extend comment to end of line
compare−windows(2) Compare buffer windows, ignore whitespace
compare−windows−exact(3) Compare buffer windows, with whitespace
compile(3) Start a compilation process
copy−region(2) (**esc w**) Copy a region of the buffer
count−words(2) (**esc C−c**) Count the number of words in a region
cpp(9) C++ programming language
create−callback(2) Create a timer callback
create−frame(2) Create a new frame
crlf(2m) File's line feed style
crypt(2m) Encrypted file mode
csh(9) C−Shell file
ctags(3f) Generate a C tags file
ctrlz(2m) File's termination style
cvs(3) MicroEmacs CVS interface
cvs−add(3) MicroEmacs CVS interface − add file
cvs−checkout(3) MicroEmacs CVS interface − checkout files
cvs−commit(3) MicroEmacs CVS interface − commit changes
cvs−diff(3) MicroEmacs CVS interface − diff changes
cvs−gdiff(3) MicroEmacs CVS interface − graphical diff changes
cvs−log(3) MicroEmacs CVS interface − log changes
cvs−remove(3) MicroEmacs CVS interface − remove file
cvs−resolve−conflicts(3) MicroEmacs CVS interface − resolve conflicts
cvs−state(3) MicroEmacs CVS interface − list state of directory files
cvs−update(3) MicroEmacs CVS interface − update directory files
cygnus(3) Open a Cygwin BASH window
dbx(3) UNIX Debugger
def(9) C or C++ definition file
define−help(2) Define help information
define−macro(2) Define a new macro
define−macro−file(2) Define macro file location
del(2m) Flag buffer to be deleted
delete−blank−lines(2) (**C−x C−o**) Delete blank lines about cursor
delete−buffer(2) (**C−x k**) Delete a buffer
delete−dictionary(2) Remove a spelling dictionary from memory
delete−frame(2) Delete the current frame
delete−global−mode(3) Remove a global buffer mode
delete−indentation(3) Join 2 lines deleting white spaces
delete−mode(3) Remove a local buffer mode
delete−other−windows(2) (**C−x 1**) Delete other windows
delete−registry(2) Delete a registry tree
delete−some−buffers(2) Delete buffers with query
delete−window(2) (**C−x 0**) Delete current window
describe−bindings(2) (**C−h b**) Show current command/key binding
describe−key(2) (**C−x ?**) Report keyboard key name and binding
describe−variable(2) (**C−h v**) Describe current setting of a variable
describe−word(3) Display a dictionary definition of a word

diff(3) Difference files or directories
diff−changes(3) Find the differences from a previous edit session
dir(2m) Buffer is a directory listing
directory−tree(2) Draw the file directory tree
display−matching−fence(3) Display the matching bracket
display−white−chars(3) Toggle the displaying of white characters
doc(9) ASCII plain text document file
dos2unix(3f) Convert DOS format files to UNIX format files
draw(3) Simple line drawing utility
eaf(8) MicroEmacs abbreviation file format
edf(8) MicroEmacs spelling dictionary file
edit(2m) Buffer has be changed
edit−dictionary(3) Insert a dictionary in a buffer
ehf(8) MicroEmacs help file
ehf(9) MicroEmacs '02 help file
ehftools(3f) Generate a MicroEmacs help file
emf(8) MicroEmacs macro file
emf(9) MicroEmacs '02 Macro File
emftags(3f) Generate a MicroEmacs macro tags file
end−kbd−macro(2) (**C−x)**) Stop recording keyboard macro
end−of−buffer(2) (**esc** >) Move to end of buffer/file
end−of−line(2) (**C−e**) Move to end of line
erf(8) MicroEmacs registry file
erf(9) MicroEmacs '02 registry file
etf(8) MicroEmacs template file format
etfinsrt(3) Insert template file into current buffer
exact(2m) Searching and sorting case sensitivity
exchange−point−and−mark(2) (**C−x C−x**) Exchange the cursor and marked position
execute−buffer(2) Execute script lines from a buffer
execute−file(2) (**esc** /) Execute script lines from a file
execute−kbd−macro(2) (**C−x e**) Execute a keyboard macro
execute−line(2) Execute a typed in script line
execute−named−command(2) [**esc x**] Execute a named command
execute−string(2) Execute a string as a command
execute−tool(3) Execute a user defined shell tool
exit−emacs(2) Exit MicroEmacs
expand−abbrev(2) Expand an abbreviation
expand−abbrev−handle(3) (**esc esc**) Expand an abbreviation handler
expand−iso−accents(3) Expand an ISO accent
expand−look−back(3) Complete a word by looking back for a similar word
expand−word(3) Complete a word by invocation of the speller
f(9) Fortran File
f77(9) Fortran 77 File
f90(9) Fortran 90 File
fence(2m) Auto fence matching mode
file−attrib(3) Set the current buffers system file attributes
file−browser(3) (**f10**) Browse the file system
file−browser−close(3) Close the file−browser
file−browser−swap−buffers(3) Swap between file−browser windows

file−op(2) File system operations command
fileHooks(2) File Hooks
fill−paragraph(2) (**esc o**) Format a paragraph
filter−buffer(2) (**C−x #**) Filter the current buffer through an O/S command
find−bfile(3) (**C−x 9**) Load a file as binary data
find−buffer(2) (**C−x b**) Switch to a named buffer
find−cfile(3) Load a crypted file
find−file(2) (**C−x C−f**) Load a file
find−registry(2) Index search of a registry sub−tree
find−tag(2) (**esc t**) Find tag, auto−load file and move to tag position
find−word(3) Find a using spelling dictionaries
find−zfile(3) Compressed file support
fold−all(3) (**f3**) (Un)Fold all regions in the current buffer
fold−current(3) (**f2**) (un)Fold a region in the current buffer
forward−char(2) (**C−f**) Move the cursor right
forward−delete−char(2) (**C−d**) Delete the next character at the cursor position
forward−kill−word(2) (**esc d**) Delete the next word at the cursor position
forward−line(2) (**C−n**) Move the cursor to the next line
forward−paragraph(2) (**esc n**) Move the cursor to the next paragraph
forward−word(2) (**esc f**) Move the cursor to the next word
ftp(3) Initiate an FTP connection
fvwm(9) FVWM configuration file
fvwmrc(9) FVWM configuration file
gawk(9) GNU AWK File
gdb(3) GNU Debugger
gdiff(3) Graphical file difference
gdiff(3f) Command line graphical file difference
generate−tags−file(3) Generate a tags file
get−next−line(2) (**C−x `**) Find the next command line
get−registry(2) Retrieve a node value from the registry
global−abbrev−file(2) Set global abbreviation file
global−bind−key(2) (**esc k**) Bind a key to a named command or macro
global−mode(2) (**esc m**) Change a global buffer mode
global−unbind−key(2) (**esc C−k**) Unbind a key from a named command or macro
goto−alpha−mark(2) (**C−x a**) Move the cursor to a alpha marked location
goto−line(2) (**esc g**) Move the cursor to specified line
goto−matching−fence(2) (**esc C−f**) Move the cursor to matching fence
goto−position(2) Restore a stored position
goto−window(2) Restore a saved window to the current window (historic)
grep(3) Execute grep command
grow−window−horizontally(2) Enlarge current window horizontally (relative)
grow−window−vertically(2) Enlarge the current window (relative change)
h(9) C programming language header
help(2) (**esc ?**) Help; high level introduction to help
help−command(2) (**C−h C−c**) Help; command information
help−item(2) (**C−h C−i**) Help; item information
help−variable(2) (**C−h C−v**) Help; variable information
hide(2m) Hide buffer
hilight(2) Manage the buffer hilighting schemes

hpj(9) MS−Windows Help Project File
htm(9) HyperText Markup Language File
html(9) HyperText Markup Language File
hunt−backward(2) (**C−x C−h**) Resume previous search in backward direction
hunt−forward(2) (**C−x h**) Resume previous search in forward direction
Installation(1) Installation details for MicroEmacs
Interfacing(2) Interfacing to external components
i(9) C/C++ preprocessor output file
ifill−paragraph(3) (**esc q**) Format a paragraph
imakefile(9) Make file
indent(2) Manage the auto−indentation methods
indent(2m) Automatic indentation
info(3) Display a GNU Info database
info(9) GNU Info file
info−goto−link(3) Display Info on a given link
info−on(3) Display Info on a given topic
ini(9) MS−Windows Initialization File
insert−file(2) (**C−x C−i**) Insert file into current buffer
insert−file−name(2) (**C−x C−y**) Insert filename into current buffer
insert−macro(2) Insert keyboard macro into buffer
insert−newline(2) (**C−o**) Insert new line at cursor position
insert−space(2) Insert space(s) into current buffer
insert−string(2) Insert character string into current buffer
insert−tab(2) (**C−i**) Insert tab(s) into current buffer
ipipe−kill(2) Kill a incremental pipe
ipipe−shell−command(2) (**esc backslash**) Incremental pipe (non−suspending system call)
ipipe−write(2) Write a string to an incremental pipe
isearch−backward(2) (**C−r**) Search backwards incrementally (interactive)
isearch−forward(2) (**C−s**) Search forward incrementally (interactive)
ishell(3) Open a Cygwin BASH window
iso−accents−mode(3) ISO accent expansion short−cut mode
item−list(3) (**F7**) Abbreviated search and list buffer contents
item−list−close(3) (**esc F7**) Close the item list
item−list−find(3) Find the selected item in the item list
jav(9) Java programming language
java(9) Java programming language
javatags(3f) Generate a C tags file from Java sources
justify(2m) Justification Mode
kbd−macro−query(2) (**C−x q**) Query termination of keyboard macro
keyNames(2) Key Binding Names
kill−line(2) (**C−k**) Delete all characters to the end of the line
kill−paragraph(2) Delete a paragraph
kill−rectangle(2) (**esc C−w**) Delete a column of text
kill−region(2) (**C−w**) Delete all characters in the marked region
ksh(9) Korn shell file
l(9) LEX programming language
languageTemplates(2) File Language Templates
latex(9) TeX Documentation
letter(2m) Letter kill policy

line(2m) Line kill policy
line−scheme−search(3) Search and annotate the current buffer
list−buffers(2) (**C−x C−b**) List all buffers and show their status
list−commands(2) (**C−h c**) List available commands
list−registry(2) Display the registry in a buffer
list−variables(2) (**C−h v**) List defined variables
localeSupport(2) Locale Support
lock(2m) Pipe cursor position lock
login(9) Shell user login file
lower−case−region(2) (**C−x C−l**) Lowercase a region (downcase)
lower−case−word(2) (**esc l**) Lowercase word (downcase)
MacroArguments(4) Arguments to macros
MacroNumericArguments(4) Numeric arguments to macros
Mahjongg(3) MicroEmacs '02 version of the solitaire Mah Jongg game
MainMenu(3) The top main menu
Match−It(3) MicroEmacs '02 version of the Match−It game
MessageLineVaraibles(4) Prompt the user for input on message line
MetaFont(9) MetaFont/MetaPost File
Metris(3) MicroEmacs '02 version of the falling blocks game
m4(9) M4 Macro Processor
magic(2m) Regular expression search
mail(3) Compose and send an email
mail−check(3) Check for new email
makefile(9) Make file
man(3) UNIX manual page viewer
man(9) UNIX Manual Page
man−clean(3) Clean UNIX manual page
mark−registry(2) Modify the operating mode of a registry node
me(1) MicroEmacs '02 text editor
me32.ini(8) Microsoft Window's Initialization (ini) File
memsdev(1) Microsoft Developer Studio Add−in for MicroEmacs '02
mf(9) MetaFont File
ml−bind−key(2) Create key binding for message line
ml−clear(2) Clear the message line
ml−unbind−key(2) Remove key binding from message line
ml−write(2) Write message on message line
mp(9) MetaPost File
nact(2m) Buffer not active
name−kbd−macro(2) Assign a name to the last keyboard macro
named−buffer−mode(2) Change a named buffer mode
narrow(2m) Buffer contains a narrow
narrow−buffer(2) Hide buffer lines
nawk(9) New AWK File
newline(2) (**return**) Insert a new line
next−buffer(2) (**C−x x**) Switch to the next buffer
next−frame(2) Change the focus to the next frame
next−window(2) (**C−x o**) Move the cursor to the next window
next−window−find−buffer(2) [] Split the current window and show new buffer
next−window−find−file(2) (**C−x 4**) Split the current window and find file

normal−tab(3) Insert a normal tab
nroff(9) UNIX nroff file
ntags(3f) Generate a nroff tags file
occur(3) Regular expression search for occurrences
organizer(3) Calendar and address organizer
osd(2) Manage the On−Screen Display
osd−bind−key(2) Create key binding for OSD dialog
osd−dialog(3) OSD dialog box
osd−entry(3) OSD entry dialog box
osd−help(3) GUI based on−line help
osd−unbind−key(2) Remove key binding from OSD dialog
osd−xdialog(3) OSD Extended dialog box
over(2m) Over−strike Mode
Patience(3) MicroEmacs '02 version of Patience (or Solitaire)
p(9) Pascal File
paragraph−to−line(3) Convert a paragraph to a single line
pas(9) Pascal File
perl(9) Practical Extraction and Report Language File
perldb(3) Perl Debugger
pipe(2m) Incremental Pipe running
pipe−shell−command(2) (**esc @**) Execute a single operating system command
pl(9) Practical Extraction and Report Language File
pm(9) Practical Extraction and Report Language File
popup−window(2) Pop−up a window on the screen
prefix(2) Key prefix command
previous−window(2) (**C−x p**) Move the cursor to the previous window
print−buffer(2) Print buffer, with formatting
print−color(2) Create a new printer color
print−region(2) Print region, with formatting
print−scheme(2) Create a new printer color and font scheme
print−setup(3) Configure (*mS's printer interface
printall(3f) Formatted print job
profile(9) Shell user profile
py(9) Python Language File
python(9) Python Language File
query−replace−all−string(3) Query replace string in a list of files
query−replace−string(2) (**esc C−r**) Search and replace a string − with query
quick−exit(2) (**esc z**) Exit the editor writing changes
quiet(2m) Quiet mode
quote−char(2) (**C−q**) Insert literal character
RegisterVariables(4) Register variables
RegularExpressions(2) Regular Expressions
rbin(2m) Reduced binary editor mode
rc(9) Microsoft Developer resource file
rcs−file(2) (**C−x C−q**) Handle Revision Control System (RCS) files
read−file(2) (**C−x C−r**) Find and load file replacing current buffer
read−history(2) Read in session history information
read−registry(2) Read in a registry definition file
recenter(2) (**C−l**) Recenter the window (refresh the screen)

reg(9) Registry file
regex−backward(3) Search for a magic string in the backward direction
regex−forward(3) Search for a magic string in the forward direction
replace−all−pairs(3) Replace string pairs in a list of files
replace−all−string(3) Replace string with new string in a list of files
replace−string(2) (**esc r**) Replace string with new string
reread−file(3) Reload the current buffer's file
resize−all−windows(2) Resize all windows (automatic change)
resize−window−horizontally(2) Resize current window horizontally (absolute)
resize−window−vertically(2) Resize the current window (absolute change)
restore−dictionary(3) Save dictionary user changes
restyle−buffer(3) Automatically reformat a buffer's indentation
restyle−region(3) Automatically reformat a regions indentation
reyank(2) (**esc y**) Restore next yank buffer
rgrep(3) Execute recursive grep command
rgy(9) Registry file
rul(9) Install Shield Rules
SearchGroups(4) Last search group values
s(9) Assembler File
save(2m) Flag buffer to be saved
save−all(3) Save all modified files (with query)
save−buffer(2) (**C−x C−s**) Save contents of changed buffer to file
save−buffers−exit−emacs(2) (**esc z**) Exit the editor prompt user to write changes
save−dictionary(2) Save changed spelling dictionaries
save−history(2) Write history information to history file
save−registry(2) Write a registry definition file
save−some−buffers(2) Save contents of all changed buffers to file (with query)
sch(9) Scheme File
scheme(9) Scheme File
scheme−editor(3) Color Scheme Editor
scm(9) Scheme File
screen−poke(2) Immediate write string to the screen
screen−update(2) (**redraw**) Force screen update
scroll−down(2) (**C−n**) Move the window down (scrolling)
scroll−left(2) (**C−x <**) Move the window left (scrolling)
scroll−next−window−down(2) (**esc C−v**) Scroll next window down
scroll−next−window−up(2) (**esc C−z**) Scroll next window up
scroll−right(2) (**C−x >**) Move the window right (scrolling)
scroll−up(2) (**C−p**) Move the window up (scrolling)
search−backward(2) (**C−x r**) Search for a string in the backward direction
search−forward(2) (**C−x s**) Search for a string in the forward direction
set−alpha−mark(2) (**C−x C−a**) Place an alphabetic marker in the buffer
set−char−mask(2) Set character word mask
set−cursor−to−mouse(2) Move the cursor to the current mouse position
set−encryption−key(2) (**esc e**) Define the encryption key
set−mark(2) (**esc space**) Set starting point of region
set−position(2) Store the current position
set−registry(2) Modify a node value in the registry
set−scroll−with−mouse(2) Scroll the window with the mouse

set−variable(2) (**C−x v**) Assign a new value to a variable
set−window(2) Save the current window for restore (historic)
sh(9) Bourne shell file
shell(2) [**C−x c**] Create a new command processor or shell
shell−command(2) Perform an operating system command
show−cursor(2) Change the visibility of the cursor
show−region(2) Show the current copy region
shrink−window−horizontally(2) Shrink current window horizontally (relative)
shrink−window−vertically(2) Shrink the current window (relative change)
shut−down(3) Editor exit callback command
so(9) UNIX t/nroff include file
sort−lines(2) Alphabetically sort lines
sort−lines−ignore−case(3) Alphabetically sort lines ignoring case
spell(2) Spell checker service provider
spell−add−word(3) Add a word to the main dictionary
spell−buffer(3) Spell check the current buffer
spell−edit−word(3) Edits a spell word entry
spell−word(3) (**esc $**) Spell check a single word
split−window−horizontally(2) (**C−x 5**) Split current window into two (horizontally)
split−window−vertically(2) (**C−x 2**) Split the current window into two
sql(9) SQL File
start−kbd−macro(2) (**C−x (**) Start recording keyboard macro
start−up(3) Editor startup callback command
stop−mail−check(3) Disable the check for new email
suspend−emacs(2) Suspend editor and place in background
symbol(3) Insert an ASCII character
Triangle(3) MicroEmacs '02 version of Triangle patience game
tab(2) (**tab**) Handle the tab key
tab(2m) Tabulation mode
tabs−to−spaces(3) Converts all tabs to spaces
tcl(9) TCL programming language
tcltags(3f) Generate a Tcl/Tk tags file
tcshrc(9) T−Shell start up file
tex(9) TeX Documentation
tex2nr(3) Convert a Latex file into nroff
texi(9) GNU Texinfo documentation file
texinfo(9) GNU Texinfo documentation file
textags(3f) Generate a LaTeX/BibTeX tags file
time(2m) File time stamping
time(3) Command time evaluator
tk(9) TK programming language
tni(9) UNIX t/nroff include file
translate−key(2) Translate key
transpose−chars(2) (**C−t**) Exchange (swap) adjacent characters
transpose−lines(2) (**C−x C−t**) Exchange (swap) adjacent lines
troff(9) UNIX troff file
txt(9) ASCII plain text file
UserProfiles(2) Defining a user profile
uncomment−line(3) Uncomment current line

undo(2) (**C−x u**) Undo the last edit
undo(2m) Retain edit modifications
uniq(3) Make lines in a sorted list unique
universal−argument(2) (**C−u**) Set the command argument count
unmark−buffer(3) Remove buffer edited flag
unset−variable(2) Delete a variable
upper−case−region(2) (**C−x C−u**) Uppercase a region (upcase)
upper−case−word(2) (**esc u**) Uppercase word (upcase)
user−setup(3) Configure MicroEmacs for a specific user
usr(2m) User buffer modes
Variables(4) User defined macro variables
vb(9) Visual Basic
vhdl(9) VHDL hardware simulation File
view(2m) Read only
view−file(2) (**C−x C−v**) Load a file read only
vm(3) Email viewer
void(2) Null command
vrml(9) VRML File
which(3) Program finder
wish(9) TCL shell file
wrap(2m) Line wrap entered text
wrap−word(2) Wrap word onto next line
write−buffer(2) (**C−x C−w**) Write contents of buffer to named (new) file
x86(9) Intel .x86 Assembler File
y(9) YACC programming language
yank(2) (**C−y**) Paste (copy) kill buffer contents into buffer
yank−rectangle(2) (**esc C−y**) Insert a column of text
zfile−setup(3) Compressed file support setup
zsh(9) Z−Shell file

!return(4)

NAME

!return, !abort − Exit macro

SYNOPSIS

!return [*n*]
!abort [*n*]

DESCRIPTION

The **!return** directive causes the current macro to exit with a `TRUE` status, either returning to the caller (if any) or to interactive mode. If an argument *n* is specified then the return status is determined by the value of *n*.

!abort has the same effect as **!return** only always returning a `FALSE` status to halt the execution of any calling macro. If an argument *n* is given to **!abort** the bell is also rung, the valid values of `n` are the same as for the !bell(4) directive.

EXAMPLE

The following example checks the current language and warns if it has not be set, i.e. `Default`.

```
;       Check the current language

!if &not &seq %language "Default"
    !return
!endif
ml-write "Warning - you have not setup the Language - use user-setup"
```

The following example is shows the logic of the **!return** directive:−

```
; !return example
define-macro i-will-return
    ml-write "you will see me"
    !return
    ml-write "you wont see me"
!emacro

define-macro test-return
    ml-write "you will see me"
    i-will-return
    ml-write "you will see me"
!emacro
```

Similarly, for the **!abort** directive

```
; !abort example
define−macro i−will−abort
    ml−write "you will see me"
    !abort
    ml−write "you wont see me"
!emacro

define−macro test−abort
    ml−write "you will see me"
    i−will−abort
    ml−write "you wont see me"
!emacro
```

For the last two examples above, all the "**will**"s are displayed and none of the "**wont**"s are.

SEE ALSO

define−macro(2), !bell(4), !if(4), !goto(4).

!bell(4)

NAME

!bell − Sound audio alarm

SYNOPSIS

!bell [*n*]

DESCRIPTION

!bell gives a warning (audible or visual) to alert the user of a problem. **!bell** honors the quiet(2m) mode, as such if **quiet** mode is disabled an audible warning is given, otherwise a visual warning is given to the user (usually the message "*[BELL]*" in the bottom left hand corner).

The optional numerical argument *n* can be used to over−ride the current setting of the **quite**, a value of `0` specifies a quite bell, `2` an audible one, when omitted the default is `1` for honoring the quite mode.

!bell is generally used in conjunction with !abort, the !bell function warning the user and the !abort function to quit the macro.

EXAMPLE

The following example checks for incoming mail and is taken from mail.emf. If any mail has arrived an audible warning is assured by toggling the **quiet** mode.

```
;
; Mail checker
define−macro mail−check
    !if &seq &set %vm−mail−src &reg "/history" &cat $platform "/mail−src" "" ""
        ml−write "[Incoming mail file not setup! Use Help/User setup]"
        !abort
    !endif
    600000 create−callback mail−check
    ml−write &spr "Checking for mail in %s..." %vm−mail−src
    set−variable #l0 &cond &gre &stat "s" %vm−mail−src 0 "M" "−"
    !if &not &seq &mid $mode−line 2 1 #l0
        set−variable #l1 &rig $mode−line &cond &seq &mid $mode−line 2 1 "%" 4 3
        set−variable $mode−line &cat &cat &lef $mode−line 2 #l0 #l1
        screen−update
        !if &seq #l0 "M"
            ; use no argument to the global−mode so it toggles it back to its orig
            !bell
            global−mode "quiet"
            !bell
            global−mode "quiet"
```

```
                !endif
            !endif
            ml-clear
        !emacro
```

SEE ALSO

!abort(4), abort−command(2), quiet(2m).

!while(4)

NAME

!while, !continue, !done − Conditional loop

SYNOPSIS

!while *condition*

... loop body ...
[**!continue**]
!done **DESCRIPTION**

The **!while** directive allows statements only to be executed if a *condition* specified in the directive is met. Every line following the **!while** directive, until the first **!done** directive, is only executed if the expression following the **!while** directive evaluates to a `TRUE` value.

A **!continue** may be used in the loop, this immediately returns control to the **!while** statement and skips the rest of the section.

!while statement may not be nested. That is, only one **!while** statement may be outstanding at a time, a !repeat(4) statement may be used within the **!while** to create an inner loop if required. Alternatively the !goto(4) used in conjunction with the !if(4) statement may be used to construct loops.

EXAMPLE

For example, the following macro segment fills to the fill column with spaces.

```
!while &less $curcol $fill-col
    insert-string " "
    !if &equal %example "1"     ; Silly to show continue
        !continue               ; Goto !while
    !endif
    ml-write "You wont see me if %example = 1"
!done
```

SEE ALSO

!if(4), !goto(4), !repeat(4).

!emacro(4)

NAME

!emacro − Terminate a macro definition
!ehelp − Terminate a help definition

SYNOPSIS

define−macro *macro−name*

... macro body ...

!emacro
define−help *item−name*

... help body ...

!ehelp

DESCRIPTION

!emacro terminates the storage of an open macro, (opened with define−macro(2)). Only the lines between **define−macro** and the **!emacro** directive comprise the new macro *macro−name*.

Similarly **!ehelp** terminates the storage of an open help definition, (opened with define−help(2)). Only the lines between **define−help** and the **!ehelp** directive comprise the new help text for item *item−name*.

!emacro and **!ehelp** may not be used in any other context.

EXAMPLE

For example if a file is being executed contains the text:

```
;
; Read in a file in view mode, and make the window red
;
define-macro view-a-file
    find-file @ml"File to view: "
    1 buffer-mode "view"
    set-variable $buffer-bcol %red
!emacro

define-help view-a-file
    This is the help text for the macro view-a-file.
!ehelp
```

```
    ml-write "[view-a-file macro has been loaded]"
```

then only the lines between the **define−macro** command and the **!emacro** directive are stored in macro *view−a−file* and the lines between the **define−help** command and the **!ehelp** directive are stored as help for *view−a−file*. The ml−write line is executed when the file is loaded, and the message will appear on the message line, this does not however form part of the macro or help.

SEE ALSO

Operating Modes, define−macro(2), define−help(2).

!if(4)

NAME

!if, !elif, !else, !endif − Conditional statements

SYNOPSIS

!if *condition*

... condition body ...
[**!elif** *condition*

... condition body ...
]
[**!else**

... condition body ...
]
!endif **DESCRIPTION**

The conditional directives allow statements to be executed only if a condition specified in the directive is met, as follows:−

- Every line following the **!if** directive, until the first **!elif**, **!else** or **!endif** directive, is only executed if the expression following the **!if** directive evaluates to a TRUE value (non−zero).
- If the **!if** evaluates to FALSE and a **!elif** directive is next then the expression following the **!if** is evaluated and following statements are executed if TRUE.
- If no **!if** or **!elif** is found to be TRUE and a **!else** is found then the statements following it are executed.

The *condition* may be any logical condition as evaluated by the variable functions (e.g. &equal(4)) returning TRUE or FALSE. An integer value, non−zero evaluates TRUE, zero evaluates to FALSE. A non−numerical argument, such as a string is always FALSE.

The *conditional body* may be any **MicroEmacs '02** function, macro or directive with the exception of **define−macro** and **!emacro**. All directives that alter the execution of the macro are handled correctly within the **!if** statement (e.g. !goto, !return etc.

EXAMPLE

The following macro segment creates the portion of a text file automatically. (yes believe me, this will be easier to understand then that last explanation....)

```
!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
        ml-write "Contact U.N.I.T."
    !elif &sequal %time "pre 20th century"
        ml-write "start praying for a miracle"
    !else
        insert-string "Investigate the situation....\n"
        insert-string "(SAY 'stay here Sara')\n"
    !endif
!else
    set-variable %conditions @ml"Atmosphere conditions outside? "
    !if &sequal %conditions "safe"
        insert-string &cat "Go outside......" "\n"
        insert-string "lock the door\n"
    !else
        insert-string "Dematerialize..try somewhere else"
        newline
    !endif
!endif
```

SEE ALSO

Variable Fuctions, !goto(4), &equal(4), !return(4), $status(5).

!force(4)

NAME

!force − Ignore command or macro status

SYNOPSIS

!force [*n*] *command*

DESCRIPTION

!force ignores the return status of a *command* while executing a macro. When MicroEmacs '02 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. $status(5) may be used following **!force** to determine if the command failed or not.

A double **!force** can be used to catch a user termination (via the abort−command(2) bound to `C−g`). A macro command aborted by the user will be terminated even with a single !force directive, but not with two. See the example below.

When specifying a numerical argument with a *command*, it is placed after the *!force* directive and before the *command* i.e.

```
!force 1 forward−char
```

EXAMPLE

The following example shows how **!force** is used in conjunction with **$status**.

```
;       Merge the top two windows

push−position           ;remember where we are
1 next−window           ;go to the top window
delete−window           ;merge it with the second window
!force pop−position     ;This will continue regardless
!if $status
    ml−write "Call PASSED"
!else
    ml−write "Call FAILED"
!endif
```

The following example creates an infinite loop that can only be broken out of by a user abort. The calling macro catches this by using a double **!force** and continues. This concept is used by commands which take a considerable amount of time yet cannot be simply aborted by the user such as the spell−checker's best guess list generator.

```
define-macro infinite-loop
    set-variable #l0 1
    !while 1
        ml-write &cat "In loop, C-g to exit: " &pinc #l0 1
    !done
!emacro

define-macro catch-abort
    !force !force infinite-loop
    ml-write "You will see this"
!emacro
```

SEE ALSO

$status(5).

!goto(4)

NAME

!goto − Unconditional labeled jump
!tgoto − Conditional labeled jump

SYNOPSIS

!goto *label*

...
***label**

!tgoto *condition label*

...
***label** **DESCRIPTION**

Flow can be controlled within a MicroEmacs '02 macro using the **!goto** directive. It takes as an argument a *label*. 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, trying to jump to a non−existing label terminates execution of a macro. *labels* may be located at any position within the macro (forwards or backwards from the **!goto**).

A conditional jump may be implemented with a **!tgoto**, this takes an additional argument *condition*, which may be a literal numeric value, a variable or an evaluated expression (see Variable Functions). If the *condition* evaluates to TRUE (or non−zero) then the branch is taken and control continues from the *label*.

!tgoto is an ideal replacement for !while(4) and !repeat(4) where nested loops are required.

EXAMPLE

For example, create a block of DATA statements for a BASIC program:

```
                insert-string "1000 DATA "
                set-variable %linenum 1000
        *nxtin
                screen-update           ;make sure we see the changes
                set-variable %data @ml"Next number: "
                !if &equal %data 0
                    !goto finish
                !endif
                !if &greater $curcol 60
                    2 backward-delete-char
                    newline
                    set-variable %linenum &add %linenum 10
                    insert-string &cat %linenum " DATA "
                !endif
                insert-string &cat %data ", "
                !goto nxtin
        *finish
                2 backward-delete-char
                newline
```

Not that any of us are writing basic programs these days !!

NOTES

!goto and **!tgoto** are expensive operations because a symbolic name lookup is performed in the macro file. For time critical macros then the !jump(4) and !tjump(4) directives should be used as these do not perform a symbolic name search. The *jump* equivalents are source sensitive since a line displacement rather than a *label* is used − this makes them a little dangerous to use.

SEE ALSO

Variable Functions, !if(4), !jump(4), !repeat(4), !return(4), !tjump(4), !while(4).

!jump(4)

NAME

!jump − Unconditional relative branch
!tjump − conditional relative branch

SYNOPSIS

!jump *offset*
!tjump *condition offset*

DESCRIPTION

Flow can be controlled within a MicroEmacs '02 macro using the **!jump** directive. It takes as a numerical argument *offset*. The *offset* is a signed relative displacement, it may be a literal numeric value, a variable or an evaluated expression (see Variable Functions). The displacement to jump starts from the current **!jump** line. (i.e. **0 goto**) would loop forever as it jumps to itself). Negative *offset* branches backwards, positive *offset* forwards.

A conditional relative branch, with a numerical displacement is specified using **!tjump**. This has an additional argument *condition* which is evaluated and if TRUE (Non−zero) then the branch is taken. The *condition* may be a variable or an evaluated expression.

!jump and **!tjump** are fast equivalents of !goto(4) and !tgoto(4), respectively. **!jump** should be used with care as these calls are source sensitive and unexpected results may be obtained if the *offset*'s are specified incorrectly.

WARNING

Comments are not counted as valid lines within the relative displacement, these are stripped out when the macro is loaded. When using a relative branch ensure that ONLY the *code* lines are counted.

EXAMPLE

For some seriously dirty macro tricks then the **!jump** directive becomes very useful. The following example is taken from the **Metris** macro (which is packed with goodies if you can find time to work out what it does !!). The following example uses the random number generator $random(5) to generate a random number which scaled and used as a **!jump** offset, thereby creating a *switch* type statement.

```
0 define−macro met−select−piece
    !jump &mul 5 &add 1 &div &mod $random 71 10
    set−variable :met−np1 " X "  ; 1st 3 lines are dummies to get offset right
```

```
            set-variable :met-np1 " X "
            set-variable :met-np1 " X "
            set-variable :met-np1 " X "
            set-variable :met-np1 " X "
            set-variable :met-np2 "XX "
            set-variable :met-np3 " X "
            set-variable :met-ncol %lyellow
            !return
            set-variable :met-np1 "XX "
            set-variable :met-np2 "XX "
            set-variable :met-np3 "   "
            set-variable :met-ncol %yellow
            !return
            set-variable :met-np1 "X  "
            set-variable :met-np2 "XX "
            set-variable :met-np3 " X "
            set-variable :met-ncol %lmagenta
            !return
            set-variable :met-np1 "  X"
            set-variable :met-np2 " XX"
            set-variable :met-np3 " X "
            set-variable :met-ncol %lgreen
            !return
            set-variable :met-np1 " X "
            set-variable :met-np2 " X "
            set-variable :met-np3 " XX"
            set-variable :met-ncol %magenta
            !return
            set-variable :met-np1 " X "
            set-variable :met-np2 " X "
            set-variable :met-np3 "XX "
            set-variable :met-ncol %green
            !return
            set-variable :met-np1 " X "
            set-variable :met-np2 " X "
            set-variable :met-np3 " X "
            set-variable :met-ncol %lblue
            !return
            set-variable :met-np1 " X "
            set-variable :met-np2 " X "
            set-variable :met-np3 "X X"
            set-variable :met-ncol %lred
        !emacro
```

SEE ALSO

Variable Fuctions, !goto(4), !if(4), !repeat(4), !return(4), !tgoto(4), !while(4).

!nmacro(4)

NAME

!nmacro − Execute line as if not in a macro

SYNOPSIS

!nmacro *command*

DESCRIPTION

!nmacro causes *command* to be executed as if it were initiated from the command line by the user, rather than from the macro context. When MicroEmacs '02 executes a macro, by default any input the command requires is expected on the same line immediately following the command. If a line is preceded by a **!nmacro** (or **!nma**) directive, the command is executed as if it was invoked from the command line by the user, as such, the rest of the line is ignored and all input is obtained directly from the user, as per normal command interaction.

EXAMPLE

The following example is taken from macro file `meme3_8.emf` and shows how to add a buffer mode.

```
; Add a buffer mode
define-macro add-mode
    ; Has the require mode been given as an argument, if so add it
    !force 1 buffer-mode @1
    !if &not $status
        ; No - use 1 buffer-mode to add a mode
        !nma 1 buffer-mode
    !endif
!emacro
```

The first line checks that the mode to add has not already been given as a macro argument, e.g. by executing the following line

```
buffer-add-mode "view"
```

If this line fails then the argument was not specified and must be obtained from the user as normal.

NOTES

Individual arguments may be obtained from the user using the @mn(4) interactive macro variables.

SEE ALSO

@mn(4).

!repeat(4)

NAME

!repeat, !until − Conditional loop (post testing)

SYNOPSIS

!repeat

... loop body ...
!until *condition* **DESCRIPTION**

Th **!repeat** command operates in a similar fashion to !while/!done except the condition is tested at the end. Control finishes if the condition is met. As with the !while(4) there is no nesting of multiple **!repeat** statements.

EXAMPLE

For example, the following macro segment fills to the fill column with spaces.

```
!repeat
    insert-string " "
!until &equal $curcol $fill-col
```

SEE ALSO

!if(4), !goto(4), !repeat(4).

info(3)

NAME

info − Display a GNU Info database
info−on − Display Info on a given topic
info−goto−link − Display Info on a given link
$INFOPATH − GNU info files base directory
.info.path − Cached info search path

SYNOPSIS

info

info−on *topic−str*

info−goto−link *link−str*

$INFOPATH *string*

.info.path *string*

DESCRIPTION

info interprets the GNU *info* pages, and presents the info file information within a buffer window called `*info XXXXX`, where `XXXXX` is the name of the info file. The root of the info page is displayed and may be traversed by selecting the links with the mouse, or by using the standard *info* traversal keys.

The root of the *info* tree is, by default, a file called **dir**, which points to the other information sources. The default search paths for the *info* directories are:−

> `c:/info` − MS−DOS and MS−Windows (all).
> `/usr/local/info` − All UNIX platforms.

The root directory may also be specified with the `$INFOPATH` environment variable. This is a colon (`:`) or semi−colon (`;`) separated list of directory paths which specify the locations of the info files, for UNIX and Microsoft DOS/Windows environment's, respectively.

info−on gets info on a user specified top level topic, e.g. "`gcc`", the info file "*topic−str*`.info`" must be found in the info search path.

info−goto−link gets and displays info on a user specified link or subject. The link may be within the currently displayed topic (the *link−str* need only specify the subject node name) or a subject within another topic (in which case the *link−str* takes the following form "`(`*topic*`)` *subject*").

NOTES

info is a macro implemented in file `info.emf`.

When an **info** command is run for the first time, the info search path is constructed and stored locally in the command variable **.info.path**. This variable must be directly changed by the user if changes to the info search path are required.

SEE ALSO

info(9).

$MENAME(5)

NAME

$MENAME − MicroEmacs user name
$LOGNAME − System user name (UNIX)

SYNOPSIS

$MENAME *string*; Default is `guest`

$LOGNAME *string*

DESCRIPTION

$MENAME is an environment variable used to initialize the MicroEmacs '02 environment for a given user. At start−up, if **$MENAME** is defined then the user's configuration and history file "`name`.*erf*" is located and read, where `name` is the variable value.

If at start−up **$MENAME** is not defined then **$MENAME** is assigned the value of **$LOGNAME**, if **$LOGNAME** is not defined the file `default.emf` is located and executed. This macro file is created by user−setup(3) to set **$MENAME** to the default user. If this fails then **$MENAME** defaults to `guest` and a default configuration is used.

The user configuration and history file has many uses, see user−setup(3) and read−history(2) for more information.

Microsoft Windows Environments

Within Microsoft Windows environments, if *login* is enabled then the users login name is automatically used as the first choice login name. No environment variables need to be set. If login is not enabled then one of the aforementioned methods should be used.

UNIX

In UNIX environments, **$LOGNAME** is typically defined.

NOTES

The three variables must be defined before start−up for them to have any effect.

$LOGNAME is often defined by the system and should not be altered. If a different user name is required, setting of **$MENAME** is preferable.

SEE ALSO

user−setup(3), read−history(2), $MEPATH(5).

$buffer−backup(5)

NAME

$buffer−backup − Buffer backup file name

SYNOPSIS

$buffer−backup *FileName*

DESCRIPTION

$buffer−backup is automatically set to the file name the current buffer's file would be backed up to if required. If the current buffer has no file name the variable will be set to "".

The value depends on whether DOS compliant file names are being used (see $system(5)), whether multiple backups are being kept (see $kept−versions(5)) and the setting of the environment variables **$MEBACKUPPATH** and **$MEBACKUPSUB**. The variable does not take into consideration the current setting of the buffer's backup(2m) mode which determine whether a backup will be made.

The environment variable **$MEBACKUPPATH** can be used to change the location of the backup files, it can also be used to prepend the backup filename with a string. **$MEBACKUPPATH** can specify an absolute path (e.g. "`c:/temp/mebackup/`") or a relative path (e.g. "`mebackup/`" which will move all backup files into a sub−directory automatically in the files directory).

The trailing '`/`' is important as the file name is simple appended, i.e. is creating a backup for "c:/foo/bar.txt" and $MEBACKUPPATH is set the "backup" the backup file name will be "c:/foo/backupbar.txt".

The environment variable **$MEBACKUPSUB** can be used to substitute strings within the backup filename for another. The format of the value is a list of **sed(1)** string substitutions, i.e.

```
$MEBACKUPSUB="s/from1/to1/ s/from2/to2/ s/fr..."
```

The 3 divide characters do not have to be '/'s, they can be any character as long as they are the same, e.g. "sXfrom1Xto1X". When define MicroEmacs performs a simple search for string "`from1`" (i.e. no regex support) and replaces any match with the string "`to1`" etc.

EXAMPLE

The following example compares the differences between the current version and the bucked up version using the diff(3) macro. The **diff−changes** macro is defined in `tools.emf`.

```
define−macro diff−changes
    !if &seq $buffer−fname ""
```

```
                ml-write "[Current buffer has no file name]"
                !abort
            !endif
            !if &bmod "edit"
                !if &iseq @mc1 "Save buffer first [y/n]? " "nNyY" "y"
                    save-buffer
                !endif
            !endif
            ; get the real file name - this only has effect on unix, copes with symbolic l
            set-variable #l0 &stat "a" $buffer-fname
            ; get the backup name
            set-variable #l1 $buffer-backup
            diff #l1 #l0
        !emacro
```

NOTES

The variable **$buffer−backup** can not be set, any attempt to set it will result in an error.

On Windows and DOS platforms if the $MEBACKUPPATH and $MEBACKUPSUB variables are used all remaining '`:`' characters are changed to '`/`'s as these are illegal in the middle of a filename.

SEE ALSO

backup(2m), $system(5), $kept−versions(5).

$search−path(5)

NAME

$search−path − MicroEmacs search path $MEPATH − MicroEmacs search path

SYNOPSIS

$search−path *string*

[Microsoft Windows/MS−DOS]
MEPATH= *<path1>***;***<path2>***;....;***<pathn>*

[UNIX]
MEPATH= *<path1>***:***<path2>***:....:***<pathn>*

DESCRIPTION

$search−path is initialized to the environment variable **$MEPATH**, and identifies the search paths which are searched to locate editor specific files. Multiple search paths may be specified, separated by the platform path separator (semi−colon ('**;**') on Microsoft Windows or MS−DOS environments and a colon ('**:**') on UNIX environments). Where multiple search paths are defined then they are search left to right.

The search paths are generally ordered from highest priority to lowest priority and might be arranged such as:−

> **MEPATH=***<user>*:*<company>*:*<me>*

where *<user>* represents the users path; *<company>* is the company file path (e.g. template files) and *<me>* are the standard MicroEmacs '02 files.

This would correspond to a directory installation, of user **foo** such as:−

```
/usr/foo/microemacs   - User files.
/usr/group/microemacs - Company wide files
/usr/local/microemacs - MicroEmacs installation directory
```

and a **$MEPATH** such as:−

```
MEPATH=/usr/foo/microemacs:/usr/group/microemacs:/usr/local/microemacs
```

USAGE

The current working directory is checked first for the location of a file.

$search−path is used to locate all macro files, and other files located with operators such as &find(4).

NOTES

If **$MEPATH** is not set then **$search−path** is initialized to the environment variable **$PATH**.

On UNIX systems the path */usr/local/microemacs* is automatically added to the end of **$MEPATH**, or if not defined, to the beginning of **$PATH**.

SEE ALSO

Variable Functions, execute−file(2), $MENAME(5), &find(4).

ishell(3)

NAME

ishell − Open a interactive shell window
$ME_ISHELL − Windows ishell command comspec

PLATFORM

Windows '95/'98/NT − win32
Unix − All variants.

SYNOPSIS

ishell

[Windows Only]
$ME_ISHELL = <*comspec*>

DESCRIPTION

ishell creates an interactive shell window within the a MicroEmacs buffer window, providing access to the native operating systems command shell. Within the window commands may be entered and executed, the results are shown in the window.

On running **ishell** a new buffer is created called `*shell*` which contains the shell. Executing the command again creates a new shell window called `*shell1*`, and so on. If a `*shell*` window is killed off then the available window is used next time the command is run.

Additional controls are available within the shell window to control the editors interaction with the window. The operating mode is shown as a digit on the buffer mode line, this should typically show "3", which corresponds to *F3*. The operating modes are mapped to keys as follows:−

F2

Locks the window and allows local editing to be performed. All commands entered into the window are interpreted by the editors. **F2** mode is typically entered to cut and paste from the window, search for text strings etc. In mode 2, a **2** is shown on the mode line.

F3

The normal operating mode, text typed into the window is presented to the shell window. Translation of MicroEmacs commands (i.e. beginning−of−word) are translated and passed to the shell. For interactive use this is the default mode. In mode 3, a **3** is shown on the mode line.

F4

All input is passed to the shell, no MicroEmacs commands are interpreted and keys are passed straight to the shell window. This mode is used where none of the keys to be entered are to be interpreted by MicroEmacs. Note that you have to un−toggle the F4 mode before you can swap buffers as this effectively locks the editor into the window.

F5

Clears the buffer contents. This simply erases all of the historical information in the buffer. The operation of the shell is unaffected.

To exit the shell then end the shell session using the normal exit command i.e. "`exit`" or "`C−d`" as normal and then close the buffer. A short cut "`C−c C−k`" is available to kill off the pipe. However, it is not recommended that this method is used as it effectively performs a hard kill of the buffer and attached process

UNIX

The UNIX environment uses the native **pty** support of the operating system. The shell that is opened is determined by the conventional `$SHELL` environment variable.

The shell window assumes that the user is running some sort of Emacs emulation on the command line (i.e. `VISUAL=emacs` for **ksh(1)**, **zsh(1)**, **bash(1)**, **tsch(1)**) and passes Emacs controls for command line editing.

The shell window understands re−size operations and provides a limited decoding of the *termio* characters for a VT100 screen. From within the shell window it is possible to run the likes of **top(1)** correctly. It is even possible to run another MicroEmacs terminal session !!

WINDOWS

The Windows environment provides a very poor command shell facility, this is more of a fundamental problem with the operating system than anything else. Unfortunately NT is no better than Windows '95/'98, stemming from the fact that the Windows is not actually an O/S but a huge window manager, hindered by legacy issues of MS−DOS.

For those familiar with the UNIX command shell then it is strongly recommended that the cygnus(3) BASH shell is used as an alternative. This is a far more responsive shell window and provides the familiar Emacs editing of the command line.

The command shell under Windows is slow and very unresponsive, this would appear to be a problem with the *command.com* as the same problems are not apparent with the cygwin environment. However, the shell window is good for kicking off command line utilities (such as *make*), or any command line processes that generate output on *stdout* as all of the output is captured in the buffer window which can be scrolled backwards for post analysis. For this very reason it is more preferable to the standard MS−DOS box.

It is not possible to run any utilities that use embedded screen control characters as these are not interpreted by the editor.

Changing the Shell

The default shell that is executed is defined by the environment variable **$COMSPEC**. Where the user is using a different command shell (i.e. 4−DOS), then problems may arise if this is an old 16−bit executable. The shell that MicroEmacs executes may be overridden by setting the environment variable **$ME_ISHELL**. This is typically set in the me32.ini(8) file i.e.

```
[username]
ME_ISHELL=c:\windows\command.com
```

Bugs

WinOldAp

Winoldap is created by the Microsoft environment whenever a shell is created. On occasions where processes have terminated badly the user may be prompted to kill these off; this is the normal behaviour of Windows. It is strongly advised that the shell is always exited correctly (i.e. `exit`) before leaving the editor. The Windows operating system for '95/'98 is not particularly resilient to erroneous processes can bring the whole system down. I believe that NT does not suffer from these problems (much).

Locked Input

There are occasions after killing a process the editor appears to lock up. This is typically a case that the old application has not shut down correctly. Kill off the erroneous task (`Alt−Ctrl−Del` − *End Task*) then bring the editor under control using a few `C−g` abort−command(2) sequences. **NOTES**

The **ishell** command uses the ipipe−shell−command(2) to manage the pipe between the editor and the shell. The window is controlled by the macro file `hkipipe.emf` which controls the interaction with the shell.

SEE ALSO

ipipe−shell−command(2), cygnus(3), me32.ini(8).

pipe−shell−command(2)

NAME

pipe−shell−command − Execute a single operating system command
$ME_PIPE_STDERR − Command line diversion to stderr symbol

SYNOPSIS

n **pipe−shell−command** "*command*" ["*buffer−name*"] (**esc @**)

[MS−DOS and Win32s Only]
$ME_PIPE_STDERR "*string*"; Default is undefined.

DESCRIPTION

pipe−shell−command executes one operating system command *command* and pipes the resulting output into a buffer with the name of ***command***.

The argument *n* can be used to change the default behavior of pipe−shell−command described above, *n* is a bit based flag where:−

0x01

Enables the use of the default buffer name ***command*** (default). If this bit is clear the user must supply a buffer name. This enables another command to be started without effecting any other command buffer.

0x02

Hides the output buffer, default action pops up a window and displays the output buffer in the new window.

0x04

Disable the use of the command−line processor to launch the program (win32 versions only). By default the "**command**" is launched by executing the command:

```
%COMSPEC% /c command
```

Where `%COMSPEC%` is typically command.com. If this bit is set, the "**command**" is launched directly.

0x08

Detach the launched process from MicroEmacs (win32 versions only). By default the command is launched as a child process of MicroEmacs with a new console. With this bit set the process is completely detached from MicroEmacs instead.

0x10

Disable the command name mangling (win32 versions only). By default any '`/`' characters found in the command name (the first argument only) are converted to '`\`' characters to make it Windows compliant.
NOTES

On MS−DOS and *Win32s* the standard shell **command.com(1)** does not support the piping of *stderr* to a file. Other shells, such as **4Dos.com(1)**, do, using the command−line argument "`>&`". If the environment variable "`ME_PIPE_STDERR`" is defined (the value is not used) then MicroEmacs assumes that the current shell supports piping of stderr.

SEE ALSO

ipipe−shell−command(2), shell−command(2).

$auto−time(5)

NAME

$auto−time − Automatic buffer save time

SYNOPSIS

$auto−time *seconds*; Default is `300` seconds

0 <= *seconds* <= t

DESCRIPTION

Sets the number of seconds to wait until an edited buffer is auto−saved to temporary file to t seconds. A setting of 0 disables the auto−saving command. Auto−saving can be enabled and disabled on a per buffer basis using buffer mode autosv(2m).

The auto−save file naming convention is the same as the backup name only using hash ('#') instead of tilde ('~') and is automatically removed on saving a buffer.

On unlimited length file name systems (UNIX), the following file naming conventions are used for file `xxxxx`:

```
xxxxx -> xxxxx#
```

On systems with an `xxxxxxxx.yyy` file name (DOS etc), the following file naming conventions are used:

```
xxxxxxxx     -> xxxxxxxx.###
xxxxxxxx.y   -> xxxxxxxx.y##
xxxxxxxx.yy  -> xxxxxxxx.yy#
xxxxxxxx.yyy -> xxxxxxxx.yy#
```

NOTES

The user is warned to be extra careful if files ending in '~' or '#'s are used, it is advisable to disable backup creation (see global−mode(2)) and auto−saving (`$auto-time = 0`). The author denies all responsibility (yet again) for any loss of data! Please be careful.

Auto−save files of URL files (i.e. "`ftp://...`" and "`http://...`") are written to the system's temporary directory. This avoids potentially slow auto−saves. This can however lead to recovery problems as the buffer name must be used to avoid auto−saving conflict with other buffers with the same base file name but different paths.

SEE ALSO

autosv(2m), backup(2m), buffer−mode(2) find−file(2), ftp(3).

$box−chars(5)

NAME

$box−chars − Characters used to draw lines

SYNOPSIS

$box−chars "*string*"; Default is "|+++++++++−"

DESCRIPTION

$box−chars is a fixed length string that defines the set of characters used to render lines to the screen. Osd(2), directory−tree(2), list−registry(2) and many macros use these characters as a platform independent method of drawing lines. The characters have fixed indices defined as follows:−

Index 0

Line joining north to south (vertical line).

Index 1

Line joining south to east.

Index 2

Line joining south to west.

Index 3

Line joining north to east.

Index 4

Line joining north to west.

Index 5

Line joining east to south to west.

Index 6

Line joining north to east to south.

Index 7

Line joining north to east to south to west.

Index 8

Line joining north to south to west.

Index 9

Line joining north to east to south.

Index 10

Line joining east to west. **EXAMPLE**

The **$box−chars** is typically platform dependent, it's setting is determined by the characters available in character set of the hosting platform. MS−DOS and Microsoft Windows environments might use a string such as:−

```
"\xB3\xDA\xBF\xC0\xD9\xC2\xC3\xC5\xB4\xC1\xC4"
```

X−Windows environments might use a string such as:−

```
"\x19\x0D\x0C\x0E\x0B\x18\x15\x0F\x16\x17\x12"
```

Both utilize platform specific characters.

SEE ALSO

Osd(2), directory−tree(2), list−registry(2) $window−chars(5).

$buffer−fhook(5)

NAME

$buffer−fhook − Buffer macro hook command name (buffer creation)
$buffer−dhook − Buffer macro hook command name (buffer deletion)
$buffer−bhook − Buffer macro hook command name (buffer current)
$buffer−ehook − Buffer macro hook command name (buffer swapped)

SYNOPSIS

$buffer−fhook *FunctionName*
$buffer−dhook *FunctionName*
$buffer−bhook *FunctionName*
$buffer−ehook *FunctionName*

DESCRIPTION

Sets the buffer create, delete, begin and end hook command which are executed:

buffer−fhook

When the buffer is created.

buffer−dhook

When the buffer is deleted.

buffer−bhook

When the buffer becomes the current buffer.

buffer−ehook

When the buffer is swapped out from being the current buffer.

The variable **$buffer−fhook** is largely redundant as the file hook is executed only once and before it can be sent. Its main use is within macros which wish to ascertain what type of buffer it is executing on, i.e. if a command was to be executed only on c file then the follow ensures that this is the case:

```
!if &not &seq $buffer−fhook "fhook−cmode"
    !abort
!endif
```

Where the command *fhook−cmode* is the c file hook.

dhooks are executed when a buffer is deleted, but before the contents of the buffer are lost. Note that dhooks will not be called if the buffer never becomes active, or if MicroEmacs '02 quits due to the receipt of a panic signal.

bhooks and **ehooks** are usually used to set and restore global variables which require different setting in the current buffer.

The order of The default settings of these variable are determined by the command add−file−hook(2).

SEE ALSO

add−file−hook(2).

$buffer−bname(5)

NAME

$buffer−bname − Name of the current buffer
$buffer−fname − Name of the current buffer's file name

SYNOPSIS

$buffer−bname *BufferName*
$buffer−fname *FileName*

DESCRIPTION

$buffer−bname the string name of the current buffer. Buffer names are unrestricted in length, but must be unique. By default the buffer name is derived from the buffer's file name without the path. But this can lead to conflicts, caused by identical file names but different paths. In these situations a counter is appended to the end of the buffer name and is incremented until a unique buffer name is created. For example:

```
File Name                Buffer Name
_______________________________________

/etc/file.c              file.c
/tmp/file.c              file.c<1>
/usr/file.c              file.c<2>
```

$buffer−fname contains the name of the current buffer's file name complete with path.

SEE ALSO

change−buffer−name(2).

$buffer−fmod(5)

NAME

$buffer−fmod − Buffer file modes (or attributes)
$global−fmod − Global file modes (or attributes)

SYNOPSIS

$buffer−fmod *FileMode*
$global−fmod *FileMode*

DESCRIPTION

$buffer−fmod is bit based variable setting the buffers file system modes or attributes. If the buffer was loaded from an existing file then the value of **$buffer−fmod** is taken directly from the file. But if the buffer was created then the buffer inherits the default file modes, **$global−fmod**, which is determined from the users umask on UNIX or a default on others.

The definition of the file mode bits are platform specific and are considered independently, as follows:

UNIX

The file modes of Unix are the standard read, write and execute permissions for user, group and global. See **chmod(1)** for a full description of their use and effect.

The variable is displayed in octal.

Microsoft Windows and DOS

On Microsoft platforms each file attribute (see **attrib(1)**) is assigned a bit, on windows 95 and NT the new file attributes such as compressed are also represented. The bits are assigned as follows

Bit	Attrib Flag	Attribute
0x001	R	Read Only
0x002	H	Hidden
0x004	S	System
0x010		Directory
0x020	A	Archive
0x080		Normal
0x100		Temporary
0x800		Compressed

EXAMPLE

The following example changes the $buffer−fmod so that the file will be executable (UNIX only), useful when writing a shell script.

```
set-variable $buffer-fmod 0775
```

SEE ALSO

crlf(2m), ctrlz(2m), auto(2m).

$buffer−hilight(5)

NAME

$buffer−hilight − Define current buffer hilighting scheme.

SYNOPSIS

$buffer−hilight *hilightNum*; Default is `0`

0 <= *hilightNum* <= 255

DESCRIPTION

$buffer−hilight Sets the current buffer's hi−lighting scheme (see hilight(2) for a full description of hi−lighting). The default setting is 0 which specifies no hi−lighting, when set to a non−zero, the hi−light scheme of that number MUST already be defined.

Terminals that cannot display color directly may still be able to take benefit from hi−lighting. A terminal that has fonts can use them in the same way using the add−color−scheme(2) command. The hi−light scheme is also used in printing (see print−buffer(2)). If, however, your terminal cannot display color in any way, it is recommended that hi−lighting is disabled (except when printing) as it does take CPU time.

SEE ALSO

hilight(2), print−buffer(2), $buffer−scheme(5), $buffer−indent(5).

$buffer−indent(5)

NAME

$buffer−indent − Current buffer indentation scheme.

SYNOPSIS

$buffer−indent *indentNum*; Default is `0`

0 <= *indentNum* <= 255

DESCRIPTION

$buffer−indent sets the current buffers indentation scheme. *indentNum* is the identity of the indentation scheme, as defined by indent(2), which is typically the same value as the buffers hilighting scheme number (see $buffer−hilight(5)).

The default setting is 0 which specifies no indentation scheme is present (with the exception of cmode(2m)). When non−zero, the value identifies the indentation scheme.

A buffer assigned an indentation method, MicroEmacs performs automatic line re−styling, by moving the left indentation, according to the defined indentation method. The `tab` key is typically disabled. This behavior can be altered using bit `0x1000` of the $system(5) variable, which can be changed using user−setup(3).

The use of tab characters to create the required indentation is determined by the setting of the buffers tab(2m) mode. If the mode is disabled tab characters are used wherever possible, otherwise spaces are always used.

NOTES

The commands restyle−region(3) and restyle−buffer(3) use the indentation method when defined.

The buffer indentation scheme is typically assigned in the *fhook* macro, see Language Templates.

EXAMPLE

The following example sets up an indentation scheme for a buffer within the *fhook* macro.

```
!if &sequal .hilight.foo "ERROR"
    set-variable .hilight.foo &pinc .hilight.next 1
!endif
```

```
....

; Define the indentation scheme
0 indent  .hilight.foo 2 10
indent .hilight.foo n "then" 4
indent .hilight.foo s "else" -4
indent .hilight.foo o "endif" -4

....

; File hook - called when new file is loaded.
define-macro fhook-foo
    ; if arg is 0 this is a new file so add template
    !if &not @#
        etfinsrt "foo"
    !endif
    ; Assign the hilighting
    set-variable $buffer-hilight .hilight.foo
    ; Assign the buffer indentation
    set-variable $buffer-indent .hilight.foo
    ; Set the abbreviation file
    buffer-abbrev-file "foo"
    ; Temporary comment to make sure that it works.
    ml-write "Loaded a foo file"
!emacro
```

This provides an indentation of the form:−

```
if condition
then
    XXXX
else
    if condition
    then
        XXXX
    endif
endif
```

SEE ALSO

indent(2), tab(2m), $system(5), user−setup(3), restyle−buffer(3), restyle−region(3), $buffer−hilight(5).

$buffer−input(5)

NAME

$buffer−input − Divert buffer input through macro.

SYNOPSIS

$buffer−input *commandName*

DESCRIPTION

$buffer−input allows the buffer input mechanism to be diverted through a command macro defined by *commandName*. If this variable is set to a valid command, which may be a user defined macro, this command will be called instead. The command can access the actual key−code typed by the user via the command variable @cc(4), e.g. the following macro prints out the name of the command that the user presses until the abort−command(2) is executed.

```
define−macro test−input
    ml−write &spr "Current command: %s" @cc
    !if &seq @cc "abort−command"
        set−variable $buffer−input ""
    !endif
!emacro

set−variable $buffer−input test−input
```

WARNING

Caution is advised when using this, if there is no way of reseting the variable then **MicroEmacs '02** must be killed.

SEE ALSO

abort−command(2), @cc(4).

$buffer−ipipe(5)

NAME

$buffer−input − Divert buffer incremental pipe input through macro.

SYNOPSIS

$buffer−ipipe *commandName*

DESCRIPTION

$buffer−ipipe allows the buffer incremental pipe input mechanism to be diverted through a command macro defined by *commandName*. On a buffer running an ipipe−shell−command(2) the command, set by this variable, will be called whenever new text has been inserted by the executing process. Two *alpha−marks* will be set in the buffer, '`i`' denotes the start of the newly inserted text and '`I`' denotes the end.

SEE ALSO

goto−alpha−mark(2), ipipe−shell−command(2).

$buffer−mask(5)

NAME

$buffer−mask − Current buffer word class mask.

SYNOPSIS

$buffer−mask string; Default is `luh`

DESCRIPTION

$buffer−mask sets the current buffer word class mask. MicroEmacs '02 has an internal word lookup table which defines whether a given letter is considered to be part of a word. This functionality is used in many areas such as forward−word(2), forward−kill−word(2) hilighting etc. The mask is composed with any combination of the following flags, the order in which the flags are specified is not important:

l

All lower case letters.

u

All upper case letters.

h

All hexadecimal characters (used to include numerical digits).

s

Spell extended characters, typically set to accent (`'`), hyphen (`−`) and period (`.`).

1

User set **1**, usually set to just underscore (_) for many system and programming files such as 'C'.

2

User set **2**, usually set to '`−`', '$', '&', '#', '`!`', '%', '`:`' and '@' for MicroEmacs files.

3

User set **3**, not usually defined.

4

User set **4**, not usually defined.

The character sets may be modified using the set−char−mask(2) command.

SEE ALSO

set−char−mask(2), forward−word(2).

$buffer−mode−line(5)

NAME

$buffer−mode−line − Buffer mode line string

SYNOPSIS

$buffer−mode−line "*string*"

DESCRIPTION

Sets the buffer mode line, unique to this buffer, see $mode−line(5) use, description and syntax. If this variable is NOT set for a buffer and **$mode−line** is changed, then the buffer's mode line will also change to the new value. If this variable is set, then then buffer's mode line will be unaffected by any setting of **$mode−line**.

SEE ALSO

$mode−line(5).

$buffer−names(5)

NAME

$buffer−names − Filtered buffer name list

SYNOPSIS

$buffer−names *BufferName*

DESCRIPTION

$buffer−names must first be set to the required filter string, if the variable is evaluated before it is initialized the value will be set to "*ABORT*" and the command will fail. The filter takes the form of a regex.

Once initialized, evaluating **$buffer−names** returns the name of the next buffer which matches the filter until no more buffers are found, in which case an empty string is returned.

EXAMPLE

The following example prints out the name of all buffers to the massage line one at a time. Note that &set(4) is used on the !while(4) statement to avoid evaluating **$buffer−names** twice per loop.

```
set-variable $buffer-names ".*"
!while &not &seq &set #l0 $buffer-names ""
    100 ml-write &cat "buffer: " #l0
!done
```

The following example is the same except it lists only the buffers which are not directory listings

```
set-variable $buffer-names ".*[^/]"
!while &not &seq &set #l0 $buffer-names ""
    100 ml-write &cat "buffer: " #l0
!done
```

NOTES

The list of buffers is evaluated when the variable is initialized, buffers created after the initialization will not be included in the list.

Deleting buffers which are in the list, before they are evaluated, will have undefined effects.

SEE ALSO

list−buffers(2), $buffer−bname(5), $file−names(5), $command−names(5), $mode−names(5), Regular Expressions.

$buffer−scheme(5)

NAME

$buffer−scheme − Buffer color scheme.

SYNOPSIS

$buffer−scheme *schemeNum*; Default is `0`

DESCRIPTION

$buffer−scheme sets the current buffer's color scheme to *schemeNum*, where *schemeNum* is a color scheme defined with add−color−scheme(2), which identifies the foreground and background color schemes of the buffer. The color scheme is initialized to the global color scheme settings (see $global−scheme(5)) when the buffer is created.

SEE ALSO

$buffer−hilight(5), $cursor−color(5), $trunc−scheme(5), $global−scheme(5), $ml−scheme(5), $mode−line−scheme(5), $scroll−bar−scheme(5), $system(5).

$c−brace(5)

NAME

$c−brace − C−mode; brace indentation

SYNOPSIS

$c−brace *integer*; Default is `−4`

−n <= integer <= n

DESCRIPTION

$c−brace is part of the cmode(2m) environment for C programmers.

Sets the indent of a '`{`' and a '`}`' on a new line, from the current indent. For example, using the default settings, if the current indent was 20 then a line starting with a '`{`' or a '`}`' would be indented to 16, i.e.

```
    xxxxxxxxxxx
    xxxxxxxxxxx
{ xxxxxxxxxxx
    xxxxxxxxxxx
} xxxxxxxxxxx
    xxxxxxxxxxx
```

This may seem strange, but the current indent is the indent of the last '`{`' (or "`if`", "`else`" etc.) plus $c−statement(5) which is 4, so this brings it back into line with '`{`"s, "`if`"'s and "`else`"'s etc., e.g.

```
if(xxxxxx)
{
    xxxxxxxxxx
    xxxxxxxxxx
}
```

With a setting of −2, this would become:−

```
if(xxxxxx)
  {
    xxxxxxxxxx
    xxxxxxxxxx
  }
```

This works in conjunction with $c−statement(5), a change to **$c−statement** will change the position of '`{`'s.

SEE ALSO

cmode(2m), $c−statement(5).

$c−case(5)

NAME

$c−case − C−mode; case indentation
$c−switch − C−mode; switch indentation

SYNOPSIS

$c−case *integer*; Default is `−4`
−n <= integer <= n

$c−switch *integer*; Default is `0`
−n <= integer <= n

DESCRIPTION

$c−case and **$c−switch** are part of the cmode(2m) environment for C programmers.

$c−switch sets the offset of a "`case`" entry statement from the opening brace left margin position. The default value is zero. e.g.

```
switch(xxxxxxxxx)
{
case 1:
    xxxxxxxxxx
    xxxxxxxxxx
case 2:
    xxxxxxxxxx
}
```

Setting the value to 4, increases the leading space on the "`case`" statement, e.g.

```
switch(xxxxxxxxx)
{
    case 1:
        xxxxxxxxxx
        xxxxxxxxxx
    case 2:
        xxxxxxxxxx
}
```

$c−case sets the offset of the lines following a "`case`" statement, from the current indent. For example, using the default settings, if the current indent was 20 then a line starting with a "`case`" would be indented to 16, i.e.

```
    xxxxxxxxxx
case xxxxxxxxxx
    xxxxxxxxxx
```

This is used inside "`switch`" statements, the default setting give the following lay−out:−

```
switch(xxxxxxxxxx)
{
case 1:
    xxxxxxxxxx
    xxxxxxxxxx
case 2:
```

This works in conjunction with the $c−statement(5), a change to **$c−statement** will change the position of '`{`'s.

SEE ALSO

cmode(2m), $c−statement(5).

$c−contcomm(5)

NAME

$c−case − C−mode; comment continuation string

SYNOPSIS

$c−contcomm "*string*"

DESCRIPTION

$c−contcomm is part of the cmode(2m) environment for C programmers.

This defines the string which is inserted when a new line is started while in a comment. The string is only inserted if the cursor is at the end of the line when the newline(2) command is given. For example, for the default settings, if a **newline** was entered at the end of the first line, the second line would initialize to:−

```
/* xxxxxxxxxx
   @
```

where '@' is the current cursor position. With a setting of " * ", then:−

```
/* xxxxxxxxxx
 * @
```

SEE ALSO

cmode(2m).

$c−continue(5)

NAME

$c−continue − C−mode; line continuation indent
$c−contmax − C−mode; line continuation maximum indent

SYNOPSIS

$c−continue *integer*; Default is `10`
−n <= *integer* <= *n*

$c−contmax *integer*; Default is `16`
−n <= *integer* <= *n*

DESCRIPTION

$c−continue and **$c−contmax** are part of the cmode(2m) environment for C programmers.

$c−continue sets the indent to be added to a split line, i.e. for an indent of 20, a continued statement would be indented to 30. A continued statement is a single c statement which is spread over 2 or more lines, the 2nd and any following lines would be indented to 30. For example

```
thisIsAVeryLongVariableWhichMeansAssignmentsAreSplit =
          ThisIsTheFirstContinuedStatementLine +
          ThisIsTheSecondContinuedStatementLine + etc ;
```

The indent is changed if there is an open bracket, continued statements are indented to the depth of the open bracket plus one, e.g.

```
func(firstFuncArg,
     secondFuncArg,
     anotherBracketForFun(firstAnotherBracketForFunArg,
                          secondAnotherBracketForFunArg),
     thirdFuncArg) ;
```

$c−contmax sets an upper limit of the indentation where an open bracket is encountered, in the case where the leading indent of the function name and open bracket exceeds **$c−contmax**, then the continuation is reduced to the continuation indent.

The effect of **$c−contmax** is described as follows; if **$c−contmax** is set to a large value then the default open brace offset appearence is:−

```
longVariable = LongFunctionNameWhichMeans(isSoFar,
                                          OverAndYouRunOutOfRoom) ;
```

Setting **$c−contmax** to 16 gives:

```
        longVariable = LongFunctionNameWhichMeans(isSoFar,
                                overAndYouRunOutOfRoom) ;
```

Where by the second argument indent has been artificially reduced because of it's length.

SEE ALSO

cmode(2m).

$c−margin(5)

NAME

$c−margin − C−mode; trailing comment margin

SYNOPSIS

$c−margin *integer*; Default is `−1`

−1 <= *integer* <= *n*

DESCRIPTION

$c−margin is part of the cmode(2m) environment for C programmers.

If inserting a comment at the end of a C line, it is tedious typing *x* number of spaces to the comment column (by default tab doesn't insert a tab when cmode(2m) is enabled, it reformats the indentation of the line regardless of the cursor position). This variable sets the indent column of these comments. So with the default settings and the following line,

```
xxxxxx ;/
```

when a '*' is type the line becomes

```
xxxxxx ;                /*
```

The indenting of the "`/*`" occurs only if there is text on the line before it, and none after it. If the current column is already past **$c−margin** then it is indented to the next tab stop.

A value of −1 disables this feature.

SEE ALSO

cmode(2m).

$c−statement(5)

NAME

$c−statement − C−mode; statement indentation

SYNOPSIS

$c−statement *integer*; Default is 4

−n <= integer <= n

DESCRIPTION

$c−statement is part of the cmode(2m) environment for C programmers.

The indent of the current line is derived from **$c−statement** plus the indent of the last c token (*if*, *else*, *while* etc.) or the last '`{`' (which ever was found first). i.e. if the last '`{`' was found at column 16 then the current line will be indented to 20:−

```
                {
                    xxxxxxxxxx
                    xxxxxxxxxx
```

or

```
                if(xxxxx)
                    xxxxxxxxxx
```

C tokens are only used to indent the next line, whereas '`{`' are used in indenting every line to it's partnering '`}`'.

SEE ALSO

cmode(2m).

$command−names(5)

NAME

$command−names − Filtered command name list

SYNOPSIS

$command−names *CommandName*

DESCRIPTION

$command−names must first be initialized to the required filter string, if the variable is evaluated before it is initialized the value will be set to "*ABORT*" and the command will fail. The filter takes the form of a regex.

Once initialized, evaluating **$command−names** returns the name of the next command which matches the filter until no more commands are found, in which case an empty string is returned.

EXAMPLE

The following example prints out the name of all commands to the massage line one at a time. Note that &set(4) is used on the !while(4) statement to avoid evaluating **$command−names** twice per loop.

```
set-variable $command-names ".*"
!while &not &seq &set #l0 $command-names ""
    100 ml-write &cat "command: " #l0
!done
```

The following example is an alternative implementation of command−apropos(2).

```
define-macro alt-commad-apropos
    set-variable #l1 @ml "Apropos string"
    set-variable $command-names &cat &cat ".*" #l1 ".*"
    !force 0 delete-buffer "*commands*"
    1 popup-window "*commands*"
    !while &not &seq &set #l0 $command-names ""
        insert-string &spr "    %s\n" #l0
    !done
    beginning-of-buffer
    -1 buffer-mode "edit"
    1 buffer-mode "view"
!emacro
```

NOTES

$command−names does not differentiate between built in commands and macros.

The list of commands is evaluated when the variable is initialized, macros created after the initialization will not be included in the list.

SEE ALSO

list−commands(2), command−apropos(2), $buffer−names(5), $file−names(5), $mode−names(5), $variable−names(5), Regular Expressions.

$cursor−blink(5)

NAME

$cursor−blink − Cursor blink rate $cursor−color − Cursor foreground color

SYNOPSIS

$cursor−blink integer; Default is `0`

$cursor−color colorNum; Default is `0`

0 <= *colorNum* <= *n*

DESCRIPTION

$cursor−blink sets the cursor's flash rate, i.e. the period in which the cursor is drawn, hidden and then redrawn. The default setting of 0 disables cursor blinking. When set to a none zero value the variable is split into two componants, the first 16 bits, or lower short, sets the cursor visible time in milliseconds, and the higher short sets the hidden time. If the hidden time is set to 0 then the cursor will be hidden for the same length of time it is visible.

The cursor blink rate can be setup in the platform section of user−setup(3).

$cursor−color sets the cursor's fore−ground color, and can greatly improve cursor visibility. *colorNum* is a integer palette number created using add−color(2), the default is 0.

PLATFORM

UNIX termcap interface does not support **$cursor−color**.

EXAMPLE

The following example sets the cursor visible time to 600 ms (0x258) and a hidden time to 200 ms (0xc8):

```
set-variable $cusror-blink 0x00c80258
```

SEE ALSO

user−setup(3), add−color(2), $global−scheme(5), $ml−scheme(5), $mode−line−scheme(5), $system(5).

$cursor−x(5)

NAME

$cursor−x − Cursor X (horizontal) position
$cursor−y − Cursor Y (vertical) position

SYNOPSIS

$cursor−x *integer*

0 <= *integer* <= $frame−width − 1

$cursor−y *integer*

0 <= *integer* <= $frame−depth − 1

DESCRIPTION

$cursor−x and **$cursor−y** are automatically set to the position of the cursor at the last screen update (i.e. the variables are not updated between screen updates). The top left character of the screen is coordinate 0,0 bottom right is $frame−width, $frame−depth.

NOTES

These variables can not be set. Any attempt to set them will result in an error.

SEE ALSO

$mouse−x(5), $frame−depth(5), $frame−width(5).

$debug(5)

NAME

$debug − Macro debugging flag

SYNOPSIS

$debug *debugLevel*; Default is `0`

−2 <= *debugLevel* <= 2

DESCRIPTION

$debug is a flag to trigger macro debugging. A setting of 1 or 2 enables debugging, 0 disables debugging (default). A **$debug** setting of 2 debugs all macro lines encountered, whereas a setting of 1 debugs only the lines executed, i.e. if a false **!if** was encountered the lines within the **!if** would not be printed. Problems arise with **!elif** and **!else** and a *debugLevel* setting of 1 as the **!elif** and **!else** lines are never printed.

A −ve setting disables debugging and has no immediate effect. However as soon as the bell is rung the value is inverted (−1 to 1, −2 to 2) enabling debugging. This can be invaluable when tracing problems, for example the following macro code will loop infinitely:−

```
!repeat
    beginning-of-line
    backward-char
    !force forward-line
!until &not $status
```

This is a fairly obvious bug, but if buried in a thousand lines of macro code it could be very difficult to spot and to find it during execution would be very tedious if not impossible. But by setting **$debug** to −1 the macro can be executed as normal and as soon as the macro is stuck the user can simply press "`C-g`" (**abort−command**) which rings the bell and starts macro debugging at the current execution point.

SEE ALSO

execute−file(2).

$delay−time(5)

NAME

$delay−time − Mouse time event delay time
$repeat−time − Mouse time event repeat time

SYNOPSIS

$delay−time *milliseconds*; Default is `500`
$repeat−time *milliseconds*; Default is `25`

10 <= *milliseconds* <= t

DESCRIPTION

$delay−time sets the time waited between the user picking a mouse button and the generation of a `mouse−time−?` key event.

When user presses the left button (say) a `mouse−pick−1` key event is generated, If this key is bound then the command it is bound to is executed. If the user then holds down the button for **$delay−time** or more milliseconds then MicroEmacs checks the binding of the special `mouse−time−1` key, if this pseudo key is bound then the command it is bound to will be executed.

If the user continues to hold down the button for a further **$repeat−time** milliseconds another **mouse−time−1** key event will be generated. A **mouse−time−1** key event will be generated after every **$repeat−time** milliseconds until the user releases the button, at which point a `mouse−drop−1` key event is generated.

EXAMPLE

The following example implements the vertical scroll−bar up and down scrolling arrows for a buffer window:−

```
define−macro mouse−pick−command
    set−cursor−to−mouse
    !if &equ &band $mouse−pos 15 5
        ml−write "Mouse on up−arrow"
        1 scroll−up
        1 global−bind−key scroll−up "mouse−time−1"
    !elif &equ &band $mouse−pos 15 9
        ml−write "Mouse on down−arrow"
        1 scroll−down
        1 global−bind−key scroll−down "mouse−time−1"
    !endif
!emacro
```

```
define-macro mouse-drop-command
    !force global-unbind-key "mouse-time-1"
!emacro

global-bind-key mouse-pick-command "mouse-pick-1"
global-bind-key mouse-drop-command "mouse-drop-1"
```

SEE ALSO

$idle−time(5), set−cursor−to−mouse(2), $mouse−pos(5).

$file−ignore(5)

NAME

$file−ignore − File extensions to ignore

SYNOPSIS

$file−ignore "*string*"; Default is ""

DESCRIPTION

$file−ignore specifies a space separated list of file endings which the file completion is to ignore. This is used by any function which prompts the user for a file name, such as find−file(2). A file ending in this case is NOT the extension but the last *n* characters where *n* is the number of characters in the specified ignore file.

EXAMPLE

To ignore all files which have the extension "o", using:

```
set-variable $file-ignore "o"
```

would not only ignore "foo.o", but also "foo.oo", "foo.po" and "foo" as well as any file that ends in an "o". What is really required is

```
set-variable $file-ignore ".o"
```

It is useful to ignore the "./" and "../" directories so that a directory containing one file will auto−complete to that one file. This is achieved by using:

```
set-variable $file-ignore "./"
```

To ignore MicroEmacs '02 backup files ("~"), C object files (".o"), "./" and "../" directories try using:

```
set-variable $file-ignore "~ .o ./"
```

NOTES

The file completion only completes further than the first non−unique point in the current list of possibles if and only if it can ignore all but one file, so if the current directory contains:

```
./ ../ foo foo.c foo.c~ foo.o
```

using the above ignore list, completing with "" has no effect as "`foo`" and "`foo.c`" cannot be ignored; completing with "`foo.`" will however complete to "`foo.c`".

SEE ALSO

find−file(2).

$file−names(5)

NAME

$file−names − Filtered file name list

SYNOPSIS

$file−names *FileName*

DESCRIPTION

$file−names must first be initialized to the required filter string, if the variable is evaluated before it is initialized the value will be set to "*ABORT*" and the command will fail.

The filter takes the form of a regex. The filter string should also contain the path to the required directory, the path many not contain wild−cards. If no path is specified the the path of the current buffers file name is taken, if the current buffer has no file name then the current working directory is used.

On initialization, $result(5) is set to the absolute path of the directory being evaluated.

Once initialized, evaluating **$file−names** returns the name of the next buffer which matches the filter until no more buffers are found, in which case an empty string is returned.

EXAMPLE

The following example creates a list of all files in the current directory to a fixed buffer "**files**". Note that &set(4) is used on the !while(4) statement to avoid evaluating **$file−names** twice per loop.

```
set-variable $file-names ".*"
!force 0 delete-buffer "*files*"
1 popup-window "*files*"
insert-string &spr "Directory listing of %s\n\n" $result
!while &not &seq &set #l0 $file-names ""
    insert-string &spr "    %s\n" #l0
!done
beginning-of-buffer
-1 buffer-mode "edit"
1 buffer-mode "view"
```

NOTES

Unlike MS−DOS and Windows systems, to match every file a filter of just "*" is required. A filter of "*.*" only matches file names with a '.' in them.

The list of files is evaluated when the variable is initialized, files created after the initialization will not be included in the list.

SEE ALSO

$result(5), find−file(2), $buffer−fname(5), $buffer−names(5), $command−names(5), $mode−names(5), Regular Expressions.

$file−template(5)

NAME

$file−template − Regular expression file search string

SYNOPSIS

$file−template "*string*"; Default is ""

DESCRIPTION

$file−template defines a regular expression search string used to identify a file in the grep(3) and compile(3) buffers. The format of the string is the same as magic mode search strings (see search−forward(2)).

EXAMPLE

A UNIX file name may be considered to contain any ASCII character except a space or a '`:`' (used as a divider in many programs). Thus **$file−template** should be:

```
set-variable $file-template "[!-9;-z]+"
```

This will correctly identify "`foo.c`" in the following example.

```
foo.c: 45:      printf("hello world\n") ;
```

SEE ALSO

$line−template(5), compile(3), get−next−line(2), grep(3), search−forward(2).

$fill−bullet(5)

NAME

$fill−bullet − Paragraph filling bullet character set
$fill−bullet−len − Paragraph filling bullet search depth

SYNOPSIS

$fill−bullet "*string*"; Default is "`*)].-`"
$fill−bullet−len *length*; Default is `5`

0 <= *length* <= $fill−col

DESCRIPTION

$fill−bullet contains the set of characters which are classified as bullet markers for fill−paragraph(2). If these characters are encountered in the first **$fill−bullet−len** characters of the paragraph AND the character is followed by a `SPACE` or a `tab` character then the user is given the option to indent to the right of the bullet.

$fill−bullet−len determines the maximum depth into the paragraph (in characters) the filling routines should search for a bullet character. The default value is 15. Note that the paragraph starts at the first non−white space character. e.g. to detect "`xviii) `" as a bullet then the bullet length must be set to at least 6 to detect the bullet character "`)`".

EXAMPLE

Examples of filled bullet paragraphs are shown as follows, based on the default **$fill−bullet** character set.

```
a) This is an  example of a  fill-paragraph.  The  closing
   bracket is classified as a bullet character and filling
   optionally takes place to the right of the bullet.

a] Another paragraph

*  A bullet paragraph

1. A numbered paragraph.

item - A dashed bullet.
```

SEE ALSO

$fill−col(5), $fill−ignore(5), $fill−mode(5), fill−paragraph(2), justify(2m).

$fill−col(5)

NAME

$fill−col − Paragraph Mode; right fill column

SYNOPSIS

$fill−col *columnNumber*; Default is `78`

−1 <= *columnNumber* <= 32767

DESCRIPTION

$fill−col defines the current fill column number. *columnNumber* defaults to 78 when undefined. This value is used in conjunction with justify(2m) and wrap(2m) modes.

SEE ALSO

buffer−mode(2), fill−paragraph(2), justify(2m), wrap(2m).

$fill−eos(5)

NAME

$fill−eos − Paragraph filling; end of sentence fill characters
$fill−eos−len − Paragraph filling; end of sentence padding length

SYNOPSIS

$fill−eos "*string*"; Default is "`.!?`"

$fill−eos−len *integer*; Default is `1`
0 <= *integer* <= *n*

DESCRIPTION

$fill−eos defines the end of sentence character set. Sentences ending in these characters are padded with additional *end−of−sentence* spaces, as defined by **$fill−eos−len**.

$fill−eos−len sets the number of spaces inserted after a full stop during paragraph filling. The default is 1 space.

SEE ALSO

fill−paragraph(2).

$fill−ignore(5)

NAME

$fill−ignore − Ignore paragraph filling character(s)

SYNOPSIS

$fill−ignore "*string*"; Default is ">_@"

DESCRIPTION

$fill−ignore describes a set of characters used by fill−paragraph(2) which disable paragraph filling when they appear at the start of a paragraph. An obvious example is an inserted mail message which is usually quoted with ">" characters. Any attempt to fill the paragraph causes **fill−paragraph** to skip to the end of it.

EXAMPLE

This is an example of an ignored paragraph when encountered by **fill−paragraph** with the default ignore character set.

```
> This is an example of a paragraph that
> is ignored.
```

SEE ALSO

$fill−col(5), $fill−bullet(5), $fill−mode(5), fill−paragraph(2), justify(2m).

$fill−mode(5)

NAME

$fill−mode − Paragraph mode; justification method

SYNOPSIS

$fill−mode *justification*; Default is `N`

justification `b` | `c` | `l` | `n` | `o` | `r` | `B` | `C` | `L` | `N` | `R`

DESCRIPTION

$fill−mode defines the justification mode i.e. *left*/*right*/*both*/... The default value is none automatic (**N**). The modes available are:−

b Both

Enables left and right margin justification.

c Center

Enables center justification.

l Left

Enables left justification.

n None

No filling is performed, adjacent lines are not merged into a single line. This subtly different from *left* justification which fills lines to the $fill−col(5).

o One Line

Enables the filling of the paragraph to a single line. Typically used to prepare a file for transfer to a word processing package.

r Right

Enables right justification.

B Both (automatic)

Automatically determines the mode, defaulting to left and right (both) justification.

C Center (automatic)

Automatically determines the mode, defaulting to center justification.

L Left (automatic)

Automatically determines the mode, defaulting to left justification.

N None (automatic)

Automatically determines the mode, defaults to *both* and not *none*.

R Right (automatic)

Automatically determines the mode, defaulting to right justification.

The lines are automatically justified only when the justification mode justify(2m) is enabled. Justification is performed between the left and right margins, defined as 0 and $fill−col(5) respectively.

Automatic Filling

Automatic filling is performed when the mode **$fill−mode** is specified in upper case. The format of the line (and adjacent lines) is interrogated and an *informed* guess is made as to the expected formating which is then adopted. The criteria for automatic formatting is defined as follows:−

center

If the left and right margins contain approximately the same amount of white space +/−1 character then the paragraph is centered.

right

If the text commences past half of the $fill−col(5) (i.e. first half of the line comprises white space) AND the line extends to, or past, the `$fill-col` then the paragraph is assumed to be right justified.

none

If the text commences in column 0 and occupies less than half of the line then the paragraph is assumed to be not justified. (i.e. left justified, but consecutive lines of the paragraph are not filled)

default

If none of the above criteria are met then the default mode is adopted, as determined by the lower−case value of the **$fill−mode** value. **SEE ALSO**

$fill−col(5), buffer−mode(2), fill−paragraph(2), justify(2m).

$find−words(5)

NAME

$find−words − Filtered word list

SYNOPSIS

$find−words *word*

DESCRIPTION

$find−words must first be initialized to the required filter string, if the variable is evaluated before it is initialized the value will be set to "*ABORT*" and the command will fail.

The filter string can contain wild−card characters compatible with most file systems, namely:−

?

Match any character.

[abc]

Match character only if it is *a*, *b* or *c*.

[a−d]

Match character only if it is *a*, *b*, *c* or *d*.

[^abc]

Match character only if it is not *a*, *b* or *c*.

*****

Match any number of characters.

Note that these are not the same characters used by exact(2m) mode.

Once initialized, evaluating **$find−words** returns the next word found in the main spell dictionaries which matches the filter until no more words are found, in which case an empty string is returned.

EXAMPLE

The following example finds all the words with "*foo*" in it (e.g. "*footnote*"), printing them to the massage line one at a time. Note that &set(4) is used on the !while(4) statement to avoid evaluating **$find−words** twice per loop.

```
set-variable $find-words "*foo*"
!while &not &seq &set #l0 $find-words ""
    100 ml-write &cat "Word: " #l0
!done
```

NOTES

The order of the words is undefined.

Due to the way words are derived, it is possible to have two or more copies of a word in the dictionary. If this is a matching word **$find−words** will return the word two or more times.

SEE ALSO

spell(2).

$fmatchdelay(5)

NAME

$fmatchdelay− Fence matching delay time

SYNOPSIS

$fmatchdelay *delayTime*; Default is `2000`

0 <= *delayTime* <= *n*

DESCRIPTION

The number of milliseconds to wait in a fence match operation. When a closing fence '`)`' '`]`' or '`}`' is added the opening fence is searched for, scrolling the screen up where necessary, this is the time that the opening fence is displayed, interruptible by typing any key.

When cmode(2m) is enable the search algorithm used is '*C*' aware and if a matching fence is not found then the bell is rung as a warning. The automatic matching of fences can be enabled/disabled via the fence(2m) mode.

A cursor can be moved to the matching fence using the goto−matching−fence(2) command.

SEE ALSO

fence(2m), cmode(2m), goto−matching−fence(2).

$frame−depth(5)

NAME

$frame−depth − Number of lines on the current frame canvas
$frame−width − Number of columns on the current frame canvas

SYNOPSIS

$frame−depth *integer*

3 <= *integer* <= 400

$frame−width *integer*

8 <= *integer* <= 400

DESCRIPTION

These variables allow the viewable size of the current frame canvas to be determined.

$frame−depth identifies depth of the current frame given as the number of character lines. This is the whole frame width, not just what is currently visible. The value returned is in the range 3 − *n*, *n* is system dependent but no greater than 400.

$frame−width identifies the width of the current frame as the number of character columns. The value returned is in the range 8 − *n*, *n* is system dependent but no greater than 400.

NOTES

The name of these variables changed from **$screen−depth** and **$screen−width** due to the support for multiple frames introduced in April 2002.

SEE ALSO

change−frame−depth(2), change−frame−width(2).

$global−scheme(5)

NAME

$global−scheme − Default global buffer color scheme.

SYNOPSIS

`$global−scheme` `schemeNum`; Default is `0`

DESCRIPTION

$global−scheme defines the default buffer color scheme to *schemeNum*, a color scheme defined by add−color−scheme(2).

SEE ALSO

add−color(2), add−color−scheme(2), $buffer−hilight(5), $buffer−scheme(5), $cursor−color(5), $trunc−scheme(5), $ml−scheme(5), $osd−scheme(5), $mode−line−scheme(5), $scroll−bar−scheme(5), $system(5).

$home(5)

NAME

$home − Users `home' directory location

SYNOPSIS

$home *directory*

DESCRIPTION

The file naming convention utilizes tilde ('~') to identify the users home directory (`$HOME`). When entering a file name:

```
~/xxx    -> $home/xxx
~yyy/xxx -> $home/../yyy/xxx
```

On most systems this is automatically set to the environment variable "HOME" if it is defined or may be defined explicitly in the start−up file. '~' may be used in the `me.emf` files but must be specified as '~'. It may be picked up in command files as **$home**.

$idle−time(5)

NAME

$idle−time − System idle event delay time

SYNOPSIS

$idle−time *milliseconds*; Default is `1000`

10 <= *milliseconds* <= t

DESCRIPTION

$idle−time sets the time waited between the last user event and the generation of a `idle−pick` key event. When user input stops for **$idle−time** milliseconds MicroEmacs checks the binding of the special `idle−pick` key, if this pseudo key is bound then the command it is bound to will be executed. MicroEmacs will then cycle, generating a `idle−pick` every **$idle−time** milliseconds until user activity starts. At this point a `idle−drop` key event is generated, if this pseudo key is bound then the command it is bound to will be executed.

This system is useful for things which can be done in the background.

EXAMPLE

The following example is taken from `ssaver.emf` and implements a simple screen saver:−

```
set−variable %screen−saver 0
define−macro screen−saver
    !if &not &pinc %screen−saver 1
        !if &seq @cck "idle−pick"
            ; default is to switch on in 5 minutes time
            &cond @? @# 300000 create−callback screen−saver
        !else
            !if &seq @cck "callback"
                @# create−callback screen−saver
            !elif @?
                ; user has suppled argument, install or remove
                !if &gre @# 0
                    &mul @# 60000 global−bind−key screen−saver "idle−pick"
                !else
                    !force global−unbind−key "idle−pick"
                !endif
                set−variable %screen−saver &sub %screen−saver 1
                !return
            !endif
            set−variable @# $frame−depth
            !while &dec @# 1
```

```
                    2 screen-poke @# 0 $global-scheme &spr "%n" $frame-width " "
                !done
                0 screen-poke 0 0 $global-scheme &spr "%n" $frame-width " "
                -1 show-cursor
                ; must set this to stop recursion when waiting for a key!
                set-variable %screen-saver 0
                set-variable @# @cg
                set-variable %screen-saver 1
                1 show-cursor
                screen-update
                ml-clear
            !endif
        !endif
        set-variable %screen-saver &sub %screen-saver 1
    !emacro
```

NOTES

Care must be taken to ensure that a recursive loop is not created, consider the following example:−

```
define-macro bored
    !if &iseq @mc1 "Are you bored (y/n)? " "nNyY" "y"
        ml-write "Play a silly game!"
    !endif
!emacro
global-bind-key bored idle-pick
```

If this was executed MicroEmacs would very quickly crash! As soon as a second past **bored** would execute, which will prompt the user and wait for input. If a second passes without input **bored** will be executed again and again and again until stack memory runs out! To avoid this `idle-pick` should be unbound before waiting for user input, i.e.:−

```
define-macro bored
    global-unbind-key idle-pick
    !if &iseq @mc1 "Are you bored (y/n)? " "nNyY" "y"
        ml-write "Play a silly game!"
    !endif
    global-bind-key bored idle-pick
!emacro
global-bind-key bored idle-pick
```

SEE ALSO

$delay−time(5).

$kept−versions(5)

NAME

$kept−versions − Number of backups to be kept

SYNOPSIS

$kept−versions *integer*; Default is 0

0 <= *integer* <= n

DESCRIPTION

$kept−versions allows the user to specify the number of backup versions that are required for each file. For file "`XXXX`", each backup version is renamed to "`XXXX.~?~`", where `?` is the backup number. If **$kept−versions** is set to 0 this feature is disabled and the default single backup file is created.

The most recent backup will always be `.~0~` and the last version will be `.~n~` where `n` is **$kept−versions** − 1. when the file is next saved the `.~0~` backup file is moved to `.~1~`, `.~1~` to `.~2~` etc, backup `.~n~` is removed. Evidently if **$kept−versions** it set to a large number this can effect performance.

RESTRICTIONS

$kept−versions may only be used when DOS file name restrictions are not enabled. This means that some systems (such as DOS) cannot use this functionality, see $system(5) for more information. Backup files are only created when buffer mode backup(2m) is enabled.

NOTES

This feature is not supported when writing ftp files, a single backup file is created when backup files are enabled.

SEE ALSO

$system(5), autosv(2m), backup(2m), ftp(3), save−buffer(2).

$line−scheme(5)

NAME

$line−scheme − Set the current line color scheme

SYNOPSIS

$line−scheme *schemeNum*; Default is `−1`

DESCRIPTION

$line−scheme sets the color scheme to be used for the current line of the current window. The given *schemeNum* can be any scheme number previously defined by the function add−color−scheme(2).

A line's $line−scheme setting is removed by setting the variable to −1.

A $line−scheme setting takes precedence over the buffer's color scheme ($buffer−scheme(5)) and the buffer's hilighting scheme ($buffer−hilight(5)).

EXAMPLE

c−hash−eval(3) greys out lines of text by doing:

```
set-variable $line-scheme %lblack
```

The lines are rest by doing

```
set-variable $line-scheme -1
```

The gdb(3) interface hilights the current line of source by doing:

```
set-variable $line-scheme %yellow-lblack
```

NOTES

Due to line storage restrictions, only 15 different color schemes can be used in a buffer at any one time. When the 16th color scheme is used it replaces the first color scheme, all lines using the first color scheme will be colored using the new color scheme.

SEE ALSO

add−color−scheme(2), c−hash−eval(3), $buffer−scheme(5), $buffer−hilight(5), $mode−line−scheme(5), $scroll−bar−scheme(5), $system(5).

$line−template(5)

NAME

$line−template − Command line regular expression search string

SYNOPSIS

$line−template "*string*"; Default is ""

DESCRIPTION

$line−template defines a regular expression search string used to identify a line number in the grep(3) and compile(3) buffers. The format of the string is the same as magic mode search strings (see search−forward(2)).

EXAMPLE

The line number may be considered to contain any numeric number, thus **$line−template** is defined as:

```
set-variable $line-template "[0-9]+"
```

This correctly identifies "45" in the following ***grep*** output example:

```
foo.c: 45:      printf("hello world\n") ;
```

SEE ALSO

$file−template(5), compile(3), get−next−line(2), grep(3), search−forward(2).

$ml−scheme(5)

NAME

$ml−scheme − Message line color scheme

SYNOPSIS

$ml−scheme *schemeNum*; Default is `0`

DESCRIPTION

$ml−scheme defines the color scheme to be used on the message line, the color scheme *schemeNum* identifies the foreground and background color and is defined by an invocation to add−color−scheme(2).

The background color is always defined by $global−scheme(5).

SEE ALSO

$global−scheme(5), $osd−scheme(5), $mode−line−scheme(5), $scroll−bar−scheme(5), $system(5), add−color−scheme(2).

$mode−line(5)

NAME

$mode−line − Mode line format

SYNOPSIS

$mode−line "*string*"; Default is "`%s%r%u me (%e) - %l %b (%f) `"

DESCRIPTION

$mode−line defines the format of the mode line printed for every window, where the character following a percent ('%') has the following effect:−

`D`	Prints the current day.
`M`	Prints the current month.
`Y`	Prints the current year (2 digits).
`y`	Prints the current year (4 digits).
`b`	Prints the current buffer's name.
`c`	Prints the current buffer's column number.
`e`	Prints the current buffer's editing modes.
`f`	Prints the current buffer's file name.
`h`	Prints the current hour of the day.
`k`	Prints the current keyboard macro status.
`l`	Prints the current buffer's line number.
`m`	Prints the current minute of the hour.
`n`	Prints the current buffer's total number of lines.
`r`	Prints the current root user status (UNIX only).
`s`	Prints the horizontal window split character.
`u`	Prints the current buffer's (un)changed or view mode flag.
`%`	Prints a percentage escape character.
`−`	Prints a literal minus character ('`−`') − see NOTES.
`*`	All other characters are printed literally.

NOTES

- Refer to $window−chars(5) for the characters utilized in the mode line. Typically a the '`−`' character is changed to a '`=`' if it is the current window. If a '`−`' is always required, use "`%−`".
- A buffer can have its own mode−line, and be uneffected be the global mode line, see $buffer−mode−line(5).

SEE ALSO

$buffer−mode−line(5), $mode−line−scheme(5), $window−chars(5).

$mode−line−scheme(5)

NAME

$mode−line−scheme − Mode line color scheme

SYNOPSIS

$mode−line−scheme *schemeNum*; Default is `1`

DESCRIPTION

Sets the window mode−line color scheme, defining the foreground and background colors. The *schemeNum* is defined by a previous invocation to add−color−scheme(2).

SEE ALSO

add−color−scheme(2), $global−scheme(5), $ml−scheme(5), $scroll−bar−scheme(5), $system(5).

$mode−names(5)

NAME

$mode−names − Filtered mode name list

SYNOPSIS

$mode−names *ModeName*

DESCRIPTION

$mode−names must first be initialized to the required filter string, if the variable is evaluated before it is initialized the value will be set to "*ABORT*" and the command will fail. The filter takes the form of a regex.

Once initialized, evaluating **$mode−names** returns the name of the next mode which matches the filter until no more modes are found, in which case an empty string is returned.

EXAMPLE

The following example prints out the name of all modes to the massage line one at a time. Note that &set(4) is used on the !while(4) statement to avoid evaluating **$mode−names** twice per loop.

```
set-variable $mode-names "*"
!while &not &seq &set #l0 $mode-names ""
    100 ml-write &cat "mode: " #l0
!done
```

SEE ALSO

buffer−mode(2), &bmode(4), $buffer−names(5), $command−names(5), Regular Expressions.

$mouse(5)

NAME

$mouse − Mouse configuration variable

SYNOPSIS

$mouse *bitmask*; Default is system dependent

DESCRIPTION

The **$mouse** is used to define and configure the MicroEmacs mouse support, it is a bit based flag where:−

0x00f

Defines the number of button the mouse has, only values 1, 2 & 3 are useful. By default MicroEmacs uses the system information to determine the number of buttons on the mouse, this is not fool proof so the user can set these bits to the appropriate number if the initial value is incorrect.

0x010

If set the mouse is enabled, if clear the mouse will not function. On systems which do not support mice (such as UNIX Termcap) this bit will be clear and can not be altered.

0x020

If set the buttons are reversed, i.e. the left button becomes the right and vice versa. By default this bit is clear.

0xf0000

Defines the current mouse icon to used, valid values are as follows:

0x00000 − Set mouse to default icon.
0x10000 − Set mouse to arrow icon.
0x20000 − Set mouse to text I−beam icon.
0x30000 − Set mouse to crosshair icon.
0x40000 − Set mouse to the grab icon.
0x50000 − Set mouse to the wait icon.
0x60000 − Set mouse to the stop icon.

This feature is not supported on some systems and on others some icons are not obvious due to platform limitations.

EXAMPLE

The following example checks that the mouse is currently available, if not, it aborts.

```
!if &not &band $mouse 0x10
    ml-write "[Mouse support is not currently available]"
    !abort
!endif
```

NOTES

The mouse can be easily configured using user−setup(3).

SEE ALSO

user−setup(3), $system(5), $platform(5).

$mouse−pos(5)

NAME

$mouse−pos − Mouse position information

SYNOPSIS

$mouse−pos *integer*

DESCRIPTION

$mouse−pos is generated by invocation of the command set−cursor−to−mouse(2). The variable is set to a value that indicates the position of the mouse within a window. The values to the mouse intersection are interpreted as follows:−

0 − Text area

Intersection with the window text area.

1 − Message Line

Intersection with the message line.

2 − Mode Line

Intersection with the mode line.

3 − Horizontal Separator

Intersection with the horizontal window separator. This value is only set if a scroll bar is not present.

4 − Up Arrow

Intersection with the scroll bar up−arrow character.

5 − Upper Shaft

Intersection with the scroll bar upper shaft (above the scroll box).

6 − Scroll Box

Intersection with the scroll bar scroll box.

7 − Lower Shaft

Intersection with the scroll bar lower shaft (below the scroll box).

8 − Down Arrow

Intersection with the scroll bar down−arrow character.

9 − Corner

Intersection with the window corner, that is the character at the intersection of the scroll bar (or separator) and the mode line.

10 − Menu Line

Intersection with the menu line.

255 − Error

The position of the mouse could not be determined. This value should not arise, if it does then it is an indication that the window structure is probably corrupted. A delete−other−windows(2) is suggested or rapid exit from the editor after a save−some−buffers(2) command to save any edits (latter option is preferred).

Bit 4 − 2nd Column

Bit 4 (16) is set if 2 character column scroll bar or vertical window separator is in effect and the cursor exists in the second column This value is bitwise OR'ed with the aforementioned intersection values. **EXAMPLE**

The following macro can be used to print out the current position of the mouse, try binding the macro to the "`mouse−move`" key:

```
define−macro print−mouse−position
    !force set−cursor−to−mouse
    set−variable #l0 &band $mouse−pos 15
    !if &equ #l0 0
        ml−write "Mouse in text window"
    !elif &equ #l0 1
        ml−write "Mouse on message line"
    !elif &equ #l0 2
        ml−write "Mouse on Mode line"
    !elif &and &gre #l0 2 &les #l0 10
        ml−write "Mouse on scroll bar"
    !elif &equ #l0 10
        ml−write "Mouse on corner"
    !elif &equ #l0 11
        ml−write "Mouse on menu line"
    !endif
!emacro

global−bind−key print−mouse−position mouse−move
```

$mouse−pos is utilized by the mouse picking code, found in macro file `mouse.emf`.

SEE ALSO

$mouse−x(5), $mouse−y(5), set−cursor−to−mouse(2), set−scroll−with−mouse(2).

$mouse−x(5)

NAME

$mouse−x − Mouse X (horizontal) position
$mouse−y − Mouse Y (vertical) position

SYNOPSIS

$mouse−x *integer*

0 <= *integer* <= $frame−width − 1

$mouse−y *integer*

0 <= *integer* <= $frame−depth − 1

DESCRIPTION

$mouse−x and **$mouse−y** are automatically set to the position of the mouse at the last mouse event, where an event is a button press or release. Initialized to 0,0. The top left character of the screen is coordinate 0,0 bottom right is $frame−width, $frame−depth.

NOTES

These variables can not be set. Any attempt to set them will result in an error.

SEE ALSO

set−cursor−to−mouse(2), $mouse−pos(5), $cursor−x(5), $frame−depth(5), $frame−width(5).

$osd−scheme(5)

NAME

$osd−scheme − OSD color scheme

SYNOPSIS

$osd−scheme *schemeNum*; Default is `1`

DESCRIPTION

$ml−scheme defines the color scheme by default on an osd(2) dialog, the color scheme *schemeNum* identifies the foreground and background color and is defined by an invocation to add−color−scheme(2). Every osd dialog can over−ride this value by using the '*S*' flag.

SEE ALSO

osd(2), add−color−scheme(2), $global−scheme(5), $ml−scheme(5), $mode−line−scheme(5), $scroll−bar−scheme(5), $system(5).

$platform(5)

NAME

$platform − MicroEmacs host platform identifier
%platform − MicroEmacs host platform type identifier

SYNOPSIS

$platform "*string*"; Default is platform specific
%platform "*string*"; Default is platform specific

DESCRIPTION

The **$platform** variable is a fixed ASCII string used to identify the current working platform, attempts to set this variable result in an error returned from set−variable(2).

Possible values are:

"**aix**"

All IBM AIX O/S.

"**dos**"

All IBM−PCs and compatibles running MS−DOS.

"**freebsd**"

All FreeBSD O/S.

"**hpux**"

All Hewlett Packard's with HP−UX O/S.

"**irix**"

All Silicon Graphics (SGI) IRIX platforms 4.x, 5.x, 6.x.

"**linux**"

All LINUX O/S.

"**sunos**"

All Sun's with SUNOS O/S.

"**unixwr1**"

PC based UNIX platform (Consensus and Unixware).

"**win32**"

Microsoft Windows based systems including Windows 3.x (with Win32s), Windows '95 and NT.

$platform is often used in **.emf** files to allow portability of macro files across platforms, allowing macro files to perform platform specific operations. $system(5) is also often used for this purpose as its value is easier to assess.

%platform is created at start−up when `me.emf` is executed, its value is identical to **$platform** except when the platform is a console in which case a '`c`' is appended to the $platform value, e.g. for MicroEmacs running a termcap version on LINUX the value will be "`linuxc`". The variable is used when the console and window based versions need to be distinguish, e.g. some of the user−setup settings.

EXAMPLE

The following example is taken from the **me.emf** file which uses the **$platform** variable to load the platform specific initialization files.

```
;
; load in the platform specific stuff
execute-file $platform
```

This could be more explicitly done by:

```
;
; load in the platform specific stuff
!if   &seq $platform "dos"            ; is it an IBM-PC running dos ?
    execute-file "dos"
!elif &seq $platform "irix"           ; is it an sgi ?
    execute-file "irix"
!elif &seq $platform "hpux"           ; is it an hp ?
    execute-file "hpux"
    .
    .
!endif
```

NOTES

The **$platform** variable can not be set. Any attempt to set it will result in an error.

SEE ALSO

$system(5), set−variable(2).

$progname(5)

NAME

$progname − Program file name

SYNOPSIS

$progname *string*

DESCRIPTION

$progname is set the the MicroEmacs '02 program file name currently being run. This can be used by macros for many purposes, from spawning another MicroEmacs '02 session to working out where MicroEmacs '02 is running from.

EXAMPLE

The following example is used to spawn of another MicroEmacs '02 command to create a C tags file:−

```
shell−command &cat $progname " \"@ctags\" *.c *.h"
```

SEE ALSO

me(1).

$random(5)

NAME

$random − Generate a random number

SYNOPSIS

`$random` *integer*

0 <= *integer* <= 65535

DESCRIPTION

The **$random** variable returns a unique random number in the range 0 − *n* on reference to the variable.

The random number is derived from the system's random number generator (the quality of which is often dubious so try to avoid using the bottom bits). Setting this variable with any value resets the random sequence using the system time as the seed.

The range of the random number generator is system dependent. The value is typically capped using the &mod(4) arithmetic operator.

EXAMPLE

The variable may be assigned to generate a new seed as follows:−

```
set-variable $random 0        ; Set it so we get a new seed
```

The returned value is used with the **&mod** operator to limit the value to a desired range:−

```
set-variable %random0to9 &mod $random 10
```

SEE ALSO

&mod(4).

$rcs−file(5)

NAME

$rcs−file − RCS (and SCCS) file name
$rcs−ci−com − RCS (and SCCS) check in command
$rcs−cif−com − RCS (and SCCS) check in first command
$rcs−co−com − RCS (and SCCS) check out command
$rcs−cou−com − RCS (and SCCS) check out unlock command
$rcs−ue−com − RCS (and SCCS) unedit file command

SYNOPSIS

$rcs−file "*string*"; Default is ""
$rcs−ci−com "*string*"; Default is ""
$rcs−cif−com "*string*"; Default is ""
$rcs−co−com "*string*"; Default is ""
$rcs−cou−com "*string*"; Default is ""
$rcs−ue−com "*string*"; Default is ""

DESCRIPTION

RCS (Revision Control System) and SCCS (Source Code Control System) are programmers source code history data−bases. RCS introduces a system in which only one programmer can edit a source file at any one time, enforcing some form of stability in the global environment. The fact that this interface was developed for the RCS system is irrelevant, and should be usable under any other control systems such as SCCS.

When using RCS, finding a file (see find−file(2)) checks for the existence of the actual file. If this is not found then it checks for the existence of an RCS **$rcs−file** variable, and if present then it constructs the RCS file name and checks for its existence. If this file does not exist then it really is a new file and an new buffer is created. If the file does exist then the file is checked out using the **$rcs−co−com** which executes to create a file with the original file name, ready for loading.

$rcs−file is the name of the file when it is fully check in, as opposed to when it is ready to be viewed or edited. In RCS, this is usually in the `RCS` directory with an appended "`,v`", i.e. for the file `foo.c` in the `/test` directory, when fully checked in, the file will not be found at "`/test/foo.c`", but at "`/test/RCS/foo.c,v`". When testing for an RCS file, the file name is split into two parts, the path name and the file name, the path is always inserted at the start, and the file name can inserted in the rcs string by using the special "`%f`" token, thus if **$rcs−file** is set to "`RCS/%f,v`", the RCS file name is constructed from "`/test/`" + "`RCS/`" + "`foo.c`" + "`,v`".

If the RCS file is found then the **$rcs−co−com** (RCS **C**heck **O**ut **COM**mand) which is a simple system command line with the exception for `%f` which is replaced by the file name, is executed. This is expected to create the file (with the correct file name) ready for viewing.

Once a file is loaded, then the rcs−file(2) command has one of two effects:−

> If the file is in view mode then the **$rcs−cou−com** (RCS **C**heck **O**ut **U**nlock **COM**mand) is executed (system command line using the "%f" as the file name). If the RCS file does not exist then is simply toggles the view mode, allowing editing.
>
> If the file is not in view mode MicroEmacs attempts to check the file back into RCS using either **$rcs−ci−com** (if the RCS file already exists) or the the **$rcs−cif−com** (RCS **C**heck **I**n **F**irst **COM**mand). The "`%f`" is again used for the file name, the "`%m`" can also be used to get a comment from the user at check in time which will be inserted (without quotes) into the **$rcs−ci−com** command line. For example, one possible **$rcs−ci−com** setting is "`ci −m\"%m\" %f`" which uses the **ci(1)** program with the **−m** option to give a check in message.

If **rcs−file** is given a −ve argument instead of checking in or out the current buffer's file it executes the command specified by **$rcs−ue−com** to unedit or abort any changes made to the file. After the command has been executed the file is reloaded.

NOTES

The RCS variables are by default undefined and must be explicitly enabled in the start−up files.

EXAMPLE

The following are typical variable definitions for the RCS interface:−

```
set-variable $rcs-file     "RCS/%f,v"
set-variable $rcs-co-com   "co %f"
set-variable $rcs-cou-com  "co -l %f"
set-variable $rcs-ci-com   "ci -u -m\"%m\" %f"
```

Note that the **$rcs−cif−com** variable is usually left unassigned and **$rcs−ci−com** is used by default.

The following are typical variable definitions for the SCCS interface:−

```
set-variable $rcs-file     "SCCS/s.%f"
set-variable $rcs-co-com   "sccs get %f"
set-variable $rcs-cou-com  "sccs edit %f"
set-variable $rcs-ci-com   "sccs delget -y\"%m\" %f"
set-variable $rcs-ci-com   "sccs create %f"
set-variable $rcs-ue-com   "sccs unedit %f"
```

The following variable definitions can be used for MicroSoft's Visual Source Safe:−

```
set-variable $rcs-file    "%f"
set-variable $rcs-cou-com "ss.exe checkout %f"
set-variable $rcs-co-com  "ss.exe checkout %f"
set-variable $rcs-ci-com  "ss.exe checkin %f \"-c%m\""
```

The above definitions can check a file out for edit and commit changes back.

SEE ALSO

find−file(2), rcs−file(2).

$recent−keys(5)

NAME

$recent−keys − Recent key history.

SYNOPSIS

$recent−keys *string*

DESCRIPTION

$recent−keys is a system variable that displays the last 100 keys entered into the system in reverse order. This variable is typically used to solve keyboard mapping problems when keys are not bound etc. allowing a visual inspection of the input into the editor.

SEE ALSO

buffer−bind−key(2), global−bind−key(2), translate−key(2).

$result(5)

NAME

$result − Various command return values

SYNOPSIS

$result *returnValue*

DESCRIPTION

$result is used to return the results of several commands:

buffer−info(2) **$result** is set to the same output string as printed to the message−line by this command.

change−font(2)

$result is used to return the user select font when hte windows font selection dialog is used (Windows systems only).

count−words(2)

$result is set to the same output string as printed to the message−line by this command.

find−registry(2)

$result is used to return the name of a registry child node given the parent and index from the user.

get−registry(2)

$result is used to return the current value of a user supplied registry entry.

mark−registry(2)

$result is used to return the full name of the given registry node.

osd(2)

> **$result** is used to give and return information to osd item commands, information depends on the type of **osd** item.

osd−dialog(3)

osd−xdialog(3)

> **$result** is used to return the button pressed by the user.

shell−command(2)

> **$result** is set to the exit status of the **system** call. The combination of **shell−command** calls and return value checking can be used in a variety of ways, for example, to test the existence of a file:

```
set-variable %filename @ml"Enter file name"
shell-command &cat "test -f " %filename
!if &equ $result 0
    ml-write "file exists"
!else
    ml-write "file does not exists"
!endif
```

show−region(2)

> **$result** is set to the current status of the region when an argument of *0* is given to **show−region**.

spell(2)

> **$result** is used to return information on the current word, the information depends on the argument given to **spell**.

$file−names(5)

> **$result** is set to the absolute path of the **$file−names** query directory when the variable is set.

For more information see the help pages on referenced commands and variables.

NOTES

The current value of **$result** is lost on the next command call which uses it. As a call to create−callback(2) can cause the execution of a macro to interrupt another which is waiting for user input, the value of **$result** should be copied before getting user input.

SEE ALSO

buffer−info(2), change−font(2), count−words(2), find−registry(2), get−registry(2), mark−registry(2), osd(2), shell−command(2), show−region(2), spell(2), $file−names(5), create−callback(2), $status(5).

$scroll(5)

NAME

$scroll − Screen scroll control

SYNOPSIS

`$scroll` *scrollNum*; Default is `1`

0 <= *scrollNum* <= n

DESCRIPTION

$scroll controls the horizontal and vertical scrolling method used to display long lines and buffers. The variable is split into two componants, the first nibble (`0x0f`) sets the horizontal scroll, and the second nibble (`0xf0`) sets the vertical. For the purpose of documentation these parts are kept separate, but when setting the variable a single combined value must be given.

The horizontal settings are defined as follows:

`0x00`

Scroll method 0 will only scroll the current line, this is the fastest method in execution time.

`0x01`

Scroll method 1 (the default) will scroll the whole page horizontally when the scroll−left(2) and scroll−right(2) commands are used. However, when the current line must be scrolled to display the cursor due to a forward−char(2) type cursor movement, only the current line is scrolled and the rest are reset.

`0x02`

Scroll method 2 always scrolls the whole page horizontally, keeping the cursor in the current column range. If the cursor moves out of this range then all the page is scrolled to the new position. This is particularly useful when editing long lined tables.

`0x03`

Scroll method 3 fixes the scroll column using the **scroll−left** and **scroll−right** functions. If the current cursor position is not visible in the column range then only the current line is scrolled to the new position.

The vertical settings are defined as follows:

`0x00`

Scroll method 0 (the default) will scroll the current line to the middle of the current window whenver it is moved off screen, this is the fastest method in execution time.

`0x10`

Scroll method 1 will scroll the current line to the the top of the window whenver the current line is moved off the screen using backward−line(2) and to the bottom of the window when forward−line(2) is used. This creates the effect of a smooth scroll. **EXAMPLE**

The following example sets the scrolling method to be the default horizontally (`0x01`) and smooth method (`0x10`) vertically :

```
set−variable $scroll 0x11
```

SEE ALSO

scroll−left(2), forward−line(2), $window−x−scroll(5), $window−y−scroll(5).

$scroll−bar(5)

NAME

$scroll−bar − Scroll bar configuration

SYNOPSIS

$scroll−bar "*bitmask*"; Default is platform specific

DESCRIPTION

$scroll−bar defines the configuration of the scroll bar and/or the horizontal window separator for both main text windows and osd(2) dialogs. The variable is interpreted as a bit mask and defines which components of the scroll bar (or separator) should be rendered in a window. The characters used to render the scroll bar or separator are defined by $window−chars(5). The bit mask is defined as follows:−

0x001 − Vertical Scroll Bar Width

Bit 0 controls the width of the vertical scroll bar (or separator). A value of 0 corresponds to a single column width, a value of 1 is a double column width.

0x002 − Upper end cap

Bit 1 set indicates that the scroll bar has an upper end cap. This is the up arrow character at the top of a scroll bar.

0x004 − Lower end cap

Bit 2 set indicates that the scroll bar has a lower end cap. This is the down arrow character at the bottom of a scroll bar.

0x008 − Corner

Bit 3 set indicates that separate corner character is used at the intersection of the mode line and the separator.

0x010 − Scroll Box Enable

Bit 4 determines if the scroll bar has a scrolling box, when the bit is set each scroll bar will have a scroll box. When clear, scroll bars are rendered according to bits 0−3 & 7 only and the main area of the bar is left empty.

0x020 − Reverse Video Box

Bit 5 when set enables the scroll box to be rendered in reverse video, that is the background and foreground/hilight scroll colors are interchanged. This bit is typically set on X−Window platforms allowing the scroll box to comprise of `SPACE` characters allowing a solid box to be rendered in the foreground color.

Bit 5 is only enacted if scroll boxes are enabled.

0x040 − Horizontal Scroll Bar Width

Bit 6 controls the width of the horizontal scroll bar, used only by osd(2). A value of 0 corresponds to a single column width, a value of 1 is a double column width.

0x080 − Splitter

Bit 7 set indicates that the scroll bar has a splitter. This is the split bar character at the top of a scroll bar.

0x100 − Enable window Scroll Bars

When Bit 8 is clear, scroll bars are not present on windows. If a horizontal split has been performed then the window separator is rendered plain. This is useful when performance is important, as scroll bars require constant up−date.

0x200 − Horizontal Scroll Bar Width

Bit 9 enables scroll bars, when the bit is set each window is assigned a scroll bar in the right−hand column(s) of the window with a scroll box. **SEE ALSO**

$mouse−pos(5), $scroll−bar−scheme(5), set−scroll−with−mouse(2), $window−chars(5).

$scroll−bar−scheme(5)

NAME

$scroll−bar−scheme − Scroll bar color scheme

SYNOPSIS

$scroll−bar−scheme *schemeNum*; Default is `1`

DESCRIPTION

Sets the horizontal window scroll bar color scheme, assigning the foreground, background and selection colors which are used to render the vertical separator / scroll bars (see add−color−scheme(2). The separator is rendered in reverse video, i.e. the foreground color of the color scheme is used as the background color, and vice versa.

The separator is rendered in the standard colors when the associated buffer is not active, and in the current color when the buffer is active.

The scroll−bar is the window separator constructed by split−window−horizontally(2) or when the scroll bars are enabled via $scroll−bar(5).

SEE ALSO

$global−scheme(5), $ml−scheme(5), $mode−line−scheme(5), $scroll−bar(5), $system(5), $window−chars(5), split−window−horizontally(2).

$show−modes(5)

NAME

$mode−line − Select buffer modes to display

SYNOPSIS

$show−modes "*bit−string*"; Default is ""

DESCRIPTION

$show−modes defines which buffer modes are displayed on the mode−line.

SEE ALSO

$user−setup(3), $mode−line(5).

$show−region(5)

NAME

$show−region − Enable the hilighting of regions

SYNOPSIS

$show−region *flag*; Default is `1`

DESCRIPTION

$show−region enables or disables the current region hilighting, normally associated with mouse interaction in a buffer. Region hilighting occurs between the *mark* (see set−mark(2)) and *point* (current cursor) positions within the current buffer. An argument *n* of 0 disables region hilighting, an argument of 1 enables region hilighting between the two positions. If it is set to 3 then region hilighting will be enabled and a defined region (created using copy−region(2)or yank(2)) will continue to be hilighted until the region is changed.

A defined region can be redisplayed (if still valid) using the command show−region(2). The color of the region hilighting is defined by add−color−scheme(2) and is determined by $buffer−scheme(5), $global−scheme(5) or $buffer−hilight(5).

SEE ALSO

show−region(2), $buffer−hilight(5), $buffer−scheme(5), $global−scheme(5), $buffer−scheme(5), add−color−scheme(2), set−mark(2).

$status(5)

NAME

$status − Macro command execution status

SYNOPSIS

$status *boolean*

boolean TRUE (1) | FALSE (0)

DESCRIPTION

$status contains the return status of the last command executed (TRUE or FALSE). **$status** is generally used with the !force directives in macros.

NOTES

This variable can not be set, any attempt to set it will result in an error.

EXAMPLE

The following example shows how the variable is used within a macro construct, it converts all tab characters to their SPACE equivalent.

```
;
; tabs-to-spaces.
; Convert all of the tabs to spaces.
define-macro tabs-to-spaces
    ; Remember line
    set-variable #l0 $window-line
    beginning-of-buffer
    !force search-forward "\t"
    !while $status
        set-variable #l1 $window-acol
        backward-delete-char
        &sub #l1 $window-acol insert-space
        !force search-forward "\t"
    !done
    goto-line #l0
    screen-update
    ml-write "[Converted tabs]"
!emacro
```

In this case **$status** monitors the search−forward command which is searching for a tab character. The command returns a status value of TRUE if a tab is found, otherwise FALSE.

The **!force** statement prevents the macro from terminating when a FALSE condition is detected, if omitted the macro would terminate with an error as soon as the FALSE status is encountered. The definition of tabs−to−spaces(3) can be found in format.emf.

SEE ALSO

execute−file(2), !force(4), $result(5), tabs−to−spaces(3).

$system(5)

NAME

$system − System configuration variable

SYNOPSIS

$system *bitmask*; Default is system dependent

DESCRIPTION

The **$system** is used to define and configure the MicroEmacs environment, it is a bit based flag where:−

0x001

This bit is set if MicroEmacs is running in Console mode. On UNIX systems the default is to use X whenever possible, in which case this bit will be clear. If X is not used then a TERMCAP base interface is used instead and this bit will be set (see notes below on how to set which interface to use). On all other systems this bit will be clear.

0x002

If this bit is set then the current system supports definable RGB colors allowing any color to be created and used in a color scheme. This bit cannot be set, typically Windows and UNIX X−Windows systems support this.

0x004

If this bit is set then the current system supports ANSI colors (8 colors, black, red, green, yellow, blue, magenta, cyan & white), bits 0x002 and 0x004 are mutually exclusive. On UNIX systems if the TERMCAP interface is being used then this bit can be changed to (de)select the used of color. Many unix terminals do not support color so this should be set appropriately. On all other systems this bit cannot be changed and MS−DOS is currently the only other system to use ANSI colors.

0x008

If this bit is set then the current system supports Extended ANSI colors, brighter versions of the 8 ANSI colors doubling the number of colors available to 16. On UNIX systems if the TERMCAP interface is being used then this bit can be changed to (de)select the used of bold with color to create this extended color set for foreground colors. But many UNIX terminals do not support this use of color with the bold font so this should be set appropriately. On all other systems this bit cannot be changed and MS−DOS is currently the only other system to support this.

0x010

If this bit is set then the current system supports the use of fonts (bold, italic, light and underline). Whether these fonts can be successfully utilized depends upon the platform and the system font being used, for UNIX TERMCAP systems it will also depend on the terminal being used. This option is not supported on MS_DOS.

0x080

This bit is set if the current system is a UNIX based system such as LINUX or HPUX. This bit cannot be altered, its use is within macros.

0x100

This bit is set if the current system is a Microsoft based system such as DOS or Windows '95. This bit cannot be altered, its use is within macros.

0x200

If this bit is set then the current system uses the concept of drives (i.e. `c:/` on DOS systems). This bit cannot be altered, its use is within macros.

0x400

If this bit is set then a DOS style `8.3` file naming system should be used (i.e. "`BBBBBBBB.XXX`"), otherwise an unlimited file name length is used. This effects the backup and auto−save file names generated by MicroEmacs, the bit can be altered on systems that support unlimited file name length.

0x800

If this bit is set then the current system supports and uses ipipe−shell−command(2) when required. For systems such as DOS which cannot support ipipes, this bit will be clear and cannot be altered. For systems which do support ipipes, this bit can be cleared to disable their use.

0x1000

If this bit set, the then execution of the tab(2) command (bound to `tab`) always checks and adjusts the indentation of the current line when the current buffer is in cmode(2m) or has an indentation method. If the bit is clear then the `tab` may only checks the indentation when the cursor is in column zero depending on the setting of bit **0x200000**.

0x2000

If this bit is set the main menu Alt hot−key bindings are enabled. These are dynamic bindings automatically generated from the main menu. Typically the first item in the main menu is "`File`" with a hot key of '**F**', with this bit set '`A−f`' will open this menu item. Note that global and local key bindings override these. Also see bit **0x4000**.

0x4000

If this bit is set the Alt key acts as a prefix 1 modifier key. By default '`A-n`' is not bound, with this bit set the key is inferred to '`esc n`' which is bound to **forward−paragraph**. Note that global, local and menu hot−key bindings override these. Also see bit 0x2000.

0x8000

If this bit is set the undo history is kept after a save allowing the undo(2) command to back−up changes beyond the last save. When clear the undo history is discarded after the buffer is saved.

0x10000

Enable box character rendering fix, supported on Win32 and XTerm interfaces only. Windows ANSI fonts and many XTerm ISO−8859−1 fonts do not have well formed box characters which are used by osd(2) and other commands to create a better looking interface. When this bit is enabled MicroEmacs traps the printing of characters with an ASCII value of less than 32 and renders them directly. Following is a table of supported characters, other characters in the range of `0x00` to `0x1f` not listed are rendered as a space:

`0x08`	Special Character; Backspace
`0x09`	Special Character; Tab
`0x0b`	Box Character; Bottom right
`0x0c`	Box Character; Top right
`0x0d`	Box Character; Top left
`0x0e`	Box Character; Bottom left
`0x0f`	Box Character; Center cross
`0x10`	Arrows; Right

`0x11`

Arrows; Left

`0x12`

Box Character; Horizontal line

`0x15`

Box Character; Left Tee

`0x16`

Box Character; Right Tee

`0x17`

Box Character; Bottom Tee

`0x18`

Box Character; Top Tee

`0x19`

Box Character; Vertical Line

`0x1e`

Arrows; Up

`0x1f`

Arrows; Down

0x20000

Enables the client server, default is disabled (UNIX and Win32 NT or Win95+ platforms only). When enabled a hidden "`*server*`" buffer is created which monitors commands written to the server, the socket "`/tmp/mesrv`**uid**" on UNIX systems and the command input file "**$TEMP**`/me`**$MENAME**`.cmd`" on Win32 systems. Commands can be written out using the command ipipe−write(2) while in the "*server*" buffer, the command is written to the same socket on UNIX systems and to the response file and response file "**$TEMP**`/me`**$MENAME**`.rsp`" on Win32 systems. This functionality is used by the **−m** and **−o** command−line options and by the MicroSoft DevStudio interface.

0x40000

Enables the capture of the Alt space key ("`A−space`"), default is enabled (Win32 platform only). In the Windows environment the Alt Space key is used to activate the main window's pull down menu at the top left. if this bit is set MicroEmacs captures this key and executes it as normal, thereby disabling this standard windows binding.

0x80000

Enables the drawing of visible white spaces, i.e. space, tab and new−line characters. When disabled (default) white spaces are drawn using spaces (' ') which means the user cannot distinguish between a tab and spaces or determine the last character of the line by merely looking at the display. When enabled MicroEmacs uses visible characters to draw the white spaces, the characters used are set with the variable $window−chars(5).

0x100000

Enables hiding MicroEmacs generated backup files. On Windows and Dos platforms the Hidden file attribute is used to hide the file, whereas on UNIX the backup file name is prepended with a '`.`'.

0x200000

If this bit set, the then execution of the tab(2) command (bound to `tab`) checks and adjusts the indentation of the current line when the cursor is in column zero and current buffer is in cmode(2m) or has an indentation method. The setting of this bit has no effect if bit **0x1000** is set. If this and bit **0x1000** are clear then the `tab` will not check the indentation.

0x400000

When this bit is set the external clipboard (Windows & XTerm platforms) will never be set to empty, if the current yank buffer is the empty string the cut buffer will be set to a space (i.e. " "). This feature has been added to avoid problems with other software (e.g. **exceed(1)** which can crash if given an empty cut buffer).

0x800000

When this bit is set all use of the external clipboard (Windows & XTerm platforms) is disabled, this means that MicroEmacs will not attempt to retrieve or set the content of the system clipboard. **EXAMPLE**

The follow example works out the current buffer's backup file name using **$system** to determine the naming system being used by MicroEmacs:−

```
set−variable #l0 &stat "a" $buffer−fname
; Is an 8.3 dos style naming system being used?
!if &band $system 0x400
    !if &not &set #l1 &sin "." #l0
        set−variable #l1 &cat #l0 ".~~~"
    !elif &gre &set #l1 &sub &len #l0 #l1 2
        set−variable #l1 &cat &lef #l0 &sub &len #l0 1 "~"
    !else
        set−variable #l1 &spr "%s%n" #l0 &sub 3 #l1 "~"
    !endif
```

```
        !elif $kept-versions
            set-variable #l1 &cat #l0 ".~0~"
        !else
            set-variable #l1 &cat #l0 "~"
        !endif
```

The following macro can be used to toggle the visible drawing of white spaces:

```
        define-macro toggle-visible-white-spaces
            set-variable $system &bxor $system 0x80000
            screen-update
        !emacro
```

NOTES

Most of the **$system** functionality can be set using the $user−setup(3) dialog.

UNIX X verses Termcap

By default, on X supporting systems MicroEmacs creates a new X window. This feature may be disabled in one of two ways:

- The environment variable `$TERM` is set to "`vt...`", in this case it is assumed that the machine is a server, and the host cannot support X.
- The `-n` option is used on the command line (see me(1)) to disable the windowing interface.

If X is disabled then the **termcap** interface is used instead, still allowing the use of colors through the ANSI standard, or the use of fonts (see bits **0x004** and **0x008**).

X provides the following features over and above a **termcap** based version of MicroEmacs '02:

- R,G,B style color creator giving access to up to 256 different colors for the ultimate hilighting schemes (see bit **0x002** and add−color(2)).
- Full mouse support, allowing user definable bindings to every mouse event (see global−bind−key(2)).
- Copy from and pasting to X's selection buffer (see yank(2)).

SEE ALSO

user−setup(3), $mouse(5), $platform(5), add−color(2), add−color−scheme(2), ipipe−shell−command(2), $global−scheme(5).

$tabsize(5)

NAME

$tabsize − Tab character width

SYNOPSIS

$tabsize *integer*; Default is 4

−0 < *integer* <= *n*

DESCRIPTION

$tabsize defines the width of a tab character.

Setting tabs to arbitrary widths is possible in MicroEmacs '02 but you must be aware of a subtle difference that it makes to your file and hence to your editing. When you start MicroEmacs '02, the tab width is set to the default (usually every 8th column) for the tab character (`CTRL−I`). As long as you stay with the default, every time you insert the tab character, a CTRL−I get inserted. Hence, you logically have a single character which might appear to be several spaces on the screen (or the output) depending upon the column location of the tab character. This means that to remove the spacing you have to delete a *single* character −− the tab character.

On the other hand, the moment you explicitly set the tab interval (even if it is to the default value), MicroEmacs '02 handles the tab character by expanding the character into the required number of spaces to move you to the appropriate column. In this case, to remove the spacing you have to delete the appropriate number of spaces inserted by M−e to get you to the right column.

The operating mode of the tab expansion is controlled by the tab(2m)mode.

SEE ALSO

buffer−mode(2) tab(2m), $tabwidth(5).

$tabwidth(5)

NAME

$tabwidth − Tab character interval

SYNOPSIS

$tabwidth `integer`; Default is `8`

−0 < *integer* <= *n*

DESCRIPTION

$tabwidth defines the interval of a tab character.

The tab interval is set to the given numeric argument. As always, the numeric argument precedes the command. Hence to get tabs every 4 spaces you would set the **$tabwidth** to 4.

SEE ALSO

buffer−mode(2) tab(2m), $tabsize(5), tabs−to−spaces(3).

$temp−name(5)

NAME

$temp−name − Temporary file name

SYNOPSIS

$temp−name *FileName*

DESCRIPTION

$temp−names is automatically set to a nonexistent file name in the systems temporary file directory. On UNIX systems the temporary directory is fixed to "`/tmp/`", on other systems the temporary directory is set by the **$TEMP** environment variable.

EXAMPLE

The following example uuencodes a given file into a temporary file and then inserts this file into the current buffer.

```
set-variable #l0 @ml04 "Uuencode and insert file"
set-variable #l1 $temp-name
!force shell-command &spr "uuencode %s < %s > %s" #l0 #l0 #l1
insert-file #l1
!force shell-command &cat "rm " #l1
```

NOTES

This variable can not be set, any attempt to set it will result in an error.

The returned file name is not guaranteed to be unique between calls, only that the file does not currently exist.

SEE ALSO

shell−command(2), file−op(2).

$time(5)

NAME

$time − The current system time

SYNOPSIS

$time "*string*"

DESCRIPTION

$time is a constantly changing variable which is set to the current system time. The format of **$time** is "`YYYYCCCMMDDWhhmmssSSS`", where:−

YYYY

The current year (full 4 digits so should be millennium bug free).

CCC

Day of the year (0−366).

MM

The month of the year (1−12).

DD

The day of the month (1−31).

W

The day of the week (0−6 Sunday=0).

hh

The hour (0−23).

mm

The minute (0−59).

ss

The second (0−59).

SSS

The millisecond (0−999).

$time can be set to an integer value which is a time offset in seconds, for example if the following was executed;−

```
set-variable $time "3600"
ml-write &cat "$time is " $time
set-variable $time "0"
```

The written time would one hour ahead of the system time.

EXAMPLE

The following macro times the time taken to execute a user command:−

```
define-macro time
    !force set-variable #l2 @1
    !if &not $status
        set-variable #l2 @ml00 "Time command"
    !endif
    set-variable #l0 $time
    !force execute-line #l2
    set-variable #l1 $time
    set-variable #l2 &add &mid #l0 16 2 &mul 60 &add &mid #l0 14 2 &mul 60 &mid #l
    set-variable #l3 &add &mid #l1 16 2 &mul 60 &add &mid #l1 14 2 &mul 60 &mid #l
    !if &les &set #l4 &sub &rig #l1 18 &rig #l0 18 0
        set-variable #l2 &add #l2 1
        set-variable #l4 &add 1000 #l4
    !endif
    ml-write &spr "Command took %d sec %d msec" &sub #l3 #l2 #l4
!emacro
```

time(3) is a macro defined in misc.emf.

organizer(3) uses **$time** to work out the current month.

SEE ALSO

time(3), organizer(3).

$timestamp(5)

NAME

$timestamp − Time stamp string

SYNOPSIS

$timestamp "*string*"; Default is "`<%Y%M%D.%h%m>`"

DESCRIPTION

$timestamp defines the file time−stamping string. MicroEmacs '02 searches for, and modifies, the string to the current time and date whenever the file is saved (written to disk) and time(2m) mode is enabled.

Time stamp string is defined, by default, as "`<%Y%M%D.%h%m>`". The first occurrence of the string in the file is up−dated with the time and date information when the buffer is written. The **$timestamp** string may contain any text, and includes the following, magic characters escaped by a percentage (`` `%' ``) character:−

`D` − Day.
`M` − Month.
`Y` − Year.
`h` − Hour.
`m` − Minute.
`s` − Second.

The format string may be redefined into any format. The '`%`' character has to be delimited by another '`%`' if it is to be used in the text (i.e. "`%%`").

The year component (`%Y`) may be a 2 or 4 digit string, depending whether it includes the century. When the time stamping searches for the `%Y` component it searches for either variant and replaces appropriately.

EXAMPLE

The startup file may define the time stamp required as follows:−

```
set-variable $timestamp "Last Modified : %Y/%M/%D %h:%m:%s"
```

Time stamping is performed on the string :−

```
Last Modified : 90/11/23 10:12:01
```

Where the time stamp is modified according to the file (buffer) type then the time stamp string may be modified within the buffer hooks. This allows different files to utilize different time stamping strings. The following example shows how the entry and exit buffer hooks are defined to modify the string:

```
0 define−macro bhook−nroff
    set−variable .timestamp $timestamp
    ; Buffer specific time stamp string.
    set−variable $timestamp "[%Y/%M/%D %h:%m:%s]"
!emacro
0 define−macro ehook−nroff
    ; Restore the existing time stamp.
    set−variable $timestamp .bhook−nroff.timestamp
!emacro
```

On entry to the buffer (buffer becomes current) the buffer hook **bhook−nroff** is executed which stores the current setting and then modifies the time stamp string. On exit from the buffer the buffer hook **ehook−nroff** is executed restoring the time stamp string.

SEE ALSO

buffer−mode(2) time(2m).

$trunc−scheme(5)

NAME

$trunc−scheme − Truncation color scheme.

SYNOPSIS

$trunc−scheme *schemeNum*; Default is `0`

DESCRIPTION

$trunc−scheme sets the color scheme used when drawing a line truncation indicator. The left truncation character (usually a '`$`' char) drawn at the start of the line indicates that the line has been scrolled to the right and therefore the start of the line has been truncated. A right truncation char (also usually a '`$`') drawn at the end of the line indicates the remainder of the line is too long to fit onto the width of the window so the end has been truncated and the indicator drawn.

The *schemeNum* selected must be a color scheme defined with add−color−scheme(2), which identifies the foreground and background color schemes. A hilight scheme can define its own truncation color scheme, see hilight(2) for more information.

NOTES

The truncation characters used are set by the $window−chars(5) variable.

SEE ALSO

$buffer−scheme(5), $global−scheme(5), add−color−scheme(2), hilight(2), $window−chars(5).

$variable−names(5)

NAME

$variable−names − Filtered variable name list

SYNOPSIS

$variable−names *VariableName*

DESCRIPTION

$variable−names must first be initialized to the required filter string, if the variable is evaluated before it is initialized the value will be set to "*ABORT*" and the command will fail.

The filter string can contain wild−card characters compatible with most file systems, namely:−

?

Match any character.

[abc]

Match character only if it is *a*, *b* or *c*.

[a−d]

Match character only if it is *a*, *b*, *c* or *d*.

[^abc]

Match character only if it is not *a*, *b* or *c*.

*****

Match any number of characters.

Note that these are not the same characters used by exact(2m) mode.

Once initialized, evaluating **$variable−names** returns the name of the next variable which matches the filter until no more variables are found, in which case an empty string is returned.

EXAMPLE

The following example prints out the name of all variables to the massage line one at a time. Note that &set(4) is used on the !while(4) statement to avoid evaluating **$variable−names** twice per loop.

```
set-variable $variable-names "*"
!while &not &seq &set #l0 $variable-names ""
    100 ml-write &cat "variable: " #l0
!done
```

NOTES

The list of variables is evaluated when the variable is initialized, variables defined after the initialization will not be included in the list. The list can contain the current buffer's buffer variables (See Variables(4) for more information on the different types of variables).

Using unset−variable(2) to delete a variable which are in the list, before it has be evaluated, will have undefined effects.

SEE ALSO

list−variables(2), $command−names(5).

$version(5)

NAME

$version − MicroEmacs version date−code

SYNOPSIS

$version "*YYYYMMDD*"

DESCRIPTION

$version is a system variable which is defined as the MicroEmacs build date code. This value is fixed at compile time and cannot be changed. The variable may be used in macros to identify incompatibility issues.

EXAMPLE

Given a macro that only operates with a MicroEmacs executable built on or after 1st August 2001 then this macro should check that `$version` is not less than `20010801`. The check may be performed as follows:

```
!if &les $version "20010801"
    ml−write "[Error: MicroEmacs executable is incompatible]"
    !abort
!endif
```

NOTES

This variable was introduced in 2001−08−01, evaluating this variable on an earlier version of MicroEmacs would return the string "`ERROR`" unless an environment variable $version has been defined. "`ERROR`" evaluates to `0` hence the test still operates correctly.

This variable is used in the macro file `me.emf` to check for any macro − executable incompatibility issues.

$window−col(5)

NAME

$window−col − Window cursor column (no expansion)
$window−line − Window cursor line (with narrows)
$window−acol − Window cursor actual column (expansion)
$window−aline − Window cursor actual line (ignore narrows)

SYNOPSIS

$window−col *integer*

0 <= *integer* <= 65535

$window−line *integer*

1 <= *integer* <= n

$window−acol *integer*

0 <= *integer* <= n

$window−aline *integer*

1 <= *integer* <= n

DESCRIPTION

$window−col is defined as the current position of the cursor in the current line in the current window. Column zero is the left hand edge. This differs from **$window−acol** in that tab and special characters only count for 1 character. **$window−col** is valid in the range 0 − *n*.

$window−line is defined as the current buffer line number the cursor is on in the current window. Line numbering starts from 1. **$window−line** is valid in the range 1 − *n*.

$window−aline is identical to **$window−line** except when the current buffer contains narrowed out sections before the current line. In this case **$window−line** will be set to the line number without counting the number of lines in the narrow, whereas **$window−aline** will return the current line number including all lines narrowed out before it. When this variable is set, the line required may lie in a narrowed out section in which case the narrow is automatically removed. See narrow−buffer(2) for more information on narrowing.

$window−acol is defined as the current column of the cursor in the current window. Column zero is the left hand edge. This differs from **$window−col** in that tab and special characters may not count for 1 character.

NOTES

Variable **$window−wcol** was renamed to **$window−acol** in June 2000. Variable **$window−wline** was also removed and a new variable **$window−y−scroll** introduced at this time. The following macro code can be used to calculate the value of the original **$window−wline** variable:

```
&sub &sub $window−line $window−y−scroll 1
```

SEE ALSO

$frame−depth(5), $window−depth(5), $window−width(5), $window−y−scroll(5), narrow−buffer(2).

$window−chars(5)

NAME

$window−chars − Character set used to render the windows

SYNOPSIS

$window−chars "*sting*"; Default is
`"=−#*%=^|#|v*==^^||##||vv**|<−#−>*||<<−−##−−>>** x*[]>\.$$\"`

DESCRIPTION

$window−chars is a fixed length string that defines the set of characters used to render the windows. The characters have fixed indices defined as follows:−

Index 0

The active window mode line separator character, This replaces all *Index 1* characters when the window is current. Default is '='.

Index 1

The inactive window mode line separator character. This character is replaced by *Index 0* characters when the window becomes current. Default is '**−**'.

Index 2

UNIX based platforms only. The **root** or **superuser** indicator character that appears on the mode line. Default is '**#**'.

Index 3

The buffer changed indicator character that appears on the mode line. Default is '*****'.

Index 4

The buffer in view(2m) mode indicator character that appears in the mode line. Default is '**%**'.

Index 5

Single column vertical scroll bar split window horizontally character. Default is '='.

Index 6

Single column vertical scroll bar up−arrow character. Default is '**^**'.

Index 7

Single column vertical scroll bar upper−shaft character. Default is '**|**'.

Index 8

Single column vertical scroll box character. Default is '**#**'.

Index 9

Single column vertical scroll bar lower−shaft character. Default is '**|**'.

Index 10

Single column vertical scroll bar down−arrow character. Default is '**v**'.

Index 11

Single column vertical scroll bar corner character. Default is '*****'.

Index 12−13

Double column vertical scroll bar split window horizontally character. Default is '**==**'.

Index 14−15

Double column vertical scroll bar up−arrow characters. Default is "**^**".

Index 16−17

Double column vertical scroll bar upper−shaft characters. Default is "**||**".

Index 18−19

Double column vertical scroll box characters. Default is "**##**".

Index 20−21

Double column vertical scroll bar lower−shaft characters. Default is "**||**".

Index 22−23

Double column vertical scroll bar down−arrow characters. Default is "**vv**".

Index 24−25

Double column vertical scroll bar corner characters. Default is "******".

Index 26−32

Single column horizontal scroll bar. Default is "**|<−#−>***".

Index 33−46

Double column horizontal scroll bar. Default is "**||<<−−##−−>>****".

Index 47

Osd title bar blank character. Default is ' '.

Index 48

Osd title bar right corner kill character. Default is '**x**'.

Index 49

Osd dialog bottom right corner resize character. Default is '*****'.

Index 50

Osd open button character. Default is ' '.

Index 51

Osd close button character. Default is ' '.

Index 52

Displayed tab character (used when $system(5) bit 0x80000 is set). Default is '>'.

Index 53

Displayed new−line character (used when $system(5) bit 0x80000 is set). Default is '\'.

Index 54

Displayed space character (used when $system(5) bit 0x80000 is set). Default is '**.**'.

Index 55

Displayed truncated text to left character (used when the current line is scrolled to the right). Default is '**$**'.

Index 56

Displayed truncated text to right character (used when the current line is longer than the window width). Default is '**$**'.

Index 57

Inserted end of wrapped line character in an ipipe−shell−command(2) buffer. Default is '\'. **EXAMPLE**

The **$window−chars** is typically platform dependent, it's setting is determined by the characters available in character set of the hosting platform. MS−DOS and Microsoft Windows use an OEM font might use the following value:

```
"=-#*%=\C^\xB1 \xB1\C_\CD==\C^\C^\xB1\xB1  \xB1\xB1\C_\C_\C[
\CZ|\CQ\xB1 \xB1\CP\CD||\CQ\CQ\xB1\xB1  \xB1\xB1\CP\CP\C[
\CZ x*  >\\.$$\\"
```

This utilizes character−set specific characters to render some of the window components.

NOTES

- $scroll−bar(5) allows the scroll box to be rendered in reverse video allowing a space to be used for the scroll box.
- Use symbol(3) to determine the displayable characters on the host platform.
- The use of MicroEmacs's extended character set on Windows and XTerm platforms can greatly improve the look and usability of MicroEmacs, see the Extend Char Set option in the Platform page of user−setup(3) and bit `0x10000` of variable $system(5).

SEE ALSO

split−window−horizontally(2), symbol(3), $box−chars(5), $global−scheme(5), $mode−line(5), $mode−line−scheme(5), $scroll−bar(5), $scroll−bar−scheme(5), $system(5).

$window−depth(5)

NAME

$window−depth − Number of text lines in a window
$window−width − Number of character columns in a window

SYNOPSIS

$window−depth *integer*

1 <= *integer* <= $frame−depth

$window−width *integer*

0 <= *integer* <= $frame−width − 1

DESCRIPTION

$window−depth returns the depth (height) of the current window, excluding the mode line, specified in text lines. (i.e. the number of lines of text in the window). The returned value is an integer in the range:

0 − ($frame−depth − **3**)

$window−width returns the width, in characters, of the current window. The returned value is an integer in the range:

0 − $frame−width.

NOTES

These variables can not be set, any attempt to set them results in an error.

SEE ALSO

$frame−depth(5), $frame−width(5), $window−scroll−bar(5), $window−mode−line(5),

$window−flags(5)

NAME

$window−flags − Current window setup flags

SYNOPSIS

$window−flags *bitmask*; Default is `0`

DESCRIPTION

The **$window−flags** variable is used to set or get various behavioural characteristic settings of the current window, it is a bit based flag where:

0x001

If set the width of the window is locked, calls to resize−all−windows(2) will maintained the width of this window whenever possible.

0x002

If set the depth of the window is locked, calls to resize−all−windows(2) will maintained the depth of this window whenever possible.

0x004

If set the buffer being displayed by the window is locked, the user can still manually change the buffer being displayed (by using commands like find−buffer(2)) but commands that pop−up buffers (such as help(2) or find−tag(2)) will not use this window.

0x008

When set the command compare−windows(2) will ignore this window.

0x010

When set the commands like previous−window(2) and next−window(2) will skip this window unless the numeric argument given to the command is used to override the flag setting.

0x020

When set the command delete−other−windows(2) will not delete this window unless the numeric argument given to the command is used to override the flag setting.

0x040

When set the command delete−window(2) will not delete this window unless the numeric argument given to the command is used to override the flag setting.

0x080

When set the window cannot be split using either the split−window−horizontally(2) or split−window−vertically(2) commands.

0x100

If not set the window cannot be deleted if it is the only window without this bit set. This more esoteric feature is utilized by the toolbar, all toolbar windows have this bit set which means that the main user window cannot be delete. **NOTES**

The $window−flags setting is not preserved during a window splitting operation (i.e. using a command like split−window−vertically(2)) as the persistence of these settings can lead to unexpected behaviour.

The toolbar uses bit 0x1000 to indicate that the window is displaying a toolbar tool, this bit should not be used by users and its value should be maintained.

SEE ALSO

next−window(2), delete−other−windows(2), compare−windows(2).

$window−mode−line(5)

NAME

$window−mode−line − Window mode line position
$window−scroll−bar − Window scroll bar (or separator) position

SYNOPSIS

$window−mode−line *integer*

1 <= *integer* <= $frame−depth − 2

$window−scroll−bar *integer*

0 <= *integer* <= $frame−width − 1

DESCRIPTION

$window−mode−line stores the screen line of the current windows mode−line, where screen lines are counted from 0 at the top of the screen. Often used in conjunction with set−cursor−to−mouse(2) and $mouse−y(5) to add more complex mouse functionality.

$window−scroll−bar stores the screen position of the right−hand horizontal window separator line or scroll−bar (see split−window−horizontally(2) and $scroll−bar(5)). A value of greater than $frame−width(5) indicates that there is no right−hand separator column or scroll bar present. Often used in conjunction with $mouse−x(5).

EXAMPLE

In the following example the position of the mouse is checked to see if it is on the mode line of the window, if so then a different action is taken.

```
set-cursor-to-mouse
;   If we are on the mode line then interpret position of
;   the cursor on line to control the screen.
!if &equal $window-mode-line $mouse-y
    !if &less $mouse-x "2"
        menu-main     ; Inform buffer to pop up menu.
    !elif &equal $mouse-x "2"
        delete-window
    !elif &equal $mouse-x "3"
        delete-other-windows
    !elif &equal $mouse-x "4"
        backward-page
    !elif &equal $mouse-x "5"
        forward-page
    !elif &equal $mouse-x "6"
        recenter
    !elif &equal $mouse-x "7"
        undo
    !endif
!else
    .....
!endif
```

SEE ALSO

$mode−line(5), $mouse−x(5), $mouse−y(5), $scroll−bar(5), $mouse−pos(5), set−cursor−to−mouse(2), split−window−horizontally(2).

$window−x−scroll(5)

NAME

$window−x−sroll − Current window X scroll
$window−xcl−sroll − Current window current line X scroll
$window−y−sroll − Current window Y scroll

SYNOPSIS

$window−x−sroll *integer*
$window−xcl−sroll *integer*

0 <= *integer* <= 65535

$window−y−sroll *integer*

0 <= *integer* <= n

DESCRIPTION

$window−x−sroll defines the horizontal scroll position in the current window for all lines except the current line, **$window−xcl−sroll** defines the scroll position for the current line. The variables set how many characters are scrolled off the left hand edge of the current window, the variables are indirectly set by commands such as scroll−left(2), forward−char(2) etc.

$window−y−sroll defines the vertical scroll position in the current window. It sets the number of lines are scroll up off the top of the current window, it is indirectly set by commands such as scroll−up(2), forward−line(2) etc.

EXAMPLE

The following example first stores the current window's buffer position and the window layout. The middle `'...'` section could be replaced with macro code performing any number of operations before the last section which restores the initial position:

```
set-variable #l0 $window-line
set-variable #l1 $window-col
set-variable #l2 $window-xcl-scroll
set-variable #l3 $window-x-scroll
set-variable #l4 $window-y-scroll
    .
    .
    .
set-variable $window-line #l0
set-variable $window-col #l1
```

```
set-variable $window-xcl-scroll #l2
set-variable $window-x-scroll #l3
set-variable $window-y-scroll #l4
```

NOTES

If these variables are set by the user or a macro the value is validated against the $scroll(5) method and the current cursor position which may lead to the variable being reset if found to be invalid. For example, if the current line is 10 when the **$window−y−scroll** is set to 20 the variable will be reset to 0 as a value of 20 will mean the current line is not displayed in the current window.

SEE ALSO

scroll−left(2), scroll−up(2), $scroll(5), $window−line(5), $window−col(5), $window−acol(5).

etfinsrt(3)

NAME

etfinsrt − Insert template file into current buffer

SYNOPSIS

etfinsrt "*template*"

DESCRIPTION

etfinsrt is generally called by file hooks when the new buffer has been created as opposed to loaded from a file (see $buffer−fhook(5)).

etfinsrt uses &find(4) to locate and insert the required "*template*.etf" file. If successful, **etfinsrt** then replaces the following strings in the template:

`$ASCII_TIME$`

To the current time. Inserts the output of ascii−time(3).

`$BUFFER_NAME$`

To the buffer name. The name is capitalized, '.'s are replaced with '_' and any trailing "<##>" digits (used to make the buffer name unique) are removed.

`$COMPANY_NAME$`

To the value of **%company−name**, or if not defined to the value used for `$USER_NAME$`. **%company−name** is usually set up in the company setup file defined in User setup.

`$USER_NAME$`

To the value of the registry entry "`/history/user-name`", or if not defined to the value "`<unknown>`". The user name is usually set up in the User setup dialog.

`$YEAR$`

To the current year (4 digit number).

`$CURSOR$`

To leave the cursor at this point, only one of these tokens should be used in the template and the token is removed. **EXAMPLE**

The following is taken from hkmake.emf and inserts the "*makefile.etf*" template if the buffer has been created.

```
define-macro fhook-make
    ; if arg is 0 this is a new file so add template
    !if &not @#
        etfinsrt "makefile"
    !endif
    set-variable $buffer-hilight .hilight.make
    -1 buffer-mode "tab"                ; Normal tabs please !!!
    1 buffer-mode "indent"
    1 buffer-mode "time"
!emacro
```

NOTES

etfinsrt is a macro defined in `etfinsrt.emf`.

magic(2m) mode is always used to perform the the search/replace so the replace strings should be appropriate for **magic**.

SEE ALSO

$buffer−fhook(5), &find(4), ascii−time(3).

%compile−com(5)

NAME

%compile−com − Default system compile command line

SYNOPSIS

%compile−com "*string*"; Default is "`make`"

DESCRIPTION

Sets the default command−line inserted into the message line when the compile(3) command is executed. **%compile−com** does not need to be defined to run the **compile** command.

SEE ALSO

compile(3), %grep−com(5).

cygnus(3)

NAME

cygnus − Open a Cygwin BASH window
%cygnus−bin−path − Cygwin BASH directory
%cygnus−hilight − Cygwin shell hilight enable flag
%cygnus−prompt − Cygwin shell prompt

PLATFORM

Windows '95/'98/NT − win32 ONLY

SYNOPSIS

cygnus

%cygnus−bin−path "*path*"
%cygnus−hilight [`0`|`1`]
%cygnus−prompt "*hilightString*"

DESCRIPTION

cygnus creates an interactive BASH shell window within a MicroEmacs buffer window, providing a UNIX command line facility within the Microsoft Windows environment. This is a preferable environment to the MS−DOS shell and is certainly far more comfortable for those people familiar with UNIX.

Within the window BASH commands may be entered and executed, the results are shown in the window. Within the context of the BASH shell window then directory naming conforms to the **cygwin** standard conventions (as opposed to the Microsoft directory naming).

On running **cygnus** a new buffer is created called `*cygnus*` which contains the shell. Executing the command again creates a new shell window called `*cygnus1*`, and so on. If a cygwin window is killed off then the available window is used next time the command is run.

Additional controls are available within the shell window to control the editors interaction with the window. The operating mode is shown as a digit on the buffer mode line, this should typically show "3", which corresponds to *F3*. The operating modes are mapped to keys as follows:−

F2

Locks the window and allows local editing to be performed. All commands entered into the window are interpreted by the editors. **F2** mode is typically entered to cut and paste from the window, search

for text strings etc. In mode 2, a **2** is shown on the mode line.

F3

The normal operating mode, text typed into the window is presented to the shell window. Translation of MicroEmacs commands (i.e. beginning−of−word) are translated and passed to the shell. For interactive use this is the default mode. In mode 3, a **3** is shown on the mode line.

F4

All input is passed to the shell, no MicroEmacs commands are interpreted and keys are passed straight to the shell window. This mode is used where none of the keys to be entered are to be interpreted by MicroEmacs. Note that you have to un−toggle the F4 mode before you can swap buffers as this effectively locks the editor into the window.

F5

Clears the buffer contents. This simply erases all of the historical information in the buffer. The operation of the shell is unaffected.

To exit the shell then end the shell session using "`exit`" or "`C−d`" as normal and then close the buffer. A short cut "`C−c C−k`" is available to kill off the pipe. However, it is not recommended that this method is used as it effectively performs a hard kill of the buffer and attached process

%cygnus−bin−path is a user defined variable that defines the file system location of the *cygwin* directory. This variable MUST be defined within the user start up script in order for the **cygnus** command to start the shell. With a default installation of *cygwin* then the settings are typically defined as:−

Release B19

```
    set−variable %cygnus−bin−path "C:/Cygnus/B19/h−i386~1/bin"
```

Release B20

```
    set−variable %cygnus−bin−path "c:/cygnus/cygwin−b20/H−i586−cygwin32/bin"
```

%cygnus−hilight is a boolean flag which controls how the cygnus command shell window is hilighted. This value MUST be defined within the user start up script prior to executing cygnus if hilighting is to be enabled; by default hilighting is disabled. A value of 1 enables shell hilighting i.e.

```
    set−variable %cygnus−hilight 1
```

%cygnus−prompt is an optional variable that is used in conjunction with **%cygnus−hilight**, it defines the hilighting string identifying the prompt. This allows the prompt to be rendered with a different color. The default prompt is `bash−2.01$` and may be hilighted using a definition:−

```
    set−variable %cygnus−prompt "bash−2.01$"
```

The user typically overrides the prompt definition within the BASH startup file, a more appropriate definition of the prompt may be:−

```
        set-variable %cygnus-prompt "^[a-z]*@[^>]*>"
```

NOTES

The **cygnus** command uses the ipipe−shell−command(2) to manage the pipe between the editor and the **bash** shell. The window is controlled by the macro file `hkcygnus.emf` which controls the interaction with the shell.

The macro **cygnus** in `hkcygnus.emf` defines the parameter setup to connect to the cygwin bash shell (Version 19), installed in the default location `c:/cygnus`. If your installation of cygnus is in a different location then correct the macro to match your install location, preferably correct by creating a *mycygnus.emf* file in your user directory simply containing a re−defined **cygnus** macro.

If you have exported some of the cygwin environment variables in your `autoexec.bat` then you will have to figure out for yourself what variables macro *cygnus* needs to export − the current configuration is for a vanilla install.

The **bash** shell is executed with options *i*, for interactive shell and *m* to enable job control.

TESTED CONFIGURATIONS

This configuration has only been tested on a Windows '98 installation, whether this works on NT and Windows '95 (OEM SR2) is unknown.

We have only been running "make" operations in the shell and do not know how the likes of "more", "man" or anything other terminal interaction works.

Tested Configurations

Windows '98 (Pentium 120MHz/Pentium Pro 200MHz/Cyrix 300MHz/Pentium II 450MHz)

> cygwin version B19.3 − this is the original "cygwin" distribution + the latest "coolview.tar.gz" patch.
> cygwin version B20 − the latest cygwin distribution.

BUGS

Break Key

A break in a bash shell is `C−c`, the macros define the key `C−c C−c` to perform the break. This sequence is sent to the process but is not enacted by the shell. This is a property of the Bash shell rather than MicroEmacs.

Slow Response

If you are getting a very slow response from the bash shell then check the directory where *bash* was started. Sometimes there are problems if the shell is started in "`c:/`" (which is typically "`/`") then the *bash* shell is very unresponsive and tends to '*ignore me*' for periods of time. If it is started in another location, i.e. "*c:/temp*" directory, then this problem does not occur.

You can see the start−up location in the top of the buffer when the shell is started.

Prompt at top of buffer

Very, very occasionally the ishell sticks at the top of the buffer with only a couple of lines showing. A swap of the buffers or a quick window resize sorts out the problem. A fix for this problem has been applied but still may occasionally occur.

WinOldAp

Winoldap is created by the Microsoft environment whenever a BASH shell is created. On occasions where processes have terminated badly the user may be prompted to kill these off; this is the normal behaviour of windows. It is strongly advised that all of the BASH processes are killed from within the Bash shell itself and the shell is always exited correctly (i.e. `exit`) before leaving the editor. The Windows operating system for '95/'98 is not particularly resilient to erroneous processes (for those of us familiar with UNIX) and can bring the whole system down. I believe that NT does not suffer from these problems (much).

Locked Input

There are occasions after killing a process the editor appears to lock up. This is typically a case that the old application has not shut down correctly. Kill off the erroneous task (`Alt−Ctrl−Del` − *End Task*) then bring the editor under control using a few `C−g` abort−command(2) sequences. **SEE ALSO**

ipipe−shell−command(2), ishell(3).
Cygnus Win32 home sites **www.cygnus.com** and **www.cygnus.co.uk**

diff(3)

NAME

diff − Difference files or directories
diff−changes − Find the differences from a previous edit session
%diff−com − Diff command line

SYNOPSIS

diff "*oldFile*" "*newFile*"
diff−changes
`%diff-com` "*string*"; Default is "`diff`"

DESCRIPTION

diff executes the **diff(1)** command with the command line set by the %diff−com(5) variable and the user supplied *oldFile* and *newFile*. The output of the command is piped into the ***diff*** buffer and is hilighted to show the changes (GNU diff only).

Your version of **diff(1)** will determine whether it is possible to difference directories.

diff−changes is a simple macro that differences the current buffer and the last backup of the associated file. It is a quick way to determine what has been modified recently. This macro only works if a backup file exists.

%diff−com is the command line that is used to execute a **diff(1)** system command.

For GNU diff then the following command line setting is recommended:−

```
diff --context --minimal --ignore-space-change \
    --report-identical-files --recursive
```

which should be defined in your personal user configuration. This is the default for Linux.

NOTES

diff and **dif−changes** are macros defined in `tools.emf`.

diff(1) must be executable on the system before diff or diff−changes can function.

diff(1) is a standard utility on UNIX systems. For Windows 95/NT a version of GNU **diff** may be found at:

<ftp.winsite.com/ftp/pub/pc/winnt/misc/gnudiff.zip>

For MS−DOS users, a DJGPP port of **diff** is also available on the net. A commercial version of **diff** is also available from MKS.

SEE ALSO

compare−windows(2), compile(3), gdiff(3), grep(3), %grep−com(5).

%ftp−flags(5)

NAME

%ftp−flags − "Configure the FTP console"
%http−flags − "Configure the HTTP console"

SYNOPSIS

%ftp−flags "[c|s|p]" ; Default is undefined.
%http−flags "[c|s|p]" ; Default is undefined.

DESCRIPTION

The **%ftp−flags** and **%http−flags** modify the behavior of the editor during FTP and HTTP file transfers, respectively. (see ftp(3) and find−file(2)).

By default, the flags are disabled, the facilities outlined below are enabled by setting the variable in the user configuration. The flag values for both flags are defined as follows:−

c

Create a console buffer (`*ftp−console*` for ftp, `*http−console*` for http) into which the FTP/HTTP command interactions with the remote server are logged.

s

Show the console whenever a FTP/HTTP operation is performed. The console is popped into the display pane and shows the current interaction status.

p

Show the download progress within the console window ('#' for every 2Kb downloaded)

Typically the following flags are enabled in the *user*.emf file:−

```
set−variable %ftp−flags "csp"
set−variable %http−flags "csp"
```

Once familiar with this facility the console pop−up becomes inconvenient and the flags are typically reduced to:−

```
set−variable %ftp−flags "cp"
set−variable %http−flags "cp"
```

This disables the pop−up feature of the console. Enabling the limited flag set allows some post mortem debugging to be performed if anything goes wrong. The console buffers are manually selected when these flags are set.

NOTES

Note that ftp and http facilities are available on UNIX by default, but must be compiled in for Windows versions.

SEE ALSO

%http−proxy−addr(5), find−file(2), ftp(3).

gdiff(3)

NAME

gdiff − Graphical file difference
%gdiff−com − Gdiff diff(1) command line

SYNOPSIS

gdiff "*version1*" "*version2*"

`%gdiff−com` "*string*"; Default is "`diff −c −w`"

DESCRIPTION

gdiff is a macro utility that facilitates the merging of two files (typically with different modification revisions). The changes between the revisions are hilighted with color, allowing modification regions and lines to be selected for the generation of a newer revision file, which might encompass selected modifications from each of the base revisions.

gdiff executes the **diff(1)** command with the command line set by the %gdiff−com(5) variable and the user supplied *version1* and *version2*. The output is displayed in two buffer windows, side by side, and the differences in the lines are hilighted to show the changes. In addition the content of the two buffers is *normalized* such that both windows are aligned at the same line position, allowing the changes in the text to be viewed in both windows at the same time.

Whilst in **gdiff** view mode then both scroll bars (if visible) are *locked*, such that either scrolls BOTH windows at the same time. Other key commands are disabled, as are the menu interactions. The short cut keys are defined as follows:−

`esc h`/`A−h` − View the help page.

Invokes the display of a OSD help box, summarizing the interaction commands

`C−up` − Move to previous difference

Moves to the previous changed region above the current cursor position.

`C−down` − Move to next difference

Moves to the next changed region below the current cursor position.

`left mouse button`
`space`
`enter`

`r` − Select difference version

Selects the difference version of the currently selected window. The region is hilighted as the required region to be incorporated into the new revision.

`R` − Select neither version.

Marks both regions as not required.

`l` − Line select current version

Selects the current line from the region as being included, without including ALL of the region modifications.

`L` − Line select neither version

Discards lines from both revisions of the file.

`g` − Globally selects the current version.

Shortcut allows ALL modifications to the current side to be accepted. This is typically the fastest method to select all changes, minor region adjustment may then be performed on those regions which are inappropriately included by the selection.

`G` − Globally selects neither version.

Marks all regions as not being acceptable.

`C−x C−s` − Save current side

Saves the current window to the specified file, merging the selected changes between the two revisions. Note that the save only operates iff all hilighted changes have been selected.

`C−x C−w` − Save current side as

Same as **Save current side** except the user is prompted to enter a new filename to which the modifications are written.

`C−x k` − Exit graphical diff

Exits the **gdiff** utility. **Hilighting**

The hilighting within the windows is dependent upon the color scheme selected, in general the following hilights apply:−

normal text

No change

cyan/grey

Addition/removal of line(s)/region(s) between files.

yellow

Modification in line(s)/region(s).

green/red

Selected region, red or green is attributed to a selection for each window. **NOTES**

gdiff is a macro defined in `gdiff.emf`, inspired by the GNU utility of the same name **gdiff(1)**

diff(1) must be executable on the system before **gdiff** can function. The **diff(1)** invocation must include the *context* difference, which annotates the differences with a `+`, **−** or `!` markers. **diff(1)** is typically invoked with the options **−c −w**.

diff(1) is a standard utility on UNIX systems. For Windows 95/NT a version of GNU **diff** may be found at:

> <*ftp.winsite.com/ftp/pub/pc/winnt/misc/gnudiff.zip*>

For MS−DOS users, a DJGPP port of GNU **diff** is also available on the net. A commercial version of **diff** is also available from MKS.

SEE ALSO

compare−windows(2), compile(3), **diff(1)**, gdiff(3f), grep(3), %grep−com(5).

%grep−com(5)

NAME

%grep−com − Grep command line

SYNOPSIS

`%grep−com` "*string*"; Default is "`grep `"

DESCRIPTION

Sets the command line used to execute a **grep(1)** system command. The output of the grep(3) execution should include both file and line number information so that the command get−next−line(2) can be used properly. This is not defined by default and the **grep** command will not execute until it is defined.

grep(1) is typically used with the **−n** option which produced line numbering information which drives the get−next−line(2) command.

EXAMPLE

The following example shows how the **grep** strings are defined.

```
set-variable %grep-com "grep -n "
0 add-next-line "*grep*"
add-next-line "*grep*" "%f:%l:"
```

This definition corresponds to a **grep** output such as:−

```
m5var000.5:13:Sets the  number of seconds to wait
m5var000.5:14:temporary file to t seconds. A
m5var000.5:15:Note than the  temporary
m5var000.5:17:saving a buffer.  Backup  files are
m5var000.5:24:On unlimited  length  file  name  systems
```

where **grep** produces file and line number information for every match.

Use add−next−line(2) to define the line pattern produced by **grep**. Some versions of **grep** place the file name on a single line matches within the file occur on subsequent lines. In this case additional *add−next−line* patterns may be defined to cater for the **grep** output as follows:

```
set-variable %grep-com "grep /n "
0 add-next-line "*grep*"
add-next-line "*grep*" "File: %f:"
add-next-line "*grep*" "%l:"
```

This definition would be used with a **grep** output such as:−

```
File:m5var000.5:
13:Sets the  number of seconds to wait
14:temporary file to t seconds. A
15:Note than the  temporary
17:saving a buffer.  Backup  files are
24:On unlimited  length  file  name  systems
File:m5var001.5:
```

NOTES

grep(1) is a standard utility on UNIX systems. For Windows 95/NT a version of GNU **grep** may be found at:

> *<ftp.winsite.com/ftp/pub/pc/winnt/misc/gnugrep.zip>*

For MS−DOS users, a DJGPP port of **grep** is also available on the net. A commercial version of **grep** is also available from MKS.

SEE ALSO

add−next−line(2), **grep(1)**, grep(3), add−next−line(2).

%http−proxy−addr(5)

NAME

%http−proxy−addr − Set HTTP proxy server address
%http−proxy−port − Set HTTP proxy server port

SYNOPSIS

%http−proxy−addr "*proxy−addr*"
%http−proxy−port "*port−number*"; Default is `80`

DESCRIPTION

If the **%http−proxy−addr** variable is set all HTTP file loading requests, using commands like find−file(2), are sent via the given proxy server. **%http−proxy−port** should be set to the proxy servers port number, defaulting to `80` if not set. These variables are typically set in your `<user>.emf` setup file, e.g.:

```
set-variable %http-proxy-addr "proxy.foobar.com"
set-variable %http-proxy-port "8080"
```

NOTES

Note that http is available on UNIX by default, but must be compiled in for win32 versions.

SEE ALSO

%http−flags(5), find−file(2), ftp(3).

%tag−file(5)

NAME

%tag−file − Tags file name
%tag−template − Tag file search string
%tag−option − Tag file search option

SYNOPSIS

%tag−file "*fileName*"
%tag−template "*string*"
%tag−option "*string*"

DESCRIPTION

The **%tag−file** and **%tag−template** variables must be defined for find−tag(2) to work, they define the information required to locate tag references.

%tag−file is the name of the tag file to be used, usually set to "**tags**". **%tag−template** is a regular expression search string used to identify tags in a tag file. For example, a tag usually consists of a name "`%[^\t]`" followed by a tab "`\t`" followed by the file name that contains the function "`%[^\t]`" followed by another tab, followed by the search string and end of line "`%[^\n]\n`", i.e.

```
set-variable %tag-template  "%[^\t]\t%[^\t]\t%[^\n]\n"
```

This would match a **vi(1)** tag string definition, as created by the UNIX utility **ctags(1)**. The tags file typically contains entries such as:−

```
$auto-time    m5var000.5 /^.XI $auto-time - "Automatic buffer"$/
$buffer-bhook m5var002.5 /^.XI $buffer-bhook - "Buffer macro"$/
$buffer-ehook m5var002.5 /^.XI $buffer-ehook - "Buffer macro"$/
```

The **tag−template** definition is modified to match the output of the **ctags(1)** utility. The format of the tags file may differ from platform to platform, typically the differences are encountered in the line contents field which is usually defined as `/ .... /` for a forward search tag and `? .... ?` for a reverse search tag. Note that a tag's search string typically starts with the character '`^`' and ends with '`$`' which indicate the start and end of the line. The variable fields are expected to be in conventional order of *label*, *filename* and *lineText*.

%tag−option is a user defined variable that modifies the behavior of find−tag(2). This is defined as a string, where each character identifies an option, when undefined then default behavior is assumed. The options are defined as:−

m − Enable multiple tags support

Allows a single tag to be present multiple times in the tag file, typically used when a function is defined multiple times. When enabled **find−tag** can be used to loop through all definitions of a given tag.

r − recursive tags file

By default, the **tags** file is assumed to reside in the current directory location. The **r** option enables an ascending search up the directory hierarchy from the current directory position in search of a recursively generated tags file.

c − Continue recursive tag search

Used in conjunction with flag **r**; when not specified, the recursive searching of a tag stops at the first tag file found, regardless of whether the given tag was located in the found tag file. If this flag is given and the tag was not found in the first tag file, the recursive search continues. This allows local tag files to be created and regularly maintained, yet still being able to access a higher level tag file when required.

Modifications to this variable should be made in the *user*.emf file, e.g. To enable multi recursive ascent tag searching define:−

```
set−variable %tag−option  "mrc"
```

NOTES

Note that GNU Emacs uses it's own tag file format generated by **etags(1)** which does not contain the appropriate information to drive the MicroEmacs '02 **find−tag** command.

The above settings should support the extended version 2 tag file format which has an extra tag type field at the end of each line.

SEE ALSO

ctags(1), ctags(3f), find−tag(2), **vi(1)**.

&abs(4)

NAME

&abs, &add, &sub, &mul, &div, &mod, &neg, &inc, &dec, &pinc, &pdec − Numeric macro operators

SYNOPSIS

&abs *num1*
&add *num1 num2*
&sub *num1 num2*
&multiply *num1 num2*
&divide *num1 num2*
&mod *num1 num2*
&negate *num*

&inc *variable increment*
&dec *variable decrement*
&pinc *variable increment*
&pdec *variable decrement*

DESCRIPTION

The numeric operators operate on variables or integers to perform integer computations, returning the integer result of the operation. The contents of the variables are interpreted as signed integers typically with a dynamic range of 2^31 <= *num* <= 2^31−1.

The operators may all be abbreviated to their three letter abbreviation (i.e. **&multiply** may be expressed as **&mul**). In all cases the first argument is completely evaluated before the second argument.

&abs *num1*

Returns the absolute value of *num1* i.e. if *num1* is positive it returns *num1*, else −*num1*

&add *num1 num2*

Addition of two numbers *num1* and *num2*. i.e. *num1* + *num2*

&sub *num1 num2*

Subtract the second number *num2* from the first *num1* i.e. *num1* − *num2*.

&multiply *num1 num2*

(Signed) Multiply *num1* by *num2*. i.e. *num1* * *num2*. **&mul** is the three letter abbreviation.

&div *num1 num2*

Divide the first number *num1* by the second *num2*, returning the integer result. i.e. *num1* / *num2*. **&div** is the three letter abbreviation.

&mod *num1 num2*

Divide the first number *num1* by the second *num2*, returning the integer remainder. i.e. *num1* % *num2*.

&negate *num*

Negate the integer (multiply by −1) i.e. −*num*. **&neg** is the three letter abbreviation.

Expression evaluation is prefix. Operators may be nested using a pre−fix ordering, there is no concept of brackets (in−fix notation). The expression `(2 * 3) + 4` is expressed as:−

```
&add &mul 2 3 4
```

conversely `2 * (3 + 4)` is expressed as:−

```
&mul 2 &add 3 4
```

The pre/post incrementing and decrementing operators provide a mechanism for stepping through indexed information without incurring the overhead of providing multiple statements to perform assignment operations. The *variable* argument MUST be the name of a variable, it cannot be an expression or an indirection. The *increment* may be any integer expression (including another auto (dec)increment). Note that *variable* is re−assigned with it's new value within the operator, therefore use with care when performing multiple (dec)increments within the same statement line. The four operators are defined as follows:

&inc *variable increment*

Pre−increment the *variable* by *increment*, returning the incremented value i.e. *variable* += *increment*.

&dec *variable decrement*

Pre−decrement the *variable* by *decrement*, returning the decrement value i.e. *variable* −= *decrement*.

&pinc *variable increment*

Post−increment the *variable* by *increment*, returning the pre−increment value i.e. *variable*++., where the ++ value is determined by *increment*. The return value is the value of *variable* as passed by the caller, the next reference to *variable* uses the *variable*+*increment* value.

&pdec *variable decrement*

Post−decrement the *variable* by *decrement*, returning the pre−decrement value i.e. *variable*−−, where the −− value is determined by *decrement*. **EXAMPLE**

Add two numbers together and assign to a variable:−

```
set-variable %result &add %num1 %num2
```

Increment `%result` by 1 and add to `%result2`

```
set-variable %result  &add %result 1
set-variable %result2 &add %result2 %result
```

The previous example could have used the increment operators to achieve the same result in a single operation e.g.

```
set-variable %result2 &add %result2 &inc %result 1
```

SEE ALSO

Variable Functions, &great(4).

&and(4)

NAME

&and, &or, &not, &equal, &sequal − Logical macro operators

SYNOPSIS

&and *log1 log2*
&or *log1 log2*
&not *log*

&equal *num1 num2*
&great *num1 num2*
&less *num1 num2*

DESCRIPTION

The logical testing operators perform comparison tests, returning a boolean value of TRUE (1) or FALSE (0).

The functions may all be abbreviated to their three letter abbreviation (i.e. **&great** may be expressed as **&gre**). In all cases the first argument is completely evaluated before the second argument. Logical operators include:−

&and *log1 log2*

TRUE if the logical arguments *log1* and *log2* are both TRUE.

&or *log1 log2*

TRUE if either one of the logical arguments *log1* and *log2* are TRUE.

&not *log*

Logical NOT. Returns the opposite logical value to *log*.

The numerical logical functions operate with integer arguments:

&equal *num1 num2*

TRUE. If numerical arguments *num1* and *num2* numerically equal. Abbreviated form of the function is **&equ**.

great *num1 num2*

`TRUE`. If numerical argument *num1* is greater than *num2*. Abbreviated form of the function is **&gre**.

&less *num1 num2*

`TRUE`. If numerical argument *num1* is less than *num2* Abbreviated form of the function is **&les**.

Evaluation of the logical operators are left to right, the leftmost argument is fully evaluated before the next argument. The operator ordering is prefix notation (see &add(4) for an example of prefix ordering).

EXAMPLE

Test for integers in the range greater than 12:

```
!if &great %i 12
    ...
```

Test for integers in the range 8−12, inclusive

```
!if &and &great 7 &less 13
    ...
```

NOTES

MicroEmacs always evaluates all arguments operators BEFORE the result is obtained, this differs from most programming languages. Consider the following example:

```
!if &and &bmod "edit" &iseq @mc1 "Save buffer first [y/n]? " "nNyY" "y"
    save−buffer
!endif
```

This would not not work as the user may expect, the user would be prompted to save every time regardless of whether the buffer has been changed. Instead the following should be used:

```
!if &bmod "edit"
    !if &iseq @mc1 "Save buffer first [y/n]? " "nNyY" "y"
        save−buffer
    !endif
!endif
```

SEE ALSO

Variable Functions, &add(4), &sequal(4), &sin(4), &cond(4).

&atoi(4)

NAME

&ato, &gmod, &bmo, &ind, &inw, &exi − Miscellaneous functions

SYNOPSIS

&atoi *char*
&itoa *num*

&gmode *mode*
&bmode *mode*
&nbmode *buffer mode*
&inword *char*

&indirect *str*

&exist *str*

DESCRIPTION

These are a selection of miscellaneous functions providing tests and exchanging of information.

The functions may all be abbreviated to their three letter abbreviation (i.e. **&indirect** may be expressed as **&ind**). In all cases the first argument is completely evaluated before the second argument.

&atoi *char*

Converts the given character *char* to it's ASCII number which is returned. (see **&itoa**). Abbreviated command is **&ato**.

&itoa *num*

Converts an integer *num* to it's ASCII character representation which is returned to the caller. Abbreviated command is **&ito**.

&gmode *mode*

Returns `1` if the given mode *mode* is globally enabled. Allows macros to test the global mode state (see Operating Modes). Abbreviated command is **&gmo**.

&bmode *mode*

Returns `1` if the mode *mode* is enabled in the current buffer. Allows macros to test the state of the buffer mode. Abbreviated command is **&bmo**.

&nbmode *buffer mode*

Returns `1` if the mode *mode* is enabled in buffer *buffer* . Allows macros to test the state of a buffer mode other than the current. Abbreviated command is **&nbm**.

&inword *char*

`TRUE`. If the given character *char* is a 'word' character, see forward−word(2) for a description of a 'word' character. Abbreviated command is **&inw**.

&indirect *str*

Evaluate *str* as a variable. The *str* argument is evaluated and takes the resulting string, and then uses it as a variable name. i.e. a variable may reference another variable which contains the data to be referenced. Abbreviated command is **&ind**.

&exist *str*

Tests for the existance of *str* which may be a variable or a command/macro name, returning `TRUE` if the variable or command does currently exist. Abbreviated command is **&exi**. **EXAMPLE**

The **&exi** function is etremely useful in initializing, for example:

```
!if &not &exi %my-init
    ; %my-init is not yet defined so this is the first call
    set-variable %my-init 1
    .
    .
```

Or in all the file hooks a user defined extension is checked for and executed if defined:

```
define-macro fhook-c
    .
    .
    ; execute user extensions if macro is defined
    !if &exi my-fhook-c
        my-fhook-c
    !endif
!emacro
```

The **&ind** function deserves more explanation. **&ind** evaluates its string argument *str*, takes the resulting string and then uses it as a variable name. For example, given the following code sequence:

```
; set up reference table

set-variable  %one      "elephant"
set-variable  %two      "giraffe"
set-variable  %three    "donkey"
```

```
        set-variable  %index "%two"
        insert-string &ind %index
```

the string "`giraffe`" would have been inserted at the point in the current buffer.

The **&bmode** invocation allows a calling macro to determine the buffer mode state (see Operating Modes). Consider the following example which is a macro to perform a case insensitive alphabetic sort using the sort−lines(2) function. **sort−list** sorts according to the state of the exact(2m) mode, hence the macro has to determine the buffer state in order to be able to do the sort.

```
        define-macro sort-lines-ignore-case
            set-variable #l0 &bmod exact
            -1 buffer-mode "exact"
            !if @?
                @# sort-lines
            !else
                sort-lines
            !endif
            &cond #l0 1 -1 buffer-mode "exact"
        !emacro
```

The **&inword** function is shown in the following example. In this case the mouse is positioned over a word. The **&inword** function is used to determine if the cursor is on a valid word character, if so the cursor is placed at the start of the word.

```
        define-macro mouse-control-drop-left
            set-cursor-to-mouse
            !if &inword @wc
                backward-word
                set-mark
                forward-word
            !else
                ...
            !endif
            copy-region
            set-cursor-to-mouse
        !emacro
```

SEE ALSO

Operating Modes, Variable Functions, &sprintf(4), &equal(4).

&band(4)

NAME

&band, &bor, &bnot, &bxor − Bitwise macro operators

SYNOPSIS

&band *num1 num2*
&bor *num1 num2*
&bxor *num1 num2*
&bnot *num*

DESCRIPTION

The bitwise operators perform bit operations on numeric values returning a numerical result of the operation.

The functions may all be abbreviated to their three letter abbreviation (i.e. **&band** may be expressed as **&ban**). In all cases the first argument is completely evaluated before the second argument.

&band *num1 num2*

Bitwise AND of *num1* and *num2* i.e. *num1* & *num2*.

&bor *num1 num2*

Bitwise (inclusive) OR of *num1* and *num2* i.e. *num1* | *num2*.

&bxor *num1 num2*

Bitwise (exclusive OR) XOR of *num1* and *num2* i.e. *num1* ^ *num2*.

&not *num*

Bitwise NOT operator of *num*, inverts the state of all bits i.e. ~*num*.

Evaluation of the bitwise operators are left to right, the leftmost argument is fully evaluated before the next argument. The operator ordering is prefix notation (see &add(4) for an example of prefix ordering).

SEE ALSO

Variable Functions, &add(4), &and(4), &negate(4), &or(4).

&cat(4)

NAME

&cat, &lef, &rig, &mid, &len, &slo, &trb − String macro operators

SYNOPSIS

&cat *str1 str2*
&lef *str len*
&right *str index*
&mid *str index len*

&len *str*

&slower *str*
&supper *str*

&trboth *str*
&trleft *str*
&trright *str*

DESCRIPTION

The string operators operate on character strings (**%** or **$** variables), performing general string manipulation, returning a string result.

The operators may all be abbreviated to their three letter abbreviation (i.e. **&right** may be expressed as **&rig**). In all cases the first argument is completely evaluated before the second argument.

&cat *str1 str2*

Concatenate two string *str1* with *str2* to form a new string. i.e. *str1str2*

&lef *str len*

Return *len* leftmost characters from *str*. If *str* length is shorter than *len* then the string itself is returned. A *len* of zero returns the empty string.

&rig *str index*

Returns the rightmost characters of string *str* from index *index*. This function causes some confusion, consider **&lef** and **&rig** to be the string equivalents of their integer counterparts &div and &mod; **&rig** returns the remainder of the equivalent **&lef** function. Invocation with *index* set to zero returns *str*.

&mid *index len*

Extracts a sub−string from string *str*, starting at position *index* of length *len*.

&len *str*

Returns the integer length of the string (number of characters).

&slower *str*

Returns the given string with all upper case characters converted to lower case.

&supper *str*

Returns the given string with all lower case characters converted to upper case.

&trboth *str*

Returns the given string trimmed of white spaces (i.e. ' ', '`\t`', '`\r`', '`\n`', '`\Cl`' and '`\Ck`') from both sides of the string.

&trleft *str*

Returns the given string trimmed of white spaces from the left side of the string only.

&trright *str*

Returns the given string trimmed of white spaces from the right side, or end, of the string only.

Evaluation of the strings is left to right, the leftmost argument is fully evaluated before the next argument. The operator ordering is prefix notation (see &add(4) for an example of prefix ordering).

EXAMPLE

Concatenate two strings `abc` and `def` together:−

```
set−variable %result &cat "abc" "def"
```

To concatenate three strings `abc`, `def` `ghi` together:

```
set−variable %result &cat "abc" &cat "def" "ghi"
```

or, a slightly different ordering:

```
set−variable %result &cat &cat "abc" "def" "ghi"
```

Retrieve the leftmost character of a string variable, modify the variable to contain the remainder.

```
set−variable %foo "abcdef"
```

```
        set-variable %c   &lef %foo 1
        set-variable %foo &rig %foo 1
```

Where `%c` = "`a`"; `%foo` = "`bcdef`" following evaluation.

To retrieve the characters `cde` into variable `%result` from the string "`abcdef`" use:

```
        set-variable %result &mid "abcdef" 2 3
```

To retrieve the rightmost character from the string:

```
        set-variable %foo "abcdef"
        set-variable %result &rig %foo &sub &len %foo 1
```

or the same result could be achieved using **&mid**:

```
        set-variable %result &mid %foo &sub &len %foo 1 1
```

To get an input string from the user which is free of spaces at the start and end:

```
        set-variable %result &trb @ml "Enter string"
```

NOTES

The original **MicroEMACS** "**&rig** *str n*" function returns the last *n* characters from the string *str* this differs from the definition of **&rig** in this implementation. As most string decomposition is performed left to right, and to make **&lef** and **&rig** complement each other, the indexing of the function has been modified.

SEE ALSO

Variable Functions, &sin(4), &sequal(4), &lget(4), &sprintf(4).

&cbind(4)

NAME

&cbind, &kbind, &nkind − Command/key binding operators

SYNOPSIS

&cbind *key*
&kbind *n command*
&nbind *key*

DESCRIPTION

&cbind returns the command bound to the given key sequence, **&kbind** can be abbreviated to **&kbi**. If the key is not bound then **&kbind** returns the string "*ERROR*".

&nbind returns the numerical argument associated with the given key binding, **&nbind** can be abbreviated to **&nbi**. If the key is not bound then **&nbind** returns the string "*ERROR*", if the binding has no argument then an empty string ("") is returned.

&kbind returns a key sequence bound to the given *command* with the given numerical argument *n*. If no binding can be found then **&kbind** returns an empty string ("").

EXAMPLE

The following example waits for the user to press a key, then prints what command the key is bound to.

```
ml-write "Enter key: "
set-variable #l0 @cgk
ml-write &spr "%s is bound to %s" #l0 &cbin #l0
```

NOTES

In March 2001 **&kbind** was renamed **&ckind** and a new **&nkind** and **&kbind** added.

SEE ALSO

Variable Functions, global−bind−key(2).

&cond(4)

NAME

&cond − Conditional expression operator

SYNOPSIS

&cond *log expr1 expr2*

DESCRIPTION

The conditional expression **&cond** provides an alternative way to write !if−!else−!endif constructs, e.g.:−

```
!if &gre %a %b
    set−variable %z %a
!else
    set−variable %z %b
!endif
```

may be replaced with a conditional expression, breaking down the components then

log is **&gre %a %b**
expr1 is **%a**
expr2 is **%b**

rewriting the expression we get:

```
set−variable %z &cond &gre %a %b %a %b
```

This is far more concise, albeit a little less readable, but does improve the performance of macros as there is less information to interpret.

The **&cond** operator accepts three fields, ALL fields are evaluated although only one of the results *expr1* or *expr2* is used. The *log* field is a logical value, if it is non−zero (`TRUE`) then the result of the *expr1* evaluation is used, otherwise the result of *expr2* is used.

It should be noted that the conditional expression may be used in any construct i.e. &add(4), &cat(4), etc. the *expr* arguments may be strings, numbers or booleans the resultant value of the *expr* arguments is simply returned to the calling expression.

SEE ALSO

Variable Functions, &add(4), &great(4).

&find(4)

NAME

&find − Find a file on the search path
&which − Find a program on the path

SYNOPSIS

&find *<basename> <extension>*
&which *<progname>*

DESCRIPTION

&find searches for a named file *<basename><extension>* on the MicroEmacs '02 search path defined by the variable $search−path(5) (initialized from the environment variable $MEPATH(5)). Each path component defined in **$search−path** is prepended to the constructed file name and it's existence is tested. If the file exists, then the FULL path name of the file is returned to the caller, otherwise `ERROR`.

<basename>

The base name of the file, excluding any extension.

<extension>

The extension of the file name, this must be specified with the extension delimiter, typically dot ('`.`'). A `NULL` string (e.g. '`""`') may be specified if no extension is required.

&which searches for the given executable program *<progname>* on the system program search path defined the the environment variable **$PATH**.

USAGE

&find is typically used with insert−file(2) and find−file(2) within macro scripts, and is used to locate user specific files.

EXAMPLE

The following example uses **&find** to locate the uses 'C' template file. Given a **$search−path** setting of `/usr/bob/emacs:/usr/local/microemacs`:−

```
        insert−file &find "c" ".etf"
```

Would insert the file `/usr/bob/emacs/c.etf` if it existed, else the file `/usr/local/microemacs/c.etf` if it exists.

SEE ALSO

Variable Functions, find−file(2), $search−path(5), insert−file(2).

&rep(4)

NAME

&rep, &irep, &xrep, &xirep − Replace string in string functions

SYNOPSIS

&rep *str1 str2 str3*
&irep *str1 str2 str3*
&xrep *str1 str2 str3*
&xirep *str1 str2 str3*

DESCRIPTION

These functions search for *str2* in *str1*, replacing it with *str3*, returning the resultant string.

The functions may all be abbreviated to their three letter abbreviation (i.e. **&xirep** may be expressed as **&xir**). In all cases the first argument is completely evaluated before the second and third arguments.

&rep *string search replace*

Searches for the *search* string in the given *string* using a simple case sensitive exact match algorithm. Any occurrences are removed from *string* and *replace* is inserted in its place. Either of the 3 input strings can be the empty string (`""`).

&irep *string search replace*

&irep is identical to **&rep** except a case insensitive search algorithm is used.

&xrep *string regex−search regex−replace*

&xrep can be used to access the more powerful regular expression searching capabilities. The function is similar to **&rep** except it takes a regex search string and the replacement string may also refer to all or part of the matched string. See Regular Expressions for information on the *regex* format.

&xirep *string regex−search regex−replace*

&xirep is identical to **&xrep** except a case insensitive regex search is used. **EXAMPLE**

The following example turns a UNIX format file name (using a '`/`' to divide directories − like MicroEmacs) into an windows format name (using a '`\`'):

```
set−variable #l0 &rep #l0 "/" "\\"
```

The following example replaces one or more white spaces in the variable with a single space, this is an easy way to remove unnecessary spaces:

```
set-variable #l0 "This   is   not   so    spacey    after   xrep"
set-variable #l0 &xrep #l0 "\\s +" " "
ml-write #l0
```

SEE ALSO

Operating Modes, Variable Functions, &sequal(4), &sin(4).

&sequal(4)

NAME

&seq, &iseq, &sle, &sgre, &xseq, &xiseq − String logical macro operators

SYNOPSIS

&sequal *str1 str2*
&isequal *str1 str2*
&sless *str1 str2*
&sgreat *str1 str2*

&xsequal *str1 regex*
&xisequal *str1 regex*

DESCRIPTION

The string logical testing operators perform string comparison tests, returning a boolean value of TRUE (1) or FALSE (0).

The functions may all be shortened to their three letter abbreviation (i.e. **&sequal** may be expressed as **&seq**). In all cases the first argument is completely evaluated before the second argument. String logical operators include:−

&sequal *str1 str2*

TRUE if the two strings *str1* and *str2* are the same. Abbreviated form of the function is **&seq**.

&sless *str1 str2*

TRUE if string *str1* alphabetically less than *str2*. Abbreviated form of the function is **&sle**.

&sgreat *str1 str2*

TRUE if string *str1* alphabetically larger than *str2*. Abbreviated form of the function is **&sgr**.

&isequal *str1 str2*

TRUE if the two strings *str1* and *str2* are the same ignoring letter case. Abbreviated form of the function is **&ise**.

&xsequal *str1 regex*

TRUE if the string *str1* matches the *regex* (case sensitive). Abbreviated form of the function is **&xse**.

See Regular Expressions for information on the *regex* format.

&xisequal *str1 regex*

TRUE if the string *str1* matches the *regex* (case insensitive). Abbreviated form of the function is **&xis**. See Regular Expressions for information on the *regex* format.

Evaluation of the string logical operators are left to right, the leftmost argument is fully evaluated before the next argument. The operator ordering is prefix notation (see &add(4) for an example of prefix ordering).

EXAMPLE

Test for variable $buffer−bname(5) is equal to `*scratch*`:

```
!if &seq $buffer-bname "*scratch*"
    ...
```

The following example tests a character is in the range `a−z`:

```
!if &not &and &sle %c "a" &sgr %c "z"
    ...
```

The following example inserts the string "`c`" into the alphabetically order string list **%test−list**:

```
set-variable %test-list "|a|b|d|e|"
set-variable %test-insert "c"

set-variable #l0 1
!while &and &not &seq &lget %test-list #l0 "" ...
        ... &sle &lget %test-list #l0 %test-insert
    set-variable #l0 &add #l0 1
!done
set-variable %test-list &lins %test-list #l0 %test-insert
```

The first test on the **!while &and** conditional checks that the current item in the list is not an empty string (""). If it is the end of the list has been reached.

The following example tests the current buffers file name for a ".c" extension:

```
!if &xse $buffer-fname ".*\\.c"
    ...
```

Note the '\' character is needed to protect the second '`.`', i.e. so that it does not match any character and the second '\' is required as the string is first parsed by the macro interpreter which changes it to "`.*\.c`" which is then interpreted as a regex.

SEE ALSO

Variable Functions, &sin(4), &slower(4), &rep(4), &add(4), &equal(4), &cond(4), Regular Expressions.

&sin(4)

NAME

&sin, &isin, &rsin, &risin − String in string test functions

SYNOPSIS

&sin *str1 str2*
&isin *str1 str2*
&rsin *str1 str2*
&risin *str1 str2*

DESCRIPTION

These functions test for the existence of *str1* in *str2*, returning the position of the string in *str2* or 0 if not found.

The functions may all be abbreviated to their three letter abbreviation (i.e. **&risin** may be expressed as **&ris**). In all cases the first argument is completely evaluated before the second argument.

&sin *str1 str2*

Returns `0` if string *str1* does not exists in string *str2*. Otherwise the function returns the character position + 1 of the location of the first character of the first occurrence of *str1*.

&isin *str1 str2*

Returns `0` if case insensitive string *str1* does not exists in string *str2*. Otherwise the function returns the character position + 1 of the location of the first character of the first occurrence of *str1*.

&rsin *str1 str2*

Returns `0` if string *str1* does not exists in string *str2*. Otherwise the function returns the character position + 1 of the location of the first character of the last occurrence of *str1*.

&risin *str1 str2*

Returns `0` if case insensitive string *str1* does not exists in string *str2*. Otherwise the function returns the character position + 1 of the location of the first character of the last occurrence of *str1*. **EXAMPLE**

The **&sin** and similar functions are useful for two different purposes. Consider the following example, this utilizes **&sin** in two different contexts. `!while &not &sin @wc " \t\n"` is a test for the end of the number, i.e. a white space character (`<tab>`, `<SPACE>` or `<NL>`).

The invocation `set-variable #l1 &isin @wc "0123456789abcdef"` is subtly different. In this case the return value is used to convert the character to it's integer hex value by using the value returned by **&isin**.

```
;
; calc-hexnum
; Convert the sting from the current position in the buffer
; to a hexadecimal number.
define-macro calc-hexnum
    forward-delete-char
    forward-delete-char
    set-variable #l0 0
    !while &not &sin @wc " \t\n"
        set-variable #l1 &isin @wc "0123456789abcdef"
        !if &not #l1
            ml-write "Bad Hex number found"
            !abort
        !endif
        set-variable #l0 &mul #l0 16
        set-variable #l0 &add #l0 &sub #l1 1
        forward-delete-char
    !done
    insert-string #l0
!emacro
```

The **&rsin** function is very similar to sin except the value return is the position of the last occurrence of the string in the given string instead of the first. This is particularly useful when extracting the path or file name from a complete file name. For example, given a UNIX style file name such as `"/usr/local/bin/me"` the path can be obtained using `set-variable %path &lef %pathfile &rsin "/" %pathfile` and the file name by using `set-variable %file &rig %pathfile &rsin "/" %pathfile`

SEE ALSO

Operating Modes, Variable Functions, &sequal(4), &rep(4).

&ldel(4)

NAME

&ldel, &lfind, &lget, &linsert, &lset − List manipulation functions

SYNOPSIS

&ldel *list index*
&lfind *list value*
&lget *list index*
&linsert *list index value*
&lset *list index value*

DESCRIPTION

The list manipulation functions perform operations on specially formatted strings called lists. A list is defined as:

```
"|value1|value2|.....|valueN|"
```

Where '`|`' is the dividing character, this is not fixed to a '`|`', but is defined by the first character of the string. Following are all valid lists.

```
"|1|2|3|4|5|"
"X1X2X3X4X5X"
"\CAHello\CAWorld\CA"
"??"
```

The functions may all be abbreviated to their three letter abbreviation (i.e. **&linsert** may be expressed as **&lin**). In all cases the first argument is completely evaluated before the second or third argument.

&ldel *list index*

Creates a new list from deleting item *index* from *list*. If *index* is out of *list*'s range (0 < index <= # items in list) then *list* is returned unchanged.

&lfind *list value*

Returns the index whose item is the same as *value* in *list*. If *value* is not found in *list* then "`0`" is returned.

&lget *list index*

Returns the value of item *index* in *list*. If *index* is out of *list*'s range (0 < index <= # items in list) then an empty string is returned.

&linsert *list index value*

Creates a new list from inserting *value* into *list* at point *index*, thereby pushing item *index* to *index*+1 etc. If *index* is 0 the *value* is inserted at the beginning of the list, if *index* is less than 0 or greater that the number of items in *list* then *value* is inserted at the end of the list.

&lset *list index value*

Creates a new list from setting *index* of *list* to *value*. If *index* is out of *list*'s range (0 < index <= # items in list) then **&lset** behaves like **&linsert**. **EXAMPLE**

The following example moves item 4 in a list to position 2:

```
set-variable #l0 &lget %list 4
set-variable #l1 &ldel %list 4
set-variable %list &lins #l1 2 #l0
```

The following example is taken from vm.emf, it firstly checks where the user has entered a vm command, if not then the key is execute as normal, otherwise the appropriate vm command is executed.

```
define-macro vm-input
    set-variable #l2 @cck
    set-variable #l3 @cc
    !if &not &set #l0 &lfi "|esc h|delete|space|return|A|a|C|c|....|z|" #l2
        !if &not &seq #l3 "ERROR"
            execute-line &spr "!nma %s %s" &cond @? @# "" #l3
            !return
        !endif
        ml-write &spr "[Key \"%s\" not bound - \"esc h\" to view help]" #l2
        !abort
    !endif
    set-variable #l1 &lget "|%osd-vm-help osd|vm-del-windows|scroll-down|....
                       ....vm-goto-list|vm-Archive-box|vm-archive-box|....
                       vm-cut-all-data|0 vm-extract-data|...|vm-forward|" #l0
    execute-line #l1
!emacro
```

SEE ALSO

Variable Functions, &mid(4), &cat(4).

&opt(4)

NAME

&opt − MicroEmacs optional feature test

SYNOPSIS

&opt *str*

DESCRIPTION

This function can be used to test the availability of optional features in the current session of MicroEmacs. Some features, like spelling checker support, are a compilation option, other options like mouse support may also be unavailable on some platforms. The **&opt** function can be used by macros to check that required base functionality is available.

The function returns `1` in the given feature "*str*" is supported, otherwise it returns `0` if the feature is unknown or not supported in the running version.

NOTES

Optional components of MicroEmacs '02 are enabled/disabled at compile time, most options are configured by `MEOPT_`<*NAME*> `#define`'s within the source file `emain.h`. Following is a complete list of options, giving the **opt** string and `#define` label:

abb − MEOPT_ABBREV

Abbreviation functionality (see expand−abbrev(2)).

cal − MEOPT_CALLBACK

Callback and idle event handling (see create−callback(2)).

cfe − MEOPT_CFENCE

Fence matching (see $fmatchdelay(5)).

cli − MEOPT_CLIENTSERVER

Client/server support (see Client−Server).

col − MEOPT_COLOR

All color support (making hilighting redundent etc, see add−color(2)).

cry − MEOPT_CRYPT

File encryption (see crypt(2m) mode).

deb − MEOPT_DEBUGM

Macro debugging (see $debug(5)).

dir − MEOPT_DIRLIST

Directory listing when loading a directory (see file−browser(3) and dir(2m) mode).

ext − MEOPT_EXTENDED

Miscellaneous more advanced commands and features such as append−buffer(2).

fho − MEOPT_FILEHOOK

File type auto−detection and configuration (see add−file−hook(2)).

fra − MEOPT_FRAME

Multiple frames (Internal or external, see opt "**mwf**" and command create−frame(2)).

has − MEOPT_CMDHASH

Use a hash table for rapid command name lookup.

hil − MEOPT_HILIGHT

Hilight and user definable indentation rules (see hilight(2) and indent(2)).

hsp − MEOPT_HSPLIT

Horizontal window splitting (see split−window−horizontally(2)).

ipi − MEOPT_IPIPES

Interactive pipes (see ipipe−shell−command(2)).

ise − MEOPT_ISEARCH

Incremental search (see isearch−forward(2)).

lbi − MEOPT_LOCALBIND

Buffer, message−line and OSD local binding overrides (see buffer−bind−key(2)).

mag − MEOPT_MAGIC

Regular expression search engine (see magic(2m) mode).

mou − MEOPT_MOUSE

Mouse support (see $mouse(5)).

mwf − MEOPT_MWFRAME

Multiple window frame support (see opt "**fra**").

nar − MEOPT_NARROW

Buffer narrowing (see narrow−buffer(2)).

nex − MEOPT_FILENEXT

Location list stepping (see get−next−line(2)).

osd − MEOPT_OSD

On Screen Display GUI support (see osd(2)).

pok − MEOPT_POKE

Direct screen poking (see screen−poke(2)).

pos − MEOPT_POSITION

Position storing and returning (see set−position(2)).

pri − MEOPT_PRINT

Printing support (see print−buffer(2)).

rcs − MEOPT_RCS

File Revision Control Support (see $rcs−co−com(5)).

reg − MEOPT_REGISTRY

Internal registry and history support (see read−registry(2) and read−history(2)).

scr − MEOPT_SCROLL

Window scroll−bar support.

soc − MEOPT_SOCKET

URL support, FTP and HTTP via sockets (see find−file(2)).

spa − MEOPT_SPAWN

External process launching (see shell−command(2)).

spe − MEOPT_SPELL

Spelling checker support (see spell(2)).

tag − MEOPT_TAGS

Tags support (see find−tag(2)).

tim − MEOPT_TIMSTMP

File timestamping on save (see time(2m) mode).

typ − MEOPT_TYPEAH

Input detect or 'type−ahead' for background processing support.

und − MEOPT_UNDO

Undo support (see undo(2)).

wor − MEOPT_WORDPRO

Word−processor style commands like fill−paragraph(2) (see forward−paragraph(2)). **EXAMPLE**

The following example checks for URL support and if not available it pops up an error:

```
!if &not &opt "soc"
    osd−dialog "Opt Test" "Error: No URL support!" "  &OK  "
!endif
```

SEE ALSO

Building MicroEmacs.

&reg(4)

NAME

&reg − Retrieve a registry value (with default)

SYNOPSIS

&reg *root subkey default*

DESCRIPTION

&reg retrieves the value of a node defined by *root*/*subkey* from the registry. The node name is specified in two components, typically required when iterating over a registry tree, where the *root* component is static and the *subkey* is dynamic, *subkey* may be specified as the null string (`""`) if an absolute registry path is specified.

The *default* value is the value of the node to return if the registry node does not exist.

EXAMPLE

The following example is taken from `me.emf` and uses the registry to retrieve some of the default configuration files:

```
; Load in the color setup
!force execute-file &reg "/history" &cat $platform "/color" "color"
; execute company setup
!if &not &seq &set #l0 &reg "/history" "company" "" ""
    !force execute-file #l0
!endif
```

SEE ALSO

get−registry(2), set−registry(2).

&set(4)

NAME

&set − In−line macro variable assignment

SYNOPSIS

&set *<var> <expr>*

DESCRIPTION

&set performs an in−line macro variable assignment assigning a variable *<var>* the value of the expression *<expr>*, returning the evaluated result to the caller. *<expr>* may be numeric, boolean or a string expression.

&set is typically used for defining (and simultaneously using) indices e.g. as with add−color(2) or add−color−scheme(2). This is a short−hand of set−variable(2).

EXAMPLE

The following example uses **&set** to define new colors (see `color.emf`):

```
; Standard colors
add-color &set %white     0 200 200 200
add-color &set %black     1 0   0   0
add-color &set %red       2 200 0   0
add-color &set %green     3 0   200 0
add-color &set %yellow    4 200 200 0
add-color &set %blue      5 0   0   200
add-color &set %magenta   6 200 0   200
add-color &set %cyan      7 0   200 200
```

SEE ALSO

Variable Functions, &inc(4), set−variable(2).

&sprintf(4)

NAME

&sprintf − Formatted string construction

SYNOPSIS

&sprintf *format args*

DESCRIPTION

The **&sprintf** function (or **&spr** in it's abbreviated form) provides a mechanism to generated a formatted string, similar to the 'C' programming language **sprintf(2)** function.

The **&sprintf** function is generally used where a number of different sources of information have to be converted and joined together to form a new string. It is possible to do this using &cat(4), but it does become complicated if the number of strings to be spliced together is greater than about 4, **sprintf** alleviates these problems and results in faster execution. Where only two, or three strings are to be concatenated **&cat** provides better execution times.

The **&sprintf** function produces a string construct for the *format* and a caller determined number of arguments *args* (variable arguments). The *format* string may contain special '**%**' formatting commands to insert strings and numbers into the base *format* string. The format for the '**%**' commands is "**%nc**" where:−

n

An optional numerical argument, the interpretation of the numeric value is determined by the following command (**c**).

c

The command determines the interpretation of the next argument *arg* which are specified as follows:

d (Decimal integer)

Expects a single numeric argument *arg* which is inserted into the *format* string as decimal text string. If *n* is specified then the inserted text string is fixed to *n* character in length.

n (Repeat String)

Expects two arguments *arg*, the first is a numeric argument giving the number of times to insert the given string (the second argument). If *n* is specified then the string is inserted *n* * *numeric−argument* times.

s (String)

Expects a single argument *arg* which is a string to be inserted into the key. If *n* is given then it is inserted*n* times.

x (Hexadecimal integer)

Expects a single numeric argument *arg* which is inserted into the format string as hexadecimal text string. If *n* is given then the inserted text string will be fixed to *n* character in length.

%

Inserts a single '%', *n* has no effect.

The **&sprintf** function may be nested (i.e. a string argument to **&sprintf** may be the result of another **&sprintf** invocation). Although this type of construct is not generally required !!

EXAMPLE

The following examples show how the command may be used:−

```
set−variable %result &sprintf "Foo [%s%s]" "a" "b"
```

generates "`Foo [ab]`"

```
set−variable %result &sprintf "Foo [%n%s]" 10 "a" "b"
```

generates "`Foo [aaaaaaaaaab]`".

```
set−variable %result &sprintf "[%d] [%3d] [%x] [%3x]" 10 11 12 13
```

generates "`[10] [ 11] [c] [  d]`"

NOTES

It is the callers responsibility to ensure that the correct number of arguments is supplied to match the requested formatting string. The results are undefined if an incorrect number of arguments are supplied.

SEE ALSO

Variable Functions, &cat(4).

&stat(4)

NAME

&stat − Retrieve a file statistic

SYNOPSIS

&stat <*stat*> <*filename*>

DESCRIPTION

&stat returns the specified <*stat*> on the given <*filename*>. Valid <*stat*> values are:−

a

Returns the absolute file name, corrects relative paths and symbolic links, i.e. on unix if the filename is a symbolic link it returns the file name the link points to (recursive), otherwise returns the file name.

d

Returns the file's modification time stamp. The value returned is an integer, larger values indicate a later time.

r

Returns a non−zero value if the user has permission to read the given file.

s

Returns the size of the file in bytes.

t

Returns the type of the file, where values returned are

```
X   File does not exist.
R   File is a regular file.
D   File is a directory.
H   File is a http URL link (see note).
F   File is an ftp URL file (see note).
N   File is an untouchable system file.
```

Note that a URL type is determined from the file name, e.g. http://..., and its existence is not verified.

w

Returns a non−zero value if the user has permission to write to the given file.

x

Returns a non−zero value if the user has permission to execute the given file. **EXAMPLE**

The following example is a macro which, given a file name, uses **&stat** to check that file file is regular:

```
define−macro test−file
    !force set−variable #l0 @1
    !if &not $status
        set−variable #l0 @ml04 "File name"
    !endif
    !if &not &equ &stat "t" #l4 1
        ml−write &spr "[%s is not a regular file]" #l0
        !abort
    !endif
!emacro

test−file "foobar"
```

The macro can be passed a file name and aborts if the file is not regular, there by returning the state.

The follow example checks that a file is not empty, this is used by **mail−check** to test for any incoming mail.

```
    !if &gre &stat "s" %incoming−mail−box
        ml−write "[You have new mail]"
    !endif
```

SEE ALSO

Variable Functions, find−file(2).

.calc.result(5)

NAME

.calc.result − Last calc calculation result

SYNOPSIS

.calc.result *integer*

DESCRIPTION

.calc.result is used to store the result of the last calculation made by calc(3).

The "*LR*" (Last Result) in the next calculation is substituted with this value.

SEE ALSO

calc(3).

which(3)

NAME

which − Program finder
.which.result − Program path

SYNOPSIS

which "*progname*"
.which.result "*string*"

DESCRIPTION

which searches for the given program "*progname*" on the system path (set by the environment variable **$PATH**). If found the location is printed on the message line, otherwise an error message is printed and the command fails.

The variable **.which.result** is set to the last found program or the string "`ERROR`" if the program was not found.

NOTES

which is a macro defined in `tools.emf`, it used the &which macro directive.

SEE ALSO

&which(4).

nroff(9)

SYNOPSIS

0−9, tni, so − UNIX t/nroff file.

FILES

hknroff.emf − UNIX t/nroff file.
nroff.etf − UNIX t/nroff template file
ntags.emf − t/nroff tags generator macro definition.

EXTENSIONS

1, **2**, **3**, **4**, **5**, **6**, **7**, **8**, **9** − UNIX t/nroff files.
tni, **so** − UNIX t/nroff include files.
sm − *[Special]* Superman t/nroff file.

MAGIC STRINGS

−*− nroff −*−

Recognized by GNU and MicroEmacs. Denotes a t/nroff type file, may be used in **.1**/**.9**, **.tni** and **.so** files.

DESCRIPTION

The **nroff** file type templates handle UNIX n/troff type files.

General Editing

On creating a new file, a new header is automatically included into the file. time(2m) is by default enabled, allowing the modification time−stamp to be maintained in the header.

Hilighting

The hilighting features allow commands, variables, logical, preprocessor definitions, comments, strings and characters of the language to be differentiated and rendered in different colors.

Tags

A C−tags file may be generated within the editor using the **Tools** −> **Nroff−Tools** −> **Create Tag File**. find−tag(2) takes the user to the file using the tag information. The tags are generated using the **.XI** keyword, this may not be standard for all nroff pages.

Folding and Information Hiding

Generic folding is enabled within the C and C++ files. The folds occur about sections **.S**[**HS**]...**.S**[**HS**] located on the left−hand margin. fold−all(3) (un)folds all regions in the file, fold−current(3) (un)folds the current region. Note that folding does not operate on K&R style code.

Tools

The nroff buffer provides a facility to toggle the hilighting of the buffer on and off. If font change inserts are used (**fB**, **fR**, etc), then the enclosed **bold** and *italic* regions are hilighted, hiding the escape sequences. This allows the nroff text to be viewed in a more representative rendered format.

The local buffer command **aman** invokes, the following command sequence (defined in `hkman`) to render a nroff **man** file into a buffer window;−

```
soelim <file> | tbl −TX | neqn | nroff −man | col −x
```

The command **tex2nr** attempts to convert a latex(9) file into an nroff file. The *latex* escape sequences are converted into their nroff equivalents. The command is only made available when an Nroff file is loaded (as the command is defined in the `hknroff.emf` file).

Short Cuts

The short cut keys used within the buffer are:−

C−c C−s − Insert a font size escape character **S0**.
C−c C−r − Insert a roman font escape character **fR**.
C−c C−b − Insert a bold font escape character **fB**.
C−c C−i − Insert a italic font escape character **fI**.
C−c C−c − Insert a courier font escape character **fC**.
C−c C−p − Insert a previous font escape character **fP**.
esc o, **esc q** − fill−paragraph(2) fills paragraph to next `.XX` command.
C−c b − Bold region by inserting **fB** .. **fR**.
C−c c − Courier region by inserting **fC** .. **fR**.
C−c c − Italic region by inserting **fI** .. **fR**.
C−c C−h − Toggle hilighting on/off.
C−c C−& − Adds nroff padding **f&** about words.
C−x C−& − Removes nroff padding **f&** about words.
esc h − Nroff help.

f2 − (un)fold the current region
f3 − (un)fold all regions

BUGS

The nroff language template is heavily biased towards the **man** macros only and includes all of the extension macros used for generating the JASSPA hypertext documentation.

The template in the current form has been used entirely by JASSPA in generating all of the documentation (**HTML**, **Winhelp**, **ehf**, **PostScript**) used by MicroEmacs '02. It does not include all of the troff/nroff keywords, or keywords for any of the standard macro packages.

The JASSPA documentation preparation tools are proprietary and have not been made publicly available.

SEE ALSO

fill−paragraph(2), find−tag(2), fold−all(3), fold−current(3), ntags(3f), time(2m).

Supported File Types

MacroArguments(4)

NAME

@?, @#, @0, @1, @2, @3, ... @p − Macro arguments

SYNOPSIS

@? − Boolean flagging if a numeric argument was supplied
@# − The value of the numeric argument

@0 − The name of the macro
@1 − The first argument of macro
@2 − The second argument of macro
@3 ... **@***n*

@p − The name of the calling (or parent) macro.

DESCRIPTION

Macros may be passed arguments, allowing a macro to be used by other macros. The **@?** and **@#** are used to determine the numeric argument given to the command. The **@***n* variable (where *n* is an integer) used in the context of a macro allows the macro body to determine it's arguments.

From a macro all commands are called in the following form

```
[num] <macro−name> "arg1" "arg2" ....
```

When executed macros do not have to be given an argument, in this case **@?** will be *0* and **@#** will be *1* (the default argument). If an argument is given then **@?** will be *1* and **@#** will be set to the numeric argument given.

The current macro command name <*macro−name*> can be obtain by using the **@0** variable, e.g.

```
define−macro Test−it
    ml−write @0
!emacro
```

When executed, writes the message "`Test−it`" which is the name of the macro.

Arguments may be passed into macro commands in the same way as standard commands are given arguments. The macro being called can access these by the **@1** to **@n** variables, where *n* is a positive integer. Any variables given as arguments are evaluated so if the variable name is required then enclose it in quotes, e.g.

```
set−variable %test−var "Hello World"
```

```
efine-macro Test-it
    ml-write &cat &cat &cat &cat @0 " " @1 " = " &ind @1
    set-variable  @1 @2
!emacro

Test-it "%test-var" "Goodbye World"
```

On execution the macro writes the message

```
"Test-it %test-var = Hello World"
```

and will set variable `%test-var` to "`Goodbye World`".

The **@p** variable can be used to obtain the name of the macro which is executing the current macro, i.e. the value of the parent's **@0** variable. If the macro was executed directly by the user then there is no parent macro and the value of **@p** is an empty string ("").

DIAGNOSTICS

If an attempt is made to access an argument which has not been given then a error occurs. This error can be trapped using the !force(4) directive, enabling the macro to take appropriate action, see example.

EXAMPLE

Consider the implementation of replace−all−string(3) macro defined in search.emf:

```
define-macro replace-all-string
    !force set-variable #l0 @3
    !if &not $status
        set-variable #l1 @ml05 "Replace all"
        set-variable #l2 @ml05 &spr "Replace [%s] with" #l1
        set-variable #l0 @ml00 "In files"
    !else
        set-variable #l1 @1
        set-variable #l2 @2
    !endif
    .
    .
    .
!emacro
```

In this example if the 3rd argument is not given then the macro gets all arguments from the user.

The **@p** variable having a value of "" when a macro is called directly by the user can be useful when determining the amount of information to feed−back to the user. For example, executing the clean macro is an easy way to remove surplus white characters, so it is often used by other macros as well as by the user. When called directly **clean** refreshes the display and prints a message of completion, but when called by other macros this would cause an unwanted screen−update and message, so clean only does this when executed by the user. This is done as follows:

```
define-macro clean
    ;
    ; Prepare to clean up file.
    .
    .
    .
    !if &seq @p ""
        screen-update
        ml-write "[Cleaned up buffer]"
    !endif
!emacro
```

NOTES

The parsing of arguments can be inefficient because of the way the arguments have to be parsed; to get the 4th argument the 1st, 2nd and 3rd arguments must be evaluated. This is because each argument is not guaranteed to be only one element, it could be an expression which needs to be evaluated. Consider the following invocation of our Test−it macro

```
Test-it &cat "%test" "-var" "Goodbye World"
```

The 2nd argument is not *"%test"* as this is part of the first argument, the 2nd argument is in fact the 4th element and the invocation will have the same effect except slower.

SEE ALSO

MacroNumericArguments, define−macro(2), replace−all−string(3), !force(4).

CommandVariables(4)

NAME

@clk, @cl − Last key or command name
@cck, @cc − Current key or command name
@cgk, @cg − Get a key or command name from the user
@cqk, @cq − Get a quoted key or command name from the user

SYNOPSIS

@clk
@cl
@cck
@cc
@cgk
@cg
@cqk
@cq

DESCRIPTION

The Command Variables allow macros to obtain MicroEmacs '02 input commands and keystrokes from the user. The general format of the command is:−

@c*i*[**k**]

Where,

i

Determines the source of the input as follows:−

l

The last input entered.

c

The current input entered.

q

Provides a low level character input mechanism, obtaining a single raw character input from the user. The input fetch does not interact with the message line and the user is NOT prompted for input (use ml−write(2) to create your own message). **@cq** is very low level, it is generally preferable to use **@cg** which provides a more intelligent binding.

g

Like **@cq**, **@cg**[**k**] gets a single character input, however if the input is bound to a function then the function name is returned instead of the character e.g. if `^F` or `<left-arrow>` is depressed then **forward−char** is returned. This has distinct advantages over **@cq** as the binding becomes device independent and executes on all platforms. In addition, it honors the users bindings, however bizarre.

k

When, omitted command input is returned to the caller (i.e. the name of the command, such as "`forward-char`"). When present, the raw keystroke is returned to the caller, i.e. "`^F` (control−F).

The **@cl**, **@clk**, **@cc** and **@cck** variables can also be set, this feature can be used by macros to change the command history. While setting the current command is limited in use, setting the last command can be immensely useful, consider the following macro code:−

```
kill-line
forward-line
set-variable @cl kill-line
kill-line
```

Without the setting of the **@cl** variable, the current kill buffer will contain only the last line. But the setting of **@cl** to kill−line fools MicroEmacs into thinking the last command was a kill command so the last kill line as appended to the current yank buffer, i.e. the kill buffer will have both lines in it.

This feature can be used for any command whose effect depends on the previous command. Such commands include forward−line(2), kill−region(2), reyank(2) and undo(2). This feature should not be abused as unexpected things may happen.

Summary

@cl

Get or set the last command.

@clk

Get or set the last key stroke.

@cc

Get or set the current command.

@cck

Get or set the current keystroke.

@cg

Get a command name from the user.

@cgk

Get a keystroke from the user.

@cq

Get a quoted command name from the user.

@cqk

Get a quoted keystroke from the user. **EXAMPLE**

The following example shows how the **@cc** and **@cl** commands are used:−

```
define−macro current−last−command
    insert−string &spr "Last key [%s] name [%s]\n" @clk @cl
    insert−string &spr "Current key [%s] name [%s]\n" @cck @cc
!emacro
```

Pressing the up key and then executing this macro using execute−named−command (esc x) will insert the lines:−

```
Last key [up] name [backward−line]
Current key [esc x] name [execute−named−command]
```

@cg like **@cq** gets a single character input, however if the keyboard input is bound to a function then the function name is returned instead of the character e.g. if `^F` or `<left−arrow>` is depressed then **forward−char** is returned. This has distinct advantages over **@cq** as the binding becomes device independent and executes on all platforms, additionally it honors the users bindings, however bizarre.

@cq provides a low level character input mechanism, obtaining a single raw character input from the user. This does not interact with the message line and the user is not prompted for input (use ml−write(2) to create your own message). **@cq** is very low level, it is generally preferable to use **@cg** which provides a more intelligent binding.

EXAMPLE

The following example is taken from `draw.emf` which uses **@cg** to obtain cursor movements from the user. Note how the input from **@cg** (stored in variable **%dw−comm**) is compared with the binding name rather than any keyboard characters.

```
    !repeat
        0 screen−update
        !force set−variable #l0 @cg
        !if &seq #l0 "abort−command"
```

```
                !if &iseq @mc1 "Really quit [y/n]? " "nNyY" "y"
                    find-buffer :dw-buf
                    0 delete-buffer "*draw*"
                    -1 buffer-mode "view"
                    !abort
                !endif
            !elif &seq #l0 "newline"
                .
                .
            !elif &seq #l0 "forward-line"
                1 draw-vert
            !elif &seq #l0 "backward-line"
                -1 draw-vert
            !elif &seq #l0 "forward-char"
                1 draw-horz
            !elif &seq #l0 "backward-char"
                -1 draw-horz
            !elif &seq #l0 "osd"
                .osd.draw-help osd
            !elif &set #l1 &sin #l0 "mdeu-="
                !if &les #l1 5
                    set-variable :dw-mode &sub #l1 1
                    set-variable :dw-modes #l0
                    draw-setmode-line
                !elif &sin #l0 "-="
                    set-variable :dw-char #l0
                    draw-setmode-line
                !endif
            !else
                ml-write "[Invalid command]"
            !endif
        !until 0
```

SEE ALSO

@wc(4), &kbind(4), define-macro(2).

@fs(4)

NAME

@fs − Frame store variable

SYNOPSIS

@fs *row column*

DESCRIPTION

The frame store variable **@fs** gives macros a way of obtaining the character currently being drawn on the screen at the given location. If the given value of *row* or *column* is out range, i.e. less than zero or greater than or equal to the screen size (see $frame−width(5)) then the value returned is the empty string (i.e. "").

This variable cannot be set and is only updated during a screen update, this means that macros that change the cursor position will need to redraw the screen before using this variable.

EXAMPLE

The following example gets the word under the current mouse position, this may not be the current cursor position:

```
define−macro word−under−mouse
    set−variable #l0 $mouse−y
    set−variable #l1 $mouse−x
    !if &not &inw @fs #l0 #l1
        ml−write "[mouse not over a word]"
        !return
    !endif
    set−variable #l2 @fs #l0 #l1
    set−variable #l1 &sub #l1 1
    !if &inw @fs #l0 #l1
        set−variable #l2 &cat @fs #l0 #l1 #l2
        !jump −3
    !endif
    set−variable #l1 $mouse−x
    set−variable #l1 &add #l1 1
    !if &inw @fs #l0 #l1
        set−variable #l2 &cat #l2 @fs #l0 #l1
        !jump −3
    !endif
    ml−write &spr "[mouse is over the word \"%s\"]" #l2
!emacro
```

SEE ALSO

$frame−width(5), screen−update(2), MacroArguments, MacroNumericArguments, define−macro(2).

MessageLineVariables(4)

NAME

@mn, @mna, @ml, @mc, @mx, @mxa − Message line input

SYNOPSIS

@mn
@mna
@ml[*f*][*h*] "*prompt*" ["*default*"] ["*initial*"] ["*com−list*"] ["*buffer−name*"]
@mc[*f*] *prompt* [*valid−list*]
@mx "*command−line*"
@mxa "*command−line*"

DESCRIPTION

The **Message Line Variables** provide a method to prompt the user for an input returning the data to the caller. The **@mn** variable cause MicroEmacs to input data from the user in the default way for that command's argument, i.e. the normal prompt with the normal history and completion etc. Similarly **@mna** causes MicroEmacs to input the current argument and any subsequent arguments in the default way.

The **@ml** variable can be used to get a string (or Line) of text from the user using the message−line in a very flexible way. The first optional flag **f** is a bitwise flag where each bit has the following meaning

`0x01`

The *default* value will be specified and this will be returned by default.

`0x02`

The *initial* value will be specified and this will be initial value given on the input line.

`0x04`

Auto−complete using the initial value, usually used with bit `0x02`.

`0x08`

Hide the input string, the characters in the current input string are all displayed as `'*'`s.

If no value is specified then default value is 0 and **h** can not be specified. The *default* value is returned when the user enters an empty string. If the *initial* string is specified the the input buffer will be initialized to the given string instead of and empty one.

The flag **h** specifies what type of data is to be entered, this specifies the history to be used and the semantics allowed, **h** can have the following values

`0`	For a general string input using the general history.
`1`	For an absolute file name, with completion and history.
`2`	For a MicroEmacs '02 buffer name, with completion and history.
`3`	For a MicroEmacs '02 command name, with completion and history.
`4`	For a file name, with completion and history.
`5`	For a search string, with history.
`6`	For a MicroEmacs '02 mode name, with completion and history.
`7`	For a MicroEmacs '02 variable name, with completion and history.
`8`	For a general string using no history.
`9`	For a user supplied completion list (`com−list`).
`a`	For a user supplied completion list (`buffer−name`).

A default value of 0 is used if no value is specified. At first glance type 1 and 4 appear to be the same. They differ only when a non absolute file name is entered, such as "foobar". Type 1 will turn this into an absolute path, i.e. if the current directory is "/tmp" then it will return "/tmp/foobar". Type 4 however will return just "foobar", this is particularly useful with the &find(4) directive to then find the file "foobar".

When a value of `9` is used the argument *com−list* must be given which specifies a list of completion values in the form of a MicroEmacs list (see help on &lget(4) for further information on lists). The user may enter another value which is not in the list, which will be returned.

Alternatively a completion list may be given in the form of a buffer using a value of `a`. The argument *buffer−name* must be given to specify the buffer name from which to extract the completion list; each line of the buffer is taken as a completion value. This option is particularly useful for large completion lists as there is no size restrictions.

The **@mc** variable can be used to get a single character from the user using the message−line. The optional flag **f** is a bitwise flag where each bit has the following meaning

`0x01`

The *valid−list* specifies all valid letters.

`0x02`

Quote the typed character, this allows keys such as '`C−g`' which is bound to the abort command to be entered.

The default value for **f** is 0. When **@mc** is used, the user is prompted, with the given prompt, for a single character. If a *valid−list* is specified then only a specified valid character or an error can be returned. For example, a yes/no prompt can be implemented by the following

```
!if &iseq @mc1 "Are you bored [y/n]? " "yYnN" "y"
```

```
            save-buffers-exit-emacs
        !endif
```

By using the &isequal(4) operator a return of "Y" or "y" will match with "y".

When the **@mx** variable is used MicroEmacs sets the system variable $result(5) to the input prompt, it will then execute the given `command-line`. If this command aborts then so does the calling command, if it succeeds then the input value is taken from the **$result** variable. Similarly **@mxa** causes MicroEmacs to get the current and any subsequent arguments in this way.

These variables are useful when trying to use existing commands in a different way, such as trying to provide a GUI to an existing command. See the **delete−buffer** example below.

EXAMPLE

The following example can be used to prompt the user to save any buffer changes, the use of **@mna** ensures the user will be prompted as usual regardless of the number of buffers changed:

```
        save-some-buffers @mna
```

The following example sets `%language` to a language supplied by the user from a given list, giving the current setting as a default

```
        set-variable %languages "|American|British|French|Spanish|"
        set-variable %language "American"

        set-variable %language @ml19 "Language" %language %languages
```

The following example is taken from `diff-changes` in tools.emf, it uses **@mc** to prompt the user to save the buffer before continuing:−

```
        define-macro diff-changes
            !if &seq $buffer-fname ""
                ml-write "[Current buffer has no file name]"
                !abort
            !endif
            !if &bmod "edit"
                !if &iseq @mc1 "Save buffer first [y/n]? " "nNyY" "y"
                    save-buffer
                !endif
            !endif
                .
                .
```

Note that the input is case insensitive. The following version would not work as the user may expect when the buffer has not been edited:

```
                .
                .
            !if &and &bmod "edit" &iseq @mc1 "Save buffer first [y/n]? " "nNyY" "y"
                save-buffer
                .
```

.

Unlike **C** and other similar languages MicroEmacs macro language always evaluates both **&and** arguments. This means that the user will be prompted to save the buffer regardless of whether the buffer has been edited.

The **@mx** variables are useful when using existing commands in a new environment. For example, consider providing a GUI for the delete−buffer(2) command, when executed the calling GUI may not be aware that changes could be lost or a process may still be active. These variables can be used as a call back mechanism to handle this problem:

```
define-macro osd-delete-buffer-callback
    !if &sin "Discard changes" $result
        2 osd-xdialog "Delete Buffer" "  Dicard changes?  " 2 10 6 "&Yes" "&No"
        set-variable $result &cond &equ $result 1 "y" "n"
    !elif &sin "Kill active process" $result
        2 osd-xdialog "Delete Buffer" "  Kill active process?  " 2 10 6 "&Yes" "&N
        set-variable $result &cond &equ $result 1 "y" "n"
    !else
        1000 ml-write &spr "[Unknown prompt %s]" $result
        !abort
    !endif
!emacro

define-macro osd-delete-buffer
    .
    . set #l0 to buffer name to be deleted
    .
    delete-buffer #l0 @mxa osd-delete-buffer-callback
!emacro
```

SEE ALSO

define−macro(2).

SearchGroups(4)

NAME

@s0, @s1, @s2, ... @s9 − Last search group values

SYNOPSIS

@s0 − Last search's whole match string
@s1 − Last search's first group value
@s2 − Last search's second group value
...
@s9 − Last search's nineth group value

DESCRIPTION

The search group variables **@s***n* return the string matches of the last regular expression search i.e. search−forward(2) (in magic(2m) mode) or regex−forward(3).

@s0 returns the whole of the matched string, **@s***n*, *n* = 1..9, returns the bracket matches corresponding to the group demarkation points indicated by `\(` and `\)` in the search regular expression.

DIAGNOSTICS

An error is generated if an attempt is made to access these variables and the last search failed or the last search did not have the specified group.

The value returned for an unused group, e.g. @s2 for the regex string "`\(a\)\|\(b\)`" if "`a`" was matched, is an empty string ("").

EXAMPLE

The following macro code gives a simple example of their potential use:

```
forward−search "Token *{\\(Start\\|End\\)}"
!if $status
    ml−write "[found \"%s\"]" @s0
    if &seq @s1 "Start"
       .
       .
```

NOTES

Remember that the regular expression escape character '\' has to be duplicated within a macro file as '\' is also the macro file escape sequence.

SEE ALSO

magic(2m), search−forward(2), regex−forward(3).

CurrentBufferVariables(4)

NAME

@wc, @wl − Extract characters from the current buffer

SYNOPSIS

@wl
@wc

DESCRIPTION

Buffer variables are special in that they can only be queried and cannot be set. Buffer variables allow text to be taken from the current buffer and placed into a variable. Two types of extraction are provided **@wl** provides a line extraction method, **@wc** provides a character extraction method.

For example, if the current buffer contains the following text:

```
Richmond
Lafayette
<*>Bloomington                (where <*> is the current point)
Indianapolis
Gary
=* me (BE..) == rigel2 == (c:/data/rigel2.txt) ===================
```

The **@wl** variable allows text from the current buffer to be accessed, a command such as:−

```
set−variable %line @wl
```

would start at the current point in the current 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 set−variable command executes, the string "`Bloomington`" is placed in the variable **%line** and the buffer rigel2 now looks like this:

```
Richmond
Lafayette
Bloomington
<*>Indianapolis               (where <*> is the current point)
Gary
=* me (BE..) == rigel2 == (c:/data/rigel2.txt) ===================
```

The buffer command **@wc** gets the current character in the buffer, it does not change the buffer position. It is important to stress that the cursor position is not modified, in general a macro will interrogate the character under the cursor and then affect the buffer (i.e. by moving the cursor, deleting the character etc.) dependent upon the value of the character returned.

EXAMPLE

The **@wc** variable provides the most useful mechanism to modify the current buffer. The following example is a macro called **super−delete** which is bound to `<CTRL−del>`. The macro deletes characters under the cursor in blocks. If a white space character is under the cursor then all characters up until the next non−white space character are deleted. If a non−white space character is under the cursor then all non−white space characters up until the next white space character are deleted, then the white space is deleted. White space in this context is a `SPACE`, `tab` or `CR` character.

```
;
;---    Macro to delete the white space, or if an a word all of the
;       word until the next word is reached.
;
define-macro super-delete
    !while &not &sin @wc " \t\n"
        forward-delete-char
    !done
    !repeat
        forward-delete-char
    !until &or &seq @wc "" &not &sin @wc " \t\n"
    !return
!emacro

global-bind-key super-delete "C-delete"
```

SEE ALSO

define−macro(2).

@y(4)

NAME

@y − Yank buffer variable

SYNOPSIS

@y − Yank buffer variable

DESCRIPTION

The *Yank Buffer Variable* **@y** retrieves the current yank(2) string from the kill buffer and returns it to the caller.

EXAMPLE

The current contents of the yank buffer can be obtained using **@y**, so to set variable `#l1` to the current or last word if the cursor is not in a word, simply use:

```
forward-char
backward-word
set-mark
forward-word
copy-region
set-variable #l1 @y
```

SEE ALSO

yank(2), MacroArguments, MacroNumericArguments, define−macro(2).

abort−command(2)

NAME

abort−command − Abort command

SYNOPSIS

abort−command (**C−g**)

DESCRIPTION

Aborts the current command, when in trouble, this command will usually limit the damage. If you find yourself in a position where you do not want to be then this command will usually take you back to a sane state. This command rings the bell and stops keyboard macros.

Avoid re−binding this key where possible as it is used in other places.

When **abort−command** is invoked a warning is automatically given alerting the user, this may be an audible or a visual warning depending on the global state of the quiet(2m) mode.

SEE ALSO

buffer−mode(2), quiet(2m).

about(2)

NAME

about − Information About MicroEmacs '02

SYNOPSIS

about

DESCRIPTION

about displays information about the current MicroEmacs '02 editing session and includes the following information:−

- Version number and date information for MicroEmacs '02.
- Global status information including the number of active buffers and global mode status information.
- Current buffer status information; buffer modes and buffer size information.

EXAMPLE

The following is an example output from **about**.

```
MicroEmacs '98 − Date 1/1/98

Global Status:
  # buffers : 21

  Modes on  : auto backup crlf exact magic quiet tab undo
  Modes off : binary cmode crypt ctrlz del dir edit hide indent
              justify letter line lock nact narrow over pipe rbin
              save time usr1 usr2 usr3 usr4 usr5 usr6 usr7 usr8
              view wrap

Current Buffer Status:
  Buffer    : m2cmd148.2
  File name : c:/emacsdoc/m2cmd148.2

  Lines     : Total     34, Current     27
  Characters: Total    759, Current    683

  Modes on  : auto backup edit exact indent justify magic quiet
              tab time undo wrap
  Modes off : binary cmode crlf crypt ctrlz del dir hide letter
              line lock nact narrow over pipe save rbin usr1 usr2
              usr3 usr4 usr5 usr6 usr7 usr8 view
```

SEE ALSO

describe−bindings(2), list−buffers(2).

add−color(2)

NAME

add−color − Create a new color
add−color−scheme − Create a new color scheme

SYNOPSIS

add−color "*col−no*" "*red*" "*green*" "*blue*"
n **add−color−scheme** "*schemeNum*" "*fore*" "*back*" "*current−fore*" "*current−back*"

"*selected−fore*" "*selected−back*"
"*current−selected−fore*" "*current−selected−back*"
["*fm−fore*" "*fm−back*" "*fm−cur−fore*" "*fm−cur−back*"
"*fm−sel−fore*" "*fm−sel−back*"
"*fm−cur−sel−fore*" "*fm−cur−sel−back*"] **DESCRIPTION**

add−color creates a new color and inserts it into MicroEmacs '02 colors table, where *red*, *green* and *blue* are the color components and *col−no* is the MicroEmacs '02 color table number. The color table contains 256 entries indexed by *col−no* in the range 0−255.

On some platforms (DOS and UNIX termcap) the number of colors is physically limited by the hardware to less than 256 (typically 16), in this case all 256 colors can be defined and for each created color the closest system color is used.

By default, only color 0 (white) and 1 (black) are defined. Once created, the colors may be used to create color schemes, this is the sole use of colors.

add−color may be used to modify an existing *col−no* index by re−assignment, the existing color definition is over−written with the new color definition. **add−color−scheme** creates a color scheme entry used by hilight(2), screen−poke(2), osd(2) and variables such as $global−scheme(5), $buffer−scheme(5), $ml−scheme(5).

The command takes an index number "*schemeNum*" and eight color values (defined by **add−color**) alternating between foreground and background colors. The 8 colors represent the 4 color paired states of foreground and background that may appear in a text buffer. The paired states correspond to current and selected lines (or permutations thereof). If an argument *n* is given to the command then *schemeNum* is set to a duplicate of the *n*th scheme, no other arguments are required.

schemeNum is the identifying index that is used to recognize the scheme. By default only two color schemes are defined at initialization, they are a monochrome scheme and inverse scheme with indices 0 and 1 using white as foreground and black as background, selected text is inverted. When defining a color scheme, if an existing *schemeNum* index is used then that scheme is modified.

The next eight arguments must be given, they specify foreground and background color pairs for the four different situations, as follows:−

Default

Color combination used when none of the following three are applicable.

Current

Color combination used when the text is on the same line as the cursor. It is also used by the $mode−line−scheme(5) for the current window's mode line and for the current selection on an osd(2) dialog.

Selected

Color combination used when the text is in the current selected region, but is not on the current line. Also used by **osd** for non−current item Hot keys.

Current−selected

Color combination used when the text is on the current line and in the current selected region. Also used by **osd** for the current item's Hot key.

The following 8 arguments set up fonts and are optional, any missing arguments are defaulted to 0. Each argument is a bitmask indicating which font should be enabled, where each bit is as follows:

`0x01`	Enable bold font.
`0x02`	Enable italic font.
`0x04`	Enable light font.
`0x08`	Enable reverse font.
`0x10`	Enable underlining.

Normally only the foreground value is used, i.e. the first, third, fifth and seventh values. But screen−poke(2) can be used to draw reversed color scheme in which case the background values are used.

EXAMPLE

The color palette is typically created at start−up via the configuration file **scheme*X*.emf**. These files are not easily read as they are automatically generated via the scheme−editor(3) dialog. A more readable form of "`schemed.emf`" would be as follows:−

```
; Standard colors
add−color &set .white     0 200 200 200
add−color &set .black     1 0   0   0
add−color &set .red       2 200 0   0
add−color &set .green     3 0   200 0
add−color &set .yellow    4 200 200 0
add−color &set .blue      5 0   0   200
```

```
add-color &set .magenta   6 200 0   200
add-color &set .cyan      7 0   200 200
; Light colors
add-color &set .lwhite    8 255 255 255
add-color &set .lblack    9 75  75  75
add-color &set .lred     10 255 0   0
add-color &set .lgreen   11 0   255 0
add-color &set .lyellow  12 255 255 0
add-color &set .lblue    13 0   0   255
add-color &set .lmagenta 14 255 0   255
add-color &set .lcyan    15 0   255 255
; Selection color
add-color &set .sel-col  16 91  78  131
; Set the required cursor-color
set-variable $cursor-color .col12
; Set up the standard schemes for the text, mode line message line, scroll bar and
add-color-scheme $global-scheme  .white .black .lwhite .black ...
    ... .white .sel-col .lwhite .sel-col 0 8 1 9 8 0 9 1
add-color-scheme $ml-scheme  .white .black .lwhite .black ...
    ... .white .sel-col .lwhite .sel-col 0 8 1 9 8 0 9 1
add-color-scheme $mode-line-scheme  .white .red .lwhite .lred ...
    ... .white .red .lwhite .red 8 0 9 1 0 8 1 9
add-color-scheme $scroll-bar-scheme .white .lblack .lwhite .lblack ...
    ...  .lblack .white .lblack .lwhite 8 0 9 1 0 8 1 9
    .
    .
```

NOTES

Color schemes can be created and altered using the scheme−editor(3) dialog, the created color scheme can then the used from start−up by using the user−setup(3) dialog. Therefore direct use of these commands is largely redundant.

The existence of a color or scheme index is checked as each entry is submitted, therefore any color or scheme used must have been previously been created, otherwise a default value is substituted.

Changing any existing color definitions causes all references to the color from a scheme to adopt the new color.

Changing any existing color−scheme definitions changes the rendered color of any hilight(2) etc., that was using that color−scheme.

A −ve color scheme value (i.e. −*n*) uses the previous '*n*'th entry that is defined in the color block. i.e. if *current−fore* was specified as −2 then it would inherit the *fore* field color.

Not all UNIX terminals support all the above fonts.

On some telnet packages color is not directly supported and some of the termcap display attributes such as bold and italic are represented by a color (e.g. italic text is shown in green). Using this translation it is possible to achieve reasonable color support on a VT100 terminal − it is a little awkward but is worth while if you have to use this type of connection frequently.

SEE ALSO

scheme−editor(3), user−setup(3), change−font(2), hilight(2), screen−poke(2), $buffer−hilight(5), $cursor−color(5), $global−scheme(5), $trunc−scheme(5), $ml−scheme(5), $osd−scheme(5), $mode−line−scheme(5), $scroll−bar−scheme(5), $system(5).

add−dictionary(2)

NAME

add−dictionary − Declare existence of a spelling dictionary

SYNOPSIS

n **add−dictionary** "*file*"

DESCRIPTION

add−dictionary adds the given dictionary (specified by the given *file*) to the dictionary list. Note that the *file* may omit the **.edf** extension, this is automatically added.

The command accepts a numeric argument '*n*' which determines the actions to be undertaken. When *n* is omitted then the dictionary is marked for loading (on demand) − this is the standard invocation used in the start up files.

If an argument of **0** is given the dictionary is created but it is not marked for loading, this can be used to create an empty dictionary.

If an argument of **−1** is given the contents of the dictionary are dumped into the current buffer, used for dictionary maintenance. The two main uses of this command are discussed below.

Dictionary Loading

A call to **add−dictionary** with no numeric argument does not perform an immediate load of the dictionary, instead the dictionary is only loaded on demand, i.e. when a call to spell(2) (usually via spell−word(3) or spell−buffer(3)) is made, this ensures that the start up time for MicroEmacs does not become too long. When the dictionary is loaded it is checked for efficiency, if found to be inefficient it is automatically optimized and flagged as changed. On exiting MicroEmacs, the user is prompted to save any dictionary that has be altered or optimized.

The spelling search order is made from the last dictionary added to the first, as soon as a word is found in a dictionary the search is halted. This implies that if a word has been defined incorrectly in one dictionary, but correct in another, the order in which the dictionaries are added determines the result.

The number of dictionaries allowed is unlimited but note that any words added are always added to the LAST dictionary. The size of the dictionary is restricted to about 16Mb, the size is NOT tested when words are added and if this size is exceeded the results are undefined. However, it is unlikely that this limit will be reached, the largest dictionary created to date is 0.8Mb.

A new main dictionary may be created as follows:−

1)

Find a file containing an **ispell(1)** compatible list of words.

2)

execute−file(2) spellutl.emf to define macro spell−add−word(3).

3)

Start up MicroEmacs '02 and execute the command **add−dictionary** giving an appropriate new dictionary name.

4)

Load up the file containing the words and execute the command spell−add−word(3) with a very large argument so all the words are added.

5)

Save the dictionary by either executing the command save−dictionary(2) or exiting. **Dictionary Dump**

A call to **add−dictionary** with a numeric argument *n* of −1 causes the contents of the given dictionary to be dumped into the current buffer (make sure you are in an empty buffer or ***scratch***) where:

`xxxx` − Good word `xxxx` with no spell rules allowed
`xxxx/abc` − Good word `xxxx` with spell rules `abc` allowed
`xxxx>yyyy` − Erroneous word with an auto−replace to `yyyy`

The dump of the dictionary may be edited, allowing erroneous entries to be removed. The macro file `spellutl.emf` contains macros edit−dictionary(3) and restore−dictionary(3) which enable the user to edit a dictionary.

NOTES

MicroEmacs '02 is supplied with a dictionaries for American and British English, it is strongly suggested that these dictionaries are **NOT** modified in anyway. Ensure that the dictionary is protected by loading the base dictionaries first, followed by a personal dictionary. New words added during spelling will then be added to the personal dictionary rather than the main dictionary.

EXAMPLE

The MicroEmacs '02 start−up file **me.emf** executes **language.emf** which in turn executes the user language setup file, for example **american.emf**, which adds the main language dictionaries and rules.

language.emf then adds the user's dictionary, this process can be simplified to:−

```
; add the main American dictionary
add−dictionary "lsdmenus"

; reset the spell rules
0 add−spell−rule
; Now add the American spell rules
−2 add−spell−rule "A" "" "" "re" ; As in enter > reenter
−2 add−spell−rule "I" "" "" "in" ; As in disposed > indisposed
    .
    .
; Now add the user dictionary
add−dictionary $MENAME
```

SEE ALSO

add−spell−rule(2), save−dictionary(2), spell−add−word(3), edit−dictionary(3), spell−buffer(3).

add−file−hook(2)

NAME

add−file−hook − Declare file name context dependent configuration

SYNOPSIS

n **add−file−hook** "*extensions*" "*fhook−name*"

DESCRIPTION

add−file−hook defines a macro binding between a file name or file type and a set of macros. This binding enables file type dependent screen highlighting and key bindings to be performed. For a higher level introduction refer to File Hooks.

add−file−hook operates in two different modes to establish the type of file:−

- Content recognition, by examination of the contents of the file.
- File extension recognition.

Content recognition has the highest priority and is used in preference to the file extension.

add−file−hook is called multiple times to add new recognition rules. The rules are interrogated in last−in−first−out (LIFO) order, hence the extension added last has a greater precedence than those added first. This ordering allows default rules to be over−ridden.

Initialization

add−file−hook must be initialized prior to the first call, using an invocation of the form:−

```
0 add−file−hook
```

with a numeric argument *n* of 0, and no arguments. This invocation resets the file hooks by deleting all of the installed hooks.

File Extension Recognition

add−file−hook with no numerical argument *n* allows the extension of a file (or the base file name if there is no extension) to be used to determine which user defined setup macro is to be executed. The *extensions* argument is a space separated list of *file endings* (as opposed to true extensions) and is usually specified with the extension separator. For example, the extension "`.doc`" may indicate that the file is a document and therefore the indent, wrap and justify buffer modes are required. This may be performed automatically by defining a macro which adds these modes and adding a file hook to automatically execute this macro whenever a file "`*.doc`" is loaded.

The command arguments are defined as follows:−

extensions

A space separated list of file extensions, which are to be checked, this list includes the extension separator (typically dot ('`.`'). It should be noted that the extension search is actually a comparison of the tail of the string, as such files such as *makefile*, which do not have an extension, are specified literally.

fhook−name

The name of the file hook to execute. This is the name of the macro to execute that initializes the buffer.

As an example:−

```
define−macro fhook−doc
    1 buffer−mode "indent"
    1 buffer−mode "wrap"
    1 buffer−mode "justify"
!emacro

add−file−hook ".doc" "fhook−doc"
```

It is quite possible that the same macro should be executed for a text file, i.e. "`*.txt`" this is achieved by a single **add−file−hook** as the space (' ') character is used as an extension separator, e.g.

```
add−file−hook ".doc .txt" "fhook−doc"
```

There are three special file hooks, which are **fhook−binary**, **fhook−rbin** and **fhook−default**, these are not predefined, but if the user defines them then they are executed whenever a file is loaded in binary or reduced binary mode (see buffer−mode(2)) or the extension does not match any of those defined.

Considering the `fhook−XXX` prefix, the initial '**f**' character must be present as this is changed to a '**b**' and an '**e**' when looking for the enter (begin) buffer and exit buffer hooks. These hooks are executed whenever the user swaps to or from a buffer (including creating and deleting). So for the given example, if the tab size of 8 is required in a document (but 4 elsewhere) then this operation this is performed by defining the `bhook−XXX` and `ehook−XXX` macros, e.g.:−

```
define−macro bhook−doc
    set−variable $tabsize 8
!emacro

define−macro ehook−doc
    set−variable $tabsize 4
!emacro
```

File hooks are often used to setup the desired *buffer modes*, *hilighting*, *local key bindings*, *abbreviation file*, etc.

Buffer hooks are usually used to set and restore conflicting global variables.

File Content Recognition

add−file−hook with a non−zero numerical argument *n* defines a macro binding between the content in a file and a set of macros. This binding enables file type dependent screen hi−lighting and key binding to be performed. For a full description of file hooks refer to File Hooks, for file extension dependent hooking refer to add−file−hook(2).

The content defined file hooks interrogate the contents of a file on loading and search for a *magic* string identifier embedded in the text which uniquely identifies the file type.

The recognition process performs a search of the first *n* (numerical argument) non−blank lines of the file, searching for the regular expression specified by the *extensions* argument. The sign of the numerical argument *n* is interpreted as follows:−

- **−ve** − Case insensitive search
- **+ve** − Case sensitive search

The command arguments are defined as follows:−

extensions

A regular expression string defining the text to be searched for.

fhook−name

The name of the file hook to execute. This is the name of the macro to execute that will initialize the buffer.

The search commences from the first non−blank line in the file, if the regular expression, defined by *extensions* is located then the file hook *fhook−name* is invoked. This is typically used to identify files which do not have file extensions i.e. UNIX shell script files. To identify a shell script file which commences with:−

```
#!/bin/sh
```

The following file hook is used:−

1 add−file−hook "**#!/.*sh**" "**fhook−shell**"

Note that "`.*sh`" also matches `/bin/csh`, `/usr/local/bin/zsh` etc, so care should be taken to ensure that the regular expression string is sufficiently well specified to recognize the file type.

The second class of embedded text are explicit identifiers embedded into the text. The embedded strings take the form:

`-*-` *mode* `-*`
`-*-` **Mode:** *mode*; ... `-*-`
`-!-` *mode* `-!-`

The `-*-` notation belongs to GNU Emacs, but MicroEmacs '02 recognizes the construct and extracts the string correctly. The `-!-` notation is MicroEmacs '02 specific and is provided so as not to cause conflict with GNU Emacs. MicroEmacs '02 searches for either construct on the first non−blank line of the file.

The explicit strings are defined with a negative numerical argument *n*, which identifies them as **explicit** rather than **magic** text strings. The *string* should be defined in lower case and matches a case insensitive string take from the file. e.g. to define a file hook for a make file:

```
#_____________________________-!-Makefile-!-________________________________
#
# Make file for MicroEmacs using the Microsoft MSCV 2.0/4.0 development kit.
#
# Author      : Jon Green
# Created     :  020197.1002
# Last Edited : <150297.1942>
# File        : makefile.w32
....
```

might be defined as:

−1 **add−file−hook** "−!−[\t]*makefile.*−!−" fhook−make

NOTES

Automatic Macro File Loading

add−file−hook performs an automatic load of a macro file if the **fhook** macro is not present in memory. The file name of the command file containing the macro is automatically derived from the *name* component of the **fhook** macro name. The **fhook−** part of the name is stripped off and prepended with **hk** and suffixed with **.emf**. Hence, macro **fhook−doc** would be searched for in file `hkdoc.emf` within the MicroEmacs '02 directory. The command file is automatically loaded and executed.

In cases where the fhook macro is not located in an equivalent hook file, the file location of the macro may be explicitly defined for auto loading via a define−macro−file(2) invocation.

As an example, consider the C−mode file hook, used to load `.c` files. The loading of a C header file (`.h`) utilizes the same highlighting modes, but it's startup sequence is slightly different when handling new files. In this case the **fhook−cmode** for `.c` and **fhook−hmode** for `.h` files are located in the same hook file namely `hkcmode.emf`.

```
define-macro-file hkcmode fhook-hmode

add-file-hook ".c .cc .cpp .def .l .y .i .ac"   "fhook-cmode"
add-file-hook ".h .hpp"                         "fhook-hmode"
```

In this case the define−macro−file has been used to inform MicroEmacs '02 of the location of the **fhook−hmode** macro thereby overriding the automatic load of a file called **hkhmode.emf**. The **fhook−cmode** macro requires no such definition as it is located in a hook file that matches the mode name, `hkcmode.emf`.

Extending a standard hook definition

The standard file hook files **hk*XXX*.emf** should not be modified. The standard file hooks may be extended with local definitions by defining a file **my*XXX*.emf**, which is an extension to the hook file **hk*XXX*.emf**. This is automatically executed after **hk*XXX*.emf**. Refer to sections Language Templates and File Hooks for details.

File Extensions

The file extensions are specified as a space separated list of file name endings. Back−up file endings such as tilde (~) are not classed as correct file endings and are skipped by the file hook search, hence a file ending "`.c~`" invokes the same hook function as a "`.c`" file. It is therefore not necessary to add the backup and auto−save endings to the file hook definition.

The extension separator, usually dot (`.`), is typically added to the *extensions* list, they may be omitted with effect where a file always ends in the same set of characters. A notable example is "`makefile`" which includes no extension, as such, MicroEmacs '02 applies the same hook function to a file called `Imakefile` as the endings are the same.

Binary Files

It is sometimes useful to associate file types as binary files, so that they are immediately loaded in binary. In this case, both file extension and content recognition methods (i.e. of a magic string) are applicable. In both cases the file is bound to the well known hook `fhook-binary` which automatically loads the file in a binary mode.

Note, that for the content recognition process for a binary hook, the load time is doubled as the file is initially loaded in the default text mode, the binary hook function forces a second load operation in binary.

SUMMARY

add−file−hook is summarized as follows:−

- Binds one or more extensions to a macro called fhook−*xxxx*.
- Extensions are typically specified with the dot (`.`) separator.
- Multiple extensions are specified as a space separated list.
- Binds a regular expression search string to a macro called fhook−*xxxx*.
- The absolute value of the numerical argument determines the number of lines in the file over which the regular expression search is made.

- The sign of the numerical argument determines if the regular expression search is case (in)sensitive.
- When one of the files with a known file extension, or recognized content, is loaded macro **fhook−xxxx** is executed.
- **fhook−xxxx**, if undefined, is automatically searched for in file **hk***xxxx***.emf**.
- When the buffer containing the known file is entered (i.e. gains focus), then entry macro **bhook−xxxx** is executed.
- When the buffer containing the known file is exited (i.e. looses focus), then the exit macro **ehook−xxxx** is executed.

EXAMPLE

The standard set of supported file types by MicroEmacs '02, at the time of writing, is defined as:−

```
; reset the file hook list
0 add−file−hook
; Add file extension hooks.
; Files loaded in binary mode do not need hook as fixed
add−file−hook "*help* *info* .ehf"                          fhook−ehf
add−file−hook "*bindings* *commands* *variables*"           fhook−lists
add−file−hook "*buffers*"                                   fhook−blist
add−file−hook "/ *directory* *files*"                       fhook−dir
add−file−hook "*registry*"                                  fhook−reg
add−file−hook "*icommand* *shell* *gdb* *dbx*"              fhook−ipipe
add−file−hook ".emf"                                        fhook−emf
add−file−hook ".doc .txt"                                   fhook−doc
add−file−hook ".1 .2 .3 .4 .5 .6 .7 .8 .9 .so .tni .sm"     fhook−nroff
add−file−hook ".c .h .def .l .y .i"                         fhook−c
add−file−hook ".cc .cpp .hpp .rc"                           fhook−cpp
add−file−hook "Makefile makefile .mak"                      fhook−make
add−file−hook "Imakefile imakefile"                         fhook−imake
add−file−hook ".sh .ksh .csh .login .cshrc .profile .tcshrc"  fhook−shell
add−file−hook ".bat .btm"                                   fhook−dos
add−file−hook ".man"                                        fhook−man
add−file−hook ".dmn"                                        fhook−dman
add−file−hook ".ini .hpj .reg .rgy"                         fhook−ini
add−file−hook ".htm .html"                                  fhook−html
add−file−hook ".htp .hts"                                   fhook−hts
add−file−hook ".tcl"                                        fhook−tcl
add−file−hook ".rul"                                        fhook−rul
add−file−hook ".awk .nawk .gawk"                            fhook−awk
add−file−hook ".p .pas"                                     fhook−pascal
add−file−hook ".vhdl .vhd"                                  fhook−vhdl
add−file−hook ".fvwm .fvwm2rc"                              fhook−fvwm
add−file−hook ".java .jav"                                  fhook−java
add−file−hook ".nsr"                                        fhook−nsr
add−file−hook ".erf"                                        fhook−erf
; Add magic hooks
 1 add−file−hook "^#!/.*sh"                   fhook−shell ; UNIX shell files
 1 add−file−hook "^#!/.*wish"                 fhook−tcl
 1 add−file−hook "^#!/.*awk"                  fhook−awk
 1 add−file−hook "^#VRML"                     fhook−vrml
−4 add−file−hook "<html>"                     fhook−html
−1 add−file−hook "−[*!]−[ \t]*c.*−[*!]−"             fhook−c     ; −*− C −*−
−1 add−file−hook "−[*!]−[ \t]*c\\+\\+.*−[*!]−"   fhook−cpp   ; −*− C++ −*−
```

```
-1 add-file-hook "-[*!]-[ \t]nroff.*-[*!]-"      fhook-nroff ; -*- nroff -*-
-1 add-file-hook "-!-[ \t]*shell.*-!-"           fhook-shell ; -!- shell -!-
-1 add-file-hook "-!-[ \t]*msdos.*-!-"           fhook-dos   ; -!- msdos -!-
-1 add-file-hook "-!-[ \t]*makefile.*-!-"        fhook-make  ; -!- makefile -!-
-1 add-file-hook "-!-[ \t]*document.*-!-"        fhook-doc   ; -!- document -!-
-1 add-file-hook "-!-[ \t]*fvwm.*-!-"            fhook-fvwm  ; -!- fvwm -!-
-1 add-file-hook "-!-[ \t]*erf.*-!-"             fhook-erf   ; -!- erf -!-
-1 add-file-hook "-!-[ \t]*fold:.*-!-"           fhook-fold  ; -!- fold:... -!-
```

OBSCURE INFORMATION

This section includes some low−level information which is so obscure it is not relevant to the typical user.

Resolving Loading Order Problems

There is a potential loading order problem involving auto−loading of file libraries and the setting up of **bhook** and **ehook**. E.g. if the main fhook function has been defined as a define−macro−file(2), but the bhook or ehooks have not the when a buffer is created as only the fhook is define, only the fhook is set, the rest remain disabled even though the execution of the macro file will define these extra hooks.

To solve this problem simply define the bhook/ehooks as well. Note that automatically loaded hooks do not suffer from this problem as the macro file is executed before the hooks are assigned, thereby ensuring the all the hooks are defined.

SEE ALSO

File Hooks, Language Templates, $buffer−bhook(5), $buffer−ehook(5), $buffer−fhook(5).

global−mode(2)

NAME

global−mode − Change a global buffer mode
add−global−mode − Set a global buffer mode
delete−global−mode − Remove a global buffer mode

SYNOPSIS

n **global−mode** "*mode*" (**esc m**)
add−global−mode "*mode*"
delete−global−mode "*mode*"

DESCRIPTION

global−mode changes the state of one of the hereditary global modes. A buffer's modes are initialized to the global modes when first created. This command is very useful in changing some of the default behavior such as case sensitive searching (see the example below). See Operating Modes for a full list and description of modes. Also see buffer−mode(2) for a full description of the use of the argument *n*.

The about(2) command gives a list of the current global and buffer modes.

add−global−mode and **delete−global−mode** are macros defined in meme3_8.emf which use global−mode to add or remove a global mode. They are defined for backward compatibility with MicroEMACS v3.8 and for ease of use; they are simple macros, add−global−mode is defined as follows:

```
define-macro add-global-mode
    ; Has the require mode been given as an argument, if so add it
    !force 1 global-mode @1
    !if &not $status
        ; No - use 1 global-mode to add a mode
        !nma 1 global-mode
    !endif
!emacro
```

EXAMPLE

The following example globally disables exact(2m) and magic(2m) modes, if these lines are copied to the user setup file then are searches will be simple and case insensitive by default:

```
-1 global-mode "exact"
-1 global-mode "magic"
```

NOTES

Globally adding binary(2m), crypt(2m) and rbin(2m) modes is strongly discouraged as any file loaded would be assigned these modes. Instead use the numeric argument of command find−file(2) or commands find−bfile(3) and find−cfile(3).

auto(2m), autosv(2m), backup(2m), exact(2m), magic(2m), quiet(2m), tab(2m) and undo(2m) modes are present on all platforms by default. On Windows and DOS platforms crlf(2m) is also present and on DOS ctrlz(2m) is also present.

SEE ALSO

Operating Modes, buffer−mode(2), find−bfile(3), find−cfile(3), about(2).

buffer−mode(2)

NAME

buffer−mode − Change a local buffer mode
named−buffer−mode − Change a named buffer mode
add−mode − Set a local buffer mode
delete−mode − Remove a local buffer mode
unmark−buffer − Remove buffer change flag

SYNOPSIS

n **buffer−mode** "*mode*" (**C−x m**)
n **named−buffer−mode** "*buffer−name*" "*mode*"
add−mode "*mode*"
delete−mode "*mode*"
unmark−buffer

DESCRIPTION

buffer−mode changes the state of a given buffer mode, affecting only the current buffer. A buffer's mode affects the behavior of MicroEmacs '02. The about(2) command gives a list of the current global and buffer modes. Refer to Operating Modes for a description of the buffer modes.

The argument *n* when given, has the following meaning:

Delete	Add	toggle	Mode
−1	1	0	Use "mode"
−2	2	130	auto
−3	3	131	autosv
−4	4	132	backup
−5	5	133	binary
−6	6	134	cmode
−7	7	135	crlf
−8	8	136	crypt
−9	9	137	ctrlz
−10	10	138	del
−11	11	139	dir
−12	12	140	edit
−13	13	141	exact
−14	14	142	hide
−15	15	143	indent
−16	16	144	justify
−17	17	145	letter
−18	18	146	line
−19	19	147	lock
−20	10	148	magic
−21	21	149	nact
−22	22	150	narrow

−23	23	151	over
−24	24	152	pipe
−25	25	153	quiet
−26	26	154	rbin
−27	27	155	save
−28	28	156	tab
−29	29	157	time
−30	30	158	undo
−31	31	159	usr1
−32	32	160	usr2
−33	33	161	usr3
−34	34	162	usr4
−35	35	163	usr5
−36	36	164	usr6
−37	37	165	usr7
−38	38	166	usr8
−39	39	167	view
−40	40	168	wrap

Note that when omitted the default argument is 0, i.e. prompt for and toggle a mode.

named−buffer−mode changes the state of a given buffer mode for a given buffer which may not be the current buffer.

add−mode and **delete−mode** are macros which use buffer−mode to add and remove a buffer mode. **unmark−buffer** is also a macro which removes the edit flag from the current buffer. They are defined for backward compatibility with MicroEMACS v3.8 and can be found in meme3_8.emf; add−mode is defined as follows:

```
define−macro add−mode
    ; Has the require mode been given as an argument, if so add it
    !force 1 buffer−mode @1
    !if &not $status
        ; No − use 1 buffer−mode to add a mode
        !nma 1 buffer−mode
    !endif
!emacro
```

NOTES

When a buffer is created it inherits the current global mode state.

SEE ALSO

Operating Modes, global−mode(2), about(2), &bmode(4).

add−next−line(2)

NAME

add−next−line − Define the searching behavior of command output

SYNOPSIS

n **add−next−line** "*buffer−name*" ["*string*"]

DESCRIPTION

add−next−line is used to set up the *next−line* functionality which is used by the get−next−line(2) command. The *next−line* feature is aimed at giving the user easy access to file locations which are stored in another buffer. This buffer may typically be the output from the **grep(1)** command or a compiler (e.g. **cc(1)**) and needs to contain the file name and line number of the required location.

As long as the format of the buffer is consistent and there is a maximum of one location per line, the *next−line* feature can be successfully configured.

The first argument, "*buffer−name*", gives the name the aforementioned buffer, this is "***grep***" for the grep(3) command etc. There is no limit on the number of next−line formats, nor on the number of **add−next−line** strings which are given. While there is no real need to initialize each new type, it is advised that the first **add−next−line** is called with a numerical argument of zero, e.g.:

```
0 add-next-line "*grep*"
add-next-line "*grep*" "....."
```

This tells MicroEmacs to reinitialize the type by freeing off any strings currently stored, note that the "*string*" argument is not used in this case. Resetting the next−line type safe guards against duplicate strings being added to it, a common problem if MicroEmacs is reinitialized.

Following is a typical output from grep:

```
foo.c: 45:      printf("hello world\n") ;
foo.c: 46:      printf("hello again\n") ;
```

If we replace the file name with "`%f`" and the line number with "`%l`", this becomes:

```
%f: %l:      printf("hello world\n") ;
```

get−next−line works on a left to right basis, as soon as it has enough information from the line it does not need to continue. Therefore the previous example can be reduced to just "`%f: %l:`". This is the string argument that should be given for the above example, i.e.:

```
add-next-line "*grep*" "%f: %l:"
```

get−next−line takes the given string and replaces the "%f" with $file−template(5) and the "%l" with the $line−template(5) and then uses the resultant string as a regular expression search string to find the next location. Crudely these could be set to "foo.c" and "45" respectively to find the first example, but this would fail to find any other. As a result the templates are usually magic search strings which will match any file and line number.

Similarly, following is an example output of the **gcc(1)** compiler:

```
basic.c:522: warning: `jj' might be used uninitialized in this command
display.c:833: warning: implicit declaration of function `ScreenPutChar'
```

In this case the **add−next−line** given needs to be:

```
add−next−line "*compile*" "%f:%l:"
```

If a −ve numerical argument is given to **add−next−line** the given 'next−line' is ignored, this can be useful when some warnings are to be ignored. For example a common warning from gcc is given when a variable might be used uninitialized, given as follows:

```
bind.c:578: warning: `ssc' might be used uninitialized in this function
```

These warnings can be ignored using the following:

```
−1 add−next−line "*compile*" ...
    ... "%f:%l: warning: `.*' might be used uninitialized in this function"
```

Some versions of **grep(1)** give the file name first and then the lines on the following lines. This is not a major problem as **get−next−line** remembers the last file name. The only problem occurs when skipping some parts of the list at which point the last file name parsed may not be the current file. Following is an example output of such a **grep** and the setup required:

```
File foo.c:
Line 45:      printf("hello world\n") ;
Line 46:      printf("hello again\n") ;
```

The configuration to locate the lines is defined as:

```
0 add−next−line "*grep*"
add−next−line "*grep*" "File %f:"
add−next−line "*grep*" "Line %l:"
```

NOTES

The reinitialize command format of this command changed in January 2001, the format changed from:

```
add−next−line "*grep*" ""
```

SEE ALSO

$file−template(5), $line−template(5), **cc(1)**, compile(3), get−next−line(2), **grep(1)**, grep(3).

add−spell−rule(2)

NAME

add−spell−rule − Add a new spelling rule to the dictionary

SYNOPSIS

n **add−spell−rule** ["*rule−letter*" "*base−ending*" "*remove*" "*derive−ending*"]

DESCRIPTION

add−spell−rule adds a new spelling rule to the speller. The rules effectively define the prefix and suffix character replacements of words, which is given an alphabetical identifier used within the speller , in conjunction with the language dictionary. The letter conventions are defined by the **Free Software Foundation** GNU **ispell(1)** package.

add−spell−rule is used in the MicroEmacs '02 dictionary initialization files called <*language*>r.emf, e.g. `americar.erf`, `britishr.erf` supplied in the MicroEmacs macros directory.

The command takes a single numeric argument *n* to control the addition of a rule to the speller, as follows:−

`0` **add−spell−rule**

Removes all existing rules and re−initializes. This is, by convention, explicitly called before instantiating a new set of rules.

`−1` **add−spell−rule** "*rule−letter*" "*base−ending*" "" "*deriv−ending*"

`−2` **add−spell−rule** "*rule−letter*" "*base−ending*" "" "*deriv−ending*"

Adds a prefix rule, an argument of −1 indicates that this prefix rule cannot be used with a suffix rule. An argument of −2 indicates it can be matched with any suffix rule which can be used with a prefix rule (e.g. argument of 2).

"*rule−letter*" is any character in the range `A−z` except '_', all rules of the given letter must be a prefix rule of the same type (i.e. same argument). The start of a base word must match the given "*base−ending*" regular expression string for the rule to be applied, the "*remove*" string must be empty for a prefix and the "*deriv−ending*" is the prefix string. Example, for the American language;−

`−2 add−spell−rule "I" "" "" "in"` ; As in disposed > indisposed

A prefix rule of type '`I`' can be applied to any base word which has rule '`I`' enabled, and it prefixes "`in`" to the word.

1 **add−spell−rule** "*rule−letter*" "*base−ending*" "*remove*" "*deriv−ending*"

2 **add−spell−rule** "*rule−letter*" "*base−ending*" "*remove*" "*deriv−ending*"

Add suffix rules. An argument of 1 indicates that this prefix rule cannot be used with a prefix rule. An argument of 2 indicates it can be matched with any prefix rule which can be used with a suffix rule (i.e. argument of −2).

"*rule−letter*" is any character in the range `A−z`, all rules of the given letter must be a suffix rule of the same type (i.e. same argument). The end of a base word must match the given "*base−ending*" regular expression string for the rule to be applied, the "*remove*" string must be a fixed string and the "*deriv−ending*" must also be a fixed string which is appended to the base−word after "*remove*" has been removed. Example, for the American language;−

```
2 add−spell−rule "N" "e" "e" "ion"     ; As in create > creation
2 add−spell−rule "N" "y" "y" "ication" ; As in multiply > multiplication
2 add−spell−rule "N" "[^ey]" "" "en"   ; As in fall > fallen
```

A suffix rule of type '`N`' can be applied to any base word which has rule '`N`' enabled, and it can be used with prefixes, e.g. with rule;−

```
−2 add−spell−rule "A" "" "" "re"       ; As in enter > reenter
```

to derive "*recreation*" from "*create*". A rule which cannot be used with prefixes, i.e.:

```
1 add−spell−rule "V" "e" "e" "ive"     ; As in create > creative
1 add−spell−rule "V" "[^e]" "" "ive"   ; As in prevent > preventive
```

While some prefix words are legal, such as "*recreative*" but some are not, such as "*collect*" where "*recollect*" is correct, so is "*collective*" but "*recollective*" is not.

SPECIAL RULES

Following are special forms of add−spell−rule used for tuning the spell support, note that an argument can not be given:−

add−spell−rule "−" "<*y*|*n*>"

Enables and disables the acceptance of hyphens joining correct words. By default the phrase "`out−of−date`" would be accepted in American even though the phrase does not exist in the American dictionary. This is because the three words making up the phrase are correct and by default hyphens joining words are allowed. Some Latin language such as Spanish do not use this concept so this feature can be disable.

add−spell−rule "*#*" "*score*"

Sets the maximum allowed error score when creating a spelling guess list. When comparing a dictionary word with the user supplied word, **spell** checks for differences, each difference or error is scored in the range of 20 to 27 points, once the maximum allowed score has been exceeded the word is ignored. The default guess error score is 60, allowing for 2 errors.

add−spell−rule "*" "*regex*"

Adds a correct word in the form of a regex if a word being spell checked is completely matched by the **regex** the word is deemed to be correct. For example, the following rule can be used to make the spell−checker allow all hex numbers:

```
add-spell-rule "*" "0[xX][[:xdigit:]]+"
```

This will completely match the words "`0x0`", "`0xff`" etc but not "`0x00z`" as the whole word is not matched, only the first 4 letters.

NOTES

The format of the dictionary is a list of base words with each word having a list of rules which can be applied to that word. Therefore the list of words and the rules used for them are linked e.g.

```
aback
abaft
abandon/DGRS
abandonment/S
abase/DGRS
abasement/S
abash/DGS
abashed/U
abate/DGRS
```

where the "`/...`" is the valid list of rules for that word.

The '_' character is used to separate different rule lists for a single word.

The format of the dictionary word list and the rule list is compatible with **ispell(1)**.

SEE ALSO

add−dictionary(2), spell(2) spell−buffer(3), spell−word(3), **ispell(1)**.

alarm(3)

NAME

alarm − Set an alarm

SYNOPSIS

alarm "*message*" "*hours*" "*minutes*"

DESCRIPTION

alarm creates an alarm for the user which will print the given "*message*" in the given number of "*hours*" and "*minutes*" time from the moment of creation.

The message is printed on the screen using osd(2).

NOTES

alarm is a macro defined in `misc.emf`.

SEE ALSO

osd(2).

append−buffer(2)

NAME

append−buffer − Write contents of buffer to end of named file

SYNOPSIS

n **append−buffer** "*file−name*"

DESCRIPTION

append−buffer is used to write the contents of the current buffer into an EXISTING file. Use save−buffer(2) if the buffer is to over−write the existing file already associated with the buffer. Use write−buffer(2) if the buffer is to be written out to a new file, or to replace an existing file.

append−buffer writes the contents of the current buffer to the named file *file−name*. But unlike write−buffer(2) the action of the write does not change the attributes associated with the file (if it exists), it also does not effect the stats of the current buffer.

On writing the file, append−buffer ignores the time(2m) and backup(2m) mode settings. The current buffer will not be time stamped and a backup will not be created for "*file−name*". If the buffer contains a narrow(2m) it will automatically be removed before saving so that the whole buffer is saved and restored when saving is complete

The argument *n* is a bit based flag, where:−

0x01

Enables validity checks (default). These include a check that the given file already exist, if not confirmation of writing is requested from the user. Without this flag the command will always succeed wherever possible. If "*file−name*" does not exist the buffer is written out in a similar fashion to using the command write−buffer(2).

0x02

Disables the expansion of any narrows (see narrow−buffer(2)) before appending the buffer.

0x04

Truncate the existing file before writing out the contents of the buffer. This means that the file will consist solely of the contents of the buffer, but it will still have the file attributes of the original file.

If *n* is not specified then the default argument of 1 is used.

EXAMPLE

The following example appends the current buffer onto the end of a file, creating the file if it does not exists

```
append−buffer "things_to_do.txt"
```

The following example truncates the users email file while maintaining the file attributes. This is taken from vm(3) where it is used to remove the current mail from the system mail box.

```
find−buffer "*vm−empty−buffer"
−1 buffer−mode "ctrlz"
5 append−buffer %vm−mail−src
delete−buffer $buffer−bname
```

Note that the macro ensures that ctrlz(2m) mode is removed. If it was enabled then the file written would not be empty.

SEE ALSO

write−buffer(2), save−buffer(2).

ascii−time(3)

NAME

ascii−time − Return the current time as a string

SYNOPSIS

ascii−time

DESCRIPTION

ascii−time returns the current time as a formatted string in `#p9` which is equivalent to `#l9` for the calling macro. The format of the time string is:

"*WWW MMM DD hh:mm:ss yyyy*"

Where: `WWW`	− Week day, `Sun` − `Sat`
`MMM`	− Month, `Jan` − `Dec`
`DD`	− Day, `1` − `31`
`hh`	− Hour, `00` − `23`
`mm`	− Minute, `00` − `59`
`ss`	− Second, `00` − `59`
`yyyy`	− Year, `1998`...

EXAMPLE

The following is taken from etfinsrt.emf, it uses **ascii−time** in replacing "`$ASCII_TIME$`" with the current.

```
0 define−macro etfinsrt
    .
    .
    ; Change the create date $ASCII_TIME$.
    beginning−of−buffer
    ; Get ASCII time in #l9
    ascii−time
    !force replace−string "\\$ASCII_TIME\\$" #l9
    .
    .
!emacro
```

NOTES

ascii−time is a macro defined in `utils.emf`.

SEE ALSO

$buffer−fhook(5), &find(4), ascii−time(3).

asm(9)

SYNOPSIS

asm, s − Assembler File

EXTENSIONS

.s − Platform specific assembler file.
.asm − Platform specific assembler file.

DESCRIPTION

The standard assembler file extensions **.s** and **.asm** are by default not bound to any hook functions as they are platform specific. The user should define a default binding for the assembler file types as appropriate to the current platform and assembler development. i.e. for the Windows environment the x86(9) file type would be conditionally bound to the file e.g.

```
!if &seq $platform "win32"
    add-file-hook ".s .asm"             fhook-asmx86
!endif
```

SEE ALSO

x86(9).
File Hooks, Supported File Types.

asn.1(9)

SYNOPSIS

asn.1 − ASN.1 File

FILES

asn1.emf − asn.1 file hook definition
ans1.etf − Template file

EXTENSIONS

asn1 − ASN.1 files.

MAGIC STRINGS

−!− asn.1 −!−

Recognized by MicroEmacs only, defines the file to be a asn.1. **DESCRIPTION**

The **asn1** file type template handles the hilighting of the asn.1 files.

General Editing

On creating a new file, a new header is automatically included into the file. time(2m) is by default enabled, allowing the modification time−stamp to be maintained in the header.

By default, TAB's are enabled as this is the syntactical feature of the file.

Hilighting

The hilighting emphasizes the keywords and comments within the asn.1. **BUGS**

None reported.

SEE ALSO

time(2m).

Supported File Types

auto(2m)

NAME

auto − Automatic source file line type detection

SYNOPSIS

auto Mode

A − mode line letter.

DESCRIPTION

When this mode is enabled the line style of the source file, with respect to `CR/LF/CTRL−Z` characters, are automatically detected and the file (if saved) is written out in the same style as it was read in. This mode is designed to solve the problems of MS−DOS which utilize a '**\r\n**' with every new line and a **ctrl−Z** as a file terminator, conversely UNIX utilizes only '**\n**' as a line terminator.

auto mode therefore allows files to be edited across file system types without corrupting the line style of the native platform.

At load time, if **auto** detects CR/LF style line feeds then it enables the buffer mode crlf(2m), and if a CTRL−Z is found at the end of the file then mode ctrlz(2m) is enabled. Otherwise these modes are cleared.

At write time, if **auto** mode is enabled then the file is written out is a style determined by modes **crlf** and **ctrlz**. For example, if crlf was enabled and ctrlz disabled then the file would be written out with new lines as '**\r\n**' and with no ending ctrl−z.

If auto is not enabled then the file is written out in the style of the current platform, regardless of the current settings on modes **crlf** and **ctrlz**.

SUMMARY

The operation on the modes may be summarized as follows:−

UNIX Systems

- **auto Enabled** UNIX and MS−DOS files may be edited normally, edits are saved in the format read by the system.
- **auto Disabled** UNIX files may be edited normally, files saved as UNIX files. MS−DOS files show a `^M` character at the end of each line (editing is not advised if the purity of the

MS−DOS is to be maintained), any edits are written back as displayed on the screen.

MS−DOS Systems

- **auto Enabled** UNIX and MS−DOS files may be edited normally, edits are saved in the format read by the system.
- **auto Disabled** on reading all files are read and editing may be undertaken normally. On writing, '**\r**'s and a **ctrl−Z** are automatically added. The act of reading a UNIX file and re−writing it translates it to an MS−DOS file.

NOTES

This mode MUST be enabled globally when the file is loaded for the file style to be correctly detected.

It is **strongly advised** that auto mode is permanently enabled.

Windows systems tend to use a '**\r\n**' style line feed but with no trailing ctrl−z.

SEE ALSO

global−mode(2), buffer−mode(2), crlf(2m), ctrlz(2m), $buffer−fmod(5).

auto−spell(3)

NAME

auto−spell − Auto−spell support
auto−spell−buffer − Auto−spell whole buffer
auto−spell−reset − Auto−spell hilight reset
auto−spell−ignore − Auto−spell ignore current word

SYNOPSIS

n **auto−spell**
auto−spell−buffer
auto−spell−reset
n **auto−spell−ignore**

DESCRIPTION

auto−spell enables and disables the auto spell checking of the current buffer. Auto spell detects word breaks as you type and checks the spelling of every completed word hilighting any erroneous words in the error color scheme (usually red).

The argument *n* determines whether auto−spell is enabled or disabled, a +ve argument enables and a −ve argument disables. If no argument or *0* is supplied the auto−spell state is toggled.

auto−spell−buffer checks all words within the current buffer for spell, hilighting any unknown or miss−spelled words found.

auto−spell−reset resets the buffer hilighting scheme, removing any added erroneous words.

auto−spell−ignore gets the current word and deletes the erroneous hilighting and adds the word to the current temporary ignore dictionary, auto−spell and the spelling−checker will now ignore the word. If an argument **n** of 2 is given to the command the word is added to the users personal dictionary instead of the temporary ignore dictionary so the word is 'ignored' in all future sessions of MicroEmacs as well.

NOTES

auto−spell, **auto−spell−buffer**, **auto−spell−reset** and **auto−spell−ignore** are macros defined in `spellaut.emf`.

SEE ALSO

user−setup(3), spell−buffer(3), spell(2).

autosv(2m)

NAME

autosv − Automatic file save

SYNOPSIS

autosv Mode

a − mode line letter.

DESCRIPTION

When this mode is enabled when the buffer is changed it will be automatically saved to a temporary file $auto−time(5) later.

Automatic saving for a buffer will not occur if

The buffer name starts with a '*', this is considered a temporary system buffer.

$auto−time(5) is set to 0, this disables auto−saving for all buffers.

The buffer does not a file name from which to generate a temporary file name. When this occurs the error message:

```
[Auto−writeout failure for buffer xxxxx]
```

MicroEmacs '02 can not write to the generated temporary file name. When this occurs the error message:

```
[Auto−writeout failure for file xxxxx#]
```

On unlimited length file name systems (UNIX), the temporary file naming convention used for file `xxxxx`:

```
xxxxx -> xxxxx#
```

On systems with an `xxxxxxxx.yyy` file name (DOS etc), the following file naming convention is used:

```
xxxxxxxx     -> xxxxxxxx.###
xxxxxxxx.y   -> xxxxxxxx.y##
xxxxxxxx.yy  -> xxxxxxxx.yy#
xxxxxxxx.yyy -> xxxxxxxx.yy#
```

NOTES

This mode MUST be enabled globally when the file is loaded for the file style to be correctly detected.

It is **strongly advised** that autosv mode is permanently enabled.

Auto−save files of URL files (i.e. "`ftp://...`" and "`http://...`") are written to the system's temporary directory. This avoids potentially slow auto−saves. This can however lead to recovery problems as the buffer name must be used to avoid auto−saving conflict with other buffers with the same base file name but different paths.

SEE ALSO

$auto−time(5), backup(2m), find−file(2), ftp(3).

awk(9)

SYNOPSIS

awk − AWK files

FILES

hkawk.emf − AWK hook definition
awk.etf − AWKL template file.

EXTENSIONS

.awk − AWK file
.gawk − GNU AWK file
.nawk − New AWK file

MAGIC STRINGS

#![\t]*/.*awk

MicroEmacs '02 recognises the magic string on the first line of the file used to locate the executable. The awk files may be extensionless and are still recognised. **DESCRIPTION**

The **awk** file type template provides simple hilighting of AWK files, the template provides minimal hilighting.

BUGS

None reported. Template could probably benifit from some form of auto indentation.

SEE ALSO

Supported File Types

Bindings(2)

DEFAULT KEY BINDINGS

The default key bindings are presented below in four alphabetical lists, one for single key bindings and one for each of the 4 bound prefixes (`esc`, `C−x`, `C−h` & `C−c`). See Key Names for a list of valid key names.

Single−Key Sequences

`backspace` backward−delete−char Delete the previous character.
`delete` forward−delete−char Delete character under the cursor.
`down` forward−line Move to next line.
`end` end−of−buffer Move to the end of the buffer.
`esc` prefix 1 Meta character prefix.
`f1` osd Open top main menu.
`home` beginning−of−buffer Move to the start of the buffer.
`insert` buffer−mode Toggle over−write mode.
`left` backward−char Move backward one character (left).
`page−down` scroll−down Move forward by one screen.
`page−up` scroll−up Move backward by one screen.
`return` newline Insert a new line.
`right` forward−char Move forward one character (right).
`tab` tab Insert a tab character.
`up` backward−line Move to previous line.

`S−backspace` backward−delete−char Delete the previous character.
`S−delete` forward−delete−char Delete character under the cursor.
`S−tab` backward−delete−tab Delete white space to previous tab−stop.

`C−a` beginning−of−line Move to beginning of line.
`C−b` backward−char Move backwards by one character
`C−c` prefix Control character prefix.
`C−d` forward−delete−char Delete character under the cursor.
`C−e` end−of−line Move to end of line.
`C−f` forward−char Move forward one character (right).
`C−g` abort−command Abort current command.
`C−h` prefix Control character prefix.
`C−i` insert−tab Insert tab character.
`C−k` kill−line Delete from cursor to the end of the line.
`C−l` recenter Redraw screen with current line in the center.
`C−m` newline Insert a new line.
`C−n` forward−line Move to next line (down).
`C−o` insert−newline Open up a blank line.
`C−p` backward−line Move to previous line (up).
`C−q` quote−char Insert literal character.

C-r isearch−backward Start incremental search backwards.
C-s isearch−forward Start incremental search forwards.
C-t transpose−chars Transpose two letters.
C-u universal−argument Repeat the next command *n* times (default is 4).
C-v scroll−down Move forward by one screen.
C-w kill−region Delete a marked region.
C-x prefix Control character prefix.
C-y yank Restore what was copied or deleted.
C-z scroll−up Move backward by one screen.
C-_ undo Undo the previous edit.
C-down forward−line Move forward five lines.
C-left backward−word Move one word backward.
C-page-down scroll−next−window−down Scroll next window down a page.
C-page-up scroll−next−window−up Scroll the next window up a page.
C-right forward−word Move one word forward.
C-up backward−line Move backward 5 lines.

A-e file−browser Browse the file system.
A-r replace−all−string Replace string with new string in a list of files.
A-down scroll−down Scroll the current window down one line.
A-left scroll−left Scroll the current window left one character.
A-right scroll−right Scroll the current window right one character.
A-up scroll−up Scroll the current window up one line.

esc Prefix Sequences

esc ! pipe−shell−command Pipe a shell command to a buffer.
esc $ spell−word Spell a word.
esc . set−mark Set the start of a region.
esc / execute−file Execute script lines from a file.
esc < beginning−of−buffer Move to the start of the buffer.
esc > end−of−buffer Move to the end of the buffer.
esc ? help Help − high level introduction to MicroEmacs.
esc @ pipe−shell−command Pipe a shell command to a buffer.
esc [backward−paragraph Goto the beginning of the paragraph.
esc \ ipipe−shell−command Incrementally pipe a shell command to a buffer.
esc] forward−paragraph Move forward one paragraph
esc ^ delete−indentation Join 2 lines deleting white spaces.
esc b backward−word Move one word backwards
esc c capitalize−word Capitalize first letter of a word
esc d forward−kill−word Delete word the cursor is on.
esc e set−encryption−key Reset the encryption key.
esc f forward−word Move one word forward.
esc g goto−line Goto a line.
esc i tab Insert a tab character.
esc k global−bind−key Bind a key to a command or macro.
esc l lower−case−word Lowercase word.
esc m global−mode Toggle a global mode.

esc n forward−paragraph Move forward one paragraph
esc o fill−paragraph Reformat (fill) current paragraph.
esc p backward−paragraph Goto the beginning of the paragraph.
esc q ifill−paragraph Reformat (fill) current paragraph.
esc r replace−string Search and replace text (no query).
esc t find−tag Find a tag.
esc u upper−case−word Uppercase word.
esc v scroll−down Move to the previous screen.
esc w copy−region Copy region to the kill buffer.
esc x execute−named−command Execute the named command.
esc y reyank Kill current yank data and restore previous kill buffer data.
esc z quick−exit Save all buffers and exit.

esc ~ buffer−mode Remove edited status from current buffer.
esc backspace backward−kill−word Delete the word under the cursor.
esc esc expand−abbrev Expand an abbreviation.
esc space set−mark Set the start of a region.

esc C−c count−words Count words in a region.
esc C−f goto−matching−fence Reposition the cursor at an opposing bracket.
esc C−g abort−command Abort current command.
esc C−i tab Insert tab character.
esc C−k global−unbind−key Unbind a key from a command or macro
esc C−n change−buffer−name Rename current buffer.
esc C−r query−replace−string Search and replace with query.
esc C−v scroll−next−window−down Scroll next window down a page.
esc C−w kill−paragraph Delete current paragraph.
esc C−z scroll−next−window−up Scroll the next window up a page.

esc A−r query−replace−all−string Query replace string in a list of files.

C−x Prefix Sequences

C−x # filter−buffer Filter the buffer through a shell filter.
C−x (start−kbd−macro Start recording a keyboard macro.
C−x) end−kbd−macro Stop recording a keyboard macro.
C−x / isearch−forward Start incremental search forwards.
C−x 0 delete−window Delete the current window.
C−x 1 delete−other−windows Delete other windows.
C−x 2 split−window−vertically Split the current window into two.
C−x 3 next−window−find−buffer Find a buffer into the next window, split if necessary.
C−x 4 next−window−find−file Load a file into the next window, split if necessary.
C−x 5 split−window−horizontally Split the current window horizontally into two.
C−x 9 find−bfile Find and load a file for binary editing.
C−x < scroll−left Scroll the window left by one screen width.
C−x = buffer−info Show cursor position information
C−x > scroll−right Scroll the window right by one screen width.
C−x ? describe−key Describe binding of command to key.

`C-x @` pipe−shell−command Pipe a shell command to buffer.
`C-x [` scroll−up Move backward by one screen.
`C-x ]` scroll−down Move forward by one screen.
`C-x ^` grow−window−vertically Enlarge the current window by a line.
`` C-x ` `` get−next−line Find the next command line.
`C-x a` goto−alpha−mark Move the cursor to an alphabetic mark.
`C-x b` find−buffer Switch window to a buffer.
`C-x c` shell Start a new command processor.
`C-x e` execute−kbd−macro Execute a macro.
`C-x h` hunt−forward Continue search in forward direction.
`C-x k` delete−buffer Delete buffer.
`C-x m` buffer−mode Toggle a local buffer mode.
`C-x n` change−file−name Rename current buffer file name.
`C-x o` next−window Move to the next window.
`C-x p` previous−window Move to the previous window.
`C-x q` kbd−macro−query Query keyboard macro.
`C-x r` search−backward Search in a reverse direction.
`C-x s` search−forward Search in a forward direction.
`C-x u` undo Undo the previous edit.
`C-x v` set−variable Assign a new value to a variable.
`C-x w` resize−window−vertically Resize the window.
`C-x x` next−buffer Switch to the next buffer.
`C-x z` grow−window−vertically Enlarge the current window.
`C-x {` shrink−window−horizontally Shrink current window horizontally.
`C-x }` grow−window−horizontally Enlarge current window horizontally.

`C-x C-a` set−alpha−mark Mark the current position with an alphabetic mark.
`C-x C-b` list−buffers Display buffer list.
`C-x C-c` save−buffer−exit−emacs Exit MicroEmacs '02.
`C-x C-d` change−directory Change the current working directory.
`C-x C-e` execute−kbd−macro Execute a macro.
`C-x C-f` find−file Find a file and load into buffer.
`C-x C-g` abort−command Abort current command.
`C-x C-h` hunt−backward Resume search in backwards direction.
`C-x C-i` insert−file Insert file into the current buffer.
`C-x C-l` lower−case−region Lowercase region.
`C-x C-n` scroll−down Scroll the current window down one line.
`C-x C-o` delete−blank−lines Delete blank lines about the cursor.
`C-x C-p` scroll−up Scroll the current window up one line.
`C-x C-q` rcs−file Interact with RCS to check in/out a file.
`C-x C-r` read−file Read a file from disk.
`C-x C-s` save−buffer Save current file to disk.
`C-x C-t` transpose−lines Swap adjacent lines.
`C-x C-u` upper−case−region Uppercase region.
`C-x C-v` view−file Read a file for viewing (read only).
`C-x C-w` write−buffer Write a file to disk witn new name.
`C-x C-x` exchange−point−and−mark Exchange cursor with mark position.
`C-x C-y` insert−file−name Insert filename into current buffer.
`C-x C-z` shrink−window−vertically Reduce size of current window.

C−h Prefix Sequences

`C−h a`	command−apropos	List commands involving a concept.
`C−h b`	describe−bindings	Show current command/key binding.
`C−h c`	list−commands	List available commands.
`C−h d`	describe−variable	Describe current setting of a variable.
`C−h k`	describe−key	Describe keyboard binding.
`C−h v`	list−variables	List defined variables.
`C−h C−c`	help−command	Display command help information.
`C−h C−i`	help−item	Display item help information.
`C−h C−v`	help−variable	Display variable help information.

Variables(4)

NAME

Variables − Macro variables

SYNOPSIS

#tn
$*variableName*
%*variableName*
.*variableName*
.*commandName***.***variableName*
:*variableName*
:*bufferName***:***variableName*

DESCRIPTION

Variables are part of MicroEmacs macro language and may be used wherever an argument is required. The variable space comprises:−

− Register Variable
$ − System Variable
% − Global Variable
. − Command Variable
: − Buffer Variable

All variables hold string information, the interpretation of the string (numeric, string or boolean) is determined when the variable is used within the context of the command. There are five types of variable, **Register Variables** (prefixed with a hash **#**), **System Variables** (prefixed with a dollar **$**), **Global Variables** (prefixed with a percentage **%**), **Buffer Variables** (prefixed with a colon **:**) and **Command Variables** (prefixed with a period **.**).

### Register Variables

Register Variables provide a set of 10 prefixed global (**#g0** .. **#g9**), parent (**#p0** .. **#p9**) and local (**#l0** .. **#l9**) register variables. The interpreted decode time of the register variables is significantly smaller than other variable types as no name space search is performed.

Register variables are assigned using set−variable(2), their value may be queried with describe−variable(2), unlike Global Buffer or Command variables they cannot be deleted.

Register variables are implemented like a stack, where the global registers are the top of the stack and every executing macro gets its own set of resister variables (**#l?**). The macro also has access to the global registers (**#g?**) and its calling, or parent macro (**#p?**). If the macro has no parent macro then the global registers are also the parent registers. Outside macros, i.e. using **set−variable** manually, the global parent and local registers are the same.

Register variables are typically used for retaining short term state, computation steps etc. As with the User Variables, the global register variables are global and care must be taken with nested macro invocations to ensure that the register usage does not conflict.

System Variables

MicroEmacs defines many System variables which are used to configure many aspects of the editors environment. The functionality of each system variable has been documented, they can be set and described but cannot be unset. If the user attempts to set or describe a non−existent MicroEmacs system variable (e.g. **$PATH**) the system environment is used instead, allowing the user to query and alter the system environment.

Global, Command and Buffer Variables

The Global variables are denoted by an initial **%** character followed by the name of the variable *variableName*. The *variableName* may be any ASCII character string up to 127 characters in length, all characters of the name are significant. Shorter names are preferred as this speeds up execution. Global Variables exist in a global context which all macros have access to.

Command variables exist within the scope of a command, they are denoted by the period (**.**) character. They can be accessed by one of two forms, either **.***variableName* or **.***commandName***.***variableName*. The first form, without the command name, assumes the scope to be the current command, as such may only be used to access internal variables. The second form qualifies the scope by specifying the command, this form is much more versatile and may be used to access any command variable from any other command, e.g.

```
define−macro foo
    set−variable .foo "Hello world"
    1000 ml−write &cat "foo1: " .foo
    1000 ml−write &cat "foo2: " .foo.foo
!emacro
define−macro bar
    foo
    1000 ml−write &cat "bar1: " .foo
    1000 ml−write &cat "bar2: " .foo.foo
!emacro

bar
```

When **bar** is executed the following messages may be observed:−

```
foo1: Hello World
foo2: Hello World
bar1: ERROR
bar2: Hello World
```

When a macro file or buffer is executed, they are executed within their own scope so local scope command variables (form 1) may be created and used in that scope. Any such variables created are automatically deleted at the end of execution. For example, the default color scheme generator macro file, `schemed.emf`, creates command variables for the created colors to aid readability:−

```
add−color &set .green     3 0   200 0
a0dd−color &set .lgreen   11 0   255 0

...

add−color−scheme .scheme.cardback   .lgreen   .green   .lgreen ...
```

The variables only exist as a file or buffer is being executed, they are not accessible by another command once the command or buffer execution has finished.

Buffer variables are similar to Command variable in function and behaviour except that their scope is of a buffer and are denoted by the colon (**:**) character. Access can be in one of two forms, either **:***variableName* where the scope is assumed to be the current buffer or **:***bufferName***:***variableName*, where the scope is explicitly given allowing access to any buffer variable, e.g.

```
find−buffer "foo"
set−variable :foo "Hello world"
find−buffer "bar"
set−variable :bar "Hello world"
1000 ml−write &cat ":foo     " :foo
1000 ml−write &cat ":foo:foo " :foo:foo
1000 ml−write &cat ":bar     " :bar
1000 ml−write &cat ":bar:bar " :bar:bar
```

When the above is executed the following messages may be observed:−

```
:foo     ERROR
:foo:foo Hello World
:bar     Hello World
:bar:bar Hello World
```

Global, Buffer and Command variables are automatically defined when they are used. A variable is assigned with set−variable(2) and may be subsequently deleted with unset−variable(2). The current assignment of a variable may be queried from the command line using describe−variable(2). e.g.

```
define−macro foo
!emacro
set−variable %foo "Some string"
set−variable :bar "Some string"
set−variable .foo.bar "Some string"

...

ml−write &spr "%s %s %s" %foo :bar .foo.bar

...

unset−variable :bar
unset−variable %foo
unset−variable .foo.bar
```

An undefined variable returns the string `ERROR`, this known state is used to advantage with the hilighting initialization, e.g.

```
!if &sequal .hilight.c "ERROR"
    set-variable .hilight.c &pinc .hilight.next 1
!endif
;
; Hi-light C Mode
;
0 hilight .hilight.c  2 50             $global-scheme
```

In this case the variable **.hilight.c** is explicitly tested for definition, if it is undefined then it is assigned a new value.

Conventionally, names are separated with a minus sign character (`-`) e.g. `foo-bar`. It is strongly advised that the name space is kept reasonably clean, since there are no restrictions on the number of macros that may be defined, problems will arise if different macros use the same variables in different contexts. Where possible, Command or Buffer Variables are preferable to Global Variables since they have no side effects on other macros or buffers. It is advised that all variable names associated with a particular macro set are prefixed with short identifier to make the variable name space unique. e.g. the **Metris** macro prefixes all variables with **:met-**; the **draw** macro uses **:dw-**, the **patience** macro **:pat-** etc.

Macro writers should endeavor to use the minimal number of variables, obviously the more variables that exist in the system, the greater the lookup time to find a variable. Use Register Variables in preference to Command, Global or Buffer variables for intimidate computation steps, temporary state etc.

Note that Buffer Variables are automatically deleted when the buffer is deleted.

EXAMPLE

The following example is the macro to convert tabs to spaces, it is shown in two forms, with User Variables and with Register Variables, the register variable implementation is obviously preferable since no new variables have been defined.

User Variable Implementation

```
;
; tabs-to-spaces.
; Convert all of the tabs to spaces.
define-macro tabs-to-spaces
    set-variable %curline $window-line        ; Remember line
    beginning-of-buffer
    !force search-forward "\t"
    !while $status
        3 drop-history
        set-variable %curcol $window-acol
        backward-delete-char
        &sub %curcol $window-acol insert-space
```

```
                !force search−forward "\t"
            !done
            3 drop−history
            goto−line %curline
            update−screen
            ml−write "Converted tabs!"
        !emacro
```

Register Variable Implementation

```
        ;
        ; tabs−to−spaces.
        ; Convert all of the tabs to spaces.
        define−macro tabs−to−spaces
            ; Remember line
            set−variable #l0 $window−line
            beginning−of−buffer
            !force search−forward "\t"
            !while $status
                set−variable #l1 $window−acol
                backward−delete−char
                &sub #l1 $window−acol insert−space
                !force search−forward "\t"
            !done
            goto−line #l0
            screen−update
            ml−write "[Converted tabs]"
        !emacro
```

SEE ALSO

@wc(4), define−macro(2), describe−variable(2), set−variable(2), unset−variable(2).

Build(2)

BUILD

MicroEmacs '02 may be compiled from the source files using the command shell build scripts *build* (UNIX Bourne Shell) or *build.bat* (DOS/Windows). A default compile sequence may be achieved with a simple:

```
build
```

from the command line. The build script attempts to detect the host system and available compiler and build the editor.

The build script recognizes the following options:−

−C

Build clean. Delete all of the object files.

−d

Build a debugging version, the output is `med` (or `med32` for 32−bit Windows versions).

−h

Display a simple help page

−l *logfile*

Redirect all compilation output to the *logfile*, this may not work on DOS or Windows systems.

−la *logfile*

Append all compilation output to the end of *logfile*, this may not work on DOS or Windows systems.

−m *makefile*

Build using the specified makefile. over−riding the auto system detect. The supplied makefiles include:−

- `aix43.mak` IBM AIX 4.3 native
- `cygwin.gmk` Cygwin using GNU tools under Windows.
- `dosdj1.mak` Microsoft DOS build using djgpp version 1.
- `dosdj2.mak` Microsoft DOS build using djgpp version 2.
- `freebsd.gmk` Free BSD using GNU tools.
- `hpux9.gmk` HP−UX 9.x using GNU tools.
- `hpux9.mak` HP−UX 9.x native
- `hpux10.gmk` HP−UX 10.x using GNU tools.

- `hpux10.mak` HP−UX 10.x native
- `hpux11.gmk` HP−UX 11.x using GNU tools.
- `hpux11.mak` HP−UX 11.x native
- `irix5.gmk` Silicon Graphics IRIX 5.x using GNU tools
- `irix5.mak` Silicon Graphics IRIX 5.x native
- `irix6.gmk` Silicon Graphics IRIX 6.x using GNU tools
- `irix6.mak` Silicon Graphics IRIX 6.x native
- `linux2.gmk` Linux 2.x using GNU tools
- `openstep.mak` Openstep 4.2 on NeXTstep (BSD 4.3).
- `sunos55.gmak` Sun Solaris 5.5 using GNU tools
- `sunos55.mak` Sun Solaris 5.5 native
- `sunos56.gmak` Sun Solaris 5.6 using GNU tools
- `sunos56.mak` Sun Solaris 5.6 native
- `sunosx86.gmk` Sun Solarais 2.6 (Intel) using GNU tools.
- `win32bc.mak` Borland C, 32−bit Windows version.
- `win32b55.mak` Borland C 5.5, 32−bit Windows version (Free compiler).
- `win32sv2.mak` Microsoft Developer v2.x, Win32s (for Win 3.xx)
- `win32sv4.mak` Microsoft Developer v4.2, Win32s (for Win 3.xx)
- `win32v2.mak` Microsoft Developer v2.x, 32−bit Windows.
- `win32v5.mak` Microsoft Developer v5.x, 32−bit Windows.
- `win32v6.mak` Microsoft Developer v6.x, 32−bit Windows.

−ne

Build NanoEmacs (a cut down version aimed as a vi replacement), the output is `ne` (or `ned32` for 32−bit Windows versions).

−S

Build spotless. Deletes all of the object files and any backup files, tag files etc.

−t *type*

Set the build type, where *type* can be one of the following:

- `c` Build a console only version (i.e. no window support), the output is `mec` (or `mec32` on Windows).
- `w` Build a windows only version (i.e. no console support), the output is `mew` (or `mew32` on Windows).
- `cw` Build a version which supports both console and windows, the output is `mecw` (or `mecw32` on Windows).

−u

Build a URL version (Windows '95/'98/NT only), constructs the executable `meu32.exe`. **Makefiles**

The supplied makefiles are provided in two forms:−

- **.gmk** − GNU make, using gcc.
- **.mak** − Native make, consistent with the compiler and platform.

The makefiles are supplied with the following targets:−

- **all** − Default build.
- **clean** − Removes intermediate files.
- **spotless** − Removes intermediate files and any backup files.
- **med** − Builds a debugging version.
- **men** − Builds console version (Windows only).
- **men** − Builds a URL version (Windows only).
- **menu** − Builds console and URL version (Windows only).

NOTES

Other UNIX ports should be fairly easy from the base set of ported platforms. If any new platform ports are performed by individuals then please submit the makefiles and any source changes back to JASSPA − see Contact Information.

backup(2m)

NAME

backup − Automatic file backup of last edit

SYNOPSIS

backup Mode

B − mode line letter.

DESCRIPTION

backup mode, when enabled, performs an automatic backup of the last edit when a save file operation is performed. The backup file name is derived from the base file name and is written into the same directory as the source file, the backup file name can be obtained from the variable $buffer−backup(5).

On unlimited file name length systems the naming convention used depends on bit **0x400** of variable $system(5), if this bit is set then the system simulates a DOS style 8.3 file naming convention. If this bit is clear then variable $kept−versions(5) can be used to create multiple backup files.

Where an existing backup file already exists, then the old backup file is removed and replaced by the newer backup file. The naming convention for backup files is defined as follows:−

> On unlimited length file name systems (UNIX and some Windows systems) with a single backup file, the following file naming conventions are used for file `xxxxx`:
>
> ```
> xxxxx -> xxxxx~
> ```
>
> On unlimited length file name systems with multiple backup files, the following file naming conventions are used for file `xxxxx`:
>
> ```
> xxxxx -> xxxxx.~?~
> ```
>
> Where `?` is the backup number, the most recent backup file is always "`.~0~`".
>
> On systems with an `xxxxxxxx.yyy` file name (MS−DOS etc), the following file naming conventions are used:
>
> ```
> xxxxxxxx      -> xxxxxxxx.~~~
> xxxxxxxx.yyy  -> xxxxxxxx.yy~
> xxxxxxxx.yyyy -> xxxxxxxx.yyy~
> ```

The environment variables $MEBACKUPPATH(5) and $MEBACKUPSUB(5) can be used to modify this behaviour.

NOTES

backup is enabled by default.

Reference should also be made to undo(2) which allows previous edits to be removed. Also see $auto−time(5) and autosv(2m) which allows a timed backup of a running edit to be periodically performed.

The user is warned to be extra careful if files ending in '~' or '#'s are used, it is advisable to disable backup creation (see global−mode(2)) and auto−saving (`$auto−time = 0`). The author denies all responsibility (yet again) for any loss of data! Please be careful.

SEE ALSO

$buffer−backup(5), $system(5), $kept−versions(5), $MEBACKUPPATH(5), $MEBACKUPSUB(5), buffer−mode(2), global−mode(2), undo(2m), autosv(2m), $auto−time(5).

forward−char(2)

NAME

forward−char − Move the cursor right backward−char − Move the cursor left

SYNOPSIS

n **forward−char** (**C−f**)
n **backward−char** (**C−b**)

DESCRIPTION

backward−char moves the cursor *n* characters to the left. Move to the end of the previous line if the cursor was at the beginning of the current line.

forward−char moves the cursor *n* characters to the right. Move to the beginning of the next line if the cursor was already at the end of the current line.

NOTES

backward−char is also bound to **left**.
forward−char is also bound to **right**.

SEE ALSO

forward−line(2), backward−line(2).

forward−delete−char(2)

NAME

forward−delete−char − Delete next character at the cursor position
backward−delete−char − Delete previous character at the cursor position

SYNOPSIS

n **forward−delete−char** (**C−d**)
n **backward−delete−char** (**backspace**)

DESCRIPTION

forward−delete−char deletes the next *n* characters from the current cursor position. If the cursor is at the end of a line, the next line is joined on the end of the current line. If an argument is given or letter(2m) mode is enabled then the character is added to the kill buffer, otherwise the kill buffer is unaltered.

backward−delete−char deletes the next *n* characters immediately to the left of the cursor (e.g. more conventionally backspace). If the cursor is at the beginning of a line, this will join the current line on the end of the previous one. If an argument is given or letter mode is enabled then the character is added to the kill buffer, otherwise the kill buffer is unaltered.

NOTES

forward−delete−char is also bound to **delete** and **S−delete**.

backward−delete−char is also bound to **S−backspace**.

SEE ALSO

backward−kill−word(2), forward−kill−word(2), letter(2m).

backward−delete−tab(2)

NAME

backward−delete−tab − Delete white space to previous tab−stop

SYNOPSIS

backward−delete−tab (**S−tab**)

DESCRIPTION

backward−delete−tab deletes all white characters left of the cursor back to the previous tab stop or non−white space, the deleted text is not added to the kill buffer.

SEE ALSO

tab(2), $tabsize(5), $tabwidth(5).

forward−kill−word(2)

NAME

forward−kill−word − Delete next word at the cursor position
backward−kill−word − Delete previous word at the cursor position

SYNOPSIS

n **forward−kill−word** (**esc d**)
n **backward−kill−word** (**esc backspace**)

DESCRIPTION

forward−kill−word deletes the next *n* words starting at the current cursor position, the deleted text is added to the kill buffer. See forward−word(2) for a description of word boundaries. If the argument *n* is `0` the command has no effect. If a −ve argument is specified, +*n* words are deleted and the text is not added to the kill buffer.

backward−kill−word deletes the previous *n* words before the cursor, the deleted text is added to the kill buffer. The numeric argument has the same effect as with **forward−kill−word**.

NOTES

backward−kill−word is also bound to **esc backspace**.

The −ve argument is typically used from macro scripts where the kill buffer is more precisely controlled.

SEE ALSO

backward−delete−char(2), forward−delete−char(2), forward−word(2), yank(2).

forward−line(2)

NAME

forward−line − Move the cursor to the next line
backward−line − Move the cursor to the previous line

SYNOPSIS

n **forward−line** (**C−n**)
n **backward−line** (**C−p**)

DESCRIPTION

forward−line moves the cursor down *n* lines, default 1. If the line is not on the current screen then display the next page and move to the line.

backward−line moves the cursor up *n* lines, if the line is not on the current screen then display the previous page and move to the line.

For both invocations a negative value reverses the sense of movement as expected.

SEE ALSO

backward−word(2), forward−word(2), scroll−down(2), scroll−up(2).

forward−paragraph(2)

NAME

forward−paragraph − Move the cursor to the next paragraph
backward−paragraph − Move the cursor to the previous paragraph

SYNOPSIS

n **forward−paragraph** (**esc n**)
n **backward−paragraph** (**esc p**)

DESCRIPTION

forward−paragraph puts the cursor at the end of the *n*th paragraph after the cursor, default is 1.

backward−paragraph puts the cursor at the beginning of the *n*th paragraph before the cursor, default is 1.

DIAGNOSTICS

The following errors can be generated, in each case the command returns a FALSE status:

[end of buffer]

When moving forwards, the given argument *n* was greater that the number of remaining paragraphs, the cursor is left at the end of the buffer.

[top of buffer]

When moving backwards, the given argument *n* was greater than the number of paragraphs before the cursor, the cursor is left at the beginning of the buffer. **NOTES**

- For both invocations a negative value reverses the sense of movement as expected.
- A paragraph break is defined as a blank line.

SEE ALSO

backward−line(2), forward−line(2), scroll−down(2), scroll−up(2).

forward−word(2)

NAME

forward−word − Move the cursor to the next word
backward−word − Move the cursor to the previous word

SYNOPSIS

n **forward−word** (**esc f**)
n **backward−word** (**esc b**)

DESCRIPTION

forward−word places the cursor at the end of the *n*th word from the current position; the default is 1.

backward−word places the cursor at the beginning of the *n*th previous word, default 1.

NOTES

Words are distinguished by non−alphanumeric characters and need not be white space such as spaces and tabs.

A character is considered to be part of a word if it is in the $buffer−mask(5) character set. The default setting for **$buffer−mask** is "`luh`" which gives a word character set of the alphanumeric characters, i.e. `0−9`, `A−Z`, `a−z`, this may be changed by setting the **$buffer−mask** variable. The character sets (including 4 user character sets `1−4`) may be altered by using the command set−char−mask(2).

SEE ALSO

backward−line(2), backward−paragraph(2), forward−line(2), forward−paragraph(2), Locale Support, $buffer−mask(5), set−char−mask(2).

vb(9)

SYNOPSIS

bas, cls − Visual Basic file

FILES

hkvb.emf − Visual Basic macro file.

EXTENSIONS

.bas, **.cls**

MAGIC STRINGS

−!− msvb −!−

DESCRIPTION

The **Visual Basic** template performs the hilighting of Visual Basic files. The file type is recognised by the standard extension, or by the inclusion of the magic string.

Hilighting

The hilighting features allows components of the language to be differentiated and rendered in different colors.

Auto Layout

The indentation mechanism is enabled which performs automatic layout of the text. restyle−region(3) and restyle−buffer(3) are available to reformat (re−layout) selected sections of the buffer, or the whole buffer, respectively. The default indentation setting is 2. **SEE ALSO**

Supported File Types

bat(9)

SYNOPSIS

bat, btm − MS−DOS batch files

FILES

hkdos.emf − MS−DOS hook definition

EXTENSIONS

.bat − MS−DOS Batch file
.btm − 4−DOS Batch file

MAGIC STRINGS

−!− msdos −!−

Recognized by MicroEmacs only. Denotes a MS−DOS batch file. **DESCRIPTION**

The **dos** file type templates provide simple hilighting of a MS−DOS batch file. The template provides minimal hilighting support of both standard and 4−DOS batch files.

The *Magic String* may be used within the **config.sys** file to force hilighting of the MS−DOS configuration file.

BUGS

None reported.

SEE ALSO

ini(9).

Supported File Types

beginning−of−buffer(2)

NAME

beginning−of−buffer − Move to beginning of buffer/file end−of−buffer − Move to beginning/end of buffer/file

SYNOPSIS

beginning−of−buffer (**esc** <)
end−of−buffer (**esc** >)

DESCRIPTION

beginning−of−buffer places the cursor at the beginning of the buffer/file.

end−of−buffer places the cursor at the end of the buffer/file.

NOTES

beginning−of−buffer is typically bound to **home**.
end−of−buffer is typically bound to **end**.

SEE ALSO

beginning−of−line(2), end−of−line(2).

beginning−of−line(2)

NAME

beginning−of−line − Move to beginning of line
end−of−line − Move to end of line

SYNOPSIS

beginning−of−line (**C−a**)
end−of−line (**C−e**)

DESCRIPTION

beginning−of−line places the cursor at the beginning of the line.

end−of−line places the cursor at the end of the line.

SEE ALSO

beginning−of−buffer(2), end−of−buffer(2).

benchmrk(3f)

NAME

benchmrk − Benchmark MicroEmacs macro processor speed

SYNOPSIS

me "@benchmrk"

DESCRIPTION

The start−up file `benchmrk.emf` may be invoked from the command line to time the macro processor variable manipulation times.

This macro suite was developed to optimize the macro processor performance, and allows comparable analysis of various macro variable manipulations. The macro is not important in it's own right and is not likely to be useful. Running it will provide an in−site into the speed of the macro language and should indicate to the user what are good and bad things to be doing.

As an aside, as MicroEmacs interprets the macro code the it is important that the processing operates at a reasonable speed. Most extensions offering additional functionality are likely to be added to MicroEmacs by way of a macro implementation − this allows speedy development of new features. Obviously core changes do occur when we find that we cannot support certain new requirements, or when the macro code becomes too convoluted. In these cases, new commands are added to help us solve the problem. However, recent evolution of the code has indicated that the core set is now reasonably complete.

SEE ALSO

start−up(3).

binary(2m)

NAME

binary − Binary editor mode

SYNOPSIS

binary Mode

b − mode line letter.

DESCRIPTION

binary mode is enabled when a file is edited in binary mode. The mode is automatically enabled when a file is loaded as a binary file via find−bfile(3).

When a file is loaded using binary mode, every 16 bytes is converted into a line of text giving the hex address of the current position in the file, the bytes as hexadecimal numbers and all printable characters at the end of the line (all non−printable characters are displayed as a '`.`'). However, This format makes it memory hungry in that every 16 bytes of the file requires a 78 character line (approximately 5 times more memory!). For a more memory efficient binary mode see rbin(2m).

When writing out a file which has binary mode enabled the format of each line must have the binary mode format which is as follows:

```
<address>: XX XX XX XX XX .... XX XX | <text-form>
```

Only the hex values (the `XX`'s) between the starting ':' marker and the terminating '|' character are used, the *<address>* and *<text−form>* are ignored.

EXAMPLE

Given a single line MSDOS file:−

```
Live long and prosper.
```

When loaded in using **binary** mode the following 2 line buffer will be produced:−

```
00000000: 4C 69 76 65 20 6C 6F 6E 67 20 61 6E 64 20 70 72  |  Live long and pr
00000010: 6F 73 70 65 72 2E 0D 0A 1A                       |  osper....
```

Note the "`0D 0A 1A`" at the end, this is due to MSDOS's "`\n\r`" carriage returns and `^Z` file termination, these characters are unprintable and are shown as '`.`' at the end of the line.

When saving a binary file, only the text between the '`:`' and '`|`' is considered and it may contain any number of hexadecimal numbers. The given file could be made UNIX compatible by editing the buffer to:−

```
00000000: 4C 69 76 65 20 6C 6F 6E 67 20 61 6E 64 20 70 72  |  Live long and pr
00000010: 6F 73 70 65 72 2E 0D                          |  osper....
```

If the word "**long**" is required to be removed, the following would not work:−

```
00000000: 4C 69 76 65 20 6C 6F 6E 67 20 61 6E 64 20 70 72  |  Live and pr
00000010: 6F 73 70 65 72 2E 0D 0A 1A                       |  osper....
```

The ASCII end letters are ignored, instead the following operation must be performed which removes the characters from the binary representation:−

```
00000000: 4C 69 76 65 20 61 6E 64 20 70 72  |  Live long and pr
00000010: 6F 73 70 65 72 2E 0D 0A 1A                       |  osper....
```

One could be nasty by doing the following:−

```
00000000: 4C 69 76 65 20 73 68 6F 72 74 20 61 6E 64 20 |
00000010: 64 6F 6E 27 74 20 70 72 6F 73 70 65 72 2E 0D 0A 1A           |
```

("Live short and don't prosper").

NOTES

rbin and **binary** modes are mutually exclusive, i.e. they cannot both be enabled at the same time.

SEE ALSO

find−bfile(3), find−file(2), rbin(2m).

bnf(9)

SYNOPSIS

bnf − Backus−Naur Form

FILES

hkbnf.emf − Backus−Naur Form hook definition

EXTENSIONS

.bnf − Backus−Naur Form file

DESCRIPTION

The **bnf** file type template provides simple hilighting of text presented in Backus−Naur Form. The hilighting allows the components of the BNF to be differentiated quickly.

The syntactical tokens that are hilighted are:−

<*[a−zA−Z].**>

Component language identifiers.

| { } ::=

Meta symbols of the BNF format. **BUGS**

None reported.

SEE ALSO

Supported File Types

global−abbrev−file(2)

NAME

global−abbrev−file, buffer−abbrev−file − Set abbreviation file(s).

SYNOPSIS

n **global−abbrev−file** "*abbrev−file*"
n **buffer−abbrev−file** "*abbrev−file*"

DESCRIPTION

The abbreviation files allow the user to define a set of short−cut expansion text, whereby a short sequence of chararacters are associated with a longer text segment. When the short sequence is entered, the user may elect to maually expand the sequnce with the associated replacement text. Provision for cursor positioning may be made in the replacement text.

buffer−abbrev−file sets the current buffer's abbreviation file (limit of one abbreviation file per buffer). **buffer−abbrev−file** does the minimal amount of work to increase speed at load−up. The first use of expand−abbrev(2) attempts to load the abbreviation file at which point errors may be reported.

An argument *n* of zero, forces the buffer abbreviation file to be uncached, such that the next abbreviation that is expanded forces a re−load of the abbreviation file. This is typically only used when an abbreviation file is being constructed and tested.

global−abbrev−file assigns a global set of abbreviations accross ALL buffers, such that the abbreviation is available regardless of the current buffer type. The global abbreviation file has a lower presidence than the **buffer−abbrev−file**, hence the currently assigned **buffer−abbrev−file** is searched before the **global−abbrev−file**.

Similarly for **global−abbrev−file**, an argument of zero forces the global abbreviation file to be uncached and re−loaded on the next use.

An abbreviation is a string which is expanded to an alternate form, e.g.

e.g. −> **for example**

or

PI −> **3.1415926536**
etc.

An abbreviation file is an ordinary text file with a strict format, it is loaded only once at the first call to expand−abbrev(2), from then on it reminds buffered. An abbreviation file has an abbreviation per line, they cannot use multiple lines. This is not a draw back as the expansion string is executed using execute−string(2) so any MicroEmacs '02 command may also be called.

For example the following expansion string inserts the string "`!continue`" and a newline:−

```
"!abort\r"
```

Note that '`\r`' is used instead of '`\n`' as **C−m** is bound to newline(2) and not **C−j**. The expansion string can also make use of a few useful abbreviations:−

\p

Mark the current position (expanded to "`C−x C−a P`")

\P

Move cursor to the marked position (expanded to "`C−x a P`")

See help on execute−string(2) for more useful abbreviations.

EXAMPLE

The abbreviation must be on the left hand side followed by at least 1 space, the expansion string must then be on the same line in quotes. So for the given examples, the abbreviation file would be:

```
|
|e.g. "for example"
|PI   "3.1415926536"
|
```

The following abbreviation could be used for a C *if−else* statement.

```
|
|if "if(\p)\r{\r\r}\relse\r{\r\r}\r\P"
|
```

This is particularly useful for email address, e.g.

```
|
|JA "\"JASSPA\" <support@jasspa.com>"
|
```

The following example is MicroEmacs '02 C−Mode abbreviation file for constructing C files. Remember **\p** is where the cursor is positioned following the expansion.

```
#i "#include <\p>\r\P"
#d "#define "
if "if(\p)\r{\r\r}\r\P"
ef "else if(\p)\r{\r\r}\r\P"
el "else\r{\r\p\r}\r\P"
wh "while(\p)\r{\r\r}\r\P"
```

```
        sw "switch(\p)\r{\rcase :\rdefault:\r}\r\P"
```

NOTES

Abbreviation files are given the extension **.eaf** in the MicroEmacs '02 home directory.

One of the easiest ways to create more complex abbreviations is to record a keyboard macro, name it and then insert the resultant macro. See notes on commands start−kbd−macro(2), name−kbd−macro(2) and insert−macro(2).

Try to avoid using named key, such as "`up`" and "`return`", as the keyboard macro equivalent is not readable and is likely to change in future releases.

FILES

c.eaf − C−Mode abbreviation file. **emf.eaf** − Macro code abbreviation file.

SEE ALSO

execute−string(2), expand−abbrev(2), insert−macro(2), iso−accents−mode(3), name−kbd−macro(2), start−kbd−macro(2), eaf(8).

buffer−bind−key(2)

NAME

buffer−bind−key − Create local key binding for current buffer
buffer−unbind−key − Remove local key binding for current buffer

SYNOPSIS

n **buffer−bind−key** "*command*" "*key*"
n **buffer−unbind−key** "*key*"

DESCRIPTION

buffer−bind−key creates a key binding local to the current buffer, binding the command *command* to the keyboard input *key*. This command is particularly useful in conjunction with file loading hooks (see add−file−hook(2)) allowing local key bindings dependent upon the context of the buffer.

The message line input is not effected by the current buffers local bindings.

buffer−unbind−key unbinds a user created local key binding, this command effects only the current buffer. If a −ve argument is given to **buffer−unbind−key** then all the current buffer's bindings are removed.

NOTES

The prefix commands cannot be rebound with this command.

Key response time linearly increases with each local binding added.

SEE ALSO

global−bind−key(2), ml−bind−key(2), osd−bind−key(2), global−unbind−key(2).

buffer−help(3)

NAME

buffer−help − Displays help page for current buffer

SYNOPSIS

buffer−help

DESCRIPTION

buffer−help opens a dialog giving the user a brief help page on tools available for the current buffer. The help page changes depending on the type of the current buffer.

SEE ALSO

buffer−setup(3).

buffer−info(2)

NAME

buffer−info − Status information on current buffer position

SYNOPSIS

buffer−info (**C−x =**)

DESCRIPTION

buffer−info reports on the current and total lines and characters of the current buffer. It also gives the hexadecimal code of the character currently under the cursor.

The output of the command is displayed on the message line e.g.

```
Line 1845/3955 Col 0.0 Char 78267/167172 (46%) Win Line 99/48 Col/0/0 char = 0xA
```

$result(5) is set to the same output string.

SEE ALSO

$result(5), $mode−line(5), about(2).

buffer−setup(3)

NAME

buffer−setup − Configures the current buffer settings

SYNOPSIS

buffer−setup

DESCRIPTION

buffer−setup provides a dialog interface to configuring the setup of the current buffer's file type within MicroEmacs. **user−setup** may be invoked from the main *help* menu or directly from the command line using execute−named−command(2).

The changes made to a configuration in **buffer−setup** are maintained in future MicroEmacs sessions by storing them within the user's setup registry file, "*<logname>*`.erf`". Note that not all file types may be supported by **buffer−setup**, if not the help menu item will not be available.

The contents of the dialog change, depending on the features the current buffer's file type supports. These features are implemented and installed within the buffer's file hook. The following buttons are always present at the bottom of the dialog:

`Save`

Saves the changes made to the configuration back to the users registry file, i.e. "*<Log−Name>*`.erf`" but does not re−initialize the current buffer. No changes made will effect the current buffer unless the **Current** button is pressed. Buffers of the same type created after the save may inherrit some of the changes.

`Current`

Makes the current buffer reflect the changes made, dismissing the **buffer−setup** dialog. This also performs the above '**Save**' operation. Some changes such as dialog creation changes, will only take effect when MicroEmacs is restarted.

`Exit`

Quits buffer−setup, if changes where not **Save**d or made **Current** they will be lost.

Following is a list of configurable features which may be available:

Create Help Page

Enables/disables the creation of a help page dialog for the tools available for the current file type.

Create Tools Menu

Enables/disables the creation of a file type specific sub menu located within the main menu's **Tools** sub−menu.

Use Author Mode

For file types which have an automatic formatter/viewer (currently only html) enabling this will simply load the file enabling the source code to be viewed and edited. When disabled files of this type will be automatically processed giving a more readable 'formatted' representation.

Insert New Template

When creating a new buffer/file of this type, a default template will be inserted if this is enabled. When disabled the buffer will remain empty.

Fence Display

Enables or disables the displaying of matching fences for this file type. Note that the way in which the matching fence is display is determined by the **Fence Display** option on the Platform page of user−setup(3); the **buffer−setup** option is ignored if this option is set to "`Never Display`".

Setup Hilighting

Creates and enables the token hilighting for the current file type.

Setup Auto Indent

Enables automatic formating (indenting) for the current file type. The indentation rules are either the built in 'C' indentation cmode(2m) or created using the indent(2) command. When enabled the tab(2m) is still adhered to, but the indent(2m) mode is ignored; when disabled the indent mode can be used.

Setup Auto Spell

Enables the setting up of auto−spell(3). When enabled the auto−spell key bindings are created and auto−spell is enabled if enabled within the user−setup dialog.

Setup Folding

Enables the setting up of section folding, when enabled the folding key bindings are created.

Add Abbreviations

Adds the file type's abbreviation file to the buffer using buffer−abbrev−file(2)

Search Modes: Exact

Enables/disables the exact(2m) mode over−riding the setting within the user−setup(3) dialog. If this setting is changed the setting within user−setup will be ignored for the current file type.

Search Modes: Magic

Enables/disables the magic(2m) mode over−riding the setting within the user−setup(3) dialog. If this setting is changed the setting within user−setup will be ignored for the current file type.

Buffer Modes: Auto

Enables/disables the auto(2m) mode.

Buffer Modes: Backup

Enables/disables the backup(2m) mode.

Buffer Modes: Indent

Enables/disables the indent(2m) mode.

Buffer Modes: Justify

Enables/disables the justify(2m) mode.

Buffer Modes: Tab

Enables/disables the tab(2m) mode over−riding the setting within the user−setup(3) dialog. If this setting is changed the setting within user−setup will be ignored for the current file type.

Buffer Modes: Time

Enables/disables the time(2m) mode.

Buffer Modes: Undo

Enables/disables the undo(2m) mode over−riding the setting within the user−setup(3) dialog. If this setting is changed the setting within user−setup will be ignored for the current file type.

Buffer Modes: Wrap

Enables/disables the wrap(2m) mode. **NOTES**

buffer−setup is a macro using osd(2), defined in `buffstp.emf`.

SEE ALSO

buffer−help(3), user−setup(3), File Hooks.

Client−Server(2)

CLIENT−SERVER

This sections describes how MicroEmacs '02 may be interfaced to external components through the **Client−Server** interface.

The **Client−Server** interface of MicroEmacs '02 provides a capability for other applications to inject commands into a running version of the editor, which are interpreted and executed. The interface is only available on multi−tasking operating systems such as UNIX and Microsoft Windows; it is not available on MS−DOS systems.

Within the following discussions, the **Server** is a running version of the MicroEmacs '02 editor; the **client** is the application (or shell script) that communicates a new command to the *server*.

The **Client−Server** interface may provide a bidirectional interface such that a *client* may submit a command to the *server* and may also retrieve a response to that command.

DESCRIPTION

The **Client−Server** interface operates by making an external interface available which is continually monitored by the *server*. The external interface may be provided by a file, named pipe or socket (depending upon the platform) with a well know location in the file system. Typically two files are provided, an input file into which the *client* writes commands (*$TEMP*/**me***$MENAME***.cmd**); and an output file where responses to those commands my be read (*$TEMP*/**me***$MENAME***.rsp**).

Within MicroEmacs, the client server interface appears as a hidden ipipe−shell−command(2) buffer, with the name `*server*`. Commands are received through this buffer and responses are written back to the buffer.

Client Commands

Clients may write directly to the *command* through the use of explicit embedded code, or may use a me(1) invocation with the **−m** option. Commands to the client interface take the form "**C:**<*client*>**:**<*command*>".

<*client*>

<*client*> is an identification string that may be used to identify the client, this information may be used when the command is handled to interpret the command if some special client specific action is required.

<*command*>

The <*command*> is an editor command (or macro) of the given name with any arguments.

Standard command escape sequences must be adhered to. i.e. to write "`Hello World`" on the message line then a client may issue the command:−

```
me −m "C:<client>:ml−write \"Hello world\"
```

The *client−server* interface is typically used to load a file, this may be performed as follows:−

```
me −m "C:<client>:find−file \"/path/foo.bar\""
```

The absolute path is specified in this type of transaction as the current working directory of the active MicroEmacs session is unknown. The **−m** option de−iconize's the existing editor session and bring it to the foreground.

Client Responses

Responses from *client* commands are written to the response file, responses take a similar form to *client* commands except they are prefixed by an **R**, i.e. "**R:**<*client*>**:**<*data*>".

As multiple *clients* may be utilizing the *client−server* mechanism then the <*client*> sting passed in the command is typically returned in the response to allow the *client* to identify it's own response (rather than any other *clients*. It is the *clients* responsibility that this string is unique in order that it may be differentiated.

The returned <*data*> format is undefined and would be generated by a macro command used to handle the *client* command; sufficient to say that the data should exist on a single line.

Server Side

On the *server* side, the **Client−Server** interface is managed like an ipipe−shell−command(2) using the hidden buffer `*server*` (as previously mentioned).

The *Client−Server* interface is enabled from the user−setup(3) interface, the user setting of the interface is confirmed by checking bit `0x20000` of the $system(5) variable.

The client server interface is typically initialized within the `me.emf` initialization file, whereby the *ipipe* input handler is bound to the client pipe buffer and the buffer is hidden, so it is not available when the buffers are swapped. (Note that the client buffer may be explicitly interrogated using find−buffer `*server*`). The client handler is installed as follows:−

```
; Setup the Client Server
!if &band $system 0x20000
    define−macro−file meserver server−input
    find−buffer "*server*"
    set−variable :last−line 2
    set−variable :client−list ":"
    set−variable $buffer−ipipe server−input
    beginning−of−buffer
    goto−alpha−mark  "I"
    −1 find−buffer "*server*"
    !endif
```

This binds a MicroEmacs macro called *server−input* to handle the client commands as they arrive on the input, an alpha−mark is used to record the processed position at the end of the buffer. The pipe handler itself decodes the client request and executes it. The default handler supplied with MicroEmacs '02 is defined within the macro file `meserver.emf`

Responses to the client are inserted into the response file by writing directly into the ipipe buffer (`*server*`) using the ipipe−write(2) command. It is the calling macros responsibility to ensure that the response string adheres to the format outlined above in the previous sections.

NOTES

It is not possible to kill the `*server*` buffer, and ipipe−kill(2) is ignored within the context of the buffer.

FILES

`meserver.emf` − Default Client−Server ipipe handler.
*$TEMP*/**me***$MENAME***.cmd** − Command file.
*$TEMP*/**me***$MENAME***.rsp** − Response file.

BUGS

The first MicroEmacs '02 session that executes becomes the editor server, additional editor sessions that are executed do not become server processes. In the event that the *server* editor is terminated, any other sessions do not take over the role of server. Subsequently issuing a client command may fail, or invoke a new editor session which adopts the role of server.

SEE ALSO

me(1), ipipe−shell−command(2)

CompanyProfiles(2)

COMPANY PROFILES

This section describes how a company profile should be incorporated into MicroEmacs '02. A company profile defines a set of extensions to MicroEmacs which encapsulate settings which are used on a company wide basis. This type of configuration is typically used with a networked (shared) installation. The company profile would typically include:−

- Name of the company.
- Standard header files including company copyright statements.
- Standard file layouts
- Company defined language extensions.

Location Of The Company Information

It is suggested that all of the company extensions applied to MicroEmacs '02 are performed in a separate directory location which shadows the MicroEmacs standard macro file directory. This enables the original files to be sourced if a user does not want to include the company files. This method also allows MicroEmacs to be updated in the future, whilst retaining the company files. For our example, we shall use a company called **JASSPA**, you should replace references to *jasspa* with your own company name. The steps involved are laid out as follows:−

Create a new company directory

You may skip this step if you are going to modify the standard installation.

Create a new directory to hold the company information. i.e.

`/usr/local/microemacs/jasspa` − UNIX
`c:\Program Files\JASSPA\MicroEmacs\jasspa` − Microsoft

Modify the $MEPATH(5) of the (of all users) to include the company directory on the search path i.e.

UNIX

Users edit their local `$MEPATH` or a base $MEPATH is added to the system `.login` or `.profile` scripts.

```
MEPATH=/usr/local/microemacs
MEPATH=/usr/local/microemacs/jasspa:$MEPATH
```

Microsoft Windows Platforms

Edit the `me32.ini` file and modify the `mepath` entry to reflect the location of the company directory:−

```
mepath=C:\Prog....\Mic...\macros\jasspa;C:\Prog...\Mic...\
```

DOS Platforms

Edit the **autoexec.bat** file and modify `MEPATH` to include the company directory location.

```
SET MEPATH=c:\me\jasspa;c:\me
```

Content Of The Company Information

Company macro file

The company file is typically called by the company name (i.e. `jasspa.emf`) create a new company file. The file includes your company name and hook functions for any new file types that have been defined for the company, an example company file for **Jasspa** might be defined as:−

```
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;
;  Author        : Jasspa
;  Created       : Thu Jul 24 09:44:49 1997
;  Last Modified : <190698.2225>
;
;  Description     Extensions for Jasspa
;
;  Notes
;
;  History
;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
; Define the name of the company.
set-variable %company-name "Jasspa"
; Add Jasspa specific file hooks
; Make-up foo file hook
add-file-hook ".foo"    fhook-foo
1 add-file-hook "-!-[ \t]*foobar.*-!-" fhook-foo ; -!- foobar -!-
; Override the make with localised build command
set-variable %compile-com "build"
```

The file contains company specific file hooks and the name of the company.

Other Company Files

Files defined on behalf of the company are included in the company directory. These would include:−

· Template header files etf(8).
· Hook file definitions (**hk***XXX***.emf**) for company specific files, see

add−file−hook(2).

- Extensions to the standard hook definitions (**my*XXX*.emf**) for company specific language extensions to the standard hook files. See File Hooks and File Language Templates.

SEE ALSO

$MENAME(5), $MEPATH(5), File Hooks, File Language Templates, Installation, user−setup(3), User Profiles.

Compatibility(2)

COMPATIBILITY

JASSPA MicroEmacs is based on the original version of **MicroEMACS** produced by Danial Lawrence at revision 3.8, the source files were obtained in approximately 1990. The exact origin of the files is unknown. In that period of time the source files have undergone an awful lot of change, without reference to the subsequent releases made of MicroEMACS by Danial Lawrence (due to no network access). As a result the JASSPA version of **MicroEmacs** does not include any modifications or features that may have been implemented since. This version of **MicroEmacs** has been tailored to suite the requirements of a small group of individuals who have used the editor on a daily basis across a limited number of platforms, for a variety of very different tasks and operating requirements.

This version of MicroEmacs is biased towards UNIX environments, MS−DOS and Microsoft Windows ports have been performed however they are heavily influenced by UNIX and inherit UNIX characteristics wherever possible. The intention is that programmers, and alike, may move across platforms using a common editor environment without being frustrated by the idiosyncrasies of different platforms. The most noticeable platform is the Microsoft Windows platform which mimics the X−Windows cut and paste mechanism within the MicroEmacs environment. If you want a Windows style environment then use **Notepad(1)** or **Wordpad(1)**, this editor is not for you !!

The gross changes to **MicroEmacs '02** are summarized as follows:−

- Macro language interpreter re−written allowing an unlimited number of named macros to be supported. The macro implementation allows new commands to be created by the user, as opposed to continually extending the underlying command set. The named macros are transparent to the user, appearing as built in commands on the command line. Macro command set significantly increased. Support for global, buffer and register variables within the macro language.
- Display drivers re−written providing color hilighting support on most platforms. A macro interface allows information to be written directly to the display canvas allowing the screen to be annotated with additional transient information.
- Support for X−Window screen type in UNIX environments. Microsoft Window's environments (3.x, '95, NT) treated in the same was as X−Windows − this may be unorthodox for existing Window's users, UNIX users will find it more comfortable.
- Introduction of integrated spell checker. Support includes correction word guessing, word auto−correction and double word detection. Ignore and personal dictionaries supported.
- Horizontal window splitting.
- Introduction of scroll bars on all platforms that support a mouse. The scroll bar implementation is platform independent.
- Command and file completion available on all platforms. Most commands support a command history allowing previous command invocations to be recalled.
- Session history file kept, allowing the previous edit session to be reinstated.
- Undo capability, allows previous edits to be undone when mistakes are made.
- Backup capability, Includes a periodic timed backup while an editing session is in progress. The timed backup is automatically recovered by the next session in situations where the system (or editor) crashes.

- A regular expression incremental search becomes the default search forward mechanism.
- Support for abbreviation files allowing frequently used constructs to be automatically expanded.
- Automatic time stamping of files, allowing the edit time to be automatically maintained in the source file(s).
- Introduction of an electric 'C' mode. Editor intelligently handles the layout of 'C' files (under user control).
- Improved documentation text mode providing left/right/center and both justification methods with inclusion for bullet points. Automatic justification may be continually performed as text is entered, thereby maintaining the paragraph in the correct format.
- Integrated on−line help facilities. All commands are documented on−line. New macros may be documented within the macro files and become part of the help system.
- File type determination system, based on either the file name or embedded file text allows file type specific macros (hooks) to be applied, thereby configuring the editor into the correct mode for the file type.
- Introduction of special MicroEmacs search path allowing all of the standard configuration files to be utilized from a shared directory.

The name space of JASSPA MicroEmacs differs from the original MicroEMACS and has become more compliant with the GNU implementation of Emacs. A list of the original MicroEMACS verses the new command name set is as follows, executing the compatibility macro file `meme3_8.emf` will create macro versions of these commands:

add−global−mode => global−mode
add−mode => buffer−mode
apropos => command−apropos
backward−character => backward−char
begin−macro => start−kbd−macro
beginning−of−file => beginning−of−buffer
bind−to−key => global−bind−key
buffer−position => buffer−info
case−region−lower => lower−case−region
case−region−upper => upper−case−region
case−word−capitalize => capitalize−word
case−word−lower => lower−case−word
case−word−upper => upper−case−word
change−screen−depth => change−frame−depth
change−screen−width => change−frame−width
clear−message−line => ml−clear
ctlx−prefix => prefix 2
delete−global−mode => global−mode
delete−mode => buffer−mode
delete−next−character => forward−delete−char
delete−next−word => forward−kill−word
delete−previous−character => backward−delete−char
delete−previous−word => backward−kill−word
end−macro => end−kbd−macro
end−of−file => end−of−buffer
execute−command−line => execute−line

execute−macro => execute−kbd−macro
execute−macro−# => *Deleted*
file−name−insert => insert−file−name
forward−character => forward−char
grow−window => grow−window−horizontally
handle−tab => tab
i−shell => shell
incremental−search => isearch−forward
kill−to−end−of−line => kill−line
meta−prefix => prefix 1
move−window−down => scroll−down
move−window−up => scroll−up
name−buffer => change−buffer−name
next−line => forward−line
next−page => scroll−down
next−paragraph => forward−paragraph
next−word => forward−word
open−line => insert−newline
pipe−command => pipe−shell−command
previous−line => backward−line
previous−page => scroll−up
previous−paragraph => backward−paragraph
previous−word => backward−word
quote−character => quote−char
redraw−display => recenter
restore−window => goto−position
reverse−incremental−search => isearch−backward
save−file => save−buffer
save−window => set−position
scroll−next−down => scroll−next−window−down
scroll−next−up => scroll−next−window−up
search−reverse => search−backward
select−buffer => find−buffer
set => set−variable
shrink−window => shrink−window−vertically
split−current−window => split−window−vertically
top−bottom−switch => *Deleted*
transpose−characters => transpose−chars
unbind−key => global−unbind−key
update−screen => screen−update
write−message => ml−write

c(9)

SYNOPSIS

C, C++ − C and C++ programming language templates

FILES

hkc.emf − C programming language hook definition
hkcpp.emf − C++ programming language hook definition

c.etf − C programming language template file
h.etf − C programming language header template file
cpp.etf − C++ programming language template file
hpp.etf − C++ programming language header template file

EXTENSIONS

.c, **.h**, **.def** − ANSI C
.cpp, **.cc**, **.hpp**, **.rc** **.C** *(UNIX only)* − C++ programming language
.l − LEX
.y − YACC
.i − C (or C++) pre−processed file (i.e. output from pre−processor).
.rc − Microsoft Developer resource file.

MAGIC STRINGS

−*− c −*−

Recognized by GNU and MicroEmacs. Denotes a 'C' programming type file, may be used in **.c**, **.def** and **.h** files.

−*− c++ −*−

Recognized by GNU and MicroEmacs. Denotes a C++ programming type file, may be used in **.c**, **.def** and **.h** files. **DESCRIPTION**

The **C** and **C++** file type templates offer the most sophisticated editing features within the MicroEmacs '02 environment.

General Editing

On creating a new file, a new header is automatically included into the file. time(2m) is by default enabled, allowing the modification time−stamp to be maintained in the header.

Hilighting

The hilighting features allow commands, variables, logical, preprocessor definitions, comments, strings and characters of the language to be differentiated and rendered in different colors.

Auto Layout

The C−Mode cmode(2m) performs automatic layout of the text, variables such as c−brace(5) allow the brace position and text formation to be modified.

> restyle−region(3) and restyle−buffer(3) are available to reformat (re−layout) selected sections of the buffer, or the whole buffer, respectively.
>
> Comments may be formatted using `esc o`, which reformats the comments according to the current fill paragraph. If a comment commences with `/***...` then the comment is automatically formatted to a box. If the comment commentces with `/**` then the comment is assumed to be a *Java Doc* comment.

Tags

A C−tags file may be generated within the editor using the **Tools** −> **C Tools** −> **Create Tag File**. find−tag(2) takes the user to the file using the tag information.

> On invoking the tag generator then the user is presented with a dialog box which specifies the generation option of the tags file. The base directory of the tags file search and tagging options may be specified to locate all of the definitions within the code space.
>
> The **tags** file is extremely useful where the user is dealing with inherited source code spread over multiple directories. Generation of a recursive tag file with all searching options enabled allows labels to be located extremely rapidly (certainly faster than IDE environments).

Folding and Information Hiding

Generic folding is enabled within the C and C++ files. The folds occur about braces **{...}** located on the left−hand margin. fold−all(3) (un)folds all regions in the file, fold−current(3) (un)folds the current region. Note that folding does not operate on K&R style code.

> The **Tools** −> **C Tools** menu allows `#define`'s to be evaluated within the buffer. Where the state of a `#if` is established to be false (using the `#define` information) then the disabled region of code is grayed out indicating which regions of the code are active.

Working Environment

compile(3) may be invoked to rebuild the source, the user is prompted to save any files.

> rcs−file(2) is automatically invoked if an RCS file is detected, the normal check−in/out operations may be performed through the editor.

Short Cuts

The short cut keys used within the buffer are:−

C−c C−c − Comment out the current line.
C−c C−d − Uncomment the current line.
C−c C−e − Comment to the end of the line with stars (*).
A−C−i − Restyle the current region.
esc q − Format a comment.
esc o − Format a comment.
f2 − (un)fold the current region
f3 − (un)fold all regions

NOTES

If the default language is C++, rather than 'C' the order of the file hooks should be over−ridden in the users local setup, using:−

```
add−file−hook ".c "                                 fhook−c
add−file−hook ".cc .cpp .hpp .rc .h .def .l .y .i"   fhook−cpp
```

This defaults all **.h** and **.def** files etc. to be C++ rather than C.

The hilighting is typically extended using a file **myc.emf** (or **mycpp.emf**) i.e. to include the usual extended types of **int32** etc, **myc.emf** might be defined as:−

```
hilight .hilight.c 1 "uint8"    .scheme.type
hilight .hilight.c 1 "int8"     .scheme.type
hilight .hilight.c 1 "uint16"   .scheme.type
hilight .hilight.c 1 "int16"    .scheme.type
hilight .hilight.c 1 "uint32"   .scheme.type
hilight .hilight.c 1 "int32"    .scheme.type
hilight .hilight.c 1 "float32"  .scheme.type
hilight .hilight.c 1 "float64"  .scheme.type
```

BUGS

The 'C' and 'C++' templates have been throughly used, there are no known issues with the templates.

The **.rc** hilighting is a little bogus and should not really be mapped onto **.cpp**. Do not attept to re−style.

SEE ALSO

c−brace(5), cmode(2m), compile(3), ctags(3f), find−tag(2), fold−all(3), fold−current(3), rcs−file(2), restyle−buffer(3), restyle−region(3), time(2m).

Supported File Types

c−hash−eval(3)

NAME

c−hash−eval − Evaluate C/C++ #defines
c−hash−del − Remove C/C++ #define evaluation
c−hash−set−define − Set a C/C++ #define
c−hash−unset−define − Unset a C/C++ #define

SYNOPSIS

n **c−hash−eval**
c−hash−del
c−hash−set−define "*variable*" "*value*"
c−hash−unset−define "*variable*"

DESCRIPTION

c−hash−eval evaluates C/C++ '#' lines, hiding sections of code which have been 'hashed' out. **c−hash−eval** evaluates the following '#' lines:−

#define <variable> ....
#ifdef <variable>
#if ...
#else
#endif

For `#defines` **c−hash−eval** creates a user variable "`%cd<variable>`", setting it to the value found. For `#ifdef` a simple check for the existence of variable "`%cd<variable>`" is made. If defined then code between the `#ifdef` and either its matching `#else` or `#endif` is displayed and code between the `#else` and `#endif` is hidden. If it is not defined then the reverse happens.

The state of `#if`'s are evaluated using calc(3), the following code is then displayed as for `#ifdef`.

Code is hidden by setting the $line−scheme(5) to a color similar to the back−ground. If an argument is given to the command the code is also narrowed out using narrow−buffer(2).

c−hash−del undoes the effect of **c−hash−eval** by restores hidden code.

c−hash−set−define and **c−hash−unset−define** can be used to manually set and unset `#define` variables.

NOTES

c−hash−eval, **c−hash−del**, **c−hash−set−define** and **c−hash−unset−define** are macros defined in `cmacros.emf`.

Executing **c−hash−eval** in a project header file (h file) which contains all used `#define` definitions will set up all `#define` variables ready for the main C files.

SEE ALSO

calc(3), $line−scheme(5), narrow−buffer(2).

calc(3)

NAME

calc − Integer calculator

SYNOPSIS

n **calc** "*string*"

DESCRIPTION

calc can perform simple integer based calculations given by "*string*", where the "*string*" takes the following form:−

```
"[b]<s>"
```

Where '*b*' is an optional letter setting the required output base which can be one of the following:

`b`	− Binary
`o`	− Octal
`d`	− Decimal
`x`	− Hexadecimal

Default when omitted is '*d*' (decimal). "*s*" is the sum to be calculated, which should be bodmas in form. Following is a list of valid symbols.

`(..)`	− Parentheses (contents calculated first)
`!`	− Logical not
`&&`	− Logical and
`||`	− Logical or
`==`	− Logical equals
`!=`	− Logical not equals
`~`	− Bitwise not
`&`	− Bitwise and
`|`	− Bitwise or
`^`	− Bitwise xor
`/`	− Divide
`*`	− Multiply
`%`	− Modulus
`+`	− Addition
`-`	− Subtraction
`0xNN`	− Hexadecimal number
`0NN`	− Octal number
`LR`	− Last calculation recall

Any MicroEmacs variables can be used in the calculation. The result of the calculation is stored in .calc.result(5). The argument *n* is a bitwise flag where:

0x01

Print out the result on the message−line.

0x02

Use string comparisons for == and != comparisons. This has the advantage of being able to calc "Foo" == "Bar" etc.

When omitted the default argument is 1.

EXAMPLE

To calculate the number of hours in a year:

```
calc "365*24"
```

To then calculate the number of seconds in the year:

```
calc "LR*60*60"
```

NOTES

calc is a macro defined in `calc.emf`.

SEE ALSO

.calc.result(5).

capitalize−word(2)

NAME

capitalize−word − Capitalize word
lower−case−word − Lowercase word (downcase)
upper−case−word − Uppercase word (upcase)
lower−case−region − Lowercase a region (downcase)
upper−case−region − Uppercase a region (upcase)

SYNOPSIS

n **capitalize−word** (**esc c**)
n **lower−case−word** (**esc l**)
n **upper−case−word** (**esc u**)

lower−case−region (**C−x C−l**)
upper−case−region (**C−x C−u**)

DESCRIPTION

capitalize−word capitalizes the next *n* words.

lower−case−word changes the next *n* words to lower case.

upper−case−word changes the next *n* words to upper case.

lower−case−region changes all alphabetic characters in the marked region to lower case (see set−mark(2)).

upper−case−region changes all alphabetic characters in the marked region to upper case

SEE ALSO

set−mark(2).

cbl(9)

SYNOPSIS

cbl − Cobol (85) files

FILES

hkcobol.emf − Cobol (85) hook definition
cobol.etf − Cobol (85) template file.

EXTENSIONS

.cbl − Cobol file

MAGIC STRINGS

−*− cobol −*−

Recognized by MicroEmacs and GNU Emacs, defines the file to be a cobol file. **DESCRIPTION**

The **cbl** file type templates provide simple hilighting of Cobol 85 files, the template provides minimal hilighting the language syntax.

NOTES

No special language features are provided within the language syntax definition.

BUGS

The Fortran hilight file is in it's infancy and a number of it's tokens may be misplaced.

SEE ALSO

Supported File Types

change−buffer−name(2)

NAME

change−buffer−name − Change name of current buffer

SYNOPSIS

n **change−buffer−name** "*buffer−name*" (**esc C−n**)

DESCRIPTION

change−buffer−name changes the name of the current buffer to *buffer−name*. Buffer names must be unique as they act as the identity handle. By default the buffer name is derived from the buffer's file name excluding the path. This can lead to conflicts, when editing files with the same name and different paths, in which case a counter is appended to the end of the buffer name to make the name unique. For example:

```
File Name                Buffer Name

/etc/file.c              file.c
/tmp/file.c              file.c<1>
```

By default, or an argument is given with bit 1 set, **change−buffer−name** will fail if a buffer with the given name already exists. This behavior can be changed by giving an argument with the first bit cleared, e.g. 0, in which case if a buffer with that name already exists then a counter as appended.

SEE ALSO

$buffer−fname(5), change−file−name(2), delete−buffer(2).

change−directory(2)

NAME

change−directory − Change the current working directory

SYNOPSIS

change−directory "*dir−name*" (**C−x C−d**)

DESCRIPTION

change−directory changes the current working directory to *dir−name*, on certain platforms (MS−DOS) this can also change the current drive. This command is largely redundant as any shell command automatically inherits the directory of the current buffer's file.

SEE ALSO

change−file−name(2).

change−file−name(2)

NAME

change−file−name − Change the file name of the current buffer

SYNOPSIS

change−file−name "*file−name*" (**C−x n**)

DESCRIPTION

change−file−name changes the file name of the current buffer to *file−name*. A validity check is made on the given file name and if found to be invalid (e.g. its a directory) the name is rejected.

SEE ALSO

change−buffer−name(2), change−directory(2), write−buffer(2).

change−font(2)

NAME

change−font − Change the screen font

SYNOPSIS

[X−Windows]
change−font "*fontName*"

[IBM−PC / MS−DOS]
change−font "*mode−no*" "*spec*"

[Microsoft Windows]
n **change−font** "*name*" *charSet weight width height*

DESCRIPTION

change−font is a platform specific command which allows the displayable font to be modified. The selection of font is determined by the monitor resolution and the capabilities of the graphics adapter.

This command is available on all systems except termcap. While MS−DOS does not support the concept of different fonts, it does (or at least the graphics card does) support the concept of changing screen resolution, which has the effect of changing the font. Each platform takes different arguments and are considered independently, as follows:

X−Windows

The X−Windows UNIX environments accept a single argument which is a fully qualified font name. Simply give the font X name and the font will change if it is available. The window size changes to attempt to retain the same number of rows and columns so ensure that when changing to a larger font then there is enough room (or a way) to resize a window which is larger than the actual screen.

The X font string describes the attributes of the font in terms of it's size name etc. as follows:−

−foundry−family−weight−slant−width−−pixels−point−hres−vres−space−av−set

Where

foundry

The type of foundry that digitized and supplied the font.

family

Font Family.

weight

Modifies the appearance of the font, the *weight* is usually **medium** or **bold**.

slant

Determines the orientation of the font. *slant* is usually **r**oman (upright), **i**talic or **o**blique.

width

Describes the proportionate width of the font. Typical widths include **normal**, **condensed**, **narrow**, **double**.

pixels

Pixel size of the font

point

The resolution of the font in tenths of a **dpi** (i.e. dpi*10)

hres

Horizontal resolution of the font in dpi.

vres

Vertical resolution of the font in dpi.

space

The spacing of the font. Typical spacing values include **m**onospaced (i.e. fixed width), **p**roportional and **c**haracter cell.

av

Mean width of all font characters, measured in tenths of a pixel.

set

Character set − character set standards e.g. **iso8859−1**.

The default font used by MicroEmacs '02 is

```
    −*−fixed−medium−r−normal−−15−*−*−*−c−90−iso8859−1
```

A good font to try is:

```
        change−font "−misc−fixed−medium−r−normal−−13−*−*−*−c−80−iso8859−1"
```

The font may also be changed in your **.Xdefaults** file by inserting the line:−

```
        MicroEmacs.font "−misc−fixed−medium−r−normal−−13−*−*−*−c−80−iso8859−1"
```

IBM−PC / MS−DOS

MS−DOS may only change the screen resolution, the standard screen resolution is either 80 columns by 25 rows or 80 by 50. A more advanced graphics card can typically support up to 132 by 60, MicroEmacs in theory has no limit but it has only been tested up to this size.

The main problem with MS−DOS machines is that there is no standard and this is no exception. The graphics mode needed to get a 132 by 60 screen (if available) varies from one card to the next so MicroEmacs '02 needs to know the graphic mode number your card uses to get your required screen resolution.

MicroEmacs '02 can also attempt a little bit of magic to double the number of rows on the screen for a given screen resolution. This is how 50 lines are obtained from the standard 25 line mode 3. If the value of "*spec*" is non−zero then this is attempted, to the authors knowledge this will either work or not depending on the direction of the wind and no harm will befall the users equipment. However the author also quickly disclaims anything and everything, the user uses this at their own peril, like everything else.

MicroEmacs '02 attempts to determine the new screen width and depth itself, in case this fails the commands change−frame−width(2) and change−frame−depth(2) may be used to correct the problem.

Following are the standard MS−DOS text modes:

```
        change−font "2" "0"       ; Simple monochrome or EGA monitor, 80 by 25.
        change−font "3" "0"       ; Simple EGA/VGA monitor, again 80 by 25.
        change−font "3" "1"       ; Simple EGA/VGA monitor using spec, 80 by 50.
```

Most Trident cards support the following text mode:

```
        change−font "86" "0"      ; Sweet 132 by 60
```

A Diamond Stealth supports the following mode:

```
        change−font "85" "1"      ; Nice 132 by 50
```

Cirrus video cards (1MB) seem to support:

```
        change−font "84" "1"      ; PT−526 (132x50)
```

Time to start digging out your graphics card manual!

Microsoft Windows

The Microsoft Windows environments utilize font files to drive the display. When **change−font** is invoked with no arguments, or a −ve argument then a font dialog is presented to the user to allow the font to be selected. The current font is not changed if a −ve argument is given, in both cases the variable [$result(5)] is set the the user selected font. The format of the returned string is "`OWwwwwhhhhhFontName`", where:−

O

The type of character set (0 for OEM and 1 for ANSI).

W

The font weight (0 − 9).

wwww

The font width.

hhhh

The font height.

FontName

The font name.

If a +ve argument is specified with **change−font** then the arguments are explicitly entered, arguments are defined as follows:−

font

> The name of the font − maximum of 32 characters. Select Fixed mono fonts only. Proportional fonts may be specified but the cursor will not align with the characters on the screen.
>
> An empty name ("") may be specified resulting in the selection of the default system OEM font. No other arguments are required when specified.
>
> Note that **Courier New** is not actually a fixed mono font as might be expected.

charSet

> The type of character set required, this is an integer value of:−
>
> `  0` − ANSI or Western (True Type etc)
> `161` − Greek

`162` − Turkish
`204` − Russian
`255` − OEM (or bitmapped)

weight

The weight of the font. The values are defined as:−

`0` − Don't care (Automatically selected).
`1` − Thin
`2` − Extra Light
`3` − Light
`4` − Normal
`5` − Medium
`6` − Semi−Bold
`7` − Bold
`8` − Extra−Bold
`9` − Heavy

Note that you may request a weight and it is not honored. Typically 4 and 7 are honored by most font definitions. 4 is typically used.

width

The width of the font. Specifies the average width, in logical units, of characters in the requested font. If this value is zero, the font mapper chooses a "closest match" value. The "closest match" value is determined by comparing the absolute values of the difference between the current device's aspect ratio and the digitized aspect ratio of available fonts.

Note that if the width is specified as zero then the height should be specified and the width will be automatically selected.

height

The height of the font. Specifies the desired height, in logical units, of the requested font's character cell or character. (The character height value is the character cell height value minus the internal−leading value.) If this value is greater than zero, the font mapper matches it against available character cell height values; if this value is zero, the font mapper uses a default height value when it searches for a match; if this value is less than zero, the font mapper matches it against available character height values.

Note: as with the weight the width and height may not be honored if the font cannot support the specified width/height in which case the closest matching height is automatically selected

Notes on the Standard Windows Configuration

For releases prior to '99, the **Terminal** font is the standard MS−DOS font used for the MS−DOS window. This is an OEM fixed width character set which contains all of the conventional symbols found in the DOS shell.

Releases of MicroEmacs post '99 may utilise any of the windows fonts, typically `Courier New` or `Lucida Console` are used, these provide the best screen rendering of characters. `Lucida Console` is slightly better with a smaller font size as this allows a '`1`' (one) and '`l`' (lower case L) to be distinguished.

The **Terminal** fonts are the same as shown in the DOS window the last 2 arguments are the width x height, the terminal equivalents (Bit Mapped) are commented here.

7x12

Regular weight seems to offer the best resolution for 14/15" monitors.

6x8

Regular weight is more suitable for 17−21" monitors which offer better resolutions.

The best options for the fonts are defined as follows:−

```
;Standard Terminal Fonts − standard weight
;change−font "Terminal" 0 4  4  6
change−font "Terminal" 0 4  6  8
;change−font "Terminal" 0 4  8  8
;change−font "Terminal" 0 4  5 12
;change−font "Terminal" 0 4  7 12
;change−font "Terminal" 0 4  8 12
;change−font "Terminal" 0 4 12 16
;change−font "Terminal" 0 4 10 18

;Standard Terminal Fonts − heavy weight
;change−font "Terminal" 0 7  4  6
;change−font "Terminal" 0 7  6  8
;change−font "Terminal" 0 7  8  8
;change−font "Terminal" 0 7  5 12
;change−font "Terminal" 0 7  7 12
;change−font "Terminal" 0 7  8 12
;change−font "Terminal" 0 7 12 16
;change−font "Terminal" 0 7 10 18
```

The "**Courier New**" font is not actually a fixed mono font as might be expected.

SEE ALSO

change−frame−width(2), change−frame−depth(2), $result(5), user−setup(3).

change−frame−depth(2)

NAME

change−frame−depth − Change the number of lines on the current frame
change−frame−width − Change the number of columns on the current frame

SYNOPSIS

n **change−frame−depth** ["*depth*"]
n **change−frame−width** ["*width*"]

DESCRIPTION

change−frame−depth changes the depth of the current frame, if the numeric argument *n* is given then the frame depth is changed by *n* lines. If *n* is not specified the user is prompted for the new *depth* and the frame depth will be changed to this value. It is assumed that the screen can draw the requested *n* lines and MicroEmacs draws the lines at the users peril.

A change in depth causes all of the internal windows currently displayed in the frame to be re−sized, the vertical position of the windows are modified to match the new screen dimension, the horizontal position of the windows remains unaltered. If the window is down−sized and the currently displayed windows are not able to fit into the new screen space then all windows are deleted with the exception of the current window.

change−frame−width changes the width of the current frame, if the numeric argument *n* is given then the frame width is changed by *n* characters. If *n* is not specified the user is prompted for the new *width* and the frame width will be changed to this value. It is assumed that the screen can draw the requested *n* columns and MicroEmacs draws them at the users peril. The windows are reorganized as **change−frame−depth** working horizontally rather than vertically.

NOTES

Within windowing environments such as **X−Windows** and **Microsoft Windows** these commands cause the canvas window to be re−sized to accommodate the change in screen size.

In MS−DOS and UNIX Termcap environments the physical size of the screen is determined by the characteristics of the display adapter. **change−frame−depth** may be used to correct anomalies (usually on portables) in the displayable screen area and the graphics mode. e.g. In DOS the graphics mode utilizes 50 lines, and only 47 lines are viewable. In this case change the screen depth to 47 and MicroEmacs will not utilize the remaining lines which are not viewable.

SEE ALSO

$frame−depth(5), $frame−width(5).

change−window−depth(2)

NAME

change−window−depth − Change the depth of the current window
grow−window−vertically − Enlarge the current window (relative change)
shrink−window−vertically − Shrink the current window (relative change)
resize−window−vertically − Resize the current window (absolute change)

SYNOPSIS

n **change−window−depth** ["*depth*"]

n **grow−window−vertically**
n **shrink−window−vertically**
n **resize−window−vertically**

DESCRIPTION

change−window−depth changes the depth of the current window, if the numeric argument *n* is given then the window depth is changed by *n* lines. If *n* is not specified the user is prompted for the new *depth* and the window depth will be changed to this value. The command aborts if the requested size cannot be achieved (the window becomes too small or a neighbouring one does).

NOTES

Commands **grow−window−vertically**, **shrink−window−vertically** and **resize−window−vertically** were replaced by the new **change−window−depth** command in April 2002. Following are macro implementations of the old commands:

```
define−macro grow−window−vertically
    @# change−window−depth
!emacro

define−macro shrink−window−vertically
    &neg @# change−window−depth
!emacro

define−macro resize−window−vertically
    !if &not @?
        !abort
    !endif
    change−window−depth @#
!emacro
```

SEE ALSO

change−window−width(2), resize−all−windows(2), split−window−vertically(2).

change−window−width(2)

NAME

change−window−width − Change the width of the current window
grow−window−horizontally − Enlarge current window horizontally (relative)
shrink−window−horizontally − Shrink current window horizontally (relative)
resize−window−horizontally − Resize current window horizontally (absolute)

SYNOPSIS

n **change−window−width** ["*width*"]

n **grow−window−horizontally**
n **shrink−window−horizontally**
n **resize−window−horizontally**

DESCRIPTION

change−window−width changes the width of the current window, if the numeric argument *n* is given then the window width is changed by *n* characters. If *n* is not specified the user is prompted for the new *width* and the window width will be changed to this value. The command aborts if the requested size cannot be achieved (the window becomes too small or a neighbouring does).

EXAMPLE

Refer to `mouse.emf` for an example of window growth using the mouse to manipulate the size of the windows.

NOTES

Commands **grow−window−horizontally**, **shrink−window−horizontally** and **resize−window−horizontally** were replaced by the new **change−window−width** command in April 2002. Following are macro implementations of the old commands:

```
define−macro grow−window−horizontally
    @# change−window−width
!emacro

define−macro shrink−window−horizontally
    &neg @# change−window−width
!emacro

define−macro resize−window−horizontally
    !if &not @?
```

```
                !abort
            !endif
            change−window−width @#
        !emacro
```

SEE ALSO

change−window−depth(2), resize−all−windows(2), split−window−horizontally(2).

charset−change(3)

NAME

charset−change − Convert buffer; between two character sets
charset−iso−to−user − Convert buffer; ISO standard to user character set
charset−user−to−iso − Convert buffer; user to ISO standard character set

SYNOPSIS

charset−change
charset−iso−to−user
charset−user−to−iso

DESCRIPTION

charset−change opens a dialog allowing the user to select a **From** and **To** character set. If the *Convert* button is selected the current buffer is converted to the destination character set. The command assumes that the current buffer is written in the **From** character set, no attempt is made to verify this.

charset−iso−to−user converts the current buffer, assumed to be in ISO−8859−1 (Latin 1) font format, to the current user's character set (defined by user−setup(3)). This process typically corrects any foreign language display problems.

Conversely, **charset−user−to−iso** converts the current buffer from the user's character set to ISO−8859−1 (Latin 1), this is typically used for the transfer of text files between different systems.

The current character set is configured using the user−setup(3) dialog (see Display Font Set). This in turn uses the command set−char−mask(2) to create the low level character conversion tables.

NOTES

charset−change, **charset−iso−to−user** and **charset−user−to−iso** are macros defined in `langutl.emf`.

SEE ALSO

user−setup(3), set−char−mask(2), Locale Support.

check−line−length(3)

NAME

check−line−length − Check the length of text lines are valid

SYNOPSIS

check−line−length

DESCRIPTION

check−line−length checks that the length of each line of the current buffer, starting with the current line, is less than or equal to fill−col(5). The command aborts if a line too long is found, leaving the cursor on the offending line. If no invalid lines are found the command succeeds leaving the cursor at the end of the buffer.

NOTES

check−line−length is a macro implemented in `misc.emf`.

SEE ALSO

$fill−col(5).

clean(3)

NAME

clean − Remove redundant white spaces from the current buffer

SYNOPSIS

n **clean**

DESCRIPTION

clean removes redundant white spaces from the current buffer, there are three types this command remove:

1)

Any space or tab character at the end of the line. All are removed until the last character is not a space or a tab, or the line is empty. Note that an empty line is not removed unless at the end of the buffer.

2)

Space characters are removed when the next character is a tab, making the space redundant, e.g. the strings "`Hello World`" and "`  Hello World`" will look identical because the tab character (' ') will indent the text to the 8th column with or without the space so the space can be removed.

3)

Superfluous empty lines at the end of the buffer are removed, leaving only one empty line.

4)

If argument *n* is given (value is not used) multiple blank lines are reduced to a single blank line.

DIAGNOSTICS

```
[Command illegal in view mode]
```

Caused by a redundant white space being found and the buffer being in view mode. Note that if clean completes while the buffer is in view mode then no superfluous white spaces where found. **NOTES**

clean is a macro defined in `format.emf`.

Most of this command's operation is performed by simple regex search and replace strings:

a)

Search for: "[\t]+$" Replace with: "\\0"

b)

Search for: "[ ]+\t" Replace with: "\t"

c)

Search for: "\n\n\n" Replace with: "\n\n" **SEE ALSO**

replace−string(2), tab(2m), delete−blank−lines(2), tabs−to−spaces(3).

cmode(2m)

NAME

cmode − C Programming language mode

SYNOPSIS

cmode Mode

C − mode line letter.

DESCRIPTION

cmode mode enters C programming language mode, providing automatic indentation and bracket matching facilities.

New users might initially find 'C−mode' a little disconcerting as the `tab` key is bound to the automatic formatting command, however the benefits of 'C−mode' far out weigh this. A lot of silly programming mistakes may be corrected at source, which are reflected in the layout. An unexpected automatic layout is a sure indication that the input syntax is incorrect − generally as a result of a missing semi−colon or bracket/brace pair.

The layout of a C program in cmode is controlled by the C−mode variables. The use of tab characters to create the required indentation is determined by the setting of the buffers tab(2m) mode. If disabled tab characters are used wherever possible.

SEE ALSO

buffer−mode(2), global−mode(2), tab(2m), $c−brace(5), $c−case(5), $c−contcomm(5), $c−continue(5), $c−margin(5), $c−statement(5).

command−apropos(2)

NAME

command−apropos − List commands involving a concept

SYNOPSIS

command−apropos "*string*" (**C−h a**)

DESCRIPTION

command−apropos compiles a list of all commands with *string* in their name, also giving their current key bindings.

EXAMPLE

To find all of the commands with "`command`" in their name space then issue the command "`C−h a command`" which generates a list of commands such as:−

```
abort−command ................. "C−g"
                                "esc C−g"
                                "C−x C−g"
command−apropos ............... "C−h a"
command−complete
execute−named−command ......... "esc x"
help−command .................. "C−h C−c"
ipipe−shell−command ........... "esc \\"
list−commands ................. "C−h c"
pipe−shell−command ............ "esc !"
                                "esc @"
                                "C−x @"
shell−command
```

SEE ALSO

describe−bindings(2).

command−wait(2)

NAME

command−wait − Conditional wait command

SYNOPSIS

n **command−wait**

DESCRIPTION

When a +ve argument *n* is given **command−wait** waits for *n* milliseconds before returning, this wait cannot be interrupted. If a −ve argument is given, **command−wait** waits for −*n* milliseconds but the command will return if the user interrupts with any input activity (i.e. presses a key).

When no argument is given **command−wait** loops getting and processing events (user input, screen updates etc) until either the calling commands **.wait** command variable is undefined or set to false (`0`). This more complex use of the command is used when a main macro must wait and process input until an exit criteria has been met, the input is best processed by setting the $buffer−input(5) variable to a second macro. The macro gdiff(3) uses this command in this way.

EXAMPLE

The following macro code will display a message on the screen for a fixed 5 seconds:

```
16 screen-poke 10 10 0 "Hello World!"
5000 command-wait
```

Similarly the following macro code will display a message for up to 5 seconds or till the user presses a key:

```
16 screen-poke 10 10 0 "Hello World!"
-5000 command-wait
```

SEE ALSO

ml−write(2), $buffer−input(5).

comment−line(3)

NAME

comment−line − Comment out the current line
uncomment−line − Uncomment current line

comment−to−end−of−line − Extend comment to end of line
comment−restyle − Reformat the current comment

comment−start − Start a new comment
comment−end − End the current comment

SYNOPSIS

n **comment−line**
n **uncomment−line**

comment−to−end−of−line
comment−restyle

comment−start
comment−end

DESCRIPTION

The action of the **comment** commands are file type specific (comments in **C** are `/* ... to ... */` where as MicroEmacs macro file comments are `; ... to the end of line`) so the commands must be configured for each file type (see the NOTES section below). The configuration is automatically performed by almost all the standard file hooks released with MicroEmacs by the file hook so these commands should be fully functional.

comment−line comments out the current and next *n*−1 lines (default when *n* is omitted is to comment out just the current line). The cursor is then moved to the start of the next line. **uncomment−line** behaves differently depending on whether the file type terminates a comment with an end token or simply by the end of the line. If an end token is used then **uncomment−line** removes the current and next *n* comments. If the end of line is used **uncomment−line** removes the first comment on the current and next *n−1* lines.

comment−to−end−of−line inserts *comment−pad*s (see NOTES) up−to the $fill−col(5) and then terminates the comment with the *comment−end* string. **comment−restyle** reformats the text within the comment, filling text lines to the **$fill−col** and regenerating any boxing and divide lines.

comment−start opens a new "***comment***" buffer which is configured for writing a text comment, the user can then type in the comment with all the benefits of MicroEmacs in a plain text editor. Once the comment is complete use the **comment−end** command to insert the comment into the source file, this comment is locally bounded to "**C−c C−c**". The styling of the comment is controlled by the *comment−flag* setting (see NOTES).

NOTES

Consider the structure of a box comment to be as follows:

```
<comment-start><comment-pad><comment-pad><comment-pad><comment-box-right>
<comment-box-left> COMMENT TEXT ... COMMENT TEXT      <comment-box-right>
<comment-box-left> COMMENT TEXT ... COMMENT TEXT      <comment-box-right>
<comment-box-left><comment-pad><comment-pad><comment-pad><comment-end>
```

The comment commands are configured by the single file hook command variable **.fhook−**<*type*>**.comment** where <*type*> is the file type label. The structure of the variable is a list with the following format:

```
    |<comment-start>|<comment-end>|<comment-pad>|<comment-box-left>|...
        ...<comment-box-right>|<comment-flags>|
```

Where **|** is the list divide character. The <*comment−flags*> are a list of character flags which are defined as follows:

`b`

Box format required, i.e. create right edge using <*comment−box−right*>.

`f`

Footer line required.

`F`

Fill footer line with <*comment−pad*> strings.

`h`

Header line required.

`H`

Fill header line with <*comment−pad*> strings.

`j`

Enable Justify mode in *comment* buffer. **EXAMPLE**

The following **comment** is the standard **C** setting:

```
set-variable .fhook-c.comment "|/*|*/|*| * | * |f|"
```

This can be used to create comments of the form:

```
/* comment-line comments out the current and next n-1 lines (default
 * when n is omitted is to comment out just the current line). The
 * cursor is then moved to the start of the next line.
 *
 * uncomment-line behaves differently depending on whether the file
 * type terminates a comment with an end token or simply by the end
 * of the line.
 */
```

A box style comment can be generated by changing the <*comment−flags*> form `f` to `bfFhH`, producing:

```
/**********************************************************************
 * comment-line comments out the current and next n-1 lines (default *
 * when n is omitted is to comment out just the current line). The   *
 * cursor is then moved to the start of the next line.               *
 *                                                                   *
 * uncomment-line behaves differently depending on whether the file  *
 * type terminates a comment with an end token or simply by the end  *
 * of the line.                                                      *
 *********************************************************************/
```

SEE ALSO

File Hooks.

compare−windows(2)

NAME

compare−windows − Compare buffer windows, ignore whitespace.
compare−windows−exact − Compare buffer windows, with whitespace.

SYNOPSIS

n **compare−windows**
compare−windows−exact

DESCRIPTION

compare−windows compares the textural content of ALL the current windows from their current cursor position. These commands are generally used to locate the next difference in the windows displayed. Returns TRUE if the buffers of the windows do not differ from the current position to the end of the file (inclusive), else returns FALSE setting the cursor of each buffer to the first point of difference.

The default mode of operation ignores white−space, a numeric argument *n* of zero (0) then an exact white−space match is performed.

compare−windows−exact is a macro short cut for *0 compare−windows*, forcing a white space comparison.

SEE ALSO

diff(3), diff−changes(3), gdiff(3).

compile(3)

NAME

compile − Start a compilation process

SYNOPSIS

n **compile** "*compile−command*"

DESCRIPTION

compile gets and executes the compile command using a pipe execution (incremental pipe on UNIX platforms), loading the output into a buffer called "***compile***", with go to error parsing using the command get−next−line(2). The default compile execution is set by variable %compile−com(5), the error parsing is setup using the command add−next−line(2).

Before the compile command is executed save−some−buffers(2) is executed to allow the user to ensure that all relevant buffers are saved. If an argument is given to compile then it is passed on to this command, so if an argument of 0 is given, all buffers are automatically saved.

NOTES

compile is a macro defined in `tools.emf`.

SEE ALSO

add−next−line(2), %compile−com(5), get−next−line(2), save−some−buffers(2), grep(3).

copy−region(2)

NAME

copy−region − Copy a region of the buffer

SYNOPSIS

copy−region (**esc w**)

DESCRIPTION

copy−region copies all the characters between the cursor and the mark set with the set−mark(2) command into the kill buffer (so they can later be yanked elsewhere).

If the last command also entered text into the kill buffer (or the @cl(4) variable is set to one of these commands) the **copy−region** text is appended to the last kill.

USAGE

To copy text from one place to another, using the **copy−region** command, the following operations are performed:

- Move to the beginning of the text you want to copy.
- Set the mark there with the set−mark (**esc−space**) command.
- Move the point (cursor) to the end of the text.
- Use **copy−region** to copy the region you just defined. The text will be saved in the kill buffer. (If you accidentally delete the text use yank (**C−y**) immediately or undo (**C−x u**) to restore the text).
- Move the point to the place you want the text to appear.
- Use the yank (**C−y**) command to copy the text from the kill buffer to the current point.

Repeat the last two steps to insert further copies of the same text.

NOTES

Windowing systems such as X−Windows and Microsoft Windows utilize a global windowing kill buffer allowing data to be moved between windowing applications (*cut buffer* and *clipboard*, respectively). Within these environments MicroEmacs '02 automatically interacts with the windowing systems kill buffer, the last MicroEmacs '02 **copy−region** entry is immediately available for a paste operation into another windowing application.

SEE ALSO

exchange−point−and−mark(2), kill−region(2), set−mark(2), yank(2).

count−words(2)

NAME

count−words − Count the number of words in a region

SYNOPSIS

count−words (**esc C−c**)

DESCRIPTION

count−words Counts the number of words between the set−mark(2) position and the current cursor position. The command also gives statistics on the number of characters and the average characters per word. The output appears on the message line in a format such as:−

```
54 Words, 345 Chars, 8 Lines
```

$result(5) is set to the same output string.

SEE ALSO

$result(5), buffer−info(2), set−mark(2).

create−callback(2)

NAME

create−callback − Create a timer callback

SYNOPSIS

n **create−callback** "*command*"

DESCRIPTION

create−callback creates a timer based callback command. The given *command* is called back in *n* milliseconds time. This can be used by the user to monitor system events (such as incoming mail). The command is called only once, but if the command creates a callback of itself a loop is created.

If a −ve argument *n* is given any pending callback for *command* is cancelled.

EXAMPLE

The following example creates a callback that is invoked every 10 minutes.

```
define−macro Example−callback
    ml−write "It was 10 minutes since you last saw me!"
    600000 create−callback Example−callback
!emacro
Example−callback
```

NOTES

A call−back cannot interrupt while MicroEmacs is active, instead the call−back is delayed until MicroEmacs becomes inactive. MicroEmacs is considered to be inactive when it is waiting for user input, this could be during the execution of another macro. If a command or macro requires no user input then once execution has started, it cannot be interrupted by a call−back macro.

The resolution of the clock is platform dependent, some platforms limit the minimum timer period to 10 milliseconds.

MicroEmacs does not guarantee to service the callbacks within any set time constraints, the resultant callback intervals may be of a slightly different duration than requested.

When a callback macro is executed, the key given by @cck(4) is "`callback`. If the current buffer has a $buffer−input(5) command set, this command will be called instead of the callback command with **@cc** and **@cck** set appropriately. It is the responsibility of the input macro to deal with the callback.

SEE ALSO

$auto−time(5), define−macro(2).

create−frame(2)

NAME

create−frame − Create a new frame

SYNOPSIS

n **create−frame**

DESCRIPTION

create−frame creates a new frame for the current MicroEmacs session. MicroEmacs support the creation of 'internal' multiple frames on all platforms and 'external' frames on windowing platforms (such as Windows and XTerm). An external frame creates a new OS window so both the existing frame and the new frame are visible, whereas an internal frame uses the same OS window or console which means that the existing frame is hidden and the new frame takes its place.

The numeric argument *n* can be used to define which type of frame is to be created. If an argument of `1` is given (the default argument) an external frame will be created, whereas an internal frame will be created if an argument of `0` is given.

NOTES

Internal frames can only be accessed via the next−frame(2) command, external frames can usually be accessed via the OS as well.

MicroEmacs is not multi−threaded in that only one frame can be active at any one time (the complexity of being able to run a command in one frame while editing in another would rapidly lead it away from the 'Micro' status). This means that if a command is left active (such as a search) in one frame and the focus is changed to another the input is 'sent' to the frame with the active command and the message '[NOT FOCUS]' will appear in the message−line of the frame with the OS focus.

create−frame may be useful in macros that rely on a window layout, this is because they can preserve the users current window layout by creating and new internal frame in which to run.

SEE ALSO

delete−frame(2), next−frame(2).

crlf(2m)

NAME

crlf − File's line feed style

SYNOPSIS

crlf Mode

c − mode line letter.

DESCRIPTION

When enabled **crlf** indicates that a line feed should be written out in the MS−DOS style of '**\r\n**'. When clear then a UNIX style of '**\n**' should be used.

NOTES

This mode only effects the style in which the buffer is written if auto(2m) mode is enabled.

SEE ALSO

auto(2m), ctrlz(2m), save−buffer(2), find−file(2), $buffer−fmod(5).

crypt(2m)

NAME

crypt − Encrypted file mode

SYNOPSIS

crypt Mode

Y − mode line letter.

DESCRIPTION

crypt mode enables encrypted files to be loaded and saved for security purposes. The key can be set at any time using the command set−encryption−key(2). Warning, take care if setting this as a global mode, it can have side−effects.

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 '02 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, use find−cfile(3) to load *ciphertext* files, you will be asked the encryption key before the file is read.

SEE ALSO

buffer−mode(2), find−cfile(3), global−mode(2), set−encryption−key(2).

sh(9)

SYNOPSIS

*sh − UNIX shell files

FILES

hkshell.emf − UNIX shell file hook definition

EXTENSIONS

.sh − Bourne shell file
.ksh − Korn shell file
.csh − C−Shell file
.zsh − Z−Shell file
.login − Shell user login file
.profile − Shell user profile
.tcshrc − T−Shell start up file

MAGIC STRINGS

#![\t]*/.*sh

MicroEmacs '02 recognizes the magic string on the first line of the file used to locate the executable. The shell files may be extension less and are still recognized. Note that this is the typical method of identifying shell files and will recognize other files not mentioned above i.e. **bash** shells. **DESCRIPTION**

The **shell** file type template provides simple hilighting of the shell files.

BUGS

None reported.

There is a heavy bias towards Bourne, Korn and Zsh shells. The author is not a csh shell user so has probably missed a lot of csh features.

SEE ALSO

fvwm(9).

Supported File Types

ctags(3f)

NAME

ctags − Generate a C tags file

SYNOPSIS

me "@ctags" [*−v%tag−option=<flags>*] [*files*]

DESCRIPTION

The start−up file `ctags.emf` may be invoked from the command line to generate a **tags** file for C and C++ source and header files.

Given a list of *files* a tags file `tags` is generated in the current directory, which may be used by the find−tag(2) command. This is a good alternative on Microsoft platforms where a utility such as **ctags(1)** is not typically available. If no *files* are specified the default file list is "`./`", i.e. process the current directory. If a directory name is given (such as the default "`./`") all C and C++ source and header files within the directory will be processed.

The value of variable **%tag−option** is used to control the tag generation process, its value <*flags*> can contain any number of the following flags:

`a`

Append new tags to the existing tag file, note that if also using flag '`m`' multiple 'tags' to the same item may be created.

`m`

Enable multiple tags. This enables the existence of 2 tags with the same tag name, but typically with different locations. See help on find−tag(2) for more information on multiple tag support.

`r`

Enables recursive mode, any sub−directory found within any given directories will also be processed.

`v`

Add global variables to the tag file. (i.e. variables marked with *extern*).

`e`

Add enumerated variables to the tag file (i.e. *enum* members).

`s`

Add structure, type definitions and classes to the tag file (i.e. *stuct*, *typedef* and *class*).

The generated tags file includes `#define` and C++ class names.

NOTES

This function is invoked from menu

Tools −> C Tools −> Create Tags File

when the user requests a tags file to be generated.

The user setup file "`myctags.emf`" is executed by ctags during start−up, this file can be used to over−ride any of the ctags configuration variables (see below).

The following variables are set within "`ctags.emf`" and are used to control the process:−

%tag−option

Tags options flag, default value is "". See above for more information.

%tag−filemask

A list of source file masks to be processed when a directory is given, default value is "`:*.[cC]:*.cpp:*.cc:*.h:*.hpp:`".

%tag−ignoredir

A list of directories to be ignored when recursive option is used, default value is "`:SCCS/:CVS/:`".

These variables can be changed using the −v command−line option or via the "`myctags.emf`" file

SEE ALSO

find−tag(2), start−up(3), c(9).

ctrlz(2m)

NAME

ctrlz − File's termination style

SYNOPSIS

ctrlz Mode

z − mode line letter.

DESCRIPTION

When enabled **ctrlz** indicates that an MS−DOS style '**ctrl−z**' file termination character should be written out. When clear, a UNIX style of no termination character should be used.

NOTES

This mode only effects the style in which the buffer is written if auto(2m) mode is enabled.

SEE ALSO

auto(2m), crlf(2m), save−buffer(2), find−file(2), $buffer−fmod(5).

cvs(3)

NAME

cvs − MicroEmacs CVS interface
cvs−add − MicroEmacs CVS interface − add file
cvs−checkout − MicroEmacs CVS interface − checkout files and directories
cvs−commit − MicroEmacs CVS interface − commit changes
cvs−diff − MicroEmacs CVS interface − diff changes
cvs−gdiff − MicroEmacs CVS interface − graphical diff changes
cvs−log − MicroEmacs CVS interface − log changes
cvs−remove − MicroEmacs CVS interface − remove file
cvs−resolve−conflicts − MicroEmacs CVS interface − resolve conflicts
cvs−state − MicroEmacs CVS interface − list state of directory files
cvs−update − MicroEmacs CVS interface − update directory files

SYNOPSIS

cvs

cvs−add
cvs−checkout
cvs−commit
cvs−diff
cvs−gdiff
cvs−log
cvs−remove
cvs−resolve−conflicts
cvs−state
cvs−update

DESCRIPTION

The cvs and sub−commands provide MicroEmacs with an interface to **cvs(1)**. **CVS** is a version control system; using it, you can record the history of your source file modifications. CVS is licensed under the GNU General Public License and is freely available on the Internet, see the documentation provided with CVS for more information on its features and use.

The MicroEmacs **cvs** command opens up a modified file−browser(3) with an additional "`*cvs-console*`" window. The "`*files*`" window includes additional columns showing the CVS state, revision and repository date. The functionality of the file−browser is the same as a non−CVS folder with the exception that additional CVS item controls are located in the mouse context menu (opened by clicking the right mouse button in the `*files*` buffer). This menu item opens another sub−menu providing access to the following items:

Checkout files

Checks out a file or directory from the repository into the current directory. The file or directory is specified by typing the name into a dialog which is opened when this option is selected. This runs the command "`cvs checkout <file>`".

Update files

Updates the currently selected files, files are selected by clicking the left button to the left of the required file name. Multiple files may be selected by 'dragging' a hilight region over the required files. This runs the command "`cvs update <files>`".

Commit files

Commits any changes made to the selected files back to the CVS repository. This runs the command "`cvs commit <files>`".

Diff files

Displays any differences between the selected files and the CVS repository version in the **cvs−console** window. This runs the command "`cvs diff <files>`".

Log files

Displays the CVS logs for the selected files in the **cvs−console** window. This runs the command "`cvs log <files>`".

Status files

Displays the CVS status for each of the selected files in the **cvs−console** window. This runs the command "`cvs status -v <files>`".

Add files

Adds the selected files to the CVS repository. Note this command only performs the local add, a **CVS commit** is required to make the addition permanent. This runs the command "`cvs add <files>`".

Remove files

This command is deliberately not implemented as its far to dangerous! Instead it opens a dialog informing the user to use the **cvs−remove** command instead.

Graphical diff

This command opens a gdiff(3) window showing the differences between the currently selected file and the CVS repository version. Note this command only works with a single file.

Resolve conflicts

This command may be used to resolve merge conflicts created by a *CVS update* operation. The command opens a gdiff(3) window showing the areas of conflict allowing the user to select the correct version and saving the resultant version back to the local file. Note this command only works with a single file.

Clear cvs console

Clears the **cvs−console** buffer.

The **cvs−add** command adds the current buffers file to the repository. Note that this command only performs the local addition, a *CVS commit* is required to make the addition permanent.

The **cvs−checkout** command checks out a file or directory from the repository into the current directory. The user specifies the file on the message line.

The **cvs−commit** command commits any changes made to the currently buffer's file (including additions) to the repository. The user is prompted for a commit log message.

The **cvs−diff** command opens a **cvs−diff** window displaying the differences between the current buffer's local file and repository version. If the current buffer is a directory list it will list all the differences found in all files within the directory.

The **cvs−gdiff** command opens a gdiff(3) window displaying the differences between the current buffer's local file and repository version.

The **cvs−log** command opens a **cvs−log** window displaying the CVS log of the current buffer's file.

The **cvs−remove** command removes the current buffer's file from the repository − PLEASE NOTE THIS CAN LEAD TO LOST DATA!!! This command only performs the local removal; as it deletes the buffer and file the **cvs−commit** command cannot be used to commit the removal to the CVS repository. Instead the main **cvs** file−browser menu or **cvs(1)** itself must be used.

The **cvs−resolve−conflicts** command may be used to resolve any conflicts created by CVS when the current buffer's file is updated. The command opens a gdiff window displaying the areas of conflict, the user may then select the correct version in each case and save the resultant new version over the local file.

The **cvs−state** command opens a **cvs−state** window listing the state of any file in the current directory which is not up−to−date. Note that unlike most cvs sub commands this command executes over all files in the current buffer's file directory.

The **cvs−update** command updates all files in the current directory, the output being reported to a new **cvs−update** window. Note that unlike most cvs sub commands this command executes over all files in the current buffer's file directory.

NOTES

cvs and sub−commands are macros defined in file `cvs.emf`.

By default MicroEmacs's **cvs** commands skip all files ignored by **cvs(1)**. This is configured by the variable **.cvs.filter**, defining this variable to `0` disables this special filtering.

SEE ALSO

file−browser(3).

gdb(3)

NAME

gdb − GNU Debugger
dbx − UNIX Debugger

SYNOPSIS

gdb "*program−name*" "*additional−args*"
dbx "*program−name*" "*additional−args*"

DESCRIPTION

gdb and **dbx** provide an editor interface to the GNU and native system debuggers, respectively. On running either command then an interactive shell window is opened to the debugger command line interface. MicroEmacs then interprets the information from the debugger interface and opens files and hilights the current line as required. The current line is maintained while single stepping through the code.

Buffers opened and referenced by the debugger have the key `F9` bound to setting a break point.

NOTES

gdb and **dbx** are macros defined in file `hkipipe.emf`.

SEE ALSO

perldb(3), ipipe−shell−command(2).

define−help(2)

NAME

define−help − Define help information

SYNOPSIS

define−help "*string*" ["*section*"]

Free form text

!ehelp

DESCRIPTION

define−help provides a mechanism to define help information for commands and variables within macro files. The command allows user defined macros to be documented with help information that is accessible from the command line via the normal help commands such as help−item(2).

The help information is typically embedded in the macro file with the macro command that it is documenting. When the macro file is loaded then the help information is loaded and integrated into the existing help database.

string is the name of the item that is being defined, *section* defines what section the item belongs to. Following is a table of standard MicroEmacs '02 sections:

`1`	MicroEmacs command line arguments.
`2`	Built−in commands.
`2m`	MicroEmacs buffer modes.
`3`	Macro commands.
`4`	Macro language commands.
`5`	MicroEmacs variables.
`8`	MicroEmacs file formats.

When *section* is omitted is defaults to the general section which is usually used for the higher level help pages.

Text following the **define−help** line contains the help information, this is a free form text area that is reproduced when the help information is requested. The end of the text area is delimited by a **!ehelp** construct. The help text is usually displayed using a special hilighting scheme to control the colors and hyper−text links to other help pages. As a result the text may contain escape ('`^[`') key sequences, see ehf(8) for more information on the format.

EXAMPLE

The following example is a define−help representation for the paragraph−to−line(3) macro.

```
define-help "paragraph-to-line" "3"

^[cENAME^[cA

     paragraph-to-line - Convert a paragraph to a single line
$a


^[cESYNOPSIS^[cA

     n paragraph-to-line


^[cEDESCRIPTION^[cA

     paragraph-to-line  reduces  each of the  next n  paragraphs  of text to a
     single  line.  This  is  used to  prepare  a  document  to go into a word
     processor environment where end of line marks represent paragraph marks.


^[cENOTES^[cA

     This command is a macro defined in format.emf.


^[cESEE ALSO^[cA

     ^[ls^[lm^[cGfill-paragraph(2)^[cA^[le.

!emacro
```

SEE ALSO

ehf(8), help−item(2), define−macro(2), help−command(2), help−variable(2).

define−macro(2)

NAME

define−macro − Define a new macro

SYNOPSIS

n **define−macro** *macro−name*

Macro body
!emacro DESCRIPTION

define−macro starts the definition of an macro named *macro−name*, only used within macro files or buffers. After the above header line, the body of the macro is added, one command or expression on a line. The macro definition is completed by the !emacro directive.

The numeric argument *n*, specified as zero, defines the macro as private such that it does not appear on a command completion list. A zero argument is generally used on helper macro's that form part of a larger macro. If the argument is omitted, or non−zero, then the macro appears in the command completion list.

See execute−file(2) for a complete definition and examples of the MicroEmacs '02 macro language.

Once the macro has been defined, it becomes indistinguishable from a standard MicroEmacs '02 command, i.e. execute−named−command(2) (`esc x`) can be used to execute the macro and global−bind−key(2) can be used to globally bind the command to a key combination.

There are no restrictions on the number of macros that may be defined, provided that the name space is managed properly. Consideration must be given as to when any additional macros that are created are loaded into MicroEmacs '02. We usually like start−up to be rapid and macros are loaded as and when requested by the user, or by the buffer hooks as new files are loaded (see add−file−hook(2) and define−macro−file(2)).

User defined macros may be documented with on−line help by including a define−help(2) construct within the macro file.

EXAMPLE

The following are two standard macros provided with MicroEmacs '02. The first is a macro called **clean**, this strips trailing white space from the ends of lines in a file and removes blank lines from the end of the file.

```
define−macro clean
    ;
```

```
        ; Prepare to clean up file.
        ; Remember line & magic mode
        set-variable #l0 $window-line
        set-variable #l1 &not &bmod magic
        !if #l1
            1 buffer-mode "magic"
        !endif
        ;
        ; Get rid of trailing white space on EOL
        beginning-of-buffer
        replace-string "[\t ]+$" "\\0"
        beginning-of-buffer
        replace-string "[ ]+\t" "\t"
        ;
        ; Strip trailing blank lines.
        end-of-buffer
        backward-line
        !while &and &gre $window-line 1 &sequal @wc "\n"
            kill-line
            backward-line
        !done
        ;
        ; Clean up - restore buffer modes etc.
        ; Move back to starting line & restore original magic mode
        !force goto-line #l0
        !if #l1
            -1 buffer-mode "magic"
        !endif
        screen-update
        ml-write "Cleaned up file."
    !emacro
```

The second example converts all of the `<tab>` characters in the file to their `<SPACE>` character equivalent.

```
    ;
    ; tabs-to-spaces.
    ; Convert all of the tabs to spaces.
    define-macro tabs-to-spaces
        ; Remember line
        set-variable #l0 $window-line
        beginning-of-buffer
        !force search-forward "\t"
        !while $status
            set-variable #l1 $window-acol
            backward-delete-char
            &sub #l1 $window-acol insert-space
            !force search-forward "\t"
        !done
        goto-line #l0
        screen-update
        ml-write "[Converted tabs]"
    !emacro
```

Both of these commands are available from the command line, they are indistinguishable from the built in commands.

Macros may also be nested, as shown in the next example, this macro contains a **define−macro** within itself, when executed the macro creates another macro dynamically − dynamic macros are generally given a prefix of **%** and are highlighted differently in describe−bindings(2).

The following example is taken from the alarm(3) macro, executing **alarm** the user is prompted for a message, and the time interval before the alarm expires in hours and minutes. It then creates a new macro with a callback so that the new macro will be called at the correct time.

```
!if &seq %alarm-numb "ERROR"
    set-variable %alarm-numb 0
    set-variable %osd-alarm &pinc %osd 1
!endif

define-macro alarm
    set-variable %alarm-numb &add %alarm-numb 1
    set-variable #l0 &cat "%alarm-" %alarm-numb
    !force set-variable #l2 @3
    !if &not $status
        set-variable &ind #l0 @ml "Message"
        set-variable #l1 @ml "Hours"
        set-variable #l2 @ml "Minutes"
    !else
        set-variable &ind #l0 @1
        set-variable #l1 @2
    !endif
    set-variable #l1 &mul 60000 &add &mul 60 #l1 #l2
    define-macro #l0
        !bell
        set-variable #l0 &add &len &ind @0 10
        osd %osd-alarm 0 "bat" 9 3
        osd %osd-alarm 1 ""
        osd %osd-alarm 2 "c" "ALARM"
        osd %osd-alarm 3 ""
        osd %osd-alarm 4 "" &ind @0
        osd %osd-alarm 5 ""
        osd %osd-alarm 6 "Bcf" " OK " f void
        %osd-alarm osd
    !emacro
    #l1 create-callback #l0
!emacro
```

SEE ALSO

Refer to !return(4) and !abort(4) for details macro termination.

!emacro(4), add−file−hook(2), define−macro−file(2), define−help(2), describe−bindings(2), execute−file(2), execute−named−command(2), global−bind−key(2), insert−macro(2), start−kbd−macro(2).

define−macro−file(2)

NAME

define−macro−file − Define macro file location

SYNOPSIS

define−macro−file "*file−name*" ["*macro−name*" "*macro2−name*" ...]

DESCRIPTION

Macros are loaded as late as possible using an on−demand mechanism, this speeds up the load time of MicroEmacs '02, it also keeps the startup file clean since macros are not defined within the start−up file. Only when the user first executes a macro defined via **define−macro−file** is the file loaded, the macro becomes defined and is executed. Subsequent calls to the macro will not reload the file as the macro will now be fully defined.

define−macro−file binds macros (*macro−name* ...) to a file name (*file−name*). This operation informs MicroEmacs '02 which file should be loaded when *macro−name* is first executed. The *macro−name* arguments may be omitted if the file contains only one exported macro which has the same name as *file−name*.

Alternatively the macro file may contain many macros all of which can be defined by a single call to **define−macro−file**, listing all macros on the same line after the *file−name*. If a *macro−name* is given then the default macro *file−name* is not created, if a macro of that name does exist it must be added to the *macro−name* list.

EXAMPLE

The following definitions are found in the `me.emf` start−up file:−

```
0 define−macro−file utils ascii−time regex−forward regex−backward
define−macro−file format clean sort−lines−ignore−case tabs−to−spaces ...
define−macro−file cvs cvs cvs−state cvs−update cvs−commit cvs−log ...
define−macro−file abbrev expand−abbrev−handle expand−iso−accents ...
define−macro−file misc symbol check−line−length alarm time
define−macro−file search replace−all−string query−replace−all−string
define−macro−file tools compile grep rgrep which diff diff−changes
define−macro−file hkdirlst file−browser file−browser−close
define−macro−file comment comment−line uncomment−line comment−to−end−of−line
define−macro−file spell spell−word spell−buffer spell−edit−word find−word
define−macro−file games Metris Patience Triangle Mahjongg Match−It
define−macro−file buffstp buffer−setup buffer−help buffer−tool
define−macro−file fattrib file−attrib
define−macro−file osd osd−main
define−macro−file gdiff
```

```
        define-macro-file calc
        define-macro-file draw
```

Hilighting a number of entries as examples; macro file **calc** is defined with no macro definition, the macro is assumed to be **calc**. The file **tools.emf** contains multiple macros **compile**, **grep**, **diff** and **diff−changes**; all can be defined by a single **define−macro−file** entry.

NOTES

- Macro files are searched for in the current directory and along the $search−path(5).
- The macro file is not loaded unless a binding has been defined using **define−macro−file**.
- Any other macros that exist in the *file−name* macro file become defined when the entry point macro is loaded and are available for use. This is potentially useful as a single *entry* macro may be defined using **define−macro−file**, when invoked other helper macros become available.

SEE ALSO

add−file−hook(2), define−macro(2), $search−path(5), start−up(3).

del(2m)

NAME

del − Flag buffer to be deleted

SYNOPSIS

del Mode

d − mode line letter.

DESCRIPTION

This mode cannot be set globally and is used to flag that the buffer is to be deleted. The state of the mode is displayed in the output of list−buffers(2), if the first column is a 'D' the mode is set, otherwise it is not. Only the execute command in list−buffers(2) (bound to 'x') uses this flag to actually delete the buffer.

SEE ALSO

list−buffers(2), save(2m).

delete−blank−lines(2)

NAME

delete−blank−lines − Delete blank lines about cursor

SYNOPSIS

delete−blank−lines (**C−x C−o**)

DESCRIPTION

delete−blank−lines deletes all the blank lines before and after the current cursor position. Note that the deleted lines are not added to a kill buffer.

SEE ALSO

delete−indentation(3), clean(3), kill−line(2).

delete−buffer(2)

NAME

delete−buffer − Delete a buffer

SYNOPSIS

n **delete−buffer** "*buffer−name*" (**C−x k**)

DESCRIPTION

delete−buffer disposes of buffer *buffer−name* in the editor and reclaim the memory. This does not delete the file that the buffer was read from.

If the buffer has been edited and its name does not start with a '*****' then the user is prompted as to whether the changes should be discarded. Also if the buffer has an active process running in it then confirmation is sort from the user before the process is killed.

The argument *n* can be used to change the default behavior of delete−buffer described above, *n* is a bit based flag where:−

0x01

Enables loss of work checks (default). These include a check that the buffer has not been modified, if so the user is prompted. Also if a process is running then user must confirm that the process can be killed. If this flag is not supplied then the buffer is killed without any user prompts (useful in macros). **SEE ALSO**

next−buffer(2).

delete−dictionary(2)

NAME

delete−dictionary − Remove a spelling dictionary from memory

SYNOPSIS

n **delete−dictionary** ["*dictionary*"]

DESCRIPTION

delete−dictionary removes the given *dictionary* from memory, where *n* is a bitwise flag determining the removal mode, defined as follows:−

0x01

Prompt the user before loosing any changes (except to the ignore dictionary).

0x02

Delete all the dictionaries other than the ignore dictionary.

0x04

Delete the ignore dictionary.

If the argument does not have bit 0x02 or 0x04 set, which specify the dictionaries to be deleted, the user is prompted for the "*dictionary*". The default argument is 1.

NOTES

The ignore dictionary is a temporary dictionary that exists in memory for duration of the MicroEmacs session; the dictionary holds words that have been ignored during any previous spell checks (see spell(2)). All of the words that have been ignored may be discarded with:−

```
4 delete−dictionary
```

i.e. **esc 4 esc x delete−dictionary**.

SEE ALSO

spell−buffer(3), add−dictionary(2), save−dictionary(2), spell(2).

delete−frame(2)

NAME

delete−frame − Delete the current frame

SYNOPSIS

n **delete−frame**

DESCRIPTION

delete−frame deletes the current frame.

SEE ALSO

create−frame(2), next−frame(2).

delete−indentation(3)

NAME

delete−indentation − Join 2 lines deleting white spaces

SYNOPSIS

n **delete−indentation**

DESCRIPTION

delete−indentation deletes all white characters between the beginning of the current line and the end of the previous line, including the line−feed. If the current line is not empty then a space is inserted to divide the two lines now joined.

If a positive argument *n* is given then the process is repeated *n* times. Note that the deleted characters are not added to a kill buffer.

NOTES

delete−indentation is a macro defined in `format.emf`.

SEE ALSO

delete−blank−lines(2), clean(3), kill−line(2).

delete−window(2)

NAME

delete−window − Delete the current window
delete−other−windows − Delete other windows

SYNOPSIS

n **delete−window** (**C−x 0**)
n **delete−other−windows** (**C−x 1**)

DESCRIPTION

delete−window attempts to delete the current window (remove window from the screen), retrieving the lines for use in the window adjacent to it. The command fails if there is no other window or if the current window is protected from deletion (see $window−flags(5)). The deletion protection can be overridden by giving the command a numerical argument *n* of 2.

The window deletion policy is determined by the formation of the windows displayed on the screen. The bias is for the *previous* window (above) the current window to be merged when split vertically, and for the left window to be merged when split horizontally.

delete−other−windows deletes all of the other windows, the current window becomes the only window, using the entire available screen area. Windows can be protected from deletion by using the $window−flags variable, giving the command a numerical argument *n* of 2 overrides this protection.

SEE ALSO

set−position(2), grow−window−vertically(2), resize−window−vertically(2), split−window−horizontally(2), split−window−vertically(2), $window−flags(5).

delete−registry(2)

NAME

delete−registry − Delete a registry tree

SYNOPSIS

delete−registry "*root*"

DESCRIPTION

delete−registry deletes a registry node *root* from the registry, any children belonging to the node are also deleted.

DIAGNOSTICS

delete−registry fails if *root* does not exist.

SEE ALSO

get−registry(2), list−registry(2), read−registry(2), set−registry(2), erf(8).

delete−some−buffers(2)

NAME

delete−some−buffers − Delete buffers with query

SYNOPSIS

n **delete−some−buffers**

DESCRIPTION

delete−some−buffers cycles through all visible buffers (buffers without mode hide(2m) set) and prompts the user [**y**/**n**] as to whether the buffer should be deleted. A **y** response deletes the buffer, a **n** response retains the buffer.

If a **y** response is given, the buffer has been edited, and its name does not start with a '*****' then the user is prompted as to whether the changes should be discarded. Also if the buffer has an active process running in it then confirmation is sort from the user before the process is killed.

The argument *n* can be used to change the default behavior of delete−some−buffers described above, *n* is a bit based flag where:−

0x01

Enables all checks (default). These include the initial y/n prompt on each buffer, the buffer has not been modified check, if so the user is prompted. Also if a process is running then user must confirm that the process can be killed. If this flag is not supplied then all visible buffers are killed without any user prompts (useful in macros). **SEE ALSO**

delete−buffer(2), next−buffer(2), hide(2m).

describe−bindings(2)

NAME

describe−bindings − Show current command/key binding

SYNOPSIS

describe−bindings (**C−h b**)

DESCRIPTION

describe−bindings pops up a window with a list of all the named commands, and the keys currently bound to them. Each entry is formatted as:

keyCode . . . . command

describe−bindings is buffer context sensitive and shows the bindings for the currently active buffer (i.e. the buffer that is active when the command is invoked). The resultant command list is divided into three sections as follows:

Buffer Bindings

The bindings for the active buffer when **describe−bindings** was invoked. These are the buffer bindings set by buffer−bind−key(2).

Ml Bindings

The message line bindings as set by ml−bind−key(2).

Global Bindings

Global binding of keys as set by global−bind−key(2). **EXAMPLE**

The following is an example of the displayed output from **describe−bindings**. This was invoked while editing buffer **m2fun038.2** which is the **Nroff** file for this manual page; the local bindings for the buffer are all Nroff related.

```
Buffer [m2cmd038.2] bindings:

    "C−c C−s" ..................... nroff−size
    "C−c C−r" ..................... nroff−roman
    "C−c C−b" ..................... nroff−bold
    "C−c C−i" ..................... nroff−italic
    "C−c C−c" ..................... nroff−mono
    "C−c C−o" ..................... nroff−para
    "esc o" ....................... nroff-para
    "esc q" ....................... nroff-para
    "C-c b" ....................... nroff-bold-block
    "C-c i" ....................... nroff-italic-block
    "C-c C-h" ..................... nroff-swap-hilight
    "C-c &" ....................... nroff-add-padding
    "C-x &" ....................... nroff-remove-padding
    "C-c C-p" ..................... nroff-prev
    "C-mouse-drop-1" .............. nroff-tag

Ml bindings:

    "esc esc" ..................... tab

Global bindings:

    "C-a" ......................... beginning-of-line
    "C-b" ......................... backward-char
    "C-c" ......................... 4 prefix
    "C-d" ......................... forward-delete-char
    "C-e" ......................... end-of-line
    "C-f" ......................... forward-char
    "C-g" ......................... abort-command
    "C-h" ......................... 3 prefix
    "C-i" ......................... insert-tab
    "C-k" ......................... kill-line
    "C-l" ......................... recenter
    "C-m" ......................... newline
    "C-n" ......................... forward-line
    "C-o" ......................... insert-newline
    "C-p" ......................... backward-line
    "C-q" ......................... quote-char
    "C-r" ......................... isearch-backward
    "C-s" ......................... isearch-forward
    "C-t" ......................... transpose-chars
    "C-u" ......................... universal-argument
    "C-v" ......................... scroll-down
    "C-w" ......................... kill-region
    "C-x" ......................... 2 prefix
    "C-y" ......................... yank
    "C-z" ......................... scroll-up
    "C-_" ......................... undo
    "A-e" ......................... file-browser
    "A-r" ......................... replace-all-string
    "esc C-c" ..................... count-words
    "esc C-f" ..................... goto-matching-fence
    "esc C-g" ..................... abort-command
    "esc C-i" ..................... goto-matching-fence
    "esc C-k" ..................... global-unbind-key
    "esc C-n" ..................... change-buffer-name
    "esc C-r" ..................... query-replace-string
    "esc C-v" ..................... scroll-next-window-down
    "esc C-w" ..................... kill-paragraph
    "esc C-z" ..................... scroll-next-window-up
    "esc space" ................... set-mark
    "esc !" ....................... pipe-shell-command
    "esc $" ....................... spell-word
    "esc ." ....................... set-mark
    "esc /" ....................... execute-file
```

```
"esc <" ....................... beginning-of-buffer
"esc >" ....................... end-of-buffer
"esc ?" ....................... help
"esc @" ....................... pipe-shell-command
"esc [" ....................... backward-paragraph
"esc \\" ...................... ipipe-shell-command
"esc ]" ....................... forward-paragraph
"esc ^" ....................... delete-indentation
"esc b" ....................... backward-word
"esc c" ....................... compile
"esc d" ....................... forward-kill-word
"esc e" ....................... set-encryption-key
"esc f" ....................... forward-word
"esc g" ....................... goto-line
"esc i" ....................... tab
"esc k" ....................... global-bind-key
"esc l" ....................... lower-case-word
"esc m" ....................... global-mode
"esc n" ....................... forward-paragraph
"esc o" ....................... fill-paragraph
"esc p" ....................... backward-paragraph
"esc q" ....................... fill-paragraph
"esc r" ....................... replace-string
"esc t" ....................... find-tag
"esc u" ....................... upper-case-word
"esc v" ....................... scroll-up
"esc w" ....................... copy-region
"esc x" ....................... execute-named-command
"esc y" ....................... reyank
"esc z" ....................... quick-exit
"esc ~" ....................... -30 buffer-mode
"esc A-r" ..................... query-replace-all-string
"C-x C-a" ..................... set-alpha-mark
"C-x C-b" ..................... list-buffers
"C-x C-c" ..................... save-buffers-exit-emacs
"C-x C-d" ..................... change-directory
"C-x C-e" ..................... execute-kbd-macro
"C-x C-f" ..................... find-file
"C-x C-g" ..................... abort-command
"C-x C-h" ..................... hunt-backward
"C-x C-i" ..................... insert-file
"C-x C-l" ..................... lower-case-region
"C-x C-o" ..................... delete-blank-lines
"C-x C-q" ..................... rcs-file
"C-x C-r" ..................... read-file
"C-x C-s" ..................... save-buffer
"C-x C-t" ..................... transpose-lines
"C-x C-u" ..................... upper-case-region
"C-x C-v" ..................... view-file
"C-x C-w" ..................... write-buffer
"C-x C-x" ..................... exchange-point-and-mark
"C-x C-y" ..................... insert-file-name
"C-x C-z" ..................... shrink-window-vertically
"C-x #" ....................... filter-buffer
"C-x (" ....................... start-kbd-macro
"C-x )" ....................... end-kbd-macro
"C-x /" ....................... isearch-forward
"C-x 0" ....................... delete-window
"C-x 1" ....................... delete-other-windows
```

```
"C-x 2" ........................ split-window-vertically
"C-x 3" ........................ next-window-find-buffer
"C-x 4" ........................ next-window-find-file
"C-x 5" ........................ split-window-horizontally
"C-x 9" ........................ find-bfile
"C-x <" ........................ scroll-left
"C-x =" ........................ buffer-info
"C-x >" ........................ scroll-right
"C-x ?" ........................ describe-key
"C-x @" ........................ pipe-shell-command
"C-x [" ........................ scroll-up
"C-x ]" ........................ scroll-down
"C-x ^" ........................ grow-window-vertically
"C-x `" ........................ get-next-line
"C-x a" ........................ goto-alpha-mark
"C-x b" ........................ find-buffer
"C-x c" ........................ shell
"C-x e" ........................ execute-kbd-macro
"C-x h" ........................ hunt-forward
"C-x k" ........................ delete-buffer
"C-x m" ........................ buffer-mode
"C-x n" ........................ change-file-name
"C-x o" ........................ next-window
"C-x p" ........................ previous-window
"C-x q" ........................ kbd-macro-query
"C-x r" ........................ search-backward
"C-x s" ........................ search-forward
"C-x u" ........................ undo
"C-x v" ........................ set-variable
"C-x w" ........................ resize-window-vertically
"C-x x" ........................ next-buffer
"C-x z" ........................ grow-window-vertically
"C-x {" ........................ shrink-window-horizontally
"C-x }" ........................ grow-window-horizontally
"C-h C-c" ...................... help-command
"C-h C-i" ...................... help-item
"C-h C-v" ...................... help-variable
"C-h a" ........................ command-apropos
"C-h b" ........................ describe-bindings
"C-h c" ........................ list-commands
"C-h d" ........................ describe-variable
"C-h k" ........................ describe-key
"C-h v" ........................ list-variables
"backspace" .................... backward-delete-char
"delete" ....................... forward-delete-char
"down" ......................... forward-line
"end" .......................... end-of-buffer
"esc" .......................... 1 prefix
"f1" ........................... menu
"home" ......................... beginning-of-buffer
"insert" ....................... 141 buffer-mode
"left" ......................... backward-char
"mouse-drop-1" ................. mouse-drop-left
"mouse-drop-2" ................. yank
"mouse-drop-3" ................. menu
"mouse-pick-1" ................. mouse-pick-left
"mouse-pick-2" ................. void
"mouse-pick-3" ................. void
"page-down" .................... scroll-down
```

```
"page-up" ..................... scroll-up
"redraw" ...................... screen-update
"return" ...................... newline
"right" ....................... forward-char
"tab" ......................... tab
"up" .......................... backward-line
"S-backspace" ................. backward-delete-char
"S-delete" .................... forward-delete-char
"S-tab" ....................... backward-delete-tab
"C-down" ...................... 5 forward-line
"C-left" ...................... backward-word
"C-mouse-drop-1" .............. mouse-control-drop-left
"C-mouse-pick-1" .............. set-cursor-to-mouse
"C-page-down" ................. scroll-next-window-down
"C-page-up" ................... scroll-next-window-up
"C-right" ..................... forward-word
"C-up" ........................ 5 backward-line
"A-down" ...................... 1 scroll-down
"A-left" ...................... 1 scroll-left
"A-right" ..................... 1 scroll-right
"A-up" ........................ 1 scroll-up
"esc backspace" ............... backward-kill-word
"esc esc" ..................... expand-abbrev
"C-c g" ....................... grep
```

Note that both internal commands and macro commands are shown in the list.

SEE ALSO

buffer−bind−key(2), command−apropos(2), describe−key(2), describe−variable(2), global−bind−key(2), list−commands(2), ml−bind−key(2).

describe−key(2)

NAME

describe−key − Report keyboard key name and binding

SYNOPSIS

describe−key (**C−x ?**)

DESCRIPTION

describe−key allows a key to be typed and it will report the name of the command bound to that key (if any) and the internal key−code. This command is useful when trying to locate the identity of keyboard keys for binding.

NOTES

describe−key is also bound to **C−h k**.

SEE ALSO

command−apropos(2), global−bind−key(2), describe−bindings(2), describe−variable(2).

describe−variable(2)

NAME

describe−variable − Describe current setting of a variable

SYNOPSIS

describe−variable (**C−h v**)

DESCRIPTION

describe−variable describes the current setting of the given variable (**%**, **:** and **$** variables), returning `ERROR` if the variable is undefined. If a `$` variable is not found then it is tested for an environment variable, i.e.

```
describe−variable $PATH
```

returns your environment `$PATH` setting. This is the easiest and best way of determining the current platform from within a Macro file.

The returned value of any undefined variable is the string `ERROR`.

NOTES

Completion is enabled on the command line for variable names.

SEE ALSO

describe−key(2), help−variable(2), set−variable(2).

describe−word(3)

NAME

describe−word − Display a dictionary definition of a word

SYNOPSIS

describe−word "*word*"

DESCRIPTION

describe−word can be used to interface to an external dictionary to get a definition of a given word. The interface has two modes of interface, the first simply launches an external program which provides the definition in its own user interface, e.g. MS Bookshelf. The second interface launches an external program which prints out the definition to `stdout`, MicroEmacs can then pull out the definition and display it in **describe−word**'s own GUI.

When executed **describe−word** will use the current word under the cursor as the initial *word* or will prompt the user if the cursor is not in a word.

When **describe−word**'s dialog is used the information presented is defined as follows:

Word

The word being defined, the entry can be edited and the new word will be automatically looked−up when the edit is completed.

Insert

The effect of this button is dependent on where describe−word was executed. If executed from the **Meaning** button within the spell checker the Word entry is changed to the current word. When executed outside the spell checker the definition of the current word is inserted into the current buffer.

Exit

Closes the dialog.

Main definition box

Displays the definition of the current word. The user can select a new word to describe by clicking the left mouse button on any word within the current definition. **NOTES**

describe−word is a macro implemented in `word.emf`.

Due to the size and availability of dictionaries etc. MicroEmacs is released without describe−word set up, the user must setup it up.

describe−word must be setup for each required language as follows:

1)

A command−line interface to a dictionary of the required language must be found. This could simply be a text file containing one word definition per line and using **grep(1)** as the command−line interface. In this example the text file could take the following form:

```
A () The first letter of the English...
Aam (n.) A Dutch and German measure of liquids...
Aardvark (n.) An edentate mammal...
.
.
```

The **grep** command−line interface required to look−up the word "`aardvark`" would be:

```
grep −i "^aardvark (" words.txt
```

The output produced from this will be the single line giving the required definition. A second common interface would be executing an external dictionary program typically using a command−line option to specify the word to define, e.g.:

```
mydict −d "aardvark"
```

2)

The MicroEmacs language name must be found, this can be done by first using user−setup(3) or spell−buffer(3) to ensure that the current language is set the the require one and then running **describe−word**. The command will probably fail, but before it does it will set the variable `.describe-word.lang`, use the command describe−variable(2) to get the value of this variable, this value is the internal language name. For example, when the current language is **American** or **American (Ext)** the language name is `american`.

3)

To execute the command−line interface the variable `.describe-word.`<*language*>`-command` must be set to the command−line required to obtain a word definition with the string "`%s`" used in place of the word and "`%%`" using in place of a single "`%`". For the first example in **(1)** above the following would be required:

```
set−variable .describe−word.american−command ...
        ... "grep −i \"^%s (\" /tmp/words.txt"
```

For the second example:

```
set−variable .describe−word.american−command "mydict −d \"%s (\""
```

4)

Only required for the second mode, for use with **describe−word**'s own GUI, the setting of another variable is required, the presence of this variable determines which mode is to be used.

> The variable `.describe−word.`<*language*>`−search` must be set to a regex search pattern which will match the required definition(s) in the command out put, the first group ("`\(...\)`") must enclose the required definition, again "`%s`" can be used in place of the word and "`%%`" for a single "`%`". **describe−word** simply uses regex−forward(3) repeatedly to find all definitions of the current word, it then uses the value of the variable @s1(4) to get the individual definitions. For example for the first example the following is required:
>
> ```
>     set−variable .describe−word.american−search  "^\(%s (.*\)\n"
> ```
>
> Note that the word being defined should be kept in the definition if possible as the spell rules are used to look−up base words when a derivitive of a word is not found, therefore the word being defined may not be clear (e.g. *deactivate* can be derived from *activate* but their meanings are very different). Also long text lines are automatically wrapped by the GUI.

The required variables should be added to the user setup file.

SEE ALSO

spell−buffer(3).

dir(2m)

NAME

dir − Buffer is a directory listing

SYNOPSIS

dir Mode

D − mode line letter.

DESCRIPTION

This mode can not be set and is used to indicate that the buffer is a directory listing, created by the find−file(2) command when the file name given is a directory.

SEE ALSO

find−file(2).

directory−tree(2)

NAME

directory−tree − Draw the file directory tree

SYNOPSIS

n **directory−tree** ["*directory*"]

DESCRIPTION

directory−tree creates or manipulates a view of the file systems directory structure. The command is quite complex to use directly so is largely used but macros such as file−browser(3).

The argument *n* is a bit based flag which is used to control the command, where the bits have the following meaning:−

0x01

If set, the focal directory of the command is set by the given "*directory*" argument. Otherwise the argument is not required and the command must be executed within the "**directory**" buffer; the current line sets the focal directory.

0x02

Specifies that the current line in resultant "**directory**" window should be set to the focal directory. If this bit is not set then the current line will be the last selected directory, or if none have been selected, the first line in the buffer.

0x04

Specifies that any evaluations required during the commands operation should be performed. Without this flag an open operation on a directory which has not previously been evaluated will not be perform an evaluation and the results will likely be incomplete.

0x08

Specifies that the current focal directory should be opened. This means that sub−directories within the current focal directory will also be drawn in the directory tree.

0x10

Specifies that the current focal directory should be closed. This means that sub−directories within the current focal directory will not be drawn in the directory tree.

0x20

Specifies that the current focal directory's open state should be toggled. This means that if the sub−directories are currently hidden they will now be drawn and vice−versa.

0x40

When specified any directory opened will be re−evaluated, ensuring the accuracy of the information.

0x80

Enables a recursive behavior, for example if this flag was specified with the open then not only will the focal directory be opened, but all of it's children, and their children etc. Note that if the Evaluation flag is not specified then only the already evaluated directories can be opened.

directory−tree creates a new buffer "**directory**" and draws the known directory tree. Every drawn directory is preceded by a character flag giving the user an indication of the directory state, where:

?

Directory has not been evaluated.

−

Directory has been evaluated and is visible.

+

Directory has been evaluated but is currently hidden.

Directories which have been evaluated and found to have no children use the '−' $box−chars(5) instead of a '−' character.

On UNIX platforms, if a directory is a symbolic link to another directory, the link name is given after the directory name.

EXAMPLE

The best example of the use of directory−tree is file−browser(3) which can be found in hkdirlst.emf.

SEE ALSO

file−browser(3), $box−chars(5).

display−matching−fence(3)

NAME

display−matching−fence − Display the matching bracket

SYNOPSIS

n **display−matching−fence**

DESCRIPTION

display−matching−fence draws the fence (or bracket) pairing the one the cursor is currently over. A fence is considered to be one of the following:

```
    {...}   (...)   [...]
```

If the matching fence is currently being drawn (i.e. it is visible) both fences are drawn in the '*Normal*' Matching Fence scheme (see scheme−editor(3)). If the matching fence is not currently visible the cursor is temporarily moved to the match fence for $fmatchdelay(5) milliseconds before returning to the starting position, the fences are hilighted using the Matching Fence 'Current' scheme. The matching fence delay can be interrupted by pressing any key. If the fence cannot be matched the fence is hilighted using the 'Select' scheme which is usually a bold red color.

The numeric argument **n** passed to the command is a bitwise flag where each bit is defined as follows:

0x01

Display fence (if not set nothing is done).

0x02

Use set−position id '`\x85`' instead of '`\x84`' (for internal use).

0x04

Don't Jump when matching fence is off screen.

0x08

Jump when closing a fence and its pair is off screen (for internal use).

0x10

Always jump to matching fence when closing a fence (for internal use).

0x20

Give preference to closing fence to left of cursor rather than character under the cursor (for internal use).

NOTES

This macro is used by the **Fence Display** setting of user−setup(3), the macro is bound to the `idle-pick` event using some of the more obscure numeric argument flags.

SEE ALSO

goto−matching−fence(2), user−setup(3), scheme−editor(3), $fmatchdelay(5).

display−white−chars(3)

NAME

display−white−chars − Toggle the displaying of white characters

SYNOPSIS

display−white−chars

DESCRIPTION

display−white−chars toggles the displaying of white characters in the main display. By default white characters, space tab and new−lines, are represented with invisible characters such as one or more ' 's for spaces and tabs and text moving to the next line for new−lines. The user can make this characters become 'visible' using this function.

When this function is first called it toggle enables the displaying of these characters, other characters are drawn in their place to make them visible. A subsequent call will disable the displaying of them.

NOTES

display−white−chars is a macro implemented in `misc.emf` and uses bit `0x80000` of the $system(5) variable.

The displaying of white characters can be enabled or disabled at start−up using user−setup(3).

This feature may be more confusing on some terminals due to the lack of characters available for displaying the white characters. The characters used when displaying white characters are defined in the variable $window−chars(5).

SEE ALSO

$system(5), user−setup(3), $window−chars(5).

txt(9)

SYNOPSIS

txt, doc − Plain text document file

FILES

hkdoc.emf − Plain text hook definition

EXTENSIONS

.txt − ASCII plain text file
.doc − ASCII plain text document file

DESCRIPTION

The **doc** file type template handles the hilighting and text formating of a plain text file. Within the text document justification and word wrapping are typically enabled. The template allows the user to format text as left, right, center or no justification.

Auto Layout

The automatic layout of the text is restricted to justification and wrapping and the detection of bulleted lists. fill−bullet(5) may be used to determine the character set used for bullet points, on encountering a bullet the left−hand justification might be modified.

Formatting rules

The default mode of operation is automatic mode which attempts to retain the document style whenever a paragraph is re−formatted. This allows rapid entry of text into a reasonable format with no special formating character embedded in the text.

The automatic formatting rules used by fill−paragraph(2) in an automatic text mode are defined as follows:−

Text on column 0

Text appearing in the first column is always assumed to be left justified, and non−wrapping, provided that the text does not extend to the fill column. This is typically used for headers and addresses.

Text on right edge

Text ending at the right edge (the fill−col(5)), which commences from more that 50% of the page width is assumed to be right justified, non−wrapping. Typically used for addresses.

Text centered

Text which is centered on the page is assumed to be centered, this is non−wrapping.

Indented

All other text, not covered by the cases above is assumed to be available for filling. In this case the text is filled by the paragraph and left/right justification is applied.

Short Cuts

The short cut keys used within the buffer are:−

C−c C−h − Help information on current mode.
C−c C−s − Spell the buffer.
C−c C−b − Fill both; perform left and right justification on the margins.
C−c C−b − Fill center; center the text on the current line.
C−c C−l − Fill left; fill the text on the paragraph (ragged right edge).
C−c C−r − Fill right; place text on right margin.
C−c C−o − Reduce a paragraph to a single line.
C−c a − Move to automatic formatting mode (default).
C−c l − Change mode to left formating
C−c r − Change mode to right formating
C−c r − Change mode to both formating
C−c c − Change mode to center formating
C−c n − Change mode to no formating

NOTES

To move text to a word processor then it is advised that all paragraphs are reduced to single lines, leading white space should be deleted (any possibly blank lines) and then import to the word processor. This saves considerable time as the word processor styles may be applied without handling spaces and band end of line characters.

MAGIC STRINGS

−!− document −!−

MicroEmacs specific tag, recognizes the file as a plain text document. No hilighting of the document is performed.

−!− document; sectioned −!−

MicroEmacs specific tag, recognizes the file as a document that contains sections. A crude section hilighting is enabled as follows:−

Lines commencing with > are assumed to be comments, typically used at the head of the document .

```
> −!− document; sectioned −!−
>
> Author:       My Self
> Created:      11/11/97
> Modified:     <211197.1003>
> Location:     /xx/yy/zz.doc
```

All lines commencing with start (`*`) are assumed to be bullet lists. Bullet is hilighted.

All lines commencing with `[a−zA−Z])` or `[0−9])` are assumed to be minor sections. The section number is hilighted. e.g.:

```
a) text
1) text
```

All text in single or double quotes is hilighted, assumed to be literal text. and are hilighted i.e. This is a **"double quote"** or **'a'** single quote.

Lines commencing with underscore (_) are hilighted to the end. typically used as demarcation breaks or for section underlining

−!− Document; pseudo−code −!−

The document contains pseudo code, and the pseudo code is hilighted. The pseudo−code tokens are defined as follows:−

// introduces a comment to the end of the line.

Command words comprise:−

`BEGIN`, `BREAK`, `CASE`, `CLEAR`, `CONTINUE`, `DO`, `DONE`, `ELIF`, `ELSE`, `END`, `ENDIF`, `FOR`, `FUNCTION`, `GOTO`, `IF`, `ONEVENT`, `ONINTERRUPT`, `PROCEDURE`, `REPEAT`, `RETURN`, `SET`, `SWITCH`, `THEN`, `TO`, `UNTIL`, `WHILE`,

Pseudo logical operators include

`AND`, `FALSE`, `MOD`, `NOT`, `OR`, `TRUE`, `XOR`,

−!− document; sectioned; pseudo−code −!−

A combination of both of the above. **BUGS**

The automatic mode sometimes mistakes an indented paragraph for a centered paragraph. This only typically occurs when the first line of the paragraph is not filled to the right. When the formatting error occurs, simply pad the line out so that it extends past the fill column and re−apply the formatting.

Unfortunately there is nothing that can be done to alleviate this problem, but it occurs infrequently.

SEE ALSO

fill−col(5), fill−paragraph(2), spell−buffer(3).

Supported File Types

dos2unix(3f)

NAME

dos2unix − Convert DOS format files to UNIX format files

SYNOPSIS

me "@dos2unix" <*files*>

DESCRIPTION

The start−up file `dos2unix.emf` may be invoked from the command line as a filter to convert all files in MS−DOS (or Windows) format into the correct UNIX format.

Each file specified on the command line is interrogated and the line ending modified to UNIX.

SEE ALSO

start−up(3), auto(2m), crlf(2m), ctrlz(2m).

draw(3)

NAME

draw − Simple line drawing utility

SYNOPSIS

draw

DESCRIPTION

draw provides a simple way of drawing lines into the current buffer, this has a variety of uses such as drawing tables. **draw** copies the current buffer into a temporary buffer and then allows the user to draw using simple commands until the user either aborts, discarding any changes, or exits insert the changes back into the buffer.

The keys for **draw** are defined as follows:−

esc h

Display a help dialog.

up, **down**, **left**, **right**

The cursor keys (or any other keys bound the the same commands) will move the cursor, drawing in the current mode.

d

Change the current mode to **d**raw (default), cursor movement will result in drawing in the current style.

e

Change the current mode to **e**rase, cursor movement will result in erasing to spaces.

m

Change the current mode to **m**ove, no drawing is performed with cursor movement.

u

Change the current mode to **u**ndo, cursor movement will result in undoing the character to the original or a space.

−

Sets the current horizontal line drawing style to use '−'s (default).

=

Sets the current horizontal line drawing style to use '='s.

C−g

Abort − changes are lost.

return

Exit, inserting any changes into the current buffer. **NOTES**

draw is a macro defined in `draw.emf`.

eaf(8)

NAME

eaf − MicroEmacs abbreviation file format

SYNOPSIS

<pattern> *<insertionString>*
<pattern> *<insertionString>*
<pattern> *<insertionString>*
<pattern> *<insertionString>*

DESCRIPTION

The MicroEmacs '02 abbreviation file, typically given the extension **.eaf**, defines a set of shorthand expansion strings which are used by the command expand−abbrev(2). buffer−abbrev−file(2) defines the abbreviation file.

The abbreviation file line based, with one abbreviation per line, with no intervening blank lines. Each line comprises of two columns, the first column *<pattern>* identifies the source pattern to be expanded, the second column *<insertionString>* defines the replacement text. The two text columns are separated by a space character.

When expand−abbrev(2) is invoked and the expansion *<pattern>* is recognized, then *<pattern>* is deleted from the buffer and replaced with *<insertionString>*.

The fields are defined as follows:−

<pattern>

The source pattern to be expanded. The data commences in text column 0 and spans to the first white space character (`SPACE` or `tab`). The pattern may not include any white space characters.

<insertionString>

The replacement string exists from the first non−white space character following the *<pattern>* to the end of the line. The replacement string may include special tokens, delimited by a backslash ('\') character which are interpreted as follows:−

\b Move cursor backwards

A positioning control. Allows the cursor to be moved backwards 1 character.

\d Delete `tab` backwards

Back tab. Deletes a tab character backwards.

\m"<*string*>**"** Macro execution

Takes the remainder of the line as a keyboard macro definition. The macro *string* is generated using insert−macro(2) and must be contained in double quotes. When invoked the keyboard macro is executed and the appropriate text is inserted into a buffer. This is typically only used for more complex operations.

\p Position

The resultant position of the cursor following the expansion. If the cursor position is not specified, the cursor is placed at the end of the expansion string by default.

\r Carriage Return (Newline)

A newline in the replacement text. Note while indent(2m) is enabled a sequence a single "\r" retains the indent on the next line, however a sequence of two "\r\r" characters does not retain the tab position and returns the cursor to the start of the second line. If blank lines are required retaining tab positioning then a keyboard macro string should be used instead. (see "\m" above).

\t Tab

A `tab` character in the replacement text. **EXAMPLE**

The following example provides abbreviations for the 'C' programming language, found in file **c.eaf**. All cursor positions in the examples are shown by <**@**>.

```
#i #include <\p>\r
#d #define \p
if if(\p)\r{\r\r}\r
ef else if(\p)\r{\r\r}\r
el else\r{\r\p\r}\r
wh while(\p)\r{\r\r}\r
sw switch(\p)\r{\rcase :\rdefault :\r}\r
```

Given that the abbreviation file has been declared then expansion of:

```
#d<@>       =>      #define <@>

if<@>       =>      if(<@>)
                    {

                    }

sw<@>       =>      switch(<@>)
                    {
                    case :
                    default :
                    }
```

Note, in all of the examples, the abbreviation replacement strings specify a resultant cursor position, typically where the next edit will take place.

The macros may alternatively be defined using keyboard macros. The aforementioned macros could have been re−written with the following definitions which are equivalent:−

```
#i \m"#include <\CX\CAP>\CM\CXaP\CX)"
#d \m"#define \CX)"
if \m"if(\CX\CAP)\CM{\CM}\CXaP\CX)"
ef \m"else if(\CX\CAP)\CM{\CM\CM}\CM\CXaP\CX)"
el \m"else\CM{\CM\CX\CAP\CM}\CM\CXaP\CX)"
wh \m"while(\CX\CAP)\CM{\CM\CM}\CM\CXaP\CX)"
sw \m"switch(\CX\CAP)\CM{\CMcase :\CMdefault:\CM}\CM\CXaP\CX)"
```

Within a macro, the cursor positioning is generally achieved by setting a mark where the resultant cursor is to be positioned (see set−mark(2)), when the macro is finished then an exchange−point−and−mark(2) is initiated to move the cursor to the correct position; alternatively a sequence of cursor movements may be used.

The "\b" and "\d" are typically used for positioning the cursor on subsequent lines. "\d" is the inverse of "\t". Consider the following Pascal definition for an *else*, *begin* and *end* sequence:−

```
el else\rbegin\r\t\p;\r\dend;
```

with indent(2m) mode enabled generates:−

```
        else
        begin
            <@>;
        end;
```

Similarly the "\b" is typically used when indent(2m) is enabled, but when the tab spacing is known. Consider the following example used in the MicroEmacs '02 **.emf** files to define a help entry. In this case the indent is known to be 5 characters. Hence to move the cursor back 5 characters then a sequence of **\b**'s are used:−

```
!h def .. \rSEE ALSO\r     <cross references>\r\b\b\b\b\b!ehelp
```

the expansion in this case is:−

```
define−help "<@>"

...

SEE ALSO
     <cross references>
!ehelp
```

FILES

The default abbreviation files are located in the MicroEmacs '02 *home* directory.

User's may specify their own abbreviation files by shadowing the *home* directory file with their own file located in a personal MicroEmacs '02 directory. See $MEPATH(5).

SEE ALSO

expand−abbrev(2), buffer−abbrev−file(2), global−abbrev−file(2), iso−accents−mode(3).

edf(8)

NAME

edf − MicroEmacs spelling dictionary file

SYNOPSIS

lsdmenus.edf
*user***.edf**

DESCRIPTION

The spelling dictionary files are given the extension **.edf**. These are binary files read by MicroEmacs '02 and cannot be edited directly.

MicroEmacs '02 is supplied with a dictionaries for various languages. It is recommended that these dictionaries are not modified, a personal dictionary is used and modified instead.

A personal dictionary, *user***.edf**, is automatically created in the users directory for additional spelling information.

FILES

The standard dictionary files lsdm<*language*><*country*>**.edf** are located in the MicroEmacs '02 *home* directory.

User's may create their own dictionary files by shadowing the *home* directory file with their local dictionary(s) located in a personal MicroEmacs '02 directory. See $search−path(5).

SEE ALSO

spell(2), add−dictionary(2), $search−path(5).

edit(2m)

NAME

edit − Buffer has be changed

SYNOPSIS

edit Mode

e − mode line letters.

DESCRIPTION

edit mode indicated that the buffer has been edited. Many commands and typing 'edit' the current buffer, automatically setting this mode. Commands which save these edits, such as save−buffer(2), automatically remove this mode.

A '*' character, 3 characters from the left on the mode line is used to indicate that this mode is set, see $mode−line(5). list−buffers(2) also displays the state of this mode in its output, as a '*' in the second column.

When this mode is set and undo(2m) mode is enabled, the undo(2) command can be used to undo all edits and the removal of this mode.

SEE ALSO

save−buffer(2), undo(2), list−buffers(2), $mode−line(5), undo(2m).

edit−dictionary(3)

NAME

edit−dictionary − Insert a dictionary in a buffer
restore−dictionary − Save dictionary user changes

SYNOPSIS

edit−dictionary "*dictionary*"
restore−dictionary

DESCRIPTION

edit−dictionary dumps the contents of "*dictionary*" into the temporary buffer "**dictionary**", if this buffer already exists then **edit−dictionary** simply swaps to this buffer. This enables the user to correct and prune the words in any dictionary. The given dictionary must have already been added as a main dictionary using add−dictionary(2).

The format of the created buffer is one word on each line, each word takes one of the following 3 forms:

`xxxx` − Good word `xxxx` with no spell rules allowed
`xxxx/abc` − Good word `xxxx` with spell rules `abc` allowed
`xxxx>yyyy` − Erroneous word with an auto−replace to `yyyy`

Executing **restore−dictionary** in a buffer created by **edit−dictionary** will first call delete−dictionary(2) to remove the original dictionary from memory. It then uses add−dictionary(2) to create a new dictionary with the same name and then uses spell−add−word(3) to add all the words in the current buffer into the new dictionary.

restore−dictionary does not save the new dictionary.

NOTES

edit−dictionary and **restore−dictionary** are macros defined in file `spellutl.emf`. They are not defined by default so *spellutl.emf* must be executed first using execute−file(2).

SEE ALSO

spell−add−word(3), add−dictionary(2), save−dictionary(2), delete−dictionary(2).

ehf(8)

NAME

ehf − MicroEmacs help file

SYNOPSIS

!<*helpTag*>
<*Text Description*>
...
|<*helpId*>
<*Text Description Line*>
...
$*?*
...
<*Text Description*>
!<*helpTag*>
!<*helpTag*>
<*Text Description*>
...

DESCRIPTION

The on−line help information is retained in the file **me.ehf**, this is an ASCII text file which holds all of the on−line help information. The help file comprises of formatted text <*Text Description*> which is literally displayed to the user when help information is requested. Each text description is delimited into pages with a **!**<*helpTag*> which identifies the block of text with a help label.

The **!**<*helpTag*> is placed before the text description and is identified by a exclamation mark (`**!**') placed at the beginning of the line. The <*helpTag*> is the identifying name used by the help system and takes the following form:

```
LSSNNNN...
```

Where:

L

Is the length of the "`NNNN...`" name which must be matched, a value of ' ' indicates that the whole name must be matched, otherwise the value must be in the range '`1`' − '`9`' indicating the number of characters to be match.

SS

Is the section number of the page, the first character should be a numeric (i.e. '`3`' for a macro) and the second is an optional section letter. A value of '` `' indicates no section number and/or letter.

NNNN...

The page name, the length is unlimited but must be on one line.

Multiple <*helpTag*>'s may be associated with a common text description by proceeding a block of text with multiple tags, each on a separate line, with no intervening non−tag lines (i.e. lines that do not commence with **!**).

The <*Text Description*> that follows is the text associated with the tag. When the help system is invoked with the tag then the text is displayed. There are 2 types of internal command lines, lines starting with a '`|`' indicate that the following line should only be displayed if the requested help page is <*helpId*>, where <*helpId*> is the the name used in the <*helpTag*>. This is a useful mechanism for pages with multiple <*helpTag*>s.

Lines which contain just "`$?`" are MicroEmacs command lines where `?` can be:

`a`

For a command help page display any global key bindings, for variables display its current value.

MicroEmacs uses a special hilighting scheme to control color schemes and hyper−text links, the special embedded tags all start with and escape character (0x1b or '`^[`') and are defined as follows:

`^[c?`

Tag used to change color where `?` can be:

`A`	white, used for main text.
`B`	red, used for underlining.
`C`	green, used for italic font.
`D`	cyan, used for bold font.
`E`	light yellow, used for a header.
`F`	light red, used for and image link.

`^[s?`

Tag used to change hilighting scheme where `?` can be:

`A`	Normal ehf hilight.
`B`	MicroEmacs macro (or emf) hilighting.

Note that other tags can only be used in the normal ehf hilighting scheme.

`^[ls<link>^[lm<name>^[le`

Used to create hyper−links, `<link>` is the help link name which can be omitted if it is the same as `<name>`. `<name>` should not contain any other tags, it is automatically displayed in the magenta color scheme.

NOTES

When the help system is invoked for the first time, **me.ehf** is loaded into internal memory and fragmented into labeled pages using the *<helpTag>* information. Hence, any edits made to **me.emf** are not visible in the help system until the next session.

Macros and alike may add additional help information to the internal help database at run−time using the define−help(2) command.

The help hilighting is applied to the help buffer from the hilighting macro's defined in **hkhelp.emf**. The hilighting is NOT part of the help file.

Special hilighting keys may be included in **me.ehf** provided that they are interpreted by the help hilighting defined in **hkehf.emf**.

<Text Description> lines cannot commence with **!**, **|** or **$** in the first column.

EXAMPLE

The following help entry defines the help for global−mode(2), add−global−mode(3) and delete−global−mode(3). It uses most features mentioned, namely multiple link names, color and scheme changes and several hyper−text links:

```
! 2 global-mode
! 3 add-global-mode
! 3 delete-global-mode
^[cE^[cENAME ^[cE^[cA


|global-mode
    global-mode - Change a global buffer mode
|add-global-mode
    add-global-mode - Set a global buffer mode
|delete-global-mode
    delete-global-mode - Remove a global buffer mode
$a


^[cE^[cESYNOPSIS ^[cE^[cA


    ^[cCn^[cA ^[cDglobal-mode^[cA "^[cCmode^[cA" (^[cDesc m^[cA)
    ^[cDadd-global-mode^[cA "^[cCmode^[cA"
    ^[cDdelete-global-mode^[cA "^[cCmode^[cA"


^[cE^[cEDESCRIPTION ^[cE^[cA


    ^[cDglobal-mode^[cA changes the state of one of the hereditary
    global modes. A buffer's modes are initialized to the global
    modes when first created. This command is very useful in changing
```

some of the default behavior such as case sensitive searching (see the example below). See ^[ls^[lmOperating Modes^[le for a full list and description of modes. Also see ^[ls^[lmbuffer-mode(2)^[le for a full description of the use of the argument ^[cCn^[cA.

The ^[ls^[lminfo(2)^[le command gives a list of the current global and buffer modes.

^[cDadd-global-mode^[cA and ^[cDdelete-global-mode^[cA are macros defined in me3_8.emf which use global-mode to add or remove a global mode. They are defined for backward compatibility and for ease of use; they are simple macros, add-global-mode is defined as follows:

```
^[sB
        define-macro add-global-mode
            ; Has the require mode been given as an argument, if so add it
            !force 1 global-mode @1
            !if &not $status
                ; No - use 1 global-mode to add a mode
                !nma 1 global-mode
            !endif
        !emacro

^[sA
```

^[cE^[cEEXAMPLE ^[cE^[cA

The following example globally disables ^[ls^[lmexact(2m)^[le and ^[ls^[lmmagic(2m)^[le modes, if these lines are copied to the user setup file then are searches will be simple and case insensitive by default:

```
^[sB
        -1 global-mode "exact"
        -1 global-mode "magic"

^[sA
```

^[cE^[cENOTES ^[cE^[cA

Globally adding ^[ls^[lmbinary(2m)^[le and ^[ls^[lmcrypt(2m)^[le modes is strongly discouraged as any file loaded would be assigned these modes. Instead the use of commands ^[ls^[lmfind-bfile(3)^[le and ^[ls^[lmfind-cfile(3)^[le are recommended.

^[ls^[lmauto(2m)^[le, ^[ls^[lmautosv(2m)^[le, ^[ls^[lmbackup(2m)^[le, ^[ls^[lmexact(2m)^[le, ^[ls^[lmmagic(2m)^[le, ^[ls^[lmquiet(2m)^[le, ^[ls^[lmtab(2m)^[le and ^[ls^[lmundo(2m)^[le modes are present on all platforms by default. On Windows and DOS platforms ^[ls^[lmcrlf(2m)^[le is also present and on DOS ^[ls^[lmctrlz(2m)^[le is also present.

^[cE^[cESEE ALSO ^[cE^[cA

^[ls^[lmOperating Modes^[le, ^[ls^[lmbuffer-mode(2)^[le, ^[ls^[lmfind-bfile(3)^[le, ^[ls^[lmfind-cfile(3)^[le, ^[ls^[lminfo(2)^[le.

FILES

The help file **me.ehf** is located in the MicroEmacs '02 *home* directory.

SEE ALSO

define−help(2), $MEPATH(5).

ehf(9)

SYNOPSIS

ehf − MicroEmacs '02 help file

FILES

hkehf.emf − MicroEmacs '02 help file.

EXTENSIONS

.ehf, ***help***

DESCRIPTION

The **ehf** file type template performs the hilighting of the help file. The **ehf** file is a computer generated file and uses special embedded text markers to indicate the required color scheme.

The macro file includes special macros to locate help information.

SEE ALSO

help(2).

Supported File Types

ehftools(3f)

NAME

ehftools − Generate a MicroEmacs help file

SYNOPSIS

me "@ehftools" *.htm

DESCRIPTION

The start−up file `ehftools.emf` may be invoked from the command line to generate a MicroEmacs help file from a set of HTML files (with the extension `.htm`).

The MicroEmacs documentation suite of tools has not been officially released as part of the distribution. For reference, the sequence of operations that are performed from the command line or shell script are:−

```
make meehf.hts
hts2html −l .htm meehf.hts
mv me.htm me/1.htm
cd me
me "@ehftools" *.htm
```

NOTES

The *nroff* to HTML generator leaves the special markers `<!-- XI: %s -->` in the generated HTML code which contain the hypertext link information.

SEE ALSO

start−up(3), ehf(8).

emf(8)

NAME

emf − MicroEmacs macro file

SYNOPSIS

DESCRIPTION

The MicroEmacs '02 macro files are ASCII text files, given the file extension **.emf**. A number of special macro files exist as follows:−

me.emf

The start−up macro file. This file is the first macro file to be invoked and is used to bootstrap MicroEmacs '02 into the correct configuration.

hk<*name*>**.emf**

Macro files prefixed with **hk** generally denote File Hook macro files which are automatically invoked when known file types are loaded.

<*logname*>**.emf**

The users start−up configuration file, typically used to configure the environment with the users preferences.

*****term.emf**

Platform specific configuration files, used to configure the environment for a specific platform.

Macro files may be any name, the more prominent macro files are:−

color.emf

Color definitions for the buffers.

mouse.emf

Mouse interaction macros.

osd.emf

OSD Menu configuration file. **FILES**

The default start−up file **me.emf** is located in the MicroEmacs '02 *home* directory.

User's may create their own start−up and files in their local MicroEmacs '02 directory. The users start−up file is called *$LOGNAME*.**emf**, and may be used to execute other macro files defined by the user.

SEE ALSO

File Hooks, emftags(3f), $MEPATH(5), execute−file(2).

emf(9)

SYNOPSIS

emf − MicroEmacs '02 Macro File

FILES

hkemf.emf − MicroEmacs '02 Macro File hook definition
emf.etf − Template file

EXTENSIONS

.emf − MicroEmacs '02 Macro File

DESCRIPTION

The **emf** file type template handles the hilighting of the MicroEmacs '02 macro files.

General Editing

On creating a new file, a new header is automatically included into the file. time(2m) is by default enabled, allowing the modification time−stamp to be maintained in the header.

Hilighting

The hilighting features allow commands, variables, logical, comments, strings and characters of the language to be differentiated and rendered in different colors.

Auto Layout

The indentation mechanism is enabled which performs performs automatic layout of the text. restyle−region(3) and restyle−buffer(3) are available to reformat (re−layout) selected sections of the buffer, or the whole buffer, respectively.

Tags

A C−tags file may be generated within the editor using the **Tools** −> **Emf−Tools** −> **Create Tag File**. find−tag(2) takes the user to the file using the tag information.

Folding and Information Hiding

Generic folding is enabled within the **emf** files. The folds occur about **define−macro** and **!emacro** text located on the left−hand margin. fold−all(3) (un)folds all regions in the file, fold−current(3) (un)folds the current region.

Short Cuts

The short cut keys used within the buffer are:−

C−c C−c − Comment out the current line.
C−c C−d − Uncomment the current line.
C−c esc esc − Command complete.
A−C−i − Restyle the current region.
f2 − (un)fold the current region
f3 − (un)fold all regions

BUGS

No bugs reported

SEE ALSO

emftags(3f), find−tag(2), fold−all(3), fold−current(3), indent(2), restyle−buffer(3), restyle−region(3), time(2m).

Supported File Types

emftags(3f)

NAME

emftags − Generate a MicroEmacs macro tags file

SYNOPSIS

me "@emftags" [*−v%tag−option=<flags>*] [*files*]

DESCRIPTION

The start−up file `emftags.emf` may be invoked from the command line to generate a **tags** file for MicroEmacs macro files, emf(8).

Given a list of *files* a tags file `tags` is generated in the current directory, which may be used by the find−tag(2) command. If no *files* are specified the default file list is "`./`", i.e. process the current directory. If a directory name is given (such as the default "`./`") all MicroEmacs macro files within the directory will be processed.

The value of variable **%tag−option** is used to control the tag generation process, its value *<flags>* can contain any number of the following flags:

`a`

Append new tags to the existing tag file, note that if also using flag '`m`' multiple 'tags' to the same item may be created.

`m`

Enable multiple tags. This enables the existence of 2 tags with the same tag name, but typically with different locations. See help on find−tag(2) for more information on multiple tag support.

`r`

Enables recursive mode, any sub−directory found within any given directories will also be processed.

NOTES

This function is invoked from menu

Tools −> Emf Tools −> Create Tags File

when the user requests a tags file to be generated.

The user setup file "`myemftags.emf`" is executed by emftags during start−up, this file can be used to over−ride any of the emftags configuration variables (see below).

The following variables are set within "`emftags.emf`" and are used to control the process:−

%tag−option

Tags options flag, default value is "". See above for more information.

%tag−filemask

A list of source file masks to be processed when a directory is given, default value is "`:*.emf:`".

%tag−ignoredir

A list of directories to be ignored when recursive option is used, default value is "`:SCCS/:CVS/:`".

These variables can be changed using the −v command−line option or via the "`myemftags.emf`" file

SEE ALSO

find−tag(2), start−up(3), emf(8).

start−kbd−macro(2)

NAME

start−kbd−macro − Start/stop recording keyboard macro end−kbd−macro − Stop recording keyboard macro

SYNOPSIS

start−kbd−macro (**C−x (**)
end−kbd−macro (**C−x)**)

DESCRIPTION

A keyboard macro is a short hand way to repeat a series of characters. In effect, a *recording* is made of the sequence of keys that you hit while defining a keyboard macro. The recording is started with **start−kbd−macro** and ended with **end−kbd−macro**. The recording is then repeated whenever you execute the keyboard macro using execute−kbd−macro(2).

Since it is key−strokes that are being saved, you can freely intermix commands and text to be inserted into the buffer.

You can save a keyboard macro for later using the name−kbd−macro(2) command, which saves the keyboard macro as a named macro. Otherwise if you start another keyboard macro recording session, the previously defined macro is lost. So make sure that you are done with the current keyboard macro before defining another one. If you have a series of commands that you would like to *record* for later use, insert−macro(2) can be used to insert the macro into a text file and can be reloaded using the execute−file(2) or execute−buffer(2) commands.

Recording commences with **start−kbd−macro** (`C−x (`) and terminates when an **end−kbd−macro** (`C−x )`) is encountered.

NOTES

Once **start−kbd−macro** has been executed, the mouse is disabled until **end−kbd−macro** is executed. This is because the mouse events cannot be successfully recorded in macros. The main menu can still be used, but only via the keyboard bindings and hot−keys (note that the layout of the menu may change).

SEE ALSO

execute−kbd−macro(2), insert−macro(2), kbd−macro−query(2), name−kbd−macro(2).

erf(8)

NAME

erf − MicroEmacs registry file

SYNOPSIS

; Comment to the end of the line
<command> ::= **"***<identifier>***"** [**=** **"***<string>***"**][**{** *<command>* **}**] *

DESCRIPTION

MicroEmacs '02 registry files are ASCII text files, given the file extension **.erf**. The registry file is a simple syntax that allows an *identifier* to be associated with a *string*. The *identifiers* are unique and allow a *string* value to be found when a search for a *identifier* is made. The *string* component is optional.

The syntax allows the *identifier*'s to be hierarchically nested, children of the *identifier* node are enclosed in a set of curly braces **{ ... }**. The enclosure itself comprises a number of *identifiers*, which may have their own enclosures, and so on.

The backslash character `\' is the escape character, the following sequences of escape character are recognized:−

`\\` − Literal backslash
`\"` − Double quote (used within a quoted string)
`\n` − New line character.
`\t` − Tab character.

The semi−colon character `;' introduces a comment which exists to the end of the line.

EXAMPLE

The following is an example of a registry file:−

```
; -!- erf -!-
; Comment on this line
"dos"
{
    "file-ignore" = "~ ./ .o"
    "font" = "85"
    "mail-dir" = "c:/mail/"
    "mail-send" = "echo from \"%f\" file \"%o\""
    "mail-src" = "c:/mail/jon"
    "nested"="value"
```

```
        {
            "foo"="bar"
        }
    }
```

The history file *username.erf* is a good example of the use of the registry. This file retains historical session information in The history registry file is automatically written at the end of a editing session when the editor is closed down (or may be saved explicitly using save−history(2)).

Every user should have their own personal history file in their personal MicroEmacs directory. The history file is located from the MicroEmacs '02 search path defined by $MEPATH(5), and is named by the environment variable $LOGNAME(5).

NOTES

- The registry files are not currently written with a backup.
- Special care should be taken when editing registry files when they are loaded into MicroEmacs. It is recommended that the registry file is not loaded as a registry item when editing the registry text file.

To edit the history registry file within MicroEmacs then the following sequence of steps should be followed:−

- Save the current history save−history(2).
- Load the history registry file *username*.erf.
- Edit the file.
- Save edits back to the file.
- Re−install the history read−history(2). This flushes the current session history and restores it from the file. The new edits should now be in the registry.
- Examine the loaded registry using list−registry(2).

SEE ALSO

list−registry(2), read−history(2), read−registry(2), save−history(2), save−history(2), $MEPATH(5).

erf(9)

SYNOPSIS

erf − MicroEmacs '02 registry file

FILES

hkerf.emf − MicroEmacs '02 registry file.

EXTENSIONS

.erf, ***registry***

DESCRIPTION

The **erf** file type template performs the hilighting of the registry file.

Hilighting

The hilighting features allows components of the language to be differentiated and rendered in different colors.

Auto Layout

The indentation mechanism is enabled which performs automatic layout of the text. restyle−region(3) and restyle−buffer(3) are available to reformat (re−layout) selected sections of the buffer, or the whole buffer, respectively. **SEE ALSO**

list−registry(2).

Supported File Types

etf(8)

NAME

etf − MicroEmacs template file format

SYNOPSIS

<Free Form Text>

DESCRIPTION

The MicroEmacs '02 template file, typically given the extension **.etf**, is a file template for a new file and defines common text that is automatically included when a new file is created.

The file inclusion is usually performed by macro etfinsrt(3), called from the File Hooks. The template file has no specific format, although **etfinsrt** replaces key strings with relevant information.

EXAMPLE

The template file is inserted with the file hooks. If a file hook is called with an argument of `0` then the buffer has been created and the template file is inserted.

```
define-macro fhook-c
    ; if arg is 0 this is a new file so add template
    !if &not @#
        ; Is it an include h file or a c file?
        !if &seq &mid $buffer-bname &rsin "." $buffer-bname 1 "h"
            etfinsrt "h"
        !else
            etfinsrt "c"
        !endif
    !endif
    1 buffer-mode "cmode"
    1 buffer-mode "time"
    .
    .
!emacro
```

See etfinsrt(3) for more information on how the template file is located and inserted into the buffer.

The default MicroEmacs '02 'C' mode template is defined as follows, but may be replaced with any other text:−

```
/* -*- C -*- ****************************************************************
 *
 *                        Copyright $YEAR$ $COMPANY_NAME$.
 *                              All Rights Reserved
 *
 *
 *  System        :
 *  Module        :
 *  Object Name   : m8fil001.8
 *  Created By    : $USER_NAME$
 *  Created       : $ASCII_TIME$
 *  Last Modified : <000719.1013>
 *
 *  Description
 *
 *  Notes
 *
 *  History
 *
 ****************************************************************************
 *
 *  Copyright (c) $YEAR$ $COMPANY_NAME$.
 *
 *  All Rights Reserved.
 *
 * This  document  may  not, in  whole  or in  part, be  copied,  photocopied,
 * reproduced,  translated,  or  reduced to any  electronic  medium or machine
 * readable form without prior written consent from $COMPANY_NAME$.
 *
 ****************************************************************************/

static const char rcsid[] = "@(#) : $Id$";
```

FILES

The default template files are located in the MicroEmacs '02 *home* directory.

User's may specify their own template files by shadowing the *home* directory file with their own file located in a personal MicroEmacs '02 directory. See $MEPATH(5).

SEE ALSO

File Hooks.
etfinsrt(3), &find(4).

exact(2m)

NAME

exact − Searching and sorting case sensitivity

SYNOPSIS

exact Mode

E − mode line letter.

DESCRIPTION

exact mode sets the searching and line sorting commands to case sensitive when enabled (case insensitive when disabled). See search−forward(2) and sort−lines(2).

SEE ALSO

buffer−mode(2), global−mode(2), search−forward(2), sort−lines(2).

exchange−point−and−mark(2)

NAME

exchange−point−and−mark − Exchange the cursor and marked position

SYNOPSIS

exchange−point−and−mark (**C−x C−x**)

DESCRIPTION

exchange−point−and−mark moves the cursor to the current marked position (see set−mark(2)) in the current window and moves the mark to where the cursor was. This is very useful in finding where a mark was, or in returning to a position previously marked.

SEE ALSO

set−mark(2), copy−region(2).

execute−buffer(2)

NAME

execute−buffer − Execute script lines from a buffer
execute−line − Execute a script line from the command line

SYNOPSIS

execute−buffer "*buffer−name*"
execute−line ["*command−line*"]

DESCRIPTION

execute−buffer executes script lines in the named buffer *buffer−name*. If the buffer is off screen and an error occurs during execution, the cursor is left on the line causing the error.

execute−line executes a in script line entered from the command line. Typically this is used in macros.

SEE ALSO

execute−file(2), execute−string(2), execute−named−command(2).

execute−file(2)

NAME

execute−file − Execute script lines from a file

SYNOPSIS

n **execute−file** "*file*" (**esc** /)

DESCRIPTION

execute−file executes script lines from the given *file n* times in succession, this is the normal way to execute a MicroEmacs '02 script. The command prompts for a file name, and will then search for <*file*>[.emf] in the search path. If the file is found then the file is loaded and the buffer is executed *n* times.

SEE ALSO

execute−buffer(2), execute−line(2), execute−named−command(2), execute−string(2).

execute−kbd−macro(2)

NAME

execute−kbd−macro − Execute a keyboard macro

SYNOPSIS

n **execute−kbd−macro** (**C−x e**)

DESCRIPTION

execute−kbd−macro executes a keyboard macro. The entire sequence of recorded key−strokes is repeated starting at the current point. The result is exactly as if you were retyping the sequence all over again. A numeric argument *n* prefixing the **execute−kbd−macro** command repeats the stored key−strokes *n* times.

Keyboard macros are recored with start−kbd−macro(2); recording is terminated with end−kbd−macro(2).

SEE ALSO

end−kbd−macro(2), kbd−macro−query(2), name−kbd−macro(2), start−kbd−macro(2).

execute−named−command(2)

NAME

execute−named−command − Execute a named command

SYNOPSIS

n **execute−named−command** "*command−string*" esc x

DESCRIPTION

execute−named−command command prompts the user for the name of a command to execute and then executes the command *n* times. MicroEmacs '02 offers command completion and history facilities, see ml−bind−key(2).

SEE ALSO

execute−buffer(2), describe−bindings(2), ml−bind−key(2).

execute−string(2)

NAME

execute−string − Execute a string as a command

SYNOPSIS

n **execute−string** "*string*"

DESCRIPTION

execute−string executes the given *string* *n* times as if it is being typed. This is the writable format of a keyboard macro, it can be placed in any **emf** file. Any characters may form the *string* (unprintables as `\xXX`) and key−strokes that are bound to a command will execute that command. This command is used by macros to store user defined keyboard macros.

EXAMPLE

The following example uses keyboard strokes with **execute−string** in a macro to format **nroff(1)** text located between `.` commands:

```
define−macro nroff−para
    beginning−of−line
    !if &not &sequal @wc "."
        1 buffer−mode "magic"
        execute−string "\CXS^\\.\CM\CB\CM\CX\CH\CN\CM"
        −1 fill−paragraph
        execute−string "\CD\CX\CH\CN\CD\CXH\CB"
    !endif
    forward−line
!emacro
```

execute−string has the advantage that execution is very fast as the amount of parsing and decoding to be performed is limited. The disadvantage is that you cannot quickly discern which operations are being performed !!

NOTES

Try to avoid using named key, such as "`up`" and "`return`", as the keyboard macro equivalent is not readable and is likely to change in future releases.

For this reason the following special abbreviations may be used

\E

The "**escape**" key.

\N

The "**return**" key.

\T

The "**tab**" key.

\b

The backspace character (0x08).

\d

The delete character (0x7f).

\e

The escape character (0x1b).

\f

The form−feed character (0x0c).

\n

The carriage−return character (0x0a).

\r

The line−feed character (0x0d). **SEE ALSO**

buffer−abbrev−file(2), global−abbrev−file(2), insert−macro(2), name−kbd−macro(2), start−kbd−macro(2).

execute−tool(3)

NAME

execute−tool − Execute a user defined shell tool

SYNOPSIS

n **execute−tool** "*tool−name*"

DESCRIPTION

execute−tool launches a predefined shell tool, the tools are typically defined by the user−setup(3) Tools page and executed using the MicroEmacs main Tools menu. See help on user−setup(3) for more information on the basic facilities given by execute−tool.

If the numeric argument *n* is supplied it is used as the tool name to be executed, otherwise the argument "*tool−name*" must be given.

A tool with a numeric name can be executed via a key binding, for example, to execute tool **3** (as defined by **user−setup**) to '`C−3`' add the following line to the user setup file:−

```
3 global-bind-key execute-tool "C-3"
```

NOTES

The registry entries for a tool must be located in registry directory "`/history/`**$platform/tool/tool−name**" where **$platform** is the current setting of variable $platform(5) and **tool−name** is the name of the tool as given to the command. The following registry entries are used:−

name

The name of the tool as displayed in the user−setup Tools dialog and the Main Tools menu. This is only used for tools `0` to `9`.

command

The command−line to be launched when the tool is executed, the following special tokens may be used in the command−line which are substituted at execution:−

%ff

The current buffer's full file name, including the path.

%fp

The current buffer's file path.

%fn

The current buffer's file name without the path.

%fb

The current buffer's file base name, i.e. the file name without the path or the extension.

%fe

The current buffer's file extension with the '.' (e.g. "*.emf*"), set to the empty string if the file name does not have an extension.

Note that "**%ff**" is always the same as "**%fp%fn**" and "**%fp%fb%fe**". If any of these tokens are used, the tool will fail to execute if the current buffer does not have a file name.

flag

A bit based flag setting the tool characteristics, where:−

0x01

Enable current buffer saving.

0x02

Enable prompt before saving current buffer.

0x04

Enable all edited buffers saving.

0x08

Enable prompt before saving an edited buffer.

0x10

Enable output capturing.

0x20

Enable concurrent running, not available on all platforms, see variable $system(5).

bname

The name of the buffer to be used if the output is captured. The following special tokens may be used in the buffer name which are substituted at execution:−

%fn

The current buffer's file name without the path, set to the buffer name if the current buffer does not have a file name.

%fb

The current buffer's file base name, i.e. the file name without the path or the extension. Set to the buffer name if the current buffer does not have a file name.

%fe

The current buffer's file extension with the '.' (e.g. "*.emf*"), set to the empty string if the current buffer does not have a file name or it does not have an extension.Note that "**%fn**" is always the same as "**%fb%fe**". Default buffer name when this field is left empty is "**command**", or "**icommand**" if `Run Concurrently` is enabled.

If more than 10 tools are required (maximum number definable by **user−setup**) or names are preferred, it is recommended that the **user−setup** dialog is used to define the tool and then use the registry copy utility bound to '`C`' in a list−registry(2) buffer.

SEE ALSO

user−setup(3), ipipe−shell−command(2), pipe−shell−command(2), shell−command(2), system(5).

exit−emacs(2)

NAME

exit−emacs − Exit MicroEmacs

SYNOPSIS

n **exit−emacs**

DESCRIPTION

Exit MicroEmacs back to the operating system. If no argument *n* is given and there are any unwritten, changed buffers, the editor prompts the user to discard changes. If an argument is specified then MicroEmacs exits immediately.

NOTES

All buffers with a name starting with a '*' are assumed to be system buffers (i.e. ***scratch***) and are not saved.

SEE ALSO

quick−exit(2), save−buffers−exit−emacs(2).

expand−abbrev(2)

NAME

expand−abbrev − Expand an abbreviation

SYNOPSIS

expand−abbrev

DESCRIPTION

expand−abbrev expands an abbreviation to an alternate form. The abbreviation must be an alpha−numeric string and the cursor must be one position to the right of the abbreviation (which must not be alpha−numeric) when this command is called. If the abbreviation is found, it is deleted and the alternate form is inserted leaving the cursor at the end of the insertion unless `\p` is used. If not found, a space is inserted.

SEE ALSO

buffer−abbrev−file(2), global−abbrev−file(2), expand−abbrev−handle(3), eaf(8).

expand−abbrev−handle(3)

NAME

expand−abbrev−handle − Expand an abbreviation handler

SYNOPSIS

expand−abbrev−handle (**esc esc**)

DESCRIPTION

expand−abbrev−handle pulls together all forms of abbreviation expansion into a single command so that it can be bound to a single key. The abbreviation must be an alpha−numeric string and the cursor must be one position to the right of the abbreviation (which must not be alpha−numeric) when this command is called. The command attempts to expand the abbreviation using the following commands in turn:

expand−abbrev(2)

Uses a buffer specific and global abbreviation files, if set, to look up the abbreviation. The use of the abbreviation file can be disabled using buffer−setup(3).

expand−iso−accents(3)

Expands ISO accent letter if the expansion mode is enabled via either the user−setup(3) General Page or by using the iso−accents−mode(3) command.

expand−look−back(3)

Looks for a word starting the same in the current buffer's last 100 lines, this can be enabled in the user−setup(3) General page.

Buffer specific expansion

Executes a buffer specific abbreviation expansion if the current buffer's file hook supports abbreviation expansion.

Word expansion

If the current buffer does not support file type specific expansion and Word Expansion is enabled via the user−setup(3) General page (`Dict'n` setting) expansion is attempted using the expand−word(3) command which expands the current partial word using the dictionary of the user's current language; warning − this can be slow!

The command exits after first command to successfully expand or if none expand the command fails. See the help in the individual expansion commands for more help.

SEE ALSO

user−setup(3), expand−abbrev(2), expand−iso−accents(3), expand−look−back(3), expand−word(3).

expand−iso−accents(3)

NAME

expand−iso−accents − Expand an ISO accent
iso−accents−mode − Enable/disable ISO accent expansion short−cut mode

SYNOPSIS

expand−iso−accents
n **iso−accents−mode**

DESCRIPTION

expand−iso−accents provides a facility to enter a plain text representation of an ISO accent and then to expand it into a proper ISO accented character. For example:−

`` `a `` => small a, grave accent
`^a` => small a, circumflex accent
`'a` => small a, acute accent
`"a` => small a, umlaut
`~a` => small a, tilde
`.a` => small a, ring
`14` => fraction, one−quater
`12` => fraction, one−half
`34` => fraction, three−quaters
`ae` => ae ligature
`sz` => small sz ligature, German.
`+-` => plus or minus (math.)
`co` => copyright
`rg` => registered trademark
`tm` => trade mark
`oe` => small oe ligature
`/o` => small o, slash

The **expand−iso−accents** can be called directly to expands the ISO abbreviated character sequence into it's ISO ASCII character equivalent. The command looks at the 2 characters to the left of the cursor and tries to find a matching abbreviation, if found the 2 characters are removed, replaced by the single ISO character.

The more typical way of using this feature is by enabling its use in the abbreviation handler which is usually bound to "`esc esc` ". It can be enabled by either by using the **iso−accents−mode** command or, for a more permanent installation, from the user−setup(3) => General => Abbrev Expansion settings.

When using the **iso−accents−mode** command, if a numeric argument *n* is given (the value is not used) then the ISO accent expansion is installed locally into the current buffer. If *n* is omitted then expansion is enabled/disabled globally (across all buffers).

expand−iso−accents is the macro command that This is by default bound to . If an ISO character is not located then expand−abbrev(2) is invoked to try a standard abbreviation.

NOTES

iso−accents−mode and **expand−iso−accents** are implemented as macros in the file `abbrev.emf`, the repertoire of expansions may be enhanced by editing this file.

Unlike the general expand−abbrev(2) command which attempts to expand the current word, **expand−iso−accents** only considers the last two characters regardless of whether they are word characters or start a word. Therefore the general **expand−abbrev** command cannot be used to implement a similar feature.

SEE ALSO

expand−abbrev−handle(3), buffer−abbrev−file(2), expand−abbrev(2), expand−look−back(3), expand−word(3).

expand−look−back(3)

NAME

expand−look−back − Complete a word by looking back for a similar word

SYNOPSIS

expand−look−back

DESCRIPTION

expand−look−back attempts to complete the word at the current position by looking backward for another word which starts the same. If such a word is found within 100 lines of the current cursor position the current partial word is replaced with the word found.

expand−look−back is automatically invoked from the expand−abbrev−handle(3) macro in response to an expansion command, it is only invoked if enabled in the user−setup(3) => General => Abbrev Expansion => Lookbk setting is enabled.

NOTES

expand−look−back is a macro implemented in `abbrev.emf`.

The **user−setup** configuration simply sets the macro variable `.expand−look−back.on` to TRUE, i.e.:

```
set-variable .expand-look-back.on 1
```

It may be subsequently disabled by setting the variable back to 0.

SEE ALSO

expand−abbrev−handle(3), user−setup(3).

expand−word(3)

NAME

expand−word − Complete a word by invocation of the speller

SYNOPSIS

expand−word

DESCRIPTION

expand−word attempts to complete the word at the current position through the use of the current language dictionary. The user is presented with a list of endings for the given word portion. These may be selected with the cursor or mouse.

expand−word is automatically invoked from the expand−abbrev−handle(3) macro in response to an expansion command, it is only invoked if enabled in the user−setup(3) => General => Abbrev Expansion => Dict'n setting is enabled.

NOTES

expand−word is a macro implemented in `abbrev.emf`.

The **user−setup** configuration simply sets the macro variable `.expand-word.on` to TRUE, i.e.:

```
set-variable .expand-word.on 1
```

It may be subsequently disabled by setting the variable back to 0.

SEE ALSO

expand−abbrev−handle(3), spell−buffer(3), find−word(3).

f(9)

SYNOPSIS

f, f77, f90 − Fortran files

FILES

hkf90.emf − Fortran hook definition
f90.etf − Fortran 90 template file.
f.etf − Fortran (77) template file.

EXTENSIONS

.f − Fortran file
.f77 − Fortran 77 file
.f90 − Fortran 90 file

DESCRIPTION

The **f90** file type templates provide simple hilighting of Fortran 77 and Fortran 90 files, the template provides minimal hilighting of both language syntaxes, which are overloaded into the same file.

The major difference between the file types, apart from the new reserved words, is the comments. In Fortran 90 comments are introduced with **!**, while the other types use a **c** in column 0.

BUGS

The Fortran hilight file is in it's infancy and a number of it's tokens may be misplaced.

SEE ALSO

Supported File Types

fence(2m)

NAME

fence − Auto fence matching mode

SYNOPSIS

fence Mode

f − mode line letter.

DESCRIPTION

fence mode can be used to enable or disable the automatic displaying of and open fence when the corresponding closing fence is typed. When the mode is enabled and the closing fence is typed the cursor is temporarily move to the position of the opening fence. The duration of the move can be controlled by the $fmatchdelay(5) variable; any user input interrupts the display.

If cmode(2m) is also enabled the search algorithm used is '*C*' aware and if a matching fence is not found then the bell is rung as a warning. If **cmode** is not enable any closing fence which cannot be matched is ignored.

NOTES

The following characters are considered closing fences:

```
} ) ]
```

These are match with the following opening fences respectively:

```
{ ( [
```

SEE ALSO

$fmatchdelay(5), cmode(2m), goto−matching−fence(2).

file−attrib(3)

NAME

file−attrib − Set the current buffers system file attributes

SYNOPSIS

file−attrib

DESCRIPTION

file−attrib opens a dialog enabling the user to change the system properties of the current buffer's file. Top of the dialog give the current buffer name and its file name. The `Save Changes` button writes the current buffer out with any current edits and changes to its file attributes. The `Ok` button closes the file−attrib dialog, any changes made to the file attributes will be applied next time the buffer is written.

The type allow the changing between UNIX, MS Windows and DOS text file formats. UNIX has a single new line character ('`\n`') where as Windows and Dos have a double new line character ('`\r\n`'). Also a Dos text file is terminated with a `C−z` (0x1A) character which the other two do not. These attribute are set in MicroEmacs by using buffer modes crlf(2m) and ctrlz(2m).

The central part of the dialog contains system dependent attributes which are defined as follows:

UNIX Platforms

Allow the setting of user, group and global, read, write and execute permissions, see man pages on **chmod(1)** for more information. This is a front end to setting the variable $buffer−fmod(5).

Win32 Platforms

Allow the setting of MS Windows file attributes, i.e. read−only, hidden, archive etc. Note that the directory attribute is displayed but cannot be altered. This is a front end to setting the variable $buffer−fmod(5).

DOS Platform

Allow the setting of MS Dos file attributes, i.e. read−only, hidden, archive etc. Note that the directory attribute is displayed but cannot be altered. **NOTES**

file−attrib is a macro implemented in `fattrib.emf`.

SEE ALSO

find−file(2), write−buffer(2), crlf(2m), ctrlz(2m), $buffer−fmod(5).

file−browser(3)

NAME

file−browser − Browse the file system file−browser−close − Close the file−browser file−browser−swap−buffers − Swap between file−browser windows

SYNOPSIS

file−browser (**f10**)
file−browser−close
file−browser−swap−buffers

DESCRIPTION

file−browser can be used to browse around the file system. When first executed **file−browser** creates 2 buffers, "`*directory*`" displaying the directory structure and "`*files*`" listing the files in the current directory with information on each file. **file−browser** displays these buffers side by side, splitting the current window horizontally if required.

Once open the user can browse through the system using the following keys in the `*directory*` buffer:

`space`

Selects the directory on the current line and up−dates the `*files*` buffer with the information on this directory. This can also be done by clicking the left mouse button on the directory name.

`return`

Selects the directory on the current line, if open (sub−directories displayed) then closes it or if closed it is opened. The `*files*` buffer is up−dated with the information on the directory. This can also be done by clicking the left mouse button on the '`+`' or '`−`' symbol just before the directory name.

`C-return`

As with `return` expect sub−directories are recursively opened or closed, note that this could take some time on large file systems. This can also be done by clicking the right mouse button on the '`+`' or '`−`' symbol just before the directory name.

`tab`

Move to the `*files*` buffer.

`delete`

Closes file−browser.

The following keys can be used in the `*files*` buffer:

`return`

If the current line is a directory, this because the current directory, updating both the `*directory*` and `*files*` buffers. If the line is a file then it is opened using find−file(2). This can also be done by clicking the left mouse button on the file name.

`space`

Toggles the tag state of the file on the current line, see `X` command. This can also be done by clicking the left mouse button anywhere before the file name, or for multiple files drag a region with the left mouse button.

`X` or `x`

Executes a shell−command(2) on all tagged files. The user is prompted for the command line which can contain the following special tokens:

`%p`	Full file name, including path.
`%f`	The file name without the path.

As the **shell−command** is executed in the directory `%f` is safe to use in a command such as "`del %f`".

`D` or `d`

Deletes all the tags in the buffer.

`tab`

Move to the `*directory*` buffer.

`delete`

Closes file−browser.

file−browser−swap−buffers swaps between the `*directory*` and `*file*` windows, making the other the current window, this is usually locally bound to the `tab` key.

file−browser−close hides both the `*directory*` and `*file*` windows, closing the file−browser, this is usually locally bound to the `delete` key.

SEE ALSO

directory−tree(2), find−file(2), shell−command(2).

file−op(2)

NAME

file−op − File system operations command

SYNOPSIS

n **file−op** [(["*from−file*" "*to−file*"]) |

(["*delete−file*"]) | (["*dir−name*"])] **DESCRIPTION**

file−op can be used to perform numerous file system operations. The given argument *n* must be used to determine the required operation, the value is a bit based flag denoting the operation as follows:

0x010

Log−off and close down the current ftp connect (not a file system operation but functionality was required and it had to go somewhere).

0x020

When this bit is set the command functionality is changed to delete−file, the single argument *delete−file* is deleted.

0x040

When this bit is set the command functionality is changed to move−file, the specified *from−file* is moved to *to−file*.

0x080

When this bit is set the command functionality is changed to copy−file, the specified *from−file* is copied to *to−file*.

0x100

When this bit is set the command functionality is changed to making a new directory, the specified *dir−name* is the name of the new directory. A file or directory of the given name must not already exist.

Only one operation can be performed per invocation. The following bits in the given argument *n* can be used to effect the behaviour of these operations:

0x01

Enables validity checks, these include a check that the proposed file does not already exist, if so confirmation of writing is requested from the user. Also MicroEmacs checks all other current buffers for one with the proposed file name, if found, again confirmation is requested. Without this flag the command will always succeed wherever possible.

0x02

Creates a backup of any file about to be deleted or over−written. Set help on $buffer−backup(5) for backup file−name generation. **NOTES**

http files are not supported except as the source file when copying. **ftp** files are fully supported with the restriction that the from and to files cannot both be url (http or ftp) files.

The command is used by file−browser(3) and ftp(3) which provides an easy to use interfaces for file manipulation.

SEE ALSO

file−browser(3), ftp(3), find−file(2), write−buffer(2), $temp−name(5).

fileHooks(2)

FILE HOOKS

File hooks provide a mechanism to automatically invoke a set of macros for a given buffer type when the following events occur:

- Loading of a file into a buffer
- Moving into a buffer (i.e. making a buffer current)
- Moving out of a buffer (i.e. making another buffer current)
- Deleting an active buffer

The file hook selection (see below) is performed on the file name / extension and on the textual content of the buffer using add−file−hook.

Refer to Language Templates for a description of how the file hooks are used to define a new template for a new text format.

The hook macros allow buffer modes and highlighting, applicable to the text type of the file, to be applied to the buffer. In addition, the associated hook macros may be located in a separate file and are loaded on demand when the file reading determines that a set of hook macros are required.

Consider a file hook definition of the form;

```
add−file−hook ".c .h" "fhook−c"
```

which binds the file hook **fhook−c** to any files that are loaded with the extension **.c** and **.h**. The operations undertaken by MicroEmacs '02 are defined as follows when a file `foo.c` is loaded:−

- Attempt to load file `foo.c`, if `foo.c` is not found then create a new buffer and assign file name `foo.c`.
- If `foo.c` is found then load file into buffer. Search the first line(s) of the buffer for magic hook text (*add−file−hook* with argument).
- If magic hook was not found then determine hook name from the file extension (*add−file−hook* information).
- If a hook command is located, assign the file hook **fhook−c** to the buffer, assign the buffer entry (begin) hook macro of **bhook−c**; assign a buffer exit hook of **ehook−c**.
- If the macro **fhook−c** is undefined then execute the macro file **hkc.emf** from the MicroEmacs home directory in an attempt to load the macro. If the file **myc.emf** is defined, then the modifications to the language template are applied after **hkc.emf** is loaded.
- If the macro **fhook−c** is (now) defined then `foo.c` is TEMPORARILY made the current buffer and the file hook macro **fhook−c** is executed to completion and the previous current buffer is restored. [*TEMPORARY* here implies that no buffer hooks are executed on the flip in/out of `foo.c`].
- The current buffer is officially swapped to `foo.c`. At this point the *ehook* of the old current buffer is executed (while its still current) and then `foo.c` is swapped in to become the current buffer; the begin buffer hook *bhook−cmode* is then executed for `foo.c` (if it exists).

- If the user moves to another buffer execute the end hook macro **ehook−cmode** (if it exists) and move to the new buffer, executing it's begin hook.
- If the user subsequently returns to buffer `foo.c` execute the previous buffers end hook macro, set the current buffer to *foo.c* and execute the begin hook macro **bhook−c** (if it exists).
- If the user kills buffer `foo.c`, if `foo.c` is the current buffer then an alternative buffer is made current, `ehook` and `bhook` executed as normal. If macro **dhook−c** is defined then `foo.c` is TEMPORARILY made the current buffer and the delete hook macro **dhook−c** is executed to completion and the previous current buffer is restored.

The name of the file hook macro name is important, hook commands must commence with the text **fhook−***mode* where *mode* is an identifier for the operating mode. The name space is decomposed as follows:−

- The initial **f** is removed and replaced with **b** for the begin hook macro and **e** for the end hook macro.
- When the **fhook** macro is undefined the *mode* component is removed and the file **hk***mode***.emf** is executed from the MicroEmacs home directory in an attempt to define the macro.

The **fhook−** nomenclature may be omitted provided that the name is less than 6 characters, however the file, begin and end hook macros MUST commence with **f**, **b** and **e** respectively. In addition the macros must be defined as no auto file loading is performed.

Buffer Hook Variables

The macros bound to a buffer may be interrogated, the variables $buffer−fhook(5), $buffer−bhook(5), $buffer−ehook(5) and $buffer−dhook(5) contain the names of any associated macro attached as a macro hooks, defining the *file*, *begin*, *end* and *delete* hooks respectively. If a macro is not bound then the empty string `""` is returned. Setting the variables has the effect of defining the hook and is a method by which the buffer hooks may be affected after the buffer has been loaded.

Determination of a new file

The *file* hook **fhook−XXX** numeric argument may be used to determine if the file associated with a buffer is a new file created by the user, or an existing file. Typically this distinction is used to determine whether a boiler template is added to the file or not. The macro argument **@#** is defined as zero (0) if this is a new file that has been created, or non−zero otherwise.

The macro argument status is typically tested on entry to the macro as follows:−

```
define-macro fhook-mode
    !if &not @#
        ; This is a new file. Do new file things
    !else
        ; This is an existing file
    !endif
    ; Set up bindings
!emacro
```

An example of a generic **hook** file is given at the end of this section which elaborates on the file hooks.

Begin and End hooks

The *begin* and *end* hooks are usually used to save and restore global states which require special settings for a particular buffer type. This typically involves saving and restoring global variables which are used by other buffers in a different configuration. For example the following is used to reformat the time stamp string; the time stamp is a global variable $timestamp(5) and if it is changed in one buffer, it must be restored ready for another. In this case the old time stamp is retained in a local buffer variable whenever the buffer is entered, the time stamp is then modified for the buffers requirements. On exit from the buffer the old time stamp format is restored to it's former state.

```
0 define-macro bhook-foo
    set-variable .timestamp $timestamp      ; Save old time stamp.
    set-variable $timestamp "19%Y/%M/%D %h:%m:%s"
!emacro

0 define-macro ehook-foo
    set-variable $timestamp .bhook-foo.timestamp
!emacro
```

Note that in both cases the define−macro(2) invocation is defined as zero, this merely hides the macro from the command line since both are private macros not normally invoked by the user.

FILE HOOK SELECTION

MicroEmacs '02 may be reconfigured to operate in different modes (referred to a *Major Modes* in GNU **emacs(1)**) using the macro file hooks. The file hooks allow the working environment to be customized for the editing of text of a particular sort, by importing text specific macros, key rebinding and highlighting.

MicroEmacs '02, by default, loads a file into a buffer with default global modes with no highlighting. There are no mode specific key bindings, variable settings, macros or highlights, buffer interaction behaves in it's default state. The state of the buffer interaction may be modified through the use of the buffer modes (see Operating Modes), for example the 'C' programming language cmode(2m) changes the characteristics of the `tab` character and performs language specific indentation of statements. When a text specific set of highlighting rules are applied to the buffer, the text becomes emphasized through the use of color applied selectively to the text i.e. comments, keywords, strings are shown in different colors, allowing them to be differentiated without studying the content.

Setting the operating mode of the buffer would be tedious to perform from the command line, instead MicroEmacs '02 uses three different prioritized criteria to endeavor to select the correct operating mode. The operating mode is applied to the buffer by execution of a set of file specific macros, referred to a hook commands. The selection criteria of the hook commands is performed as follows, ordered in lowest to highest priority:−

File Name

MicroEmacs '02 uses the filename and/or the file extension to select a start−up hook command. File names and extensions are bound to a set of macro hooks in a space separated list e.g.

> add−file−hook "**c cpp**" "**fhook−cmode**"
> add−file−hook "**doc txt README**" "**fhook−doc**"

The space separated list of names are interpreted as either file extensions or filenames. In this case any file with the extension **.c**, **.cpp** is bound to a file hook called **fhook−cmode** e.g. `foo.c`. Similarly files with the extension **.doc** or **.txt** are interpreted as plain text documents and are bound to **fhook−doc**. e.g. `foo.txt`. The entry **README** that exists in the documentation hook list may refer to a file `README` and also `foo.README`, both cases invoke the document hook.

The file selection is the lowest priority selection criteria but usually satisfies most mode selection requirements.

Magic Strings

There are cases when file extensions may be omitted from files, typically these files include an identifier, or magic string, on the first line of the file which is used to identify the file to the operating system or application e.g. shell scripts under UNIX. MicroEmacs '02 automatically interrogates the top of every file that is loaded to locate some form of identification string. The identification strings are defined in a similar way to the file name hooks, except instead of defining a file extension the location and text content of the identifier is defined:

> 1 add−file−hook "**#!/bin/sh**" "**fhook−shell**"
> 1 add−file−hook "**#!/usr/local/bin/wish**" "**fhook−tcl**"

In this case, any file that commences with "**#!/bin/sh**" is interpreted as a shell script and invokes the shell hook **fhook−shell**. Where the identifier does not appear on the first non−blank line, the argument may be increased to the number of lines to be searched. Also it the magic sting should be search for without exact(2m) mode then the argument should be negated, e.g.

> −4 **add−file−hook** "<html>" "**fhook−html**"

invokes **fhook−html** whenever "`<html>`", "`<HTML>`" etc. is found in the first 4 lines of a file header, e.g.:

```
<!-- Comment line -->
<HtMl>
```

A match on a string identifier is assigned a higher priority than the file extension. It is recommended that magic strings are only used where there are no predefined file extensions, or conflicts exist between files with the same extension containing data interpreted in a different context.

Explicit Strings

The last method allows an explicit identifier string to be embedded into the text of the file informing MicroEmacs '02 which mode it should adopt. GNU Emacs supports this (see **Major Mode** in the GNU Emacs documentation) type of operation by insertion of strings of the form:

> **−*−** *mode* **−*−**

Where *mode* represents the major mode within GNU Emacs. The same format as used by **Magic Strings** can be used to find and extract the *mode*, e.g.:

> **−1** add−file−hook "**−[*!]−[\t]nroff.*−[*!]−**" "**fhook−nroff**"

The definition would detect the GNU Emacs mode defined in an Nroff file e.g.

```
.\" −*− nroff −*− "
.TH man 1
.SH NAME
...
```

It should be stressed that the **−*−** syntax belongs to GNU Emacs and NOT MicroEmacs '02, MicroEmacs '02 provides a mechanism to locate, extract and interpret the string. The **−*−** syntax should only be applied to files if it is known that the *mode* is a GNU mode.

A MicroEmacs '02 specific string is also provided, defined as:

> **−!−** *mode* **−!−**

where *mode* is an arbitrary string defined by *add−file−hook*. User defined modes may be created and assigned to files with this syntax, this does not conflict with the GNU Emacs command. For example to assign a new mode *mymode* to a file we would define the following:−

> **−1** add−file−hook "**−!−[\t]mymode.*−!−**" "**fhook−mymode**"

Files containing a the following identifier would be loaded with *mymode* hook:

```
# −!− mymode −!−
#
# Last Modified:  <120683.1014>
```

FILE HOOK SCRIPTS

The buffer hook files **hk***name***.emf** typically follow a standard layout, and are generally associated with hi−lighting as follows, **mode** in this case is the name of the file mode associated with the file:−

```
!if &seq .hilight.mode "ERROR"
```

```
    set-variable .hilight.mode &pinc .hilight.next 1
!endif
;
; Define the hilighting
;
0 hilight .hilight.mode 1                $global-scheme
hilight .hilight.mode 2 "\*\*"           .scheme.comment
hilight .hilight.mode 4 "\"" "\"" "\\"   .scheme.string
hilight .hilight.mode 0 "'.'"            .scheme.quote

hilight .hilight.mode 1 "if"             .scheme.keyword
hilight .hilight.mode 1 "elif"           .scheme.keyword
hilight .hilight.mode 1 "else"           .scheme.keyword
...

; Reset the hilighting printer format and define the color bindings.
0 hilight-print .hilight.mode
hilight-print .hilight.mode "i"  .scheme.comment
hilight-print .hilight.mode "b"  .scheme.keyword .scheme.variable
hilight-print .hilight.mode "bi" .scheme.string .scheme.quote
...

; Define the indentation tokens
0 indent  .hilight.mode 2 10
indent .hilight.mode n "if" 4
indent .hilight.mode s "elif" -4
indent .hilight.mode s "else" -4
indent .hilight.mode o "endif" -4
indent .hilight.mode n "while" 4
...

define-macro fhook-mode
    ; if arg is 0 this is a new file so add template
    !if &not @#
        etfinsrt "mode"
    !endif
    set-variable $buffer-hilight .hilight.mode
    set-variable $buffer-indent .hilight.mode
    1 buffer-mode "time"
    1 buffer-mode "indent"
    buffer-abbrev-file "mode"
!emacro
```

The previous example shows how the **fhook−mode** numeric argument is used to determine if this is a new file. If the argument **@#** is zero then this is interpreted as a new file, in this case a standard template is inserted (from file **mode.etf**) and the generic strings such as $YEAR$ replaced with construction information. The template is generally used for standard headers and skeleton text body.

In addition an abbreviation file **mode.eaf** (see eaf(8)) is bound to the buffer using the buffer−abbrev−file(2) command and the buffer hi−lighting enabled by assignment of the $buffer−hilight(5) variable.

MODIFYING FILE HOOKS

The standard hook files supplied with MicroEmacs '02 should not be modified, changes to the file hooks may be applied using a separate macro file called **my***XXX.emf*, this is automatically executed after the **hk***XXX.emf* file is executed.

The extended hook functions may be defined company wide, or by the user, to over−ride some of the standard hook functions, or to extend the syntax of the base files with locally defined extensions. As an example, consider the following file **myc.emf** which extends the basic **hkc.emf** file set of hi−lighting tokens for the 'C' Language.

```
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;
;  Created By    : Steven Phillips
;  Created       : Thu Jun 18 15:34:05 1998
;  Last Modified : <230798.0854>
;
;  Description   Extension hilighting for the 'C' language.
;
;  Notes         Define the locally defined 'C' library types and definitions
;                as extensions to the 'C' programming language.
;
;  History
;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

; MicroEmacs specific tokens
hilight .hilight.c 1 "LINE"           .scheme.type
hilight .hilight.c 1 "BUFFER"         .scheme.type
hilight .hilight.c 1 "WINDOW"         .scheme.type
hilight .hilight.c 1 "REGION"         .scheme.type
hilight .hilight.c 1 "KEYTAB"         .scheme.type
hilight .hilight.c 1 "KILL"           .scheme.type
hilight .hilight.c 1 "KLIST"          .scheme.type
hilight .hilight.c 1 "HILNODE"        .scheme.type
hilight .hilight.c 1 "HILNODEPTR"     .scheme.type
hilight .hilight.c 1 "HILCOLOR"       .scheme.type
hilight .hilight.c 1 "SELHILIGHT"     .scheme.type
hilight .hilight.c 1 "VIDEO"          .scheme.type
hilight .hilight.c 1 "VVIDEO"         .scheme.type
hilight .hilight.c 1 "FRAMELINE"      .scheme.type
hilight .hilight.c 1 "IPIPEBUF"       .scheme.type
hilight .hilight.c 1 "DIRNODE"        .scheme.type
hilight .hilight.c 1 "UNDOND"         .scheme.type
hilight .hilight.c 1 "meVARLIST"      .scheme.type
hilight .hilight.c 1 "meVARIABLE"     .scheme.type
hilight .hilight.c 1 "meCMD"          .scheme.type
hilight .hilight.c 1 "meAMARK"        .scheme.type
hilight .hilight.c 1 "meABREV"        .scheme.type
hilight .hilight.c 1 "meMACRO"        .scheme.type
hilight .hilight.c 1 "meNARROW"       .scheme.type
hilight .hilight.c 1 "meREGISTERS"    .scheme.type
hilight .hilight.c 1 "meSTAT"         .scheme.type
hilight .hilight.c 1 "osdITEM"        .scheme.type
hilight .hilight.c 1 "osdDIALOG"      .scheme.type
hilight .hilight.c 1 "osdCHILD"       .scheme.type
hilight .hilight.c 1 "meSCROLLBAR"    .scheme.type
hilight .hilight.c 1 "osdCONTEXT"     .scheme.type
hilight .hilight.c 1 "osdDISPLAY"     .scheme.type
hilight .hilight.c 1 "RNODE"          .scheme.type
hilight .hilight.c 1 "REGHANDLE"      .scheme.type
hilight .hilight.c 1 "meDIRLIST"      .scheme.type
hilight .hilight.c 1 "meNAMESVAR"     .scheme.type
hilight .hilight.c 1 "meDICTADDR"     .scheme.type
hilight .hilight.c 1 "meSPELLRULE"    .scheme.type
hilight .hilight.c 1 "meDICTWORD"     .scheme.type
hilight .hilight.c 1 "meDICTIONARY"   .scheme.type
hilight .hilight.c 1 "meMODE"         .scheme.type
```

SEE ALSO

Operating Modes, Language Templates, add−file−hook(2), cmode(2m).

fill−paragraph(2)

NAME

fill−paragraph − Format a paragraph

SYNOPSIS

n **fill−paragraph** (**esc o**)

DESCRIPTION

fill−paragraph this takes all the text in the current paragraph (as defined by surrounding blank lines, or a leading indent) and attempts to fill it from the left margin to the current fill column as defined by $fill−col(5). When an argument *n* is supplied *n* paragraphs are filled. If *n* is positive then MicroEmacs '02 performs indented filling (i.e. indentation for a bullet mark etc). If *n* is negative then indented filling is disabled. If no argument *n* is supplied then the paragraph is filled and the *point* and *mark* positions are retained. This allows paragraphs to be filled, whilst in the middle of the paragraph and the word position is maintained.

If **justify mode** is enabled the variable $fill−mode(5) determines how the paragraph is filled (i.e. *left*, *right*, *both* or *center*). The variable $fill−eos−len(5) determines the trailing space used after a period (`.`) character (the trailing characters are specified by $fill−eos(5)), typically defined as 2.

A set of characters defined by $fill−bullet(5) enable bullet markers to be placed in the text at the beginning of the paragraph causing the left margin to be moved to the right of the bullet. The search depth for fill to locate a bullet character is defined by $fill−bullet−len(5). When the paragraph is formatted and one of the bullet characters is encountered then the user is prompted as to whether the paragraph should be indented following the marker or not. The point of indentation is shown with a **<<<<** marker.

Filling is automatically disabled on paragraphs which start with characters in the $fill−ignore(5) set.

The simple text formatting is generally used for mail messages, ASCII text README files etc.

EXAMPLE

The following examples show how the text is formatted with indented filling enabled and both justification enabled:−

```
This  is  regular  text  that  is on the
margin

    This is a regular  paragraph that is
    offset  from  the  margin.  Note how
    MicroEmacs '02 retains the indent.
```

```
* With  the  introduction  of one of
  the  special  characters,  in this
  case a  bullet,  a  format  of the
  paragraph  offsets  the text  from
  the bullet.

1) Numbered  lists   are  the  same.
   Note that the  paragraphs are all
   separated with a blank line.

1. Numbered  lists  ending  with   a
   period.

label - Or labeled lists, separated
        with a dash.

>  '>' might be an ignore
>  character so it skips the paragraph
>
>          it is up to the user to
>   format these.
```

SEE ALSO

$fill−bullet(5), $fill−bullet−len(5), $fill−col(5), $fill−eos(5), $fill−eos−len(5), $fill−ignore(5), $fill−mode(5), ifill−paragraph(3), paragraph−to−line(3).

filter−buffer(2)

NAME

filter−buffer − Filter the current buffer through an O/S command

SYNOPSIS

filter−buffer (**C−x #**)

DESCRIPTION

filter−buffer executes one operating system command, using the contents of the current buffer as input, sending the results back to the same buffer, replacing the original text.

This would typically be used in conjunction with **sort(1)**, **awk(1)** or **sed(1)** to translate the contents of the buffer.

SEE ALSO

pipe−shell−command(2).

find−bfile(3)

NAME

find−bfile − Load a file as binary data
find−cfile − Load a crypted file

SYNOPSIS

n **find−bfile** "*file−name*" (**C−x 9**)
n **find−cfile** "*file−name*"

DESCRIPTION

find−bfile and **find−cfile** provide a simple interface to loading files in binary(2m) and crypt(2m) modes respectively. The numeric argument has the same effect as with the find−file(2) command except the respective modes are always enabled. See documentation on the modes an **find−file** command for more information.

NOTES

find−bfile and **find−cfile** are macros defined in file `tools.emf`.

The command find−file(2) is bound to key "`C−x 9`" with a numeric argument of 2, this is equivalent to executing **find−bfile** with no argument.

SEE ALSO

find−file(2), binary(2m), crypt(2m).

next−buffer(2)

NAME

next−buffer − Switch to the next buffer
find−buffer − Switch to the next buffer

SYNOPSIS

n **next−buffer** (**C−x x**)
n **find−buffer** "*buffer−name*" (**C−x b**)

DESCRIPTION

next−buffer switches to the *n*th next buffer in the buffer list in the current window, the default *n* is `1`, if *n* is negative then the `0−`*n*th previous buffer is selected. If `0` or a number greater than the number of buffers is specified then the command fails.

find−buffer switches to buffer "*buffer−name*" in the current window. If the buffer does not exist and a zero argument *n* is supplied then the command fails. If the buffer does not exist but no argument or a +ve argument *n* is specified then a new buffer is created, at which point the file−hook is evaluated.

If a −ve argument *n* is given to **find−buffer** then the buffer will be hidden. Any window displaying "*buffer−name*" will find another buffer to display. This functionality is often used with the hide(2m) buffer mode. If a value of −1 is given then the buffer will not be hidden in a window whose $window−flags(5) are set to lock the buffer to the window. If a value of less than −1 is given then the buffer is hidden from all windows.

If the current buffer has an *$buffer−ehook* command set then this command is executed before the new buffer is switched in. If the new buffer has a $*buffer−bhook* command set then this command is automatically executed after the new buffer is switched in but before control returns to the user.

SEE ALSO

next−window−find−buffer(2), hide(2m).

find−file(2)

NAME

find−file − Load a file

SYNOPSIS

n **find−file** "*file−name*" (**C−x C−f**)

DESCRIPTION

find−file finds the named file *file−name*. If it is already in a buffer, make that buffer active in the current window, otherwise attempt to create a new buffer and read the file into it.

The numeric argument *n* can be used to modify the default behaviour of the command, where the bits are defined as follows:

0x01

If the file does not exist and this bit is not set the command fails at this point. If the file does not exist and this bit is set (or no argument is specified as the default argument is `1`) then a new empty buffer is created with the given file name, saving the buffer subsequently creates a new file.

0x02

If this bit is set the file will be loaded with binary(2m) mode enabled. See help on **binary** mode for more information on editing binary data files.

0x04

If this bit is set the file will be loaded with crypt(2m) mode enabled. See help on **crypt** mode for more information on editing encrypted files.

0x08

If this bit is set the file will be loaded with rbin(2m) mode enabled. See help on **rbin** mode for more information on efficient editing of binary data files.

Text files are usually thought of as named collections of text residing on disk (or some other storage medium). In MicroEmacs '02 the disk based versions of files come into play only when reading into or writing out buffers. The link between the physical file and the buffer is through the associated file name.

MicroEmacs '02 permits full file names, i.e. you can specify:

```
disk:\directories\filename.extension
```

or (UNIX)

```
/directories/filename.extension
```

If the disk and directories are not specified, the current buffers disk/ directory is used. Several points should be noted in respect to the methods that MicroEmacs utilizes in the handling of files:−

- Without explicitly saving the buffer(s) to file, all edits would be lost upon leaving MicroEmacs − you are asked to confirm whenever you are about to lose edits.
- MicroEmacs has a mechanism for "protecting" your disk−based files from overwriting when it saves files. When instructed to save a file, it proceeds to dump the file to disk, making a backup of the existing file when backup(2m) mode is enabled.
- Auto−saving files can be performed on edited buffers by setting the $auto−time(5) variable. The file is saved in the same place with a '#' appended to the file name. This can be used directly by the user or in the unlikely event of MicroEmacs crashing (or system crash), the files are automatically recovered next time it is edited.

If you do not wish to perform any edits but merely browse the file(s), add the view(2m) mode to the buffer or ask for the file to be read in for viewing only.

RCS Support

If the file does not exist and the variable $rcs−file(5) is set then the existence of the RCS file is tested. If the rcs file exists then it will be checked out using a command−line created from the variable $rcs−co−com(5). If the check−out is successful then this file is loaded.

This raw interface for supporting file revision control systems has been adapted to support SCCS and Visual Source Safe see help on variable **$rcs−file** for more information and examples.

HTTP Support

MicroEmacs supports http file loading, this is available by default on UNIX systems but must be compiled in on win32 platforms (socket libraries not available on all win95 machines so cannot be compiled in by default). When available a http file can be loaded by simply executing **find−file** and giving the http file name, i.e. "`http://user:password@address:port/file`". Only the `http://`, `address` and `/file` components are mandatory, the rest can usually be omitted. e.g.:

```
find−file "http://members.xoom.com/jasspa/index.html"
```

See help page on %http−proxy−addr(5) for information on HTTP proxy server support.

FTP support

MicroEmacs supports ftp file loading, this is identical to http except the prefix `ftp://` is used as opposed to `http://`. The user name and password defaults to *guest* in the absence of both these fields. If the user name is supplied but not the password the password will be prompted for; this can be useful as the password will not be stored or written to the history file. Connection is by default on port 21.

```
        find−file "ftp://<me>:<password>@members.xoom.com/jasspa/index.html"
```

See also ftp(3).

The progress of the FTP transfer, and the FTP commands issued, may be viewed in the `*ftp−console*` buffer. This is popped up depending on the setting of the %ftp−flags(5) variable.

NOTES

The base name part (i.e. not the path) of `file−name` can contain wild−card characters compatible with most file systems, namely:−

?

Match any character.

[abc]

Match character only if it is *a*, *b* or *c*.

[a−d]

Match character only if it is *a*, *b*, *c* or *d*.

[^abc]

Match character only if it is not *a*, *b* or *c*.

*****

Match any number of characters.

If the name matches more than one file, a buffer will be created for each matching file. Note that these are not the same wild−card characters used by regex.

For *ftp* and *http* then a ftp console window is opened up to show the progress of the transfer (when configured), this is described in ftp(3).

SEE ALSO

auto(2m), binary(2m), crypt(2m), rbin(2m), time(2m), view(2m), buffer−mode(2), find−bfile(3), ftp(3), $rcs−file(5), %ftp−flags(5), %http−flags(5), %http−proxy−addr(5), next−window−find−file(2), read−file(2), save−buffer(2), view−file(2), write−buffer(2), file−op(2), file−attrib(3).

find−registry(2)

NAME

find−registry − Index search of a registry sub−tree.

SYNOPSIS

find−registry "*root*" "*subkey*" *index*

DESCRIPTION

find−registry performs an indexed search of a registry sub−tree allowing the caller to determine the names of the children that exist as sub−nodes of the specified node. *root* and *sub−key* form the root whose children are to be determined, *subkey* may be specified as the null−string (`""`) if an absolute registry path is specified. *index* is a value from `0..n` and identifies the index number of the child node. The name of the child node is returned in $result(5) if one exists, otherwise an error status is returned.

EXAMPLE

The following example comes from `addrbook.emf` and shows how **find−registry** is used to iterate through entries in the address book. Note that **find−registry** is used with !force(4) and the $status(5) of the call is tested to determine if the invocation succeeded.

```
!force find-registry "/AddressBook" "Names" #l0
!if $status
    set-variable #l1 $result
    76 insert-string "_"
    2 newline
    insert-string &spr "Section: %s" #l1
    newline
    ; Iterate through all of the entries.
    set-variable #l2 0

    !repeat
        !force #l2 ab-buffer
        !if $status
            set-variable #l2 &add #l2 1
        !endif
    !until &not $status
    set-variable #l0 &add #l0 1
    !goto next
!endif
```

SEE ALSO

get−registry(2), list−registry(2), read−registry(2), set−registry(2), erf(8).

find−tag(2)

NAME

find−tag − Find tag, auto−load file and move to tag position

SYNOPSIS

n **find−tag** "*string*" (**esc t**)

DESCRIPTION

find−tag finds the current or given tag (*string*) in a **tags** file and goes to the given point, loading the file if necessary. The tag is either the current word under the cursor or a user supplied word if the cursor is not in a word. The buffer containing the tag is popped up in another window and the cursor moved to the tag in the new window.

A **tags** file is usually created by an external program (e.g. **ctags(1)**) which stores word references (or tags) and the name of the file containing the tag, with a search string to go to its local. It is an indexing system which is often used in programming.

The argument *n* can be used to change the default behavior of find−tag described above, *n* is a bit based flag where:−

0x01

Use popup−window to display the tag in a different window (default) when this flag is not given the current window is used to display the tag.

0x02

Disable the use of the current cursor position to determine the tag. Instead the tag must always be supplied through "*string*".

0x04

Find the next definition of the last tag (multiple tag support). This feature can only be used if multiple tag support is enabled (see flag '**m**' of variable %tag−option(5)) and **find−tag** has already been successfully executed. In this situation the last invocation of find−tag defines the current tag and executing again with an argument of `4` will jump to the next definition of the current tag or return the message "`[No more "<current>" tags]`".

The next tag is typically bound to `M−C−t`.

The **tags** file is, by default, assumed to reside in the current directory of the currently viewed file. The user variable %tag−option(5) may be specified with a value of '`r`' (recursive) and '`c`' (continue) flags, which ascends the directory tree from the current directory and attempts to locate a *recursively* generated tags file at a higher directory level. Recursive tag files are generally easier to maintain where project source files are located in a number of project sub−directories, and enable the whole of the project tree to be taggable.

Two user variables must be defined before **find−tag** will execute, if either %tag−file(5) or %tag−template(5) are not defined the error message "**[tags not setup]**" is signaled.

NOTES

A **tags** file may be generated by MicroEmacs '02 from the menu (*Tools−>XX Tools−>Create Tags File*). Alternatively a **tags** file may be generated by the **ctags(1)** utility. This is typically standard on UNIX platforms. For Windows and DOS platforms then the **Exuberant Ctags** is recommended, this is available from:−

```
http://darren.hiebert.com
```

A MicroEmacs '02 compatible tags file may be generated using the command line "`ctags −N −−format=1 .`" cataloging the current directory. To generate **tags** for a directory tree then use "`ctags −NR −format=1 .`". Refer to the **Exuberant Ctags** documentation for a more detailed description of the utility.

SEE ALSO

%tag−file(5), %tag−option(5), %tag−template(5), generate−tags−file(3), **ctags(1)**.

spell−buffer(3)

NAME

spell−buffer − Spell check the current buffer
spell−word − Spell check a single word
spell−edit−word − Edits a spell word entry
find−word − Find a using spelling dictionaries

SYNOPSIS

spell−buffer
n **spell−word** ["*word*"] (**esc $**)
spell−edit−word ["*word*"]
find−word ["*word*"]

DESCRIPTION

MicroEmacs '02 provides an integrated spell checker with the following features:−

- Different languages.
- Dialog control of the speller.
- Best guess capability.
- *Replace* and *Replace all*, *Ignore* and *Ignore All*
- Undo capability.
- Adding new words and endings to speller.
- Auto correct of commonly occurring mistakes.
- Word finder, allows words to be searched with wild cards.

spell−buffer spell checks the current buffer, from the current position, to the end of the buffer. On invocation, an osd(2) dialog is opened and any corrections are made through this interface. If an error dialog opens instead the current language is not setup, please see the Language setting in user−setup(3) and Locale Support.

The dialog provides the user with an interface from which a new spelling may be selected, in addition new words may be added to the spelling dictionary. The dialog entries are defined as follows:−

Word

The **word** entry contains the erroneous word, this is presented in a text dialog box which may be manually edited to correct. If the word is manually corrected, then it is spell checked prior to insertion, and a new guess list is created. The user may elect to replace the word, take one of the suggestions or re−edit the misspelled word.

Meaning

The meaning button provides a convenient interface to describe−word(3) for looking up the meaning of the current word. The **Insert** button within the describe−word dialog will replace the current word in the spell−buffer.

Suggestions

The suggestions entry contains a list of suggestions as to the correct spelling of the word. The list is ranked in order of the best match, typically the misspelled word appears at (or near) the top of the list, unless the word is unknown or there are gross errors in the spelling. Selecting the word in the list with a single click of the mouse selects the word as the replacement, the actual replacement is performed by the **Replace** or **Replace All** buttons. Alternatively, double selecting a guess word replaces the word.

Language

The **language** entry allows the user to select the current spelling language. The new language is chosen from the dialog box. The language may be changed at any time during the spell operation and is effective immediately. The **Ext** languages are extended dictionaries that contain additional words, it is recommended that all spelling is performed with the extended dictionaries (where available).

Replace

The **replace** button is activated when a new word has been edited or selected as a candidate for replacement. Selecting **replace** modifies the erroneous word in the buffer with the newly selected word.

Replace All

The **Replace All** button is similar to the **Replace** button, except that it automatically replaces any subsequent occurrences of the erroneous word with the newly selected word. The replacement words are retained for the MicroEmacs edit session and are lost when the editor is closed.

Ignore

The **ignore** button requests that the speller ignore the erroneous word and continue to spell the buffer.

Ignore All

The **Ignore All** button is similar to the **Ingore** button, except that it automatically ignores the erroneous word thereafter. The ignore words are retained for the MicroEmacs edit session and are lost when the editor is closed.

Add

Add adds the current erroneous word to the dictionary, thereafter the word is recognized as a valid word. **Add** should only be used for words which have no derivatives, it is generally better to add a new word through the **Edit** interface where a new *base* word may be specified with it's derivatives.

Edit

The **Edit** button executes **spell−edit−word** giving the current erroneous word. This allows new words and auto−corrections to be defined as well as existing words to be altered, see full description below.

Find

The **Find** button executes **find−word** giving the current word as a starting seed. This allows the user to search for the word using a simple search criteria, see full description below.

Undo Last

The **undo Last** button restores the user to the previous spelling so that it may be re−entered, any replacement text that was made is restored to it's original spelling.

Exit

Exits the speller and returns the user to the buffer.

spell−word checks a single word which is either supplied by the user, or if an argument is given, the word under (or to the left of) the cursor position. If the word is correct, a simple message−line print−out is given, otherwise the main spell **osd** dialog is opened and the user may check the spelling within the context of the spell dialog as described above.

The default key binding of "`esc $`" supplies an argument forcing **spell−word** to check the current buffer word. **spell−word** is often used to check the spelling of a word outside of the context of the editor (i.e. when working on paper, or when doing at that prize crossword !!).

spell−edit−word allows words in dictionaries to be altered as well as new words and auto−corrections to be defined. On invocation, an **osd** dialog is opened and changes are made through this interface, defined as follows:−

Word

The **word** entry to be changed or added. If **spell−edit−word** was executed via spell−buffer **Edit** button, this will be set to the current word.

No word set

The word entry is empty, most of the functionallity will not be available until a word is entered.

New Word

To add a new word, the derivatives of the new word should be selected using the prefix and suffix options. Note that not all derivatives are listed, only one example derivative of each spell rule is given.

> **BE CAREFUL WITH THE CASE OF THE BASE WORD:** new words that are entered are case sensitive, as a general rule the *word* in the **Word** text box should be edited to it's base form and should be presented in lower case characters (unless it is a proper name, in which case it should be capitalized, or is an abbreviation or acronym when it might be upper

case).

When the appropriate derivatives of the new word have been selected, it may be added to the dictionary using the **Add** button. This adds the word to the users personal dictionary. Please note that if there are numerous standard words missing then check that an *extended* dictionary (designated by **Ext** in the language) is being used, the extended dictionaries more than double the repertoire of words available.

Words added to the dictionary may be subsequently removed using the **Delete** button, typing the existing word in the **Word** entry and selecting **Delete** button removes the existing word.

Auto−Correct

Selection of the **Auto−Correct** button allows a replacement word to be entered in the **To** text entry. Selecting **Add** adds the automatic correction to the speller. Thereafter, whenever the erroneous word is encountered the replacement word is always used to replace it, without user intervention.

Entering an exiting *auto−correct* word into the dialog and selecting **Delete** removes an existing auto−correct entry.

Exit

Exits the **Edit** dialog.

find−word opens the word finder dialog. This allows the user to search for a word using a simple search criteria. (This is ideal for cheating at crosswords !!). The word to be searched for is entered into the **Word Mask** and may use wild cards ***** to represent any number of characters, **?** to represent an unknown character and **[..]** for a range of characters.

For example, searching for `t?e?e` presents the list *theme*, *there* and *these*. Searching for `t*n` lists all of the words beginning with `t` and ending in `n`. See $find−words(5) for a full discription of the format used by search engine.

The words that match are returned in the scrolling dialog, and may be selected with the mouse (or cursor keys). The **Insert** button inserts the selected word into the current buffer or into the **Word** entry if executed from the **spell−buffer** dialog. Note that the list presented is limited to 200 words, selecting **next** gets the next 200 words, and so on. The **Exit** button exits the dialog.

NOTES

The words added to the speller during a MicroEmacs session are saved when the editor is closed. The user is prompted to save the dictionary, if the dictionary is not saved then any words added are lost.

All *ignore* words accumulated during a spell session are lost when the editor is closed. In order to retain *ignore* words, it is suggested that they are added to the personal dictionary rather than be ignored.

The personal spelling dictionary is typically called <*user*><*type*>.edf, and is stored in the default user location. The dictionary names are specified in the user−setup(3) dialog.

find−word may claim to have found more words than are actually listed. The use of derivatives in the spell algorithm allows a single word to be present several times. **find−word** counts each occurrence but it is only listed once.

SEE ALSO

user−setup(3), Locale Support, osd(2), spell(2), describe−word(3), $find−words(5).

find−zfile(3)

NAME

find−zfile − Compressed file support
zfile−setup − Compressed file support setup

SYNOPSIS

find−zfile "*file−name*"
zfile−setup "*extension*" "*list−command*" "*cut−to*"

"*column*" "*file−end*" "*extract−command*"
"*remove−command*" **DESCRIPTION**

find−zfile provides generic support for listing and extracting the contents of compressed files. **find−zfile** also supports the extraction of the internal files into another buffer.

find−zfile must be configured for each compression format using **zfile−setup**. It relies on command−line programs to generate content lists which are used to generate the main file listing, and subsequently, the ability to extract individual files for file extraction support.

For basic content listing support the first 3 arguments must be given to zfile−setup. The first argument "*extension*" is used as the compressed file id string. The compressed file type is derived from the file extension, e.g. "`zip`" or "`Z`" for UNIX compressed files. The exact case of the extension is checked first, followed by the lower case and upper case string.

The compressed file contents list is generated from executing the user supplied "*list−command*" and dumping the output into the list buffer. The command is run from the directory containing the compressed file and the following special tags may be used within the "*list−command*" which get substituted as follows:−

%zb

The token is replaced with the compressed files base name, i.e. the file name without the path.

%zf

The token is replaced with the compressed files absolute file name, i.e. the file complete with the path.

The head of the list output is often unwanted verbose printout, this can be automatically be removed by the use of the "*cut−to*" argument. The argument, if supplied (not an empty string), must be a regex search string matching the start of the required list. If found, all text before it is removed.

For single file extraction support the last 4 arguments must be specified by **zfile−setup**. The file to extract is selected either by selecting the file name using the left mouse button or by moving the cursor to the line containing the file name and pressing the "`return`" key.

find−zfile assumes that the file name starts at a fixed column number, specified with the "*column*" argument. The end of the file name is obtained by searching for the regular expression "*file−end*" string, the file name is assumed to end at the start of the search string match.

The file is then extracted by executing the supplied "*extract−command*" and then loading the extracted file into a new buffer. The command is run from the system temporary directory (i.e. "`/tmp/`" on UNIX or `$TEMP` on Windows etc.). The following special tags may be used within the "*extract−command*" which get substituted as follows:−

%zb

The token is replaced with the compressed files base name, i.e. the file name without the path.

%zf

The token is replaced with the compressed files absolute file name, i.e. the file name complete with the path.

%fb

The name of the file to be extracted.

The file is assumed to be extracted to the temp directory due to the way the command is run, this file is then loaded into a new buffer. The temporary file should then be removed using the supplied "*remove−command*" with is run from the temp directory, the "**%fb**" special tag may be used in the command. This argument may be given as an empty string, thereby disabling the removal.

EXAMPLE

For zip file support the freely available **unzip(1)** command can be used, following is the list of arguments with suitable entries:

```
extension           zip
list−command        unzip −v %zb
cut−to              ^ Length
column              58
file−end            $
extract−command     unzip −o %zf %fb
remove−command      rm %fb
```

For the zip file "*/usr/jasspa/memacros.zip*", after substitution the list command becomes "`unzip −v memacros.zip`" which will be executed in the "*/usr/jasspa/*" directory. This will produce the following form of output:

```
Archive:  memacros.zip
 Length  Method   Size  Ratio   Date    Time   CRC−32     Name
```

```
   ------  ------   ----  -----   ----    ----   ------     ----
      565  Defl:N    258  54%  02-27-99  22:56  018a7f70   american.emf
     3409  Defl:N    872  74%  02-28-99  01:37  6a6f9722   americar.emf
     4201  Defl:N    772  82%  03-01-99  12:58  d4e3bc4a   benchmrk.emf
      565  Defl:N    258  54%  02-27-99  22:56  dd394e24   british.emf
     3408  Defl:N    872  74%  02-28-99  01:37  32f3eeca   britishr.emf
     7239  Defl:N   1923  73%  02-28-99  15:13  d408f0da   calc.emf
     7292  Defl:N   2072  72%  01-23-99  12:49  5979d6b2   cbox.emf
     7104  Defl:N   1402  80%  02-28-99  15:13  6faf4fc5   cmacros.emf
     5967  Defl:N   1239  79%  02-13-99  16:38  27601523   ctags.emf
     1097  Defl:N    489  55%  02-16-99  10:58  53a55e36   dos.emf
      562  Defl:N    310  45%  01-16-98  07:54  ec24f65e   dos2unix.emf
.
.
.
```

The top Archive line is not require, this is automatically removed by setting the "*cut−to*" to "`^ Length`" which matches the start of the next line.

For file extract, consider the file "`ctags.emf`", the first character '`c`' is at column 58 and the first character after the end of the file name is the end−of−line character ('`\n`') which is matched by the regex string "$", hence the settings on "*column*" and "*file−end*". When this and the zip file name are substituted into the extract−command, it becomes "`unzip -o /usr/jasspa/memacros.zip calc.emf`" and is run from the "*/tmp.*" directory. Note that the "`-o`" option disables any overwrite prompts, these are not required as tests and prompting have already been performed by **find−zfile**. The extracted file "*/tmp/calc.emf*" is then loaded into a new buffer.

The temporary file is removed by executing the substituted remove−command which becomes "`rm calc.emf`" from the "`/tmp/`" directory.

For gzipped tar files, extension "**tgz**" the following setup can be used on UNIX platforms:

```
extension           tgz
list-command        unzip -v %zb
cut-to
column              43
file-end            $
extract-command     gunzip -c %zf | tar xof - %fb
remove-command      rm %fb
```

For the tgz file "*/usr/jasspa/memacros.tgz*", this will produce the following listing:

```
tgz file: /usr/jasspa/memacros.tgz


rw-rw-r-- 211/200    565 Feb 27 22:56 1999 american.emf
rw-rw-r-- 211/200   3409 Feb 28 01:37 1999 americar.emf
rw-rw-r-- 211/200   4201 Mar  1 12:58 1999 benchmrk.emf
rw-rw-r-- 211/200    565 Feb 27 22:56 1999 british.emf
rw-rw-r-- 211/200   3408 Feb 28 01:37 1999 britishr.emf
rw-rw-r-- 211/200   7239 Feb 28 15:13 1999 calc.emf
rw-rw-r-- 211/200   7292 Jan 23 12:49 1999 cbox.emf
rw-rw-r-- 211/200   7104 Feb 28 15:13 1999 cmacros.emf
rw-rw-r-- 211/200   5967 Feb 13 16:38 1999 ctags.emf
rw-rw-r-- 211/200   1097 Feb 16 10:58 1999 dos.emf
```

```
rw-rw-r-- 211/200    562 Jan 16 07:54 1998 dos2unix.emf
.
.
.
```

NOTES

find−zfile and **zfile−setup** are macros defined in `zfile.emf`.

SEE ALSO

find−file(2).

fold−current(3)

NAME

fold−current − (un)Fold a region in the current buffer
fold−all − (Un)Fold all regions in the current buffer

SYNOPSIS

fold−current
fold−all

DESCRIPTION

MicroEmacs '02 provides a generic, albeit course, folding mechanism which is applied to some of the well known file modes. The folding mechanism allows parts of the buffer to be scrolled up and hidden, leaving a residue hilighting marker within the buffer indicating a folded region. A folded buffer typically allows a summary of the buffer contents to be viewed within several windows, hiding the detail of the buffer.

The folding mechanism uses well defined *start* and *end* markers which form part of the syntax of the well known file mode. i.e. in 'C' this is the open and closed braces that appear on the left−hand margin (`{` .. `}`). The intention is that the natural syntax of the text is used to determine the fold positions, requiring no additional text formating or special text tags to be inserted by the user.

fold−current opens and closes a folded region within the buffer. If the current cursor position lies between a *start* and *end* marker then the region between the start and end is folded out and hidden from view, leaving a highlight marker in the buffer. If the fold already exists then, moving the cursor to the folded line and invoking **fold−current** removes the fold marker and reveals the text.

fold−all opens and closes all folded regions within the buffer, if the current state is unfolded then all of the *start*/*end* markers are located and their regions folded. Conversely, if the buffer is currently folded and **fold−all** is invoked, then all folds are removed and the associated text revealed.

CONFIGURATION

In order to utilize the **fold−current/all** commands within a buffer, the *start* and *end* markers have to be initialized for the syntactical contents of the buffer. This is performed within the hook function for the buffer, using the hook−name. Buffer specific variables are defined within the context of the buffer to configure that start and end fold handling. The buffer specific variables are defined as follows, where *xxxx* is the file hook base name.

xxxx−**fold−open**

A regular expression search string used to locate the start of the string. For speed the search string should include a regular expression start or end of line character whenever possible. i.e. in C the open is defined as "`^{`".

xxxx−**fold−close**

A regular expression search string used to locate the end of the string. For speed the search string should include a regular expression start or end line character whenever possible. i.e. in C the close is defined as "`^}`".

xxxx−**fold−mopen**

An integer value that denotes the number of lines to move in a forward or (−ve) backward direction from the *start* line located by the search string to the position in the buffer to be folded. If default value when **mopen** is omitted is 0, starting the fold from the search string line.

xxxx−**fold−mclose**

The relative displacement from the close fold line to the fold position, this is a positive or negative displacement depending on where the fold is to be positioned.

xxxx−**fold−mnext**

Specifies the number of lines to advance before the next search is continued on the fold operation. This is only used by **fold−all**. **EXAMPLE**

The following examples show how the fold variables are set up in each of the buffer modes.

C and C++

C and **C++** fold on the open and close brace appearing in the left−hand margin. The fold variables are defined in `hkc`/`hkcpp.emf` as follows:−

```
set-variable %c-fold-open  "^{"
set-variable %c-fold-close "^}"
```

Given a 'C' function definition:−

```
static void
myfunc (int a, int b)
{
    /* Function body */
}
```

the folded version appears as follows:−

```
static void
myfunc (int a, int b)
}
```

emf

MicroEmacs macro files **emf** support folding of macro definitions, the fold variables are defined in `hkemf.emf` as follows:−

```
set-variable %emf-fold-open  "^0? ?define-macro"
set-variable %emf-fold-close "^!emacro"
set-variable %emf-fold-mopen "1"
```

Given a macro definition:−

```
0 define-macro mymacro
; This is the body of the macro
; ... and some more ...
!emacro
```

the folded version of the macro is defined as:−

```
0 define-macro mymacro
!emacro
```

nroff

nroff is configured for manual pages only and folds between `.SH` and `.SS` sections, the hook variables are defined as follows:−

```
set-variable %nroff-fold-open  "^\.S[SH]"
set-variable %nroff-fold-close "^\.S[SH]"
set-variable %nroff-fold-mopen "1"
set-variable %nroff-fold-mnext "-1"
```

Given an nroff block of text defined as:−

```
.SH SYNOPSIS
.\" Some text
.\" Some more text
.SH DESCRIPTION
```

Then the folded version appears as:

```
.SH SYNOPSIS
.SH DESCRIPTION
```

tcl/tk

tcl/tk is configured to fold procedures. The fold variables are defined as:−

```
set-variable %tcl-fold-open  "^proc "
set-variable %tcl-fold-close "^}"
set-variable %tcl-fold-mopen "1"
```

Given a tcl procedure definition:−

```
proc tixControl:InitWidgetRec {w} {
    upvar #0 $w data

    tixChainMethod $w InitWidgetRec

    set data(varInited)   0
    set data(serial)    0
}
```

The folded version of the same section appears as:−

```
proc tixControl:InitWidgetRec {w} {
}
```

NOTES

fold−current and **fold−all** are macros implemented in `fold.emf`. The folding is performed using the narrow−buffer(2) command.

fold−current may also be bound to the mouse using the user−setup(3). The typical binding is `C−mouse−drop−1`.

SEE ALSO

File Hooks, user−setup(3), narrow−buffer(2).

ftp(3)

NAME

ftp − Initiate an FTP connection

SYNOPSIS

ftp

DESCRIPTION

ftp initiates a File Transfer Protocol (FTP) connection to a remote host on the network. Using FTP, editing of files may be performed in much the same way as on the local file system. Directory listings may be retrieved and traversed using the mouse or cursor keys. Using the directory listing, files may be transfered to/from the remote host to the local machine.

On issuing the command then a dialog is presented to the user which is used to open the connection. The dialog entries are defined as follows:−

Registry File

The name of a MicroEmacs registry file which is used to store the FTP information. If a registry name is provided then all FTP address information is stored in the registry file and saved for later sessions. Be aware that password information is saved in this file as plain text if a password is entered into the site information.

If the registry information is omitted then the information is not saved between sessions.

Site Name

An ASCII pseudo name for the remote host. The pull−down menu may be used to select existing sites that have been previously entered.

Host Address

The address of the host, this may be an IP address (`111.222.333.444`) or a DNS name (i.e. `ftp.mysite.com`).

User Name

The login name for the site. If this is omitted then `guest` is used by default.

Password

The password used to enter the site for the given login name. If the password is NOT supplied then the user is prompted for the password when a transaction takes place. If the password is omitted and left to promt then it is not stored in the registry.

Take note of the comments provided above regarding the password information.

Initial Host Path

The starting directory at the remote host. If this is omitted then the root directory ('`/`') is used by default.

On selecting **Connect** then a FTP connection is opened and the initial directory appears as a directory listing, if the initial path is a file then the file is loaded into the editor.

Thereafter the file may be edited within the editor as normal, on a write operation then the file is written back to the host, via FTP.

On opening a FTP connection the progress of the transfer, and the FTP commands issued, may be viewed in the `*ftp-console*` buffer. This buffer may automatically appear depending upon the value of the %ftp−flags(5) variable.

NOTES

ftp is a macro implemented in `ftp.emf`. This uses the underlying command find−file(2) to implement the FTP transfer.

FTP files can be directly loaded and edited using the standard file commands such as find−file(2).

The FTP addresses are retained in a registry file (see erf(8)). The registry file is automatically loaded when MicroEmacs starts up each session. The current site information may be viewed using list−registry(2) and is located at the following registry addresses:−

/url

Data value is file system location of the FTP registry file.

/url/ftp/<*hostName*>

The name of the host to which the connection is to be made.

/url/ftp/<*hostName*>**/host**

The name or IP address of the remote host

/url/ftp/<*hostName*>**/user**

The user name used to log into the remote host.

/url/ftp/*<hostName>***/pass**

The user password to the remote host. If this entry is empty then the user is always prompted for the password when the connection is made.

/url/ftp/*<hostName>***/path**

The initial path at the remote site.

When a FTP connection is initiated then the connection (socket) remains open for a period of approximately 4 minutes from the last transfer time, after that the connection is automatically closed and is re−initiated if required again.

NOTE: For windows platforms then the resultant executable must be built with URL support enabled, for UNIX platforms socket support is automatically enabled.

BUGS

Directory completion is not available when the current working directory is an FTP address. To work around this from the command line, select `<RETURN>` to get a directory listing of the current directory and select the file(s) from the directory to load.

SEE ALSO

%ftp−flags(5), erf(8), find−file(2), file−op(2), list−registry(2).

fvwm(9)

SYNOPSIS

fvwm, fvwmrc − FVWM Window manager configuration files

FILES

hkfvwm.emf − FVWM configuration file hook definition

EXTENSIONS

.fvwm, **.fvwmrc** − FVWM configuration file

MAGIC STRINGS

−!− fvwm −!−

The embedded fvwm string may be used with later versions of fvwm which use a different file extension to force the hilighting of the file. **DESCRIPTION**

The **fvwm** file type template provides simple hilighting of the FVWM files, the template provides minimal hilighting.

BUGS

None reported.

SEE ALSO

Supported File Types

gdiff(3f)

NAME

gdiff − Command line graphical file difference

SYNOPSIS

me "@gdiff" "*version1*" "*version2*"

DESCRIPTION

MicroEmacs may be executed from the command line to invoke the *Graphical Difference* gdiff(3) macro, showing the difference(s) between two files.

The editor is invoked in **gdiff** mode and shows the difference between the two files on the command line.

NOTES

The macro is defined in file `gdiff.emf`.

SEE ALSO

gdiff(3), start−up(3).

generate−tags−file(3)

NAME

generate−tags−file − Generate a tags file

SYNOPSIS

n **generate−tags−file** ["*tag−command*"]

DESCRIPTION

The **generate−tags−file** command provides an interface to tag file generation. Typically the "*tag−command*" argument will not be required as the current buffer will automatically configure **generate−tags−file** on how tags are generated for the current buffer's file type. See the notes below for more information on configuration.

generate−tags−file supports two different methods of tag generation, firstly via a MicroEmacs macro file and secondly by an external shell command (such as **ctags(1)**). It is generally configured in the current buffer's setup hook.

If a macro file is used a setup dialog is opened if an argument of 0 is given to **generate−tags**. This dialog can be used to configure which type of tags are required and the starting directory (useful when using recursive tags over a source tree). Note that not all tag types are available for all file types.

The generated tags file can then be used by the find−tag(2) command.

NOTES

generate−tags−file is a macro defined in file `gentags.emf`.

generate−tags−file can be configured in one of 2 ways:

When a MicroEmacs macro file (such as `ctags.emf`) is to be used, simply give the name of the macro file to be run as the "*tag−command*" argument. Alternatively set the variable **.**<*$buffer−fhook*>**.tags** to this name, e.g. for C files

```
set-variable .fhook-c.tags "ctags"
```

Note the "`.emf`" extension is assumed.

When an external shell command is to be used, set the *tag−command* to the shell command−line prefixed with a '`!`' character, for example to use **ctags(1)** try the following:

```
set-variable .fhook-c.tags "!ctags *.c *h"
```

Note that the generate−tags dialog is not available in this mode of execution.

SEE ALSO

find−tag(2).

get−next−line(2)

NAME

get−next−line − Find the next command line

SYNOPSIS

get−next−line (**C−x `**)

DESCRIPTION

get−next−line is typically used in conjunction with the compile(3) and grep(3) commands to enable the user to step through errors/locations one by one. The command looks for lines in the form defined by add−next−line(2) in the order of definition. If a match is found the command attempts to find the next error or warning found from the current location (See compile(3)). If the buffer was not found then the next buffer set is searched for, and if found then the next expression from the cursor is automatically located. The command fails if none of the buffers exist, or the end of the buffer is reached.

SEE ALSO

$file−template(5), $line−template(5), add−next−line(2), compile(3), grep(3).

get−registry(2)

NAME

get−registry − Retrieve a node value from the registry.
set−registry − Modify a node value in the registry.

SYNOPSIS

get−registry "*root*" "*subkey*"
set−registry "*root*" *"subkey"* *"value"*

DESCRIPTION

get−registry retrieves the value of a node defined by *root*/*subkey* from the registry into the variable $result(5).

The node name is specified in two components, typically required when iterating over a registry tree, where the *root* component is static and the *subkey* is dynamic, *subkey* may be specified as the null string (`""`) if an absolute registry path is specified.

set−registry adds (or modifies) a new value to the registry. *root* is the root of the new entry and MUST exist or the call fails. *subkey* is the node name (or path) if the path does not exist then it is created. *value* is the value to assign to the node.

DIAGNOSTICS

get−registry fails if the node does not exist, otherwise the registry string is returned in $result(5).

set−registry fails if the *root* node does not exist.

EXAMPLE

The following call

```
set−registry "/history" "foo/win32/printer" "foo−bar"
```

constructs a registry hierarchy of the form:−

```
"history" {
  "foo" {
    "win32" {
      "printer"="foo−bar";
    }
  }
```

```
        }
```

The value of the registry node may be retrieved using:−

```
        get−registry "/history" "foo/win32/printer"
```

which would return "`foo−bar`".

SEE ALSO

find−registry(2), list−registry(2), read−registry(2), &reg(4), erf(8).

global−bind−key(2)

NAME

global−bind−key − Bind a key to a named command or macro
global−unbind−key − "Unbind a key from a named command or macro"

SYNOPSIS

n **global−bind−key** "*command*" "*key*" (**esc k**)
n **global−unbind−key** "*key*" (**esc C−k**)

DESCRIPTION

global−bind−key takes one of the named commands and binds it to a key. Thereafter, whenever that key is struck, the bound command is executed. If an argument *n* is given then the bound command is executed *n* times when the key is struck. (i.e. the command is passed the numeric argument '*n*').

global−unbind−key unbinds (detaches) a user entered *key* sequence (i.e. `C−x C−f`) from any command to which it may be bound. This does not work with buffer or message line key bindings, see buffer−unbind−key(2) and ml−unbind−key(2). If an argument of 0 is given to **global−unbind−key**, only a single key is obtained for the user, if the character is currently bound to the prefix command, the prefix binding and any sub−bindings are removed. **global−bind−key** calls **global−unbind−key** first if the key to be bound is already bound to something else.

If a −ve argument is given to **global−unbind−key** then all bindings are removed, **caution** − removing all bindings interactively will render the current MicroEmacs session unusable. This can only be used within macro development where new bindings are created immediately afterwards.

The **global−bind−key** command, currently bound to `esc k`, prompts the user for the named command and the key to which it is to be bound. This help file gives a complete list of all built in commands, and some useful macros, a complete list of all commands and macros can be obtained by using the command completion (type `esc x tab tab`, see ml−bind−key(2)) or using the command describe−bindings(2).

The mouse buttons are considered to be *keys*, there is a *key* for each button press and release event, use describe−key(2) to get the binding key string.

The non−ASCII standard keys such as the cursor keys have 'standard' key names to make cross platform binding support easy. Some systems such as *termcap* do not have fixed key−bindings, for these key the users must use the command translate−key(2) to convert the system key binding to the standard key binding.

Permanent changes are done indirectly through the `me.emf` file. This is a file that MicroEmacs '02 reads and executes (see execute−file(2)) during startup and hence results in the appearance of a permanent change in the key bindings. The syntax of commands in the `me.emf` file is described under the execute−file command. Of principal concern here are the two commands **global−bind−key** and **global−unbind−key**. The primary difference between the way parameters are passed to these commands in the `me.emf` file is that the keys are not typed in directly (as in the *control−I* key when you want `C−i`) but by symbolic names. Every key has a unique name which can be easily obtained with the current binding by using the command describe−key(2).

See help on Key Names for a description of the symbolic naming system and a complete list of valid key names. Also see Bindings for a complete list of default key bindings.

EXAMPLE

Alt P

```
global-bind-key "func" "A-p"
```

Control F2

```
global-bind-key "func" "C-f3"
```

Shift Alt Left Cursor

```
global-bind-key "func" "A-S-left"
```

Control Alt Delete

```
global-bind-key "func" "C-A-delete"
```

Note that binding **Control−Alt−Delete** is not recommended for MS−DOS systems for obvious reasons.

NOTES

Some ASCII keys, such as `<CR>` (C−m), `<tab>` (C−i), `<BACKSPACE>` (C−h) have non−ASCII key bindings, namely "**return**", "**tab**", "**backspace**" etc. this is to allow separate key−bindings for the real "**C−m**" etc.

Be very careful in binding and unbinding keys since you could get into some very peculiar situations such as being unable to abort out of a command (if you unbind CTRL−G or bind it to something else) or recover from the bad binding/unbinding if you unbind execute−named−command(2) or the **global−unbind−key** command. As long as you leave yourself the opportunity to do both of the last two commands, you can recover from disastrous bindings/unbindings.

SEE ALSO

buffer−bind−key(2), buffer−unbind−key(2), describe−bindings(2), describe−key(2), ml−bind−key(2), ml−unbind−key(2), translate−key(2).

goto−alpha−mark(2)

NAME

goto−alpha−mark − Move the cursor to a alpha marked location

SYNOPSIS

goto−alpha−mark "*?*" (**C−x a**)

DESCRIPTION

goto−alpha−mark prompts user for an alpha character and sets the cursor position to the preset location. Alpha marks are specified on a per buffer basis, thus the current buffer is not changed, merely the current location in the buffer. The alpha mark must already be defined using set−alpha−mark(2). This functionality is useful for rapidly returning back to locations in large files.

SEE ALSO

set−alpha−mark(2).

goto−line(2)

NAME

goto−line − Move the cursor to specified line

SYNOPSIS

n **goto−line** (**esc g**)
goto−line "*num*"

DESCRIPTION

goto−line moves the cursor to the specified line in the buffer. The user is prompted for the new line number on the command line, which may be entered as a relative displacement ([+|−]*number*) from the current position, or as an absolute line number (*number*). If the number is preceded by + or − then this is treated as a relative displacement from the current line, otherwise it is an absolute line number.

If a +ve argument *n* is supplied, **goto−line** moves to this line, e.g. to move the cursor to line 240:

```
240 goto-line
```

A special case of **goto−line** is operative if an argument of 0 is supplied, argument "*num*" must also be given as above except **goto−line** treats the line number or displacement as an absolute move, i.e. includes *narrowed out* sections when calculating the new position. If the new line lies within a narrowed out section (i.e. a section that has been hidden and is not visible on the screen) the narrow is automatically expanded. See narrow−buffer(2) for more information on narrowing.

Supplying a −ve argument to goto−line results in an error.

NOTES

After successfully calling goto−line, variable $window−line(5) is set to the required line number.

SEE ALSO

goto−alpha−mark(2), goto−matching−fence(2), narrow−buffer(2), $window−line(5).

goto−matching−fence(2)

NAME

goto−matching−fence − Move the cursor to specified line

SYNOPSIS

goto−matching−fence (**esc C−f**)

DESCRIPTION

goto−matching−fence moves the cursor to the opposing fence character of the character currently under the cursor. The set of fence characters include `[]`, `{}` and `()`. i.e. if the character under the cursor is `` `{ ``' then **goto−matching−fence** moves the cursor to the opening fence `` `} ``', and visa versa.

goto−matching−fence can also be used to move the cursor to matching C/C++ `#if`, `#elif`, `#else` and `#endif` constructs, cycling through them in the given order.

When the fence(2m) buffer mode is enabled the matching open fence is automatically displayed when the closing fence is typed. The length of time the matching fence is displayed for can be controlled by the $fmatchdelay(5) variable.

SEE ALSO

fence(2m), $fmatchdelay(5), goto−line(2).

set−position(2)

NAME

set−position − Store the current position
goto−position − Restore a stored position

SYNOPSIS

n **set−position** "*label*"
n **goto−position** "*label*"

DESCRIPTION

set−position stores current window, buffer, cursor and mark position information against the given '`label`' (a single alpha−numeric character). **goto−position** takes the positional information stored against the given '`label`' and restores the window, buffer and cursor positions from those previously **set**.

A call to **set−position** with the same label over−writes the previous stored information, a call to **goto−position** does not alter the information and may be restored multiple times.

The numerical argument to **set−position** is used to define the information that is stored in the position item. The argument is intrepreted as a bitmask, flagging what information is to be stored. The bit mask is defined as follows:

`0x001`

Store the current window.

`0x002`

Store the current window's horizonal scroll (value of $window−x−scroll(5)).

`0x004`

Store the current window's current line horizonal scroll (value of $window−xcl−scroll(5)).

`0x008`

Store the current window's vertical scroll (value of $window−y−scroll(5)).

`0x010`

Store the current buffer.

`0x020`

Store the current window's current line using an alpha mark.

`0x040`

Store the current window's current line number (value of $window−line(5)).

`0x080`

Store the current window's current column offset (value of $window−col(5)).

`0x100`

Store the current window's mark line using an alpha mark.

`0x200`

Store the current window's mark line number (value of $window−line(5) when mark was set).

`0x400`

Store the current window's mark column offset (value of $window−col(5) when mark was set).

When *n* is not specified, the default value is `0x0bf`, i.e. store all information required to return to the window, buffer and cursor position.

The argument supplied to **goto−position** similarly interpreted as a bitmask, restoring the positional information. When the numerical argument *n* is omitted the same default is used when omitted on the store. On restoring a position, information stored during the call to **set−position** which is not requested in corresponding **goto** is ignored, similarly information requested in a **goto** which was not stored in the **set** is also ignored.

EXAMPLE

The following example shows the typical use of these commands:

```
set−position "a"
    .
    .
goto−position "a"
```

The following example stores the current position at the start of a macro sequence, if `my−command` is not successful (**$status** equals 0) the original position is restored:

```
set−position "\x80"
!force my−command
!if &equ $status 0
    ; command failed, return to the original position
    goto−position "\x80"
```

```
!endif
```

Note '`\x80`' is interpreted as the character with the ASCII value of 0x80 which is a non−alphanumeric character, this is permitted in macros to avoid using alphanumerics.

The following example shows how the current position can be restored after re−reading a file:

```
0xce set−position
read−file $buffer−fname @mna
; a numeric argument of 0xce is not
; required as this is the default
goto−position
```

NOTES

The position item may store and restore the current line by using an alpha mark or the line number. The restrore strategy will determine what is required, as follows:−

The main benefit from using an alpha mark is that the position is maintained even when the buffer is edited, for example if the position is stored at line 10 and a line is subsequently inserted at the top of the buffer, if the line number was used then it would return back to the 10th line which is the old 9th line whereas if an alpha mark were used it would correctly return to the 11th line, as expected.

The disadvantage of using an alpha mark is that it is only associated with that buffer. In some cases a position may need to be restored in another buffer (e.g. when re−reading a buffer the original buffer may be deleted first), in this situation the buffer line number must be used.

Commands **set−window** and **goto−window**, which simple stored and returned to the current window, were replaced by set−position and goto−position in August 2000. The following macro implementations can be used as a replacement:

```
define−macro set−window
    1 set−position "\x80"
!emacro

define−macro goto−window
    goto−position "\x80"
!emacro
```

SEE ALSO

set−alpha−mark(2), find−buffer(2), $window−x−scroll(5), $window−xcl−scroll(5), $window−y−scroll(5), $window−line(5), $window−col(5).

grep(3)

NAME

grep − Execute grep command rgrep − Execute recursive grep command

SYNOPSIS

grep "*expression files...*" **rgrep** "*expression*" "*base−path*" "*file−mask*"

DESCRIPTION

grep executes the **grep(1)** command with the command line set by the %grep−com(5) variable and the user supplied *expression* and file list *files...*. The output of the command is piped into the ***grep*** buffer ready for the get−next−line(2) command to step through all matched lines. The syntax from the grep output must be setup using add−next−line(2).

If an argument is given then a pipe−shell−command(2) is used instead of ipipe−shell−command(2), this is useful when used in macros as it ensures that **grep** has finished before the command returns.

rgrep is simpler to **grep** in that it uses **grep(1)** to search for all occurrences of *expression*, but **rgrep** also uses **find(1)** to search for *expression* in all files matching the *file−mask* in all directories from *base−path* down.

NOTES

grep is a macro defined in `tools.emf`.

grep(1) must be executable on the system before grep or rgrep can function, **find(1)** must also be available for rgrep to work.

EXAMPLE

The **grep** command is generally set up in the startup files as follows:−

```
;
; setup the next-error stuff including grep and compiling
;
set-variable $line-template "[0-9]+"
set-variable $file-template "[a-zA-Z:]*[0-9a-zA-Z\_.]+"
;
; Definitions for GNU grep utility.
;
set-variable %grep-com "grep -n "
0 add-next-line "*grep*"
```

```
add−next−line "*grep*" "%f:%l:"
```

SEE ALSO

grep(1), %grep−com(5), add−next−line(2), get−next−line(2), compile(3).

help(2)

NAME

help − Help; high level introduction to help
help−command − Help; command information
help−variable − Help; variable information
help−item − Help; item information

SYNOPSIS

n **help** (**esc ?**)
help−command "*command*" (**C−h C−c**)
help−variable "*variable*" (**C−h C−v**)
help−item "*item*" (**C−h C−i**)

DESCRIPTION

The help commands provide a quick on−line help facility within MicroEmacs '02 without invoking a third party documentation system (e.g. a browser such as **Netscape(1)** or **winhelp(1)**).

The on−line help is assisted by a set of macros which enable key words within the help buffers to be located by either tagging (`esc t`) or by selection with the left mouse button. The tag mechanism supports command completion.

help provides general help on the philosophy and functionality of MicroEmacs '02, if an argument *n* of 0 is given to the command it changes the current buffer to the internal help buffer, typically named `"*on-line help*"`. This is a hidden system buffer used to store all the on−line help and can be used for a variety of things. Note that access to this buffer must be via the **help** command, not **find−buffer** and the help command will also ensure the system help file is loaded first.

help−command describes the purpose of the given *command*.

help−variable Describes the purpose of the given *variable*, similar to **help−command**, only for variables.

help−item Describes the purpose of any given item, where item could be a command, variable or any aspect of MicroEmacs '02.

FILES

The help files are ASCII text files located in the MicroEmacs '02 home directory. The files are defined as follows:−

`me.ehf` − Help text file.
`hkehf.emf` − Help macros.

SEE ALSO

osd−help(3), command−apropos(2), describe−bindings(2), describe−key(2), list−commands(2), list−variables(2).

hide(2m)

NAME

hide − Hide buffer

SYNOPSIS

hide Mode

H − mode line letter.

DESCRIPTION

This mode can only be set on a buffer and when enabled the buffer is effectively hidden from the user. When set the buffer is hidden from the buffer completion list used by commands such as find−buffer(2), the buffer is also ignored by commands list−buffers(2), save−some−buffers(2) and delete−some−buffers(2).

SEE ALSO

find−buffer(2), list−buffers(2).

hilight(2)

NAME

hilight − Manage the buffer hilighting schemes

SYNOPSIS

0 **hilight** "*hil−no*" "*flags*" ["*nol*"] ["*buffer−scheme*" ["*trunc−scheme*"]]

hilight "*hil−no*" "*type*" "*token*" [["*rtoken*"]

[(["*close*" ["*rclose*"] "*ignore*"]) |

(["*continue*"]) |
(["*b−hil−no*"])]
"*schemeNum*"
hilight "*hil−no*" "*0x200*" "*token*"
hilight "*hil−no*" "*0x400*" "*from−col*" "*to−col*" "*schemeNum*"

−1 **hilight** "*hil−no*" "*type*" "*token*"

DESCRIPTION

The **hilight** command creates and manages the buffer hilighting, the process of creating a new hilighting scheme is best described in File Language Templates. The command takes various forms as defined by the arguments. Each of the argument configurations is defined as follows:−

Hilight Scheme Creation

0 **hilight** "*hil−no*" "*flags*" ["*nol*"] ["*buffer−scheme*" ["*trunc−scheme*"]]

With an argument of `0`, **hilight** initializes or re−initializes the hilight scheme *hil−no* (1−255). Every buffer has a hilight scheme, the default is 0 which means no hi−lighting and only the $global−scheme(5) etc. are used. The hilighting scheme must be defined before use and is used to specify how the buffer is to be hilighted. MicroEmacs '02 supports the following hilighting concepts:−

- **hilight string**, a user specified string is hilighted in any color scheme.
- **Tokens**, same as a hilight string except that the string must be enclosed in non alpha−numeric characters.
- **Start−of−line hilights**, the start of the hilight must be the first non−white character of the line.
- **End−of−Line hilights**, the hilight starts from the current position and continues until the end of the line. Optionally, the hilight may continue onto the next line if the current line ends in a

given string. A bracket may also be searched for within the line.
- **Bracket hilight**, hi−lights from the current position until the closing bracket token is found.
- **Replace string** , allows the hilight string to be replaced with a different user specified string. (i.e. the displayed representation is different from the buffer contents)

Terminals that cannot display color directly may still be able to take advantage of the hi−lighting. A terminal that has fonts (i.e. *Termcap*) can use them in the same way using the add−color−scheme(2) command. The hi−light scheme is also used in printing (see print−buffer(2)). If your terminal cannot display color in any way, it is recommended that hi−lighting is disabled (except when printing) as it does take CPU time.

The "*hil−no*" argument specifies which hi−lighting scheme is being initialized. Once a hilighting scheme has been initialized, hi−light tokens can be added to it and it can be used by setting the current buffer's $buffer−hilight(5) variable to "*hil−no*". The "*flags*" argument is a bit based flag setting global hi−light characteristics, where:−

0x01

The hi−light scheme is case insensitive, i.e. the following tokens become equivalent:−

```
house == HOUSE == hOuSe
```

When the hilight scheme is attributed as case insensitive then the tokens must **all** be specified in **lower** case.

0x02

Set a hi−light look−back. During the process of determining the window hilighting then the hilight process has to determine whether the top of the window starts in a hi−light bracket or not. The look−back command tries looking "*nol*" lines backwards for an open bracket. If an open bracket is found then the top of the window is assumed to start with that bracket, else it is assumed that the top of the window is not in a bracket. For example, in `C', a comment starts with "`/*`" and ends with "`*/`" so if the hilight was initialized with

```
0 hilight 1 2 10 $global-scheme
```

of the following, only the first would begin hi−lighted which is correct (assuming the "`/*`" is 10 or less lines away).

```
  /* ........       /*.........         .........
     ........         .........*/       .........
----------------  ---------------   --------------- top of
     ........*/       .........         .........    window
```

The optional argument "*buffer−scheme*" specifies the default scheme to use if there is no specific hi−light, when omitted the value of $global−scheme(5) is used. The *buffer−scheme* is a reference to a set of foreground and background color pairs previously defined with add−color−scheme(2). The last argument "*trunc−scheme*" is also optional and specifies the line truncation scheme, when omitted the value of $trunc−scheme(5) is used.

The hi−lighting scheme required is based on the type of file being edited and so is usually directly related to the file extension, thus it can be automatically set using file hooks (see add−file−hook(2)).

Hilight Scheme Token Creation

hilight "*hil−no*" "*type*" "*token*" [["*rtoken*"]

[(["*close*" ["*rclose*"] "*ignore*"]) |

(["*continue*" ["*rcontinue*"]]) |
(["*b−hil−no*"])]
"*schemeNum*"
hilight "*hil−no*" "*0x200*" "*token*"
hilight "*hil−no*" "*0x400*" "*from−col*" "*to−col*" "*schemeNum*"

With the default argument of `1`, **hilight** creates a hilight token to be used in hilight color scheme identified by "*hil−no*" (1−255) (see the section on **Hilight Scheme Creation** for a overview of hi−lighting). The second argument "*type*" specifies the token type and must always be specified, it determines which other arguments required.

Typically the last argument, *schemeNum*, is also required. This identifies the color scheme to use when hilighting the token, defining the foreground, background and selection color schemes. This is an index generated from add−color−scheme(2). If the *schemeNum* argument is omitted the default hilght color scheme is used.

The token "**type**" is a bit based flag of which 0, 1 or more of the bits may be set, the effect of the bits are defined as follows:

0x001

The "*token*" must be surrounded by non−word characters (word characters are typically the alpha−numeric characters), e.g. the following defines "`if`" as a token:

```
hilight 1 1 "if" .scheme.keyword
```

this hilights the 'if' in " `if` " but not in "`aifa`".

0x002

Color this to the end of the line, often used for comments etc. For example in MicroEmacs macro language a '`;`' character signifies the rest of the line as a comment, hilighting is defined as follows:

```
; this is a comment line
hilight 1 2 ";" .scheme.comment
```

0x004

This is a bracket token, the closing bracket string "*close*" and an ignore character "*ignore*" must also be supplied. The ignore character indicates that when found it should ignore the next character; this prevents an early end on bracket miss−match. For example, in C a '`"`' character can be inserted into a string by 'protecting' it with a '`\`' character, such as "*this is a string with a \" in it*". In this example the ignore character should be '\' so the mid string '`"`' is correctly ignored, as follows:

```
hilight 1 4 "\"" "\"" "\\" .scheme.string
```

An empty value, "", effectively disables the ignore feature. If replacing bit `0x040` is set the replacement close bracket "*rclose*" must be supplied.

0x008

The token has a continuation string, usually used with 0x02 but cannot be used with token types `0x004` and `0x080`. The argument "*continue*" must be supplied and if the replacing bit `0x040` is set the replacement continue string "*rcontinue*" must also be supplied. The best example of its use can again be found in C; macros defined using the `#define` pre−processor construct may be constructed on single or multiple lines. The macro continues onto another line if the current line ends with a backslash '`\`' character, e.g.:

```
#define a_single_line_macro() printf("hello world\n")

#define a_four_lined_macro()          \
do {                                  \
    printf("hello world\n") ;         \
} while(0)
```

This can be correctly hilighted with the pre−processor scheme using:

```
; use to−end−of−line (2) and continuation (8), i.e. 2+8=10
hilight 1 10 "#" "\\" .scheme.prepro
```

0x010

If this is an end of line token (`0x002`) then

The rest of the line is checked for any valid brackets.

Else if this is a bracket token (`0x004`) then

This is still searched for after an end of line token is found.

Else

Ignored

This feature enables the searching and hilighting of specific brackets contained within a to−end−of−line scheme. For example, consider the following C code:

```
#define My_Token 0x01  /* This is a multi−lined comment
```

```
                                        * describing My_Token */
```

With the '#' pre−processor hilight (see bit 0x08 above) the #define line would all be hilighted with the pre−process scheme, the comment would be missed causing incorrect hilighting of the next line. Instead this feature may be used by both the pre−processor and comment hilight tokens to correctly hilight the above example:

```
hilight 1 26 "#" "\\" .scheme.prepro
hilight 1 20 "/\\*" "*/" "" .scheme.comment
```

0x020

This token must be the first non−white character of the line.

0x040

The token (and closing bracket tokens) are to be replaced by the given replacement strings. This is often utilized when displaying formated text such as MicroEmacs on−line help ehf(8) pages, the output from UNIX **man(1)** etc. In MicroEmacs help pages, the start of bold text is delimited with the character sequence "`\C[cD`" and ends with the character sequence "`\C[cA`", e.g.

```
"the word \C[cDbold\C[cA is in \C[cDbold\C[cA"
```

Obviously the hilight delimiters should not appear so the character sequence may be correctly drawn using a bracket token, starting with "`\C[cD`" and ending with "`\C[cA`", replacing both with an empty string:

```
hilight 1 0x44 "\C[cD" "" "\C[cA" "" "" .scheme.bold
```

0x080

This is a branch token. When this token is found, the token (or the replace string) is colored using the given color *schemeNum* and then the current hilighting scheme is changed to "*b−hil−no*" (which MUST be defined by the time it is first used). The "*b−hil−no*" hi−light scheme should also contain a branch token which branches back to "*hil−no*" or "`0`" (which branches to $buffer−hilight(5)). A branch does not have to branch back to "*hil−no*", it may branch to any other hi−light scheme. The branches are not stacked and there is no limit on the nesting.

0x100

The token must be at the start of the line.

0x200

This is an invalid token in its own right, which is used for optimizing a hi−lighting scheme.

This has the second highest precedence (see **0x400**) and all other bits are ignored. Only the first 3 arguments are required. For example, if there are 11 tokens starting with "`delete−`" as with the hi−lighting of this buffer, then adding the token "`delete−`", while invalid in its own right, means that "`delete-`" is only checked for once. This also reduces the size of the internal hilighting tables so if the message "**Table full**" appears, the hilighting scheme should be reduced by removal of the common components.

0x400

This is a column hilighting token, which allows absolute columns within a window to be hilighted (irrespective of the contents). This bit takes precedence over all other bits and all other bits are ignored. Column highlighting is a different concept to token in that it requires a "*from−col*" and a "*to−col*" column positions and a line will be hilighted in the given scheme between these two columns.

0x800

The flag is used with bracket tokens (`0x04`) and indicates that the bracket is typically contained on a single line. This information is used by MicroEmacs in trying to avoid hilighting anomalies caused when the start and end tokens of the bracket are the same (e.g. a string's start and end token is '`"`'). Problems arise when the bracket starts on one line and closes on a later line, even with a large look−back, eventually the start bracket will become too far back and only the end bracket is found. But as this is the same as the open token it is mistaken for an open bracket and the strings become out of synch. This test can reset this if further down the file an open and close bracket is found on the same line. For this to have any effect, the hilighting scheme must use look−back (flag `0x02` of **Hilight Creation**).

Note that `0x004`, `0x008` and `0x080` are mutually exclusive and more than 1 should not be set in any one hilight token, if 2 or more are set the effect is undefined. Other than this there is no restrictions placed on the types of token used, although strange combinations like `0x006` may lead to unexpected results −− hopefully not a core dump, but not guaranteed !

The token and close token of brackets may contain a limited subset of regular expression tokens as follows:−

^

When specified as the first character of the token, the token must be at the start of the line.

$

The token must be at the end of the line, must be the last character.

\{

Indicates the start of the hilighted part of the token, only one may be used per token. This token use is different from regex.

\}

Indicates the end of the hilighted part of the token, only one may be used per token. The rest of the token must be matched for it to be used but is not considered part of the token, i.e. hilighting continues on the character immediately after the "\}", not at the end of the token. Similar to the \< token, the length of the rest of the token must be fixed. This token use is different from regex.

\(.\)

Groups are supported in hilighting, but they must only enclose a single character, closures ('*', '?' and '+') must come after the closure, i.e. use "\(.\)*", not "\(.*\)". Alternatives ("`\|`") are not supported.

.

Matches any character.

[...]

Matches a single buffer character to a range of characters, for example to hilight MicroEmacs register variables (i.e. `#g0−#g9`, `#p0−#p9`, `#l0−#l9`) the following regex string may be used:

```
hilight 1 1 "#[gpl][0−9]"
```

This matches a token which starts with a '#', followed by a '`g`', '`p`' or '`l`' character and ends with a numerical digit. If the user required the replacement (bit 0x40) of the "#" to "#register" to aid readability, the replacement string some now needs to know whether the second character was a '`g`', '`p`' or '`l`' and which digit. Up to 9 groups ("`\(.\)`") can be use to store a store a single search character, which can be used later in the search string and in the replacement string by using the form "\#", where # is the range test number counting from the left, e.g. for the given example use:

```
hilight 1 65 "#\\([gpl]\\)\\([0−9]\\)" "#register\\1\\2"
```

The content of the brackets (**[...]**) include a set of special short cuts and regular expression syntax definitions as follows:−

`[abc]`

A list of characters.

`[a−z]`

A range of characters.

`[−.0−9]`

A combination of character lists and ranges.

`[[:space:]]`

A white space character. See set−char−mask(2) for a full description on MicroEmacs character range support.

`[[:digit:]]`

A digit, 0−9.

`[[:xdigit:]]`

A hexadecimal digit, 0−9, a−f, A−F.

`[[:lower:]]`

A lower case letter, by default a−z.

`[[:upper:]]`

An upper case letter, by default A−Z.

`[[:alpha:]]`

A lower or upper case letter.

`[[:alnum:]]`

A lower or upper case letter or a digit.

`[[:sword:]]`

A spell word character.

[^...]

Matches all characters except the given range of characters, e.g. "`[^[:space:]]`".

\#

The same character which matched the #th group token. This functionality is best explained using UNIX **man(1)** output as an example, to create a bold character '**X**' it produces "`X\CHX`" where `\CH` is a backspace character thereby overstriking the first 'X' with another creating a bold character. This can be checked for and simulated in MicroEmacs using the following:

```
hilight 1 64 "\\(.\\)\CH\\1" "\\1" .scheme.bold
```

The use of "`\1`" in the search string ensures that the second character is the same as the first. This is replace by a single character drawn in the bold scheme.

? + *

Matches 0 or 1, 1 or more and 0 or more of the previous character or character range respectively.

Following is a list of hilighting regular expression restrictions:

The number of characters to the left of a **{** and to the right of a **}** token must be fixed, i.e. the '**?**', '**+**' and '*****' tokens cannot be used before this token. Consider the hilighting of a C function name defined to be a token at the start of a line followed by 0 or more spaces followed by a '('. The following hilight token looks valid but the variable space match is incorrect as it is to the right of the **}**:

```
hilight 1 0 "^\\w+\\}\\s-*(" .scheme.function
```

Instead either the space match must be include in the function token hilighting (which may cause problems, particularly if printing with underlining) or by fixing the number of spaces as follows:

```
; include the spaces in the function hilighting
hilight 1 0 "^\\w+\\s-*\\}(" .scheme.function
; or fix the number of spaces to 0, 1 ...
hilight .hilight.c    0 "^\\w+\\}(" .scheme.function
hilight .hilight.c    0 "^\\w+\}\\s-(" .scheme.function
```

The **+** and ***** tokens match the longest string and do not narrow, e.g. consider the hilighting of a C goto label which takes the form of an alpha−numerical name at the start of a line followed by a ':' character. The token "`^.*:`" cannot be used as `.` will also match and move past the ending ':', ending only at the end of the line. As no narrowing is performed the final '`:`' in the token will not match and the label will not be hilighted. Instead a character range which excludes a ':' character must be used, e.g. "**^[^:]*:**".

A group should not be followed by a **?** or ***** closure, it should either be a fixed single character or followed by a **+** closure (in which case the last matching character is stored).

Following is a list of hilight type bit / token regex equivalents:

0x01

`"[^word]\{????\}[^word]"`

0x02

`"????.*"`

0x20

`"^\s-*\{????"` − (note that this is strictly incorrect as the `\s-*` is to the left of the `\{`, it is correctly handled for the ease of use).

0x100

`"^????"` **Hilight Scheme Token Deletion**

−1 **hilight** "*hil−no*" "*type*" "*token*" With a −ve argument **hilight** deletes the given "*token*" from a hi−light color scheme identified by "*hil−no*". The token "*type*" must also be specified to distinguish between normal and column token types.

EXAMPLE

Example 1

Hilighting a MicroEmacs character given in hex form, checking its validity (i.e. "`\x??`" where `?` is a hex digit):

```
hilight 1 0 "\\x[[:xdigit:]][[:xdigit:]]" .hilight.variable
```

Hilighting a C style variable length hex number (i.e. "`0x???`"):

```
hilight 1 1 "0[xX][[:xdigit:]]+" .hilight.variable
```

Example 2

Replacing a quoted character with just the character (i.e. `'x'` −> `x`)

```
hilight 1 64 "'\\(.\\)'" "\\1" %magenta
```

Example 3

The following example uses the branch hilighting feature to hilight each window line a different color to its neighbors by cycle through 3 different color schemes:

```
0 hilight .hilight.line1 0                        $global−scheme
  hilight .hilight.line1 0x80 "\\n" .hilight.line2 .scheme.no1
0 hilight .hilight.line2 0                        .scheme.no1
  hilight .hilight.line2 0x80 "\\n" .hilight.line3 .scheme.no2
0 hilight .hilight.line3 0                        .scheme.no2
  hilight .hilight.line3 0x80 "\\n" .hilight.line1 $global−scheme
```

Example 4

Simulate the hilighting from the output of a UNIX man page (taken from hkman.emf):

```
0 hilight  .hilight.man 0                                   $global−scheme
; ignore
hilight .hilight.man 64 ".\CH" ""                           $global−scheme
; normal underline/italic
hilight .hilight.man 64 "_\CH\\(.\\)\\}[^\CH]" "\\1"     .scheme.italic
hilight .hilight.man 64 "\\(.\\)\CH_\\}[^\CH]" "\\1"     .scheme.italic
; bold − first is for nroff −man
hilight .hilight.man 64 "\\(.\\)\CH\\1\\}[^\CH]" "\\1"   .scheme.bold
hilight .hilight.man 64 "_\CH_\CH_\CH_\\}[^\CH]" "_"     .scheme.header
hilight .hilight.man 64 "\\(.\\)\CH\\1\CH\\1\CH\\1\\}[^\CH]" "\\1" .scheme.header
; bold underline
hilight .hilight.man 64 "_\CH_\CH_\CH_\CH_\\}[^\CH]" "_" .scheme.italic
hilight .hilight.man 64 "_\CH\\(.\\)\CH\\1\CH\\1\CH\\1\\}[^\CH]" "\\1" .scheme.ita
```

This replaces the complex nroff character string with a single hi−lighted character (don't believe me, try it!).

NOTES

MicroEmacs hilight was written with speed and flexibility in mind, as a result the user is assumed to know what they are doing, if not the effects can be fatal.

SEE ALSO

File Language Templates, $buffer−hilight(5), add−file−hook(2), add−color−scheme(2), print−scheme(2), indent(2), $system(5), print−buffer(2).

ini(9)

SYNOPSIS

ini, hpj, reg, rgy − MS−Windows initialization and registry files

FILES

hkini.emf − MS−Windows initialization and registry files.

EXTENSIONS

.ini − MS−Windows Initialization File
.hpj − MS−Windows Help Project File
.reg − Registry File
.rgy − (Other) registry File

DESCRIPTION

The **ini** file type templates provide simple hilighting of MS−Windows initialization and registry files. The file format is similar to a number of other registry type files which are also over−loaded into the same template.

Hilighting

The template provides minimal hilighting , but allows the different components of the file to be differentiated.

Folding and Information Hiding

Generic folding is enabled within the ini files. The folds occur about lines with leading square brackets **[...[** located on the left−hand margin. fold−all(3) (un)folds all regions in the file, fold−current(3) (un)folds the current region. **BUGS**

None reported.

SEE ALSO

bat(9).

Supported File Types

html(9)

SYNOPSIS

html − HyperText Markup Language File.

FILES

hkhtml.emf − HyperText Markup Language file hook definition

EXTENSIONS

.htm, **.html** − HyperText Markup Language File.
.htp − *[Special]* Super HTML Preprocessor file.
.hts − *[Special]* Super HTML file.

MAGIC STRINGS

<html>

MicroEmacs '02 recognizes the magic string on any of the first 4 lines of the file. The HTML files may be extension−less and are still recognized. **DESCRIPTION**

The **html** file type template provides simple hilighting of HTML files. Additionally, MicroEmacs '02 is capable of rendering simple HTML files (without graphic content) into the current buffer and follow the hyper text links. The JASSPA HTML documentation may be viewed in this way.

General Editing

HTML files may be edited or processed and rendered into the buffer. The **Use Author Mode** option in the buffer−setup(3) dialog determines the edit mode on loading a HTML file; when set to 'N' the page is rendered, 'Y' and the raw HTML file is presented. The default state is 'Y'.

Hilighting

The hilighting features allow commands, variables, logical, preprocessor definitions, comments, strings and characters of the language to be differentiated and rendered in different colors.

Short Cuts

The short cut keys used within the buffer are:−

C−c C−h − Help information.

C−c C−a − Toggle the HTML author status.
html−spell−check−word − spell check the current word.

Rendered Mode

In the non−author mode, the HTML file is extracted and rendered to the buffer. The hypertext links may be followed by selecting them with the mouse or using the `<RETURN>` key.

The rendered mode is typically used to check HTML text after it has been authored from the editor. The rendered mode only caters for regular HTML 2.0 text. It does not handle tables or frames etc. *(use a browser)*.

Toggling between rendered and authoring mode, then the buffer should be killed as the translation is only performed when the file is read.

The non−author mode can be permanently enabled by setting the **Use Author Mode** option in the buffer−setup(3) dialog to `'N'`. When set to N any HTML files loaded are automatically processed, and rendered according to their HTML content.

NOTES

The print driver may be used to generate HTML from the contents of the buffer. Select the printer destination as *buffer*, and *HTML* as the driver. The buffer being printed is converted to HTML and dumped in the print buffer.

.hts and **.htp** are computer generated extended HTML files used in the MicroEmacs '02 documentation system.

BUGS

None reported.

SEE ALSO

print−buffer(2), buffer−setup(3).

Supported File Types

hunt−forward(2)

NAME

hunt−forward − Resume previous search in forward direction hunt−backward − Resume previous search in backward direction

SYNOPSIS

n **hunt−forward** (**C−x h**)
n **hunt−backward** (**C−x C−h**)

DESCRIPTION

hunt−forward repeats the last search with the last search string in a forwards direction, from the current cursor position. magic(2m) and exact(2m) modes are operational.

hunt−backward repeats the last search with the last search string in a backwards direction, as per **hunt−forward**.

The numeric argument *n* is interpreted as follows:−

n > **0**

The *n*th occurrence of the pattern is located.

n < **0**

The first occurrence of the pattern is located in the next *n* lines. **DIAGNOSTICS**

The command returns a status of FALSE if no previous search string has been established, or if the pattern could not be located (or *n*th pattern where *n* occurrences are requested). If the pattern is found within the given search criteria the return status is TRUE.

SEE ALSO

exact(2m), isearch−forward(2), magic(2m), search−backward(2), search−forward(2), Regular Expressions

Installation(1)

INSTALLATION

This page describes introductory notes for the installation and setup of MicroEmacs '02.

Quick Install

The quickest way to install MicroEmacs without reading the rest of this document is to:−

- Create a new directory i.e. `me` or `microemacs`.
- Unpack the macros archive into this directory.
- Unpack any spelling dictionaries into this directory.
- Unpack the executable into this directory.
- Run `me` from this directory.

On starting, use the mouse and configure the user from the menu bar:−

```
Help−>User Setup
```

This allows the user and screen settings to be altered. On becoming more accustomed to the editor then a fuller installation may be performed.

Getting Help

See Contact Information for full contact information. A mail archive exists at:−

```
http://groups.yahoo.com/group/jasspa/
```

If you wish to participate in the list then you must first register by sending an empty mail message body to:−

```
jasspa−subscribe@yahoogroups.com
```

You will then be able to mail any questions into the group. Registration is required in order to prevent *spam* mailings from entering into the lists.

Distribution

MicroEmacs is distributed in the following files:−

Complete Installations

The Microsoft '95/'98/NT platforms may be installed using the **Install Shield** installation utility and do not require the components specified in later sections.
`jasspame.exe` − '95/'98/NT Self Extracting Install Shield Installation

Executable Source Code

The source code release for MicroEmacs '02 contains makefiles (`*.mak`) for all supported platforms. Microsoft '95/'98/NT makefiles contain options at the top of the makefile to enable/disable console and URL support. `mesrc.zip` − Source code for all platforms
`mesrc.tar.gz` − Source code

Executable Images

`medos.zip` − DOS Executable
`mewin32.zip` − Windows 32' (95/98/NT) Executable
`mewin32s.zip` − Windows win32s (Win3.1/3.11) Executable
`meirix6.gz` − Silicon Graphics Irix 6 Executable
`meaix43.gz` − IBM's AIX 4.3 Executable
`mehpux10.gz` − Hewlett Packard HP−UX 10 Executable
`mehpux11.gz` − Hewlett Packard HP−UX 11 Executable
`mesunos55.gz` − Sun OS 5.5 Executable
`mesunos56.gz` − Sun OS 5.6 Executable
`mesolx86.gz` − Sun Solaris 2.6 Intel Platform Executable
`melinux20.gz` − Linux 2.0.0 Executable
`mefreebsd.gz` − Free BSD Executable

Help File Images (all platforms)

`mewinhlp.zip` − Windows Help file
`mehtm.zip` − HTML Help files for 8.3 file systems (.htm)
`mehtml.tar.gz` − HTML Help files (.html)

Macro File Images (all platforms)

`memacros.zip` − Macro files
`memacros.tar.gz` − Macro files

Spelling Dictionaries (all platforms)

One of the following base dictionaries is required for spelling. The extended dictionaries require the base dictionary and are recommended for a more comprehensive spelling list. Other languages are supported.

`lsdmenus.zip` − American rules and base dictionary.
`lsdxenus.zip` − American extended dictionary.
`lsdmengb.zip` − British rules and base dictionary.
`lsdxengb.zip` − British extended dictionary.
`lsdmfifi.zip` − Finnish rules and dictionary.
`lsdmfrfr.zip` − French rules and dictionary.
`lsdmdede.zip` − German rules and base dictionary.
`lsdxdede.zip` − German extended dictionary.
`lsdmitit.zip` − Italian rules and dictionary
`lsdmplpl.zip` − Polish rules and dictionary.

`lsdmptpt.zip` − Portuguese rules and dictionary.
`lsdmeses.zip` − Spanish rules and dictionary.

`lsdmenus.tar.gz` − American rules and base dictionary.
`lsdxenus.gz` − American extended dictionary.
`lsdmengb.tar.gz` − British rules and base dictionary.
`lsdxengb.gz` − British extended dictionary.
`lsdmfifi.tar.gz` − Finnish rules and dictionary.
`lsdmfrfr.tar.gz` − French rules and dictionary.
`lsdmdede.tar.gz` − German rules and base dictionary.
`lsdxdede.gz` − German extended dictionary.
`lsdmitit.tar.gz` − Italian rules and dictionary
`lsdmplpl.tar.gz` − Polish rules and dictionary.
`lsdmptpt.tar.gz` − Portuguese rules and dictionary.
`lsdmeses.tar.gz` − Spanish rules and dictionary.

NOTE: The binary versions of the executables held on the site include the platform name as part of the executable name i.e. **me** for `DOS` is called **medos.exe**. On installing the binaries onto the target machine, you should rename the executable to **me** or **me.exe**, whatever is appropriate. The ONLY exception to this rule is the Microsoft Windows executable where **mewin32.exe** should be renamed to **me32.exe**. Our reason for this naming is to allow the executables to be unpacked in the same directory and not be confused with each other.

Quick Start Guild For All Platforms

Simply create a directory, down−load the files required (see list for each platform below) and extract into this directory. From a shell or command prompt, change to the directory, making it the current one (i.e. **cd** to it), and run the executable. MicroEmacs '02 should open with the on−line help page visible.

On Windows based systems this can also be achieved by creating a short−cut and setting the Working Directory in Properties to this path.

To enable MicroEmacs to be run from any directory, simply include this directory in you **PATH** environment variable. Alternatively, copy the executable to somewhere in your PATH and set the environment variable MEPATH to point to this directory.

MicroEmacs '02 will function normally in this environment, but in multi−user environments and for up−dating purposes, it is strongly recommended that a proper installation is used, see below.

Installation

DOS

Executable:

Compiled with DJGPP V1.0

Distribution components required:

> `medos.zip`
> `memacros.zip`
> <*spelling*>`.zip`
>
> `mewinhlp.zip` if you are using windows 3.1/3.11

Recommended installed components:

`4dos` − Command shell (giving *stderr* redirection).
`grep` − Version of grep (djgpp recommended)
`make` − Version of make (djgpp recommended)
`diff` − Version of diff (djgpp recommended)

Installation:

Create the directory `c:\me` (or other location)

> Unzip the MicroEmacs components into `c:\me`
>
> Edit "`c:\autoexec.bat`" and add the following lines:−

```
SET MENAME=<name>
SET PATH=%PATH%;c:\me
SET MEPATH="c:\me"
```

> Reboot the system.
>
> MicroEmacs may be run from the command line using

```
me
```

Graphics Cards:

MicroEmacs may be configured to the text modes of your graphics card. Refer to you graphics card DOS text modes to identify the text modes supported by your monitor. The text mode number may be entered into the user monitor configuration, defined in **Help−>User Setup**.

Running From Windows (3.x)

> The DOS version of MicroEmacs may be executed from a **.pif** file. Use the pif editor to create a new **.pif** file to launch MicroEmacs. The size of the DOS window may be configured from the command line, set the terminal size using one of the following command lines:−

```
me −c −v$TERM=E80x50        - 80 x 50 window
me −c −v$TERM=E80x25        - 80 x 25 window.
```

We usually add the `-c` option so that MicroEmacs is executed with history information. This may be omitted if required.

Windows 3.1/3.11

Executable:

Compiled with Microsoft Developer 2.0

Helper DLL:

Under **Win32s** a helper DLL **methnk16.dll** is required to perform the pipe−shell−command(2) in a synchronous manner. This should be installed into the `C:\WINDOWS\SHELL` directory. This (rather inelegantly) gets around the problems of spawning a process under **win32s** due to a number of Microsoft bugs in the operating system. Note: that on a spawn operation a `MS-DOS` window is visible, this is due to the nature of the command shell on this platform which has a tendency to prompt the user at every opportunity, hence a certain amount of interaction (which is out of our control) is necessary.

The helper DLL is compiled with a 16−bit Windows compiler − MSVC 1.5.

Distribution components required:

`mewin32s.zip`
`memacros.zip`
`mewinhlp.zip`
<*spelling*>`.zip`

Recommended installed components:

`4dos` − command shell (giving *stderr* redirection)
`grep` − Version of grep (GNU port of grep recommended)
`diff` − Version of diff (GNU port of grep recommended)
`make` − use nmake or GNU port of make.

win32s

win32s is a requirement on this platform, typically taken from **pw1118.exe** which freely available on the Internet.

Installation:

This version of Windows does not have a *install* directory as '95/'98 and it is expected that the `MS-DOS` version will coexist. No *Install Shield* installation is provided. Install in a directory structure similar to `MS-DOS`. Install the helper DLL **methnk16.dll** in the `C:\WINDOWS\SHELL` directory. Create a me32.ini(8) file in the `C:\WINDOWS` directory to identify the location of the MicroEmacs '02 components, this much the same as the '95/'98 file, change the directory paths to suite the install base.

Support Status:

The **win32s** release has not been used with vengeance, although no specific problems have been reported with this release.

Windows '95/'98/NT

Executable:

Compiled with Microsoft Developer 5.0

Install Shield

An **Install Shield** version of MicroEmacs is available which includes all of the distribution components.

Distribution components required:

`mewin32.zip`
`memacros.zip`
<spelling>`.zip`
`mewinhlp.zip` (optional)

Recommended installed components:

`4dos` or `4nt` − command shell
`grep` − Version of grep (GNU port of grep recommended)
`diff` − Version of diff (GNU port of grep recommended)
`make` − use nmake or GNU port of make.

Installation:

Create the directory "`C:\Program Files\Jasspa\MicroEmacs`" (or other location)

Unzip the MicroEmacs components into "`C:\Program Files\Jasspa\MicroEmacs`"

Create the file "`c:\windows\me32.ini`" and add the following lines:−

```
[Defaults]
mepath=C:\Program Files\Jasspa\MicroEmacs
userPath=C:\Program Files\Jasspa\MicroEmacs
fontfile=dosapp.fon
```

Create a short cut to MicroEmacs for the Desktop

Right click on the desk top

```
=> New
=> Short
=> Command Line: "c:\Program Files\Jasspa\MicroEmacs\me.exe -c"
=> Short Cut Name: "MicroEmacs"
```

MicroEmacs may be executed from the shortcut.

Open Actions

Microsoft Windows 95/98/NT provide short cut actions, assigning an open action to a file. The short cuts may be installed from the **Install Shieled** installation, but may alternativelly be explictly defined by editing the registry file with **regedit(1)**.

A file open action in the registry is bound to the file file extension, to bind a file extension *.foo* to the editor then the following registry entries should be defined:−

```
[HKEY_CLASSES_ROOT\.foo]
"MicroEmacs_foo"
[HKEY_CLASSES_ROOT\MicroEmacs_foo\DefaultIcon]
"C:\Program File\JASSPA\MicroEmacs\meicons,23"
[HKEY_CLASSES_ROOT\MicroEmacs_foo\Shell\open]
"&Open"
[HKEY_CLASSES_ROOT\MicroEmacs_foo\Shell\open\command]
"C:\Program File\JASSPA\MicroEmacs\me32.exe -o "%1""
```

In the previous exaple the *DefaultIcon* entry is the icon assigned to the file. This may be an icon taken from `meicons.exe` (in this case icon number 23), or may be some other icon. The open action in the example uses the **−o** option of the *client−server*, which loads the file into the current MicroEmacs '02 session, alternatively the **−c** option may be used to retain the previous context, or no option if a new session with no other files loaded is started.

A generic open for ALL files may be defined using a wildcard, this may be used to place a *MicroEmacs* edit entry in the right−click popup menu, as follows:−

```
[HKEY_CLASSES_ROOT\*\shell]
[HKEY_CLASSES_ROOT\*\shell\MicroEmacs]
"&MicroEmacs"
[HKEY_CLASSES_ROOT\*\shell\MicroEmacs\command]
"C:\Program File\JASSPA\MicroEmacs\me32.exe -o "%1""
```

UNIX

Executable:

Compiled with native compilers.

Distribution Components Required:

`me`*<unix>*`.gz`
`memacros.tar.gz`
<spelling>`.gz`
`html.tar.gz` (optional)

Installation:

It is recommended that all files are placed in `/usr/local`, although they may be installed locally.

> Unpack the executable and placed in "`/usr/local/bin`"
>
> Create the new directory "`/usr/local/microemacs`", unpack and install the `memacros.tar.gz` into this directory.
>
> For **csh(1)** users execute a "`rehash`" command and then me(1) can be executed from the command line.
>
> By default a X−Windows terminal is displayed, ensure that `$DISPLAY` and `$TERM` are correctly configured. To execute a terminal emulation then execute **me** with the `−n` option i.e. "`me   −n`". Note that this is not required if you are using a `vt100` emulation.

Organizing a local user profile

MicroEmacs uses local user configuration profiles to store user specific information. The user information may be stored in the MicroEmacs directory, or more typically in a users private directory. The environment variable `$MENAME` is typically used to determine the identity of the user.

The location of the user profile will depend upon your installation configuration.

Single Machine

For a single user machine it is typically easiest to use the installed MicroEmacs directory where user specific files are placed. This method, although not recommended, is simple as all files that are executed are in the same location. The `$MEPATH` is not changed.

UNIX

The UNIX environment is fairly easy and operates as most other UNIX applications. The user should create a MicroEmacs directory in their home directory for their own local configuration. Assigning a suitable name such as "`microemacs`", or if the file is to be hidden "`.microemacs`".

> The `$MEPATH` environment variable of the user should be modified to include the users MicroEmacs path BEFORE the default macros MicroEmacs path i.e.
>
> Ksh/Zsh:

```
export MEPATH=$HOME/microemacs:/usr/local/bin
```

> Csh/Bash:

```
setenv MEPATH $HOME/microemacs:/usr/local/bin
```

Where `$HOME` is defined as "`/usr/<name>`" (typically by default).

DOS/Windows

DOS and Windows are a little more tricky as numerous directories at the root level are more than a little annoying. It is suggested that the user directory is created as a sub−directory of the MicroEmacs directory. i.e.

"`c:\me\`*<user>*" for DOS

or

"`c:\Program Files\Jasspa\MicroEmacs\`*<user>*" for Windows

The `$MEPATH` environment variable (see me32.ini(8) for Windows) is modified to include the user component before the MicroEmacs component where `$MEPATH` is defined i.e.

```
SET MEPATH=c:\me\<user>;c:\me
```

where `<user>` is the user name (or `$MENAME`).

Alternative Directory Configurations

Numerous other configurations exist to organize the macro directories, to take the directory organization to the extreme then it is sometimes easiest to keep all of the macro components separate. An installation layout which encompasses different macro directories for:−

- User profiles − 1 per user.
- Shared company profiles − 1 per organization.
- MicroEmacs macros which are updated from time to time.

The configuration on different systems may be defined as follows:−

UNIX

The shared files are placed in `/usr/local`

```
/usr
   \
  local
     \
  microemacs   − Spelling + standard macros
       \
     company   − Company specific files
```

The user profile is stored in the users directory

```
/usr
  \
<name>
```

```
          \
       microemacs    - User specific files
```

The user should configure the `$MEPATH` as:

```
MEPATH=$(HOME)/microemacs:/usr/local/microemacs/company:/usr/local/microema
```

DOS/WINDOWS

For DOS and MS−Windows environments, bearing in mind the problem of the root directory, then it is easier to use the "`me`" directory as a place holder for a number of sub−directories, using a configuration such as:−

```
              c:
              |
              me      - Place holder directory
            / | \
          /   |   \
     <name> macros company
```

The user should configure the `$MEPATH` as:−

```
SET MEPATH=c:\me\<name>;c:\me\company;c:\me\macros
```

User Profile Files

Files contained in the user profiles typically include:−

<name>`.emf` − The users start up profile.
<name>`.edf` − The users spelling dictionary.
<name>`.erf` − The users registry configuration file.

These files are established from the menu "**Help−>User Setup**". The "**Setup Path**" item defines the location of the files, but must be MANUALLY included in the `$MEPATH` environment.

Company Profiles

Company profiles include standard files and extensions to the standard files which may be related to a company, this is typically *<company>*.emf where *<company>* is the name of the company.

The directory may also include template files etf(8) files which defines the standard header template used in the files. Files in the "`company`" directory would over−ride the standard template files.

The company directory should be added to the `$MEPATH` after the user profile and before the MicroEmacs standard macro directory.

SEE ALSO

$MENAME(5), $MEPATH(5), Company Profiles, File Hooks, File Language Templates, User Profiles.

Interfacing(2)

INTERFACING

This sections describes how MicroEmacs '02 may be interfaced to external components.

Shells

A shell window may be opened within the context of the editor using the command ishell(3), whereby an interactive command shell is presented within a buffer.

In the Microsoft Windows environment a **cygnus** UNIX style BASH shell may be realised with the cygnus(3) command.

Debugger

Within the UNIX environment the GNU **gdb(1)** or native UNIX **dbx(1)** debuggers may be invoked from the editor using gdb(3) or dbx(3). respectively This invokes the debugger and follows the debugging process in the editor window, automatically opening the source files as the debugger calls for them.

Microsoft Developer Studio

In the Microsoft windows environment, the memsdev(1) DLL may be attached to the **Microsoft Developer Studio** to enable MicroEmacs '02 to be used in place of the in−built editor.

File Searching

File searching is performed using **grep(1)** using the grep(3) command. For Windows then the GNU grep utility is recommended, for MS−DOS then the DJGPP version of GNU grep is recommended.

File Differencing

Differencing files, or directories is performed using the **diff(1)** utility using the diff(3) command. For all platforms the GNU diff utility is recommended as this provides a comprehensive differencing that is not typically available with native UNIX diff utilities.

Tag Files

A **tag** capability exists (see find−tag(2)) such that source functions and alike may be located quickly using a **tags** file. The standard **ctags(1)** format is used by MicroEmacs. The **tags** file itself may be generated by MicroEmacs '02 from the menu (*Tools−>XX Tools−>Create Tags File*). Alternatively a **tags** file may be generated by the **ctags(1)** utility. This is typically standard on UNIX platforms. For Windows and DOS platforms then the **Exuberant Ctags** is recommended, this is available from:−

```
    http://darren.hiebert.com
```

A MicroEmacs '02 compatible tags file may be generated using the command line "`ctags −N --format=1 .`" cataloging the current directory. To generate **tags** for a directory tree then use "`ctags −NR --format=1 .`". Refer to the **Exuberant Ctags** documentation for a more detailed description of the utility.

Compilation

Compilation is performed using the compile(3) command. This invokes a command shell, typically using **make(1)** to initiate a build sequence.

Client−Server

The Client−Server interface allows other client applications to inject commands into an already existing MicroEmacs '02 session (the server), thereby controlling the editor remotely. This is typically used to inject new files into the editor to be presented to the user.

The *Client−Server* interface is available in both the UNIX and Microsoft Windows environments. This mechanism is used in the Microsoft windows environment by the memsdev(1) DLL to attach the **Microsoft Developer Studio** to MicroEmacs '02. This may be used with similar effects within the UNIX environments from the X−Window managers desktop in addition to other utilities such as **TkDesk(1)**.

Command Line Filer

MicroEmacs may be invoked as a command filter in it's own right, macro scripts have been developed to perform a **dos2unix(1)** conversion operation, generate tags files etc. See Command Line Filters.

SEE ALSO

ctags(1), compile(3), cygnus(3), dbx(3), diff(3), find−tag(2) gdb(3), grep(3), ishell(3), memsdev(1), Client−Server, Command Line Filters.

ifill−paragraph(3)

NAME

ifill−paragraph − Format a paragraph

SYNOPSIS

n **ifill−paragraph** (**esc q**)

DESCRIPTION

ifill−paragraph, like **fill−paragraph**, fills the current paragraph from the left margin to the current fill column. In addition ifill−paragraph also recognizes joined bullet lists and fills each bullet paragraph separately.

See fill−paragraph(2) for more information on the process of filling paragraphs.

EXAMPLE

Following are 2 copies of the same paragraph, the first has been filled using **ifill−paragraph**:

```
This  is the  main  paragraph  which  can be as long as  required,
following is a list of bullets, some with a sub−bullet  list. Here
is the list:
    a) The bullet paragraph can also be as long as required and it
       also can have a bullet  list  following  (sub−bullet  list)
       which will also be filled correctly. Here is the sub−bullet
       list:
       1. First  sub−bullet − again no length  restrictions,  this
          will be filled correctly.
       2. second sub−bullet − no problems.
       3. Third sub−bullet − again no length restrictions, this is
          getting boring.
    b) This is the second  major bullet and this can just carry on
       for ever, but all things must come to an
```

The following version has been filled using the normal **fill−paragraph**:

```
This  is the  main  paragraph  which  can be as long as  required,
following is a list of bullets, some with a sub−bullet  list. Here
is the  list:  a) The  bullet  paragraph  can  also  be as long as
required and it also can have a bullet list following  (sub−bullet
list) which will also be filled  correctly. Here is the sub−bullet
list: 1. First  sub−bullet  − again no length  restrictions,  this
will be filled  correctly. 2. second  sub−bullet − no problems. 3.
Third sub−bullet − again no length  restrictions,  this is getting
boring. b) This is the second major bullet and this can just carry
on for ever, but all things must come to an
```

NOTES

ifill−paragraph is a macro defined in `format.emf`.

SEE ALSO

fill−paragraph(2), paragraph−to−line(3).

imakefile(9)

SYNOPSIS

imakefile − Make file

FILES

hkimake.emf − Imakefile hook definition
imake.etf − Template file

EXTENSIONS

Imakefile, **imakefile** − Imakefiles.

DESCRIPTION

The **Imakefile** file type template handles the hilighting of the Imakefile files.

General Editing

On creating a new file, a new header is automatically included into the file. time(2m) is by default enabled, allowing the modification time−stamp to be maintained in the header.

By default, TAB's are enabled as this is the syntactical feature of the file.

Hilighting

The hilighting emphasizes the keywords and comments within the Imakefile. **BUGS**

No attempt is made to hilight any embedded shell commands.

SEE ALSO

makefile(9), time(2m).

Supported File Types

indent(2)

NAME

indent − Manage the auto−indentation methods

SYNOPSIS

0 **indent** "*ind−no*" "*flags*" "*look−back*"

indent "*ind−no*" "*type*" "*token*" ["*close*" ["*ignore*"]] ["*indent*"]

DESCRIPTION

The **indent** command creates and manages the auto−indenting methods, the process of creating a new indentation method is best described in File Language Templates. The command takes various forms as defined by the arguments. Each of the argument configurations is defined as follows:−

Indentation Method Creation

0 **indent** "*ind−no*" "*flags*" "*look−back*"

With an argument of `0`, **indent** creates a new indentation method with the integer handle *ind−no*. The indentation method is assigned to a buffer by setting $buffer−indent(5) to *ind−no*. *ind−no* cannot be 0 as setting **$buffer−indent** to zero disables indentation. If the indentation method with the same *ind−no* already exists, then the existing method is deleted and a new method may be created.

flags Sets the indent bit flags where:−

`0x01`

Indent method is case insensitive. Note that **indent** tokens must be specified in lower case.

look−back specifies the maximum number of lines, prior to the current line, considered when calculating the indentation of a line, i.e. if there are *look−back* number of lines between the line to be indented and the previous non−blank line then the current indentation is lost.

If *look−back* is set to 0 then the indentation is effectively disabled as the current indentation can never be found. The value may be specified in the range 0−255, a value of 10 is typically sufficient.

Indentation Rule Creation

indent "*ind−no*" "*type*" "*token*" ["*close*" ["*ignore*"]] ["*indent*"]

With the default argument of `1`, **indent** creates a new rule for the indentation method *ind−no* which must have previously been defined and initialized.

The indentation of a line in a buffer, which is using an indentation method, is affected by the token types matched on the line (*type* `f`, `o`, `s`) and the current indentation (if line is not of type `f`).

The current indentation is determined by searching the previous lines (look−back) for the indentation of the last indented line. This may not simply be the indentation of the last non−blank line, the exact indentation is determined by searching for tokens in the line and assessing their effect on the indentation of the current line.

The format of the regex valid in the "*token*" and "*close*" arguments are the same as at used by hilight token creation, see hilight(2) for more information.

The indent tokens may be assigned one of the following types, using the *type* argument. If the type is specified in upper case then the token must be surrounded by non−alpha−numeric characters:

Fixed (*type* = '`f`' or '`F`')

A line containing a fixed indent token will be indented to the given *indent* column from the left−hand edge. *indent* is the only argument specified. e.g. MicroEmacs macro `!goto` labels:−

```
indent .hilight.emf f "*" 0
```

producing

```
    .....
*label
    .....
```

The fixed token must be the first non−white character on the line, the rest of the line is ignored. The indentation of the previous line has no effect.

Indent−from−next−line−onward (*type* = '`n`' or '`N`')

The indentation changes by *indent* from the next line onwards from the current line. *indent* is the only argument specified. e.g. MicroEmacs macro `!if`:−

```
indent .hilight.emf n "!if" 4
```

Keeps the indentation of the `!if` line the same as the previous indentation, change the indentation on the following lines by an extra 4 characters, to produce:

```
....
!if
    ....
```

Indent−from−current−line−onward (*type* = '`o`' or '`O`')

Increment the current and following lines indentation by *indent*. *indent* is the only argument specified. e.g. MicroEmacs macro `!endif`

```
indent .hilight.emf o "!endif" -4
```

decrement the indent of the `!endif` line and following lines by 4 spaces producing:

```
    ....
!endif
....
```

Indent−single (*type* = 's' or 'S')

Changes the indentation of the current line ONLY by *indent*. *indent* is the only argument specified. e.g. MicroEmacs macro `!elif`:−

```
indent .hilight.emf o "!elif" -4
```

decrements the indentation of the `!elif` line by 4 characters, but restores the previous indentation after the current line, producing:

```
    ....
!elif
    ....
```

Bracket (*type* = 'b' or 'B')

A bracket should be used when a starting token pairs with a closing token which may span multiple lines. i.e. the opening and closing braces of a programming language. Note that the opening and closing tokens must be different otherwise they cannot be differentiated. A bracket has two main effects:

When the previous line has an unmatched open bracket

In this situation the current line is indented to the right of the mismatched bracket.

When the previous line has an unmatched close bracket

In this situation the matching open bracket is hunted for in previous lines until either the *look−back* limit (See **Indentation Method Creation**) is exhausted or the bracket is matched, in which case the indent of that line is used.

For a bracket the only other argument given is the *close*. e.g. tcl's '`(`' and '`)`' brackets

```
indent .hilight.tcl b "(" ")"
```

Which produces:

```
....
.... (....
      ....
```

```
            ....)
    ....
```

Continue (*type* = '`c`' or '`C`')

Indicates that when *token* is found on the current line, the next line is a continuation of the current line. The indentation of the next line is the indentation of the first continuation line plus the given *indent*. *indent* is the only argument specified. e.g. tcl's '\'

```
indent .hilight.tcl c "\\" 10
```

A simple example is

```
....
12345678901234567890      \
         ....
....
```

When used in conjunction with brackets, the following effect is observed:

```
....
12345678901234567890      \
          ....(....       \
               ....)      \
          ....            \
          ....
....
```

This shows why the first continuation line (the `123456...` line) must be located and used as the base line from which the indentation is derived; again the *look−back* limits the search for this line.

Exclusion (*type* = '`e`' or '`E`')

Used to exclude text between start *token* and *close* token from the indentation calculation, typically used for quotes. The *ignore* argument is also specified (see hilight(2) `type 0x004` type bracket) e.g. MicroEmacs macro quotes:−

```
indent .hilight.emf e "\"" "\"" "\\"
```

e.g. tcl's quotes

```
indent .hilight.tcl e "\"" "\"" "\\"
```

producing:−

```
....
".... ignore { ... \" ... ignore another { token ... "
....
```

Ignore (*type* = '`i`' or '`I`')

Text to the right of a line containing *token* is to be ignored; typically used for comments. e.g. MicroEmacs macro '`;`' comment:−

```
indent .hilight.emf i ";"
```

Or tcl's '#' comment

```
indent .hilight.tcl i "#"
```

producing

```
....
# ... ignore this { indent token
....
```

EXAMPLE

Examples of indentation method creations can be found in macro files `hkemf.emf`, `hktcl.emf` and `hkvrml.emf`. The following example is taken from `hkemf.emf`:−

```
!if &sequal .hilight.emf "ERROR"
    set−variable .hilight.emf &pinc .hilight.next 1
!endif

...

0 indent  .hilight.emf 0 10
indent .hilight.emf N "define−macro" 4
indent .hilight.emf n "!if" 4
indent .hilight.emf s "!eli" −4
indent .hilight.emf s "!els" −4
indent .hilight.emf o "!end" −4
indent .hilight.emf n "!whi" 4
indent .hilight.emf o "!don" −4
indent .hilight.emf n "!rep" 4
indent .hilight.emf o "!until" −4
indent .hilight.emf o "!ema" −4
indent .hilight.emf e "\"" "\"" "\\"
indent .hilight.emf i ";"
indent .hilight.emf f "*" 0
```

SEE ALSO

File Language Templates, $buffer−indent(5), add−file−hook(2), hilight(2).

indent(2m)

NAME

indent − Automatic indentation

SYNOPSIS

indent Mode

I − mode line letter.

DESCRIPTION

indent mode, when enabled, ensures that a new text line is automatically indented to the same left hand column as the previous line's first non−white character. If the previous line contains no non−white characters then the line will not be indented. Automatic indentation is disabled when using *center* or *right* justification. **Indent** is usually used in conjunction with wrap(2m) and justify(2m).

SEE ALSO

buffer−mode(2), global−mode(2), wrap(2m) justify(2m).

info(9)

SYNOPSIS

info − GNU Info File.

FILES

info.emf − Info macro file.

EXTENSIONS

No fixed extension, the root of the *info* tree is specified by `$INFOPATH`/dir. The default search paths on different platforms are:−

`c:/info` − MS−DOS and MS−Windows (all).
`/usr/local/info` − All UNIX platforms.

DESCRIPTION

The GNU *info* files are handled by the command info(3) which starts the info reader. This reads the initial info file `dir` and initializes the *info* file traversal. Where the *info* directory is not in the aforementioned locations then the `$INFOPATH` environment variable should specify the base directory.

The standard *info* navigation keys are in effect within the *info* buffers. The *mouse* may also be used to select the next info page.

BUGS

There is no support within MicroEmacs '02 to regenerate the *info* tags and indexes.

SEE ALSO

info(3).

Supported File Types

insert−file(2)

NAME

insert−file − Insert file into current buffer

SYNOPSIS

n **insert−file** "*file−name*" (**C−x C−i**)

DESCRIPTION

insert−file inserts the named file *file−name n* times into the current buffer at the beginning of the current line. The buffer mark is set to the start of the insertion and the cursor is moved to the end.

SEE ALSO

set−mark(2), find−file(2), insert−file−name(2), view−file(2).

insert−file−name(2)

NAME

insert−file−name − Insert filename into current buffer

SYNOPSIS

insert−file−name (**C−x C−y**)

DESCRIPTION

insert−file−name inserts the current buffer's file name into the current buffer or, if entering text on the message line then enters the file name into the message line buffer.

SEE ALSO

insert−file(2), yank(2).

insert−macro(2)

NAME

insert−macro − Insert keyboard macro into buffer

SYNOPSIS

insert−macro "*command*"

DESCRIPTION

insert−macro inserts the named *command* into the current buffer in the MicroEmacs '02 macro language, thus enables it to be saved, re−load and therefore re−used at a later date. This is often used in conjunction with start−kbd−macro(2), end−kbd−macro(2) and name−kbd−macro(2). The given *command* must have been defined either by a keyboard macro or in MicroEmacs '02 macro code.

NOTES

The **insert−macro** provides a good method of identifying unknown low level key codes. Simply record the unknown key as a macro and insert the macro into the scratch buffer. The low level key code appears within the string.

SEE ALSO

start−kbd−macro(2), name−kbd−macro(2), define−macro(2), execute−file(2).

insert−newline(2)

NAME

insert−newline − Move the cursor to the next word

SYNOPSIS

n **insert−newline** (**C−o**)

DESCRIPTION

insert−newline inserts *n* new lines at the current cursor position, but does not move the cursor. Any text following the cursor is moved to the newly created line.

SEE ALSO

newline(2).

insert−space(2)

NAME

insert−space − Insert space(s) into current buffer

SYNOPSIS

n **insert−space**

DESCRIPTION

insert−space inserts *n* spaces at the current cursor position, moving the cursor position.

SEE ALSO

insert−string(2), insert−tab(2), insert−newline(2).

insert−string(2)

NAME

insert−string − Insert character string into current buffer

SYNOPSIS

n **insert−string** "*string*"

DESCRIPTION

insert−string inserts a string *n* times into the current buffer, moving the cursor position.

insert−string allows text to be built in a buffer without reading it from a file. Some special escape characters are interpreted in the *string*, as follows:

`\n` − Enters a new line
`\t` − A tab character
`\b` − Backspace
`\f` − Form−feed
`\\` − Literal backslash character '\'
`\xXX` − Hexadecimal value of character ASCII value

SEE ALSO

insert−file(2), insert−newline(2), insert−space(2), insert−tab(2), newline(2).

insert−tab(2)

NAME

insert−tab − Insert tab(s) into current buffer

SYNOPSIS

n **insert−tab** (**C−i**)

DESCRIPTION

insert−tab inserts *n* tab characters at the current cursor position, moving the cursor. The command is not affected by the tab(2m) mode as it always inserts literal tab characters.

SEE ALSO

insert−space(2), insert−string(2), insert−newline(2), tab(2), normal−tab(3), tab(2m).

ipipe−shell−command(2)

NAME

ipipe−shell−command − Incremental pipe (non−suspending system call)
ipipe−kill − Kill a incremental pipe
ipipe−write − Write a string to an incremental pipe

SYNOPSIS

n **ipipe−shell−command** "*command*" ["*buffer−name*"] (**esc backslash**)
n **ipipe−write** "*string*"
n **ipipe−kill**

PLATFORM

UNIX − *irix*, *hpux*, *sunos*, *freebsd*, *linux*.

Windows NT − *win32*.

DESCRIPTION

ipipe−shell−command executes the given system command *command*, opening up a ***icommand*** buffer into which the results of the command execution are displayed. Unlike the pipe−shell−command(2), the user may continue editing during command execution. The command may be terminated by deleting the buffer or issuing a **ipipe−kill** command.

The argument *n* can be used to change the default behavior of pipe−shell−command described above, *n* is a bit based flag where:−

0x01

Enables the use of the default buffer name ***icommand*** (default). If this bit is clear the user must supply a buffer name. This enables another command to be started without effecting any other command buffer.

0x02

Hides the output buffer, default action pops up a window and displays the output buffer in the new window.

0x04

Disable the use of the command−line processor to launch the program (win32 versions only).

By default the "**command**" is launched by executing the command:

```
%COMSPEC% /c command
```

Where `%COMSPEC%` is typically command.com. If this bit is set, the "**command**" is launched directly.

0x08

Detach the launched process from MicroEmacs (win32 versions only). By default the command is launched as a child process of MicroEmacs with a new console. With this bit set the process is completely detached from MicroEmacs instead.

0x10

Disable the command name mangling (win32 versions only). By default any '`/`' characters found in the command name (the first argument only) are converted to '`\`' characters to make it Windows compliant.

0x20

Displays the new process window, by default this window is hidden.

Many other macro commands (see compile(3), grep(3)etc.) use this command.

ipipe−write writes a string *string* to an open ipipe, *n* times.

ipipe−kill terminates an open ipipe, this is automatically called when the ipipe buffer is deleted using delete−buffer(2) or when MicroEmacs is exited.. The numeric argument *n* can be used to change the signal generated, where *n* can take the following values:

1

Sends a Terminate process signal, literally a `SIGKILL` signal on unix or a `WM_CLOSE` on windows platforms. This is the default signal and is typically bound to `C−c C−k`.

2

Sends an interrupt signal, writes a Ctrl−C to the <stdin> pipe on unix or sends Ctrl−C key events on windows platforms. This is typically bound to `C−c C−c`. **NOTES**

On UNIX platforms the TERM environment variable of the new process can be set by setting the user variable **%ipipe−term** to the required value, e.g.:

```
set-variable %ipipe-term "TERM=vt100-nam"
```

Ipipe shells support a large sub−set of vt100 terminal commands, notable exceptions are color and font support and the support of auto−margins. Using the terminal type "`vt100−nam`" disables the use of auto−margins, providing better support.

On platforms which do not support **ipipe−shell−command**, such as MS−DOS, executing **ipipe−shell−command** automatically invokes pipe−shell−command, hence macros may safely use ipipes without explicitly checking the platform type. **ipipe−shell−command** does not run reliably on Windows 3.11 and Windows 95; Windows NT does support ipipes.

While the pipe command is running, mode pipe(2m) is enabled. Modes lock(2m) and wrap(2m) effect the output behavior of an **ipipe−shell−command**.

EXAMPLE

The following example is the grep(3) command macro which utilizes the **ipipe−shell−command**, diverting the output to a buffer called ***grep***.

```
define-macro grep
    !if &seq %grep-com "ERROR"
        set-variable %grep-com "grep "
    !endif
    !force set-variable #l0 @1
    !if &not $status
        set-variable #l0 @ml00 %grep-com
    !endif
    !if @?
        1 pipe-shell-command &cat %grep-com #l0 "*grep*" @mna
    !else
        1 ipipe-shell-command &cat %grep-com #l0 "*grep*" @mna
    !endif
!emacro
```

Note that if an argument is passed to **grep** then it uses pipe−shell−command instead. This is useful if another command is using **grep** which must finish before the calling command can continue, see replace−all−string(3) for an example.

BUGS

On MicroSoft Windows platforms, **ipipe−shell−command** spawns the shell (e.g. `command.com`) with the appropriate command line to make it execute the given command. If the command to be run detaches from the shell and creates its own window, for example `me.exe`, **ipipe−kill** will only kill the shell, it will not kill the actual process, i.e. the `me.exe`.

On MicroSoft Windows platforms **ipipe−shell−command** does not work on Novell's Intranet Client v2.2 networked drives, version 2.5 does appear to work.

SEE ALSO

$buffer−ipipe(5), compile(3), grep(3), pipe−shell−command(2), replace−all−string(3), shell−command(2), pipe(2m), lock(2m), wrap(2m).

isearch−forward(2)

NAME

isearch−forward − Search forward incrementally (interactive)
isearch−backward − Search backwards incrementally (interactive)

SYNOPSIS

isearch−forward (**C−s**)
isearch−backward (**C−r**)

DESCRIPTION

isearch−forward provides an interactive search in the forward direction. This command is similar to search−forward(2), but it processes the search as each character of the input string is typed in. This allows the user to only use as many key−strokes as are needed to uniquely specify the string being searched.

The follow keys can be used at the start of an incremental search only:

`C−s` − Search for last string.
`C−m` − Perform a search−forward instead.
`esc p`,
`esc n` − Scroll through history list etc (See ml−bind−key(2)).

Several control characters are active while isearching:

C−s or **C−x**

Skip to the next occurrence of the current string

C−r

Skip to the last occurrence of the current string

C−h

Back up to the last match (possibly deleting the last character on the search string)

C−w

Insert the next word into the search string.

C−g

Abort the search, return to start.

esc or **C−m**

End the search, stay here

isearch−backward is the same as **isearch−forward**, but it searches in the reverse direction.

For both commands when the end of the buffer is reached, an alarm is raised (bell etc.) a further search request (`C−s`) causes the search to commence from the start of the buffer.

NOTES

The ml−bind−key(2) bindings are used.

The incremental search supports buffer modes exact(2m) and magic(2m).

BUGS

Due to the dynamic nature of active ipipe−shell−command(2) buffers the search history cannot be stored in the same way (list of fixed locations). As a result the search history is stored as a list of searches which are not guaranteed to be consistent.

SEE ALSO

exact(2m), hunt−forward(2), magic(2m), ml−bind−key(2), search−forward(2).
Regular Expressions

item−list(3)

NAME

item−list − Abbreviated search and list buffer contents.
item−list−find − Find the selected item in the item list
item−list−close − "Close the item list"

SYNOPSIS

item−list (**F7**)
item−list−find
item−list−close (**esc F7**)

DESCRIPTION

item−list performs a regular expression search of a buffer, presenting a list of the located text and associated types in a separate window which is presented to the left of the buffer window. **item−list** is a generic function that interacts with the buffer environment variables to present abbreviated buffer information to the user.

The regular expression search strings are predefined in the language templates. To add support for a new buffer type a list of search/replace strings must be created. The search strings must use regex (magic mode) and groups `\(..\)` to place the located object string into the replace string. Within the template buffer search strings (**s**) and replace (**r**) are defined with the following syntax:−

set−variable .*hookname*.item−list−s*x* "*regexp*"
set−variable .*hookname*.item−list−r*x* "*replace*"

Where:−

hookname

The name of the file hook i.e. `fhook−c` for ANSI 'C'.

x

The search number, this is valid in the range 1..9, commencing from 1. The search is processed in the order of the numeric identity.

regexp

The regular expression to search for. One of the arguments must include a groups `\(..\)` definition to allow the string to be moved to the replace.

replace

The replace string, this typically includes a *type* and part of the search string.

On invocation of **item−list** the buffer is searched and the results are presented in the `*item−list*` window appearing at the left−hand side of the window. If there is no item list set up for the file type then an error message is displayed.

The user may interact with the `*item−list*` buffer using the mouse or `<RETURN>`, on selecting a line then the user is moved to the corresponding line in the original buffer.

item−list−find finds the current item list item and searches for the text in the original buffer. This is typically bound to a mouse or key stroke action.

item−list−close closes the item list buffer.

EXAMPLE

The following example works through the **item−list** definition for the ME macros e.g. given that the ME macro definition is:

```
define−macro macro−name
```

Searching for "`define−macro \([a−z−]+\)`" and replacing with "`Macro \1`" will work most of the time. The space between `define−macro` and the name does not have to be a single space and the *name* does not have to contain just lower case letters, so these search strings should be a flexible as possible, try

```
"define−macro\s +\(\w+\)"
```

This however is not as optimal as it could be and if you have large files this could become slow. Performance can be greatly increased if it can be anchored to the start of the line, e.g.

```
"^define−macro\s +\(\w+\)"
```

but to allow for initial white spaces and the optional numeric argument, you really need

```
"^\s *[0−9]*\s *define−macro\s +\(\w+\)"
```

To hilight the function name you need the name encased the name in a magic hilighting string,

```
"\ecBmacro−name\ecA"
```

where `\e` is an escape char, so the replace string should be

```
"Macro \ecB\1\ecA"
```

Now all thats needed is to set these variables as fhook command variables, for macro files, the file hook command is `fhook−emf`, therefore the following is required:

```
        set-variable .fhook-emf.item-list-s1 "^\\s *[0-9]*\\s *define-macro\\s +\\(\\w+\\)
        set-variable .fhook-emf.item-list-r1 "Macro \ecB\\1\ecA"
```

Note that you can have as many of these search and replace variables as you require, i.e. `.item-list-s1`, `.item-list-s2`, `.item-list-s3`, ... ; but the more you have the slower it will be, often a good regex can do the job of 2 or 3.

SEE ALSO

occur(3), search−forward(2), Regular Expressions

java(9)

SYNOPSIS

java − Java programming language templates

FILES

hkjava.emf − Java programming language hook definition
java.etf − Java programming language template file

EXTENSIONS

.java, **.jav** − Java

DESCRIPTION

The **java** file type templates share much with the c(9) template definitions, utilising the electric 'C' features for automatic layout of text.

General Editing

On creating a new file, a new header is automatically included into the file. time(2m) is by default enabled, allowing the modification time−stamp to be maintained in the header.

Hilighting

The hilighting features allow commands, variables, logical, preprocessor definitions, comments, strings and characters of the language to be differentiated and rendered in different colors.

Auto Layout

The electric C−Mode cmode(2m) performs automatic layout of the text, variables such as c−brace(5) allow the brace position and text formation to be modified.

> restyle−region(3) and restyle−buffer(3) are available to reformat (re−layout) selected sections of the buffer, or the whole buffer, respectively.
>
> Comments may be formatted using `esc o`, which reformats the comments according to the current fill paragraph. If a comment commences with `/***...` then the comment is automatically formatted to a box.

Folding and Information Hiding

Generic folding is enabled within the Java files. The folds occur about braces **{...}** located on the left−hand margin. fold−all(3) (un)folds all regions in the file, fold−current(3) (un)folds the current region. Note that folding does not operate on K&R style code, with the trailing open brace.

Tags

A c−tags file may be generated within the editor using the **Tools** −> **Java−Tools** −> **Create Tag File**. find−tag(2) takes the user to the file using the tag information.

> On invoking the tag generator then the user is presented with a dialog box which specifies the generation option of the tags file. The base directory of the tags file search and tagging options may be specified to locate all of the definitions within the code space.
>
> The **tags** file is extremely useful where the user is dealing with inherited source code spread over multiple directories. Generation of a recursive tag file with all searching options enabled allows labels to be located extremely rapidly (certainly faster than IDE environments).

Folding and Information Hiding

Generic folding is enabled within the C and C++ files. The folds occur about braces **{...}** located on the left−hand margin. fold−all(3) (un)folds all regions in the file, fold−current(3) (un)folds the current region. Note that folding does not operate on K&R style code.

> The **Tools** −> **C−Tools** menu allows `#define`'s to be evaluated within the buffer. Where the state of a `#if` is established to be false (using the `#define` information) then the disabled region of code is grayed out indicating which regions of the code are active.

Working Environment

compile(3) may be invoked to rebuild the source, the user is prompted to save any files.

> rcs−file(2) is automatically invoked if an RCS file is detected, the normal check−in/out operations may be performed through the editor.

Short Cuts

The short cut keys used within the buffer are:−

> **C−c C−c** − Comment out the current line.
> **C−c C−d** − Uncomment the current line.
> **C−c C−e** − Comment to the end of the line with stars (*).
> **A−C−i** − Restyle the current region.
> **esc q** − Format a comment.
> **esc o** − Format a comment.
> **f2** − (un)fold the current region
> **f3** − (un)fold all regions

NOTES

The hilighting is typically extended using a file **myjava.emf**

SEE ALSO

c(9), c−brace(5), cmode(2m), compile(3), find−tag(2), javatags(3f), find−tag(2), fold−all(3), fold−current(3), rcs−file(2), restyle−buffer(3), restyle−region(3), time(2m).

Supported File Types

javatags(3f)

NAME

javatags − Generate a C tags file from Java sources.

SYNOPSIS

me "@javatags" [*−v%tag−option=<flags>*] [*files*]

DESCRIPTION

The start−up file `javatags.emf` may be invoked from the command line to generate a **tags** file for java source files.

Given a list of *files* a tags file `tags` is generated in the current directory, which may be used by the find−tag(2) command. This is a good alternative on Microsoft platforms where a utility such as **ctags(1)** is not typically available to process Java files. If no *files* are specified the default file list is "`./`", i.e. process the current directory. If a directory name is given (such as the default "`./`") all Java source files within the directory will be processed.

The value of variable **%tag−option** is used to control the tag generation process, its value <*flags*> can contain any number of the following flags:

`a`

Append new tags to the existing tag file, note that if also using flag '`m`' multiple 'tags' to the same item may be created.

`m`

Enable multiple tags. This enables the existence of 2 tags with the same tag name, but typically with different locations. See help on find−tag(2) for more information on multiple tag support.

`r`

Enables recursive mode, any sub−directory found within any given directories will also be processed.

`v`

Add global variables to the tag file. (i.e. variables marked with *extern*).

`s`

Add classes definitions to the tag file (i.e. *class*).

The generated tags file includes `#define` and C++ class names.

NOTES

This function is invoked from menu

Tools −> C Tools −> Create Tags File

when the user requests a tags file to be generated.

The user setup file "`myjavatags.emf`" is executed by javatags during start−up, this file can be used to over−ride any of the javatags configuration variables (see below).

The following variables are set within "`javatags.emf`" and are used to control the process:−

%tag−option

Tags options flag, default value is "". See above for more information.

%tag−filemask

A list of source file masks to be processed when a directory is given, default value is "`:*.java:*.jav:`".

%tag−ignoredir

A list of directories to be ignored when recursive option is used, default value is "`:SCCS/:CVS/:`".

These variables can be changed using the −v command−line option or via the "`myjavatags.emf`" file

SEE ALSO

find−tag(2), start−up(3), java(9).

justify(2m)

NAME

justify − Justification Mode

SYNOPSIS

justify Mode

J − mode line letter.

DESCRIPTION

justify mode, when enabled, performs paragraph justification as designated by $fill−mode(5) − capable of *left*, *right*, *both* or *center* justification of text. Justify removes all white spaces at the end of the line, if there are no non−white characters on the line then the line is always left empty. If the justification method is *center* or *right* then all white spaces are removed at the beginning of the line. If the line is longer than the $fill−col(5) or the method is *left* then nothing more is done, else the line is appropriately justified. The method used is set by the variable $fill−mode(5). Justify is usually used in conjunction with wrap(2m) and indent(2m).

SEE ALSO

buffer−mode(2), global−mode(2), wrap(2m) indent(2m), $fill−col(5), $fill−mode(5).

kbd−macro−query(2)

NAME

kbd−macro−query − Query termination of keyboard macro

SYNOPSIS

[Definition]
kbd−macro−query (**C−x q**)

[Execution]
kbd−macro−query "**y**"|"**n**"|"**C−g**"

DESCRIPTION

kbd−macro−query queries the termination state of keyboard macro recording. If the command is executed during a keyboard macro definition, at that point during its execution the user is prompted as to whether to continue the macro execution. A reply of "**y**" continues the execution as normal, "**n**" stops execution at that point once, if executing the macro *n* times the macro will still executed a further *n−1* times. If the "`C−g`" abort command is entered then all keyboard macro execution is aborted, regardless of the number of repetitions.

SEE ALSO

start−kbd−macro(2), execute−kbd−macro(2).

keyNames(2)

KEY BINDING NAMES

Every key which can be generated in MicroEmacs '02 has a character string or name representation which can be used to bind and unbind the key to a command. The name of simple keys like "`a`" or "`$`" is simply the character, i.e. "`a`" and "`$`". Following is a list of other parts to a key name.

Modify Keys

There are 3 modifying keys, Shift, Control and Alt, these are represented as "`S−`", "`C−`", "`A−`" respectively. For example the key "`A−C−S−up`" is generated when the up cursor key is pressed when Shift, Control and Alt keys where also pressed.

The Control and Alt modifiers are case insensitive so `C−a` is the same as `C−A` and `C−S−a`.

Prefix Keys

Many binding are single stroke key sequences (e.g. "`C−a`" => beginning−of−line). However MicroEmacs '02 has a prefix(2) command which can be used to bind up to 8 single stroke keys, turning them into two stroke keys; this greatly increasing the number of available bindable key sequences. For example **prefix 1** is bound to the escape character (`esc`), this allows key sequences like "`esc a`" to be used. Following is a list of prefixes and their default bindings

prefix 1 => `esc`
prefix 2 => `C−x`
prefix 3 => `C−h`
prefix 4 => `C−c`

Special Keys

Following is a complete list of recognized keyboard key names, not all are able to be generated on every platform:−

`backspace`, `delete`, `down`, `end`, `esc`, `f1`, `f2`, `f3`, `f4`, `f5`, `f6`, `f7`, `f8`, `f9`, `f10`, `f11`, `f12`, `f13`, `f14`, `f15`, `f16`, `f17`, `f18`, `f19`, `f20`, `home`, `insert`, `kp−0`, `kp−1`, `kp−2`, `kp−3`, `kp−4`, `kp−5`, `kp−6`, `kp−7`, `kp−8`, `kp−9`, `kp−add`, `kp−begin`, `kp−decimal`, `kp−delete`, `kp−divide`, `kp−down`, `kp−end`, `kp−enter`, `kp−home`, `kp−insert`, `kp−left`, `kp−multiply`, `kp−page−down`, `kp−page−up`, `kp−right`, `kp−subtract`, `kp−up`, `left`, `page−down`, `page−up`, `return`, `right`, `space`, `tab`, `up`

The name of any key can be obtained by using describe−key(2).

Mouse Keys

Following is a list of mouse related keys:−

mouse−pick−1, **mouse−pick−2**, **mouse−pick−3**, **mouse−pick−4**, **mouse−pick−5**

These keys are generated when the user presses a mouse button, these key events are always created. On most systems button 1 is the left, 2 the middle and 3 the right mouse button. If the system only has a 2 button mouse then a `mouse−pick−2` cannot be generated. The order of the buttons can be revered (i.e. 1 becomes right) and the number of buttons can be set using the $system(5) variable. Note that X−servers support up to 5 buttons and with the growing popularity of pilot 'wheel' mice, the 4th and 5th button are often used to report wheel spin up and down events. The translate−key(2) command can be used to translate these buttons to the mouse wheel keys.

mouse−drop−1, **mouse−drop−2**, **mouse−drop−3**, **mouse−drop−4**, **mouse−drop−5**

These keys are generated when the user release a mouse button, these key events are always created.

mouse−move−1, **mouse−move−2**, **mouse−move−3**, **mouse−move−4**, **mouse−move−5**, **mouse−move**

These key events are generated when the user moves the mouse and are only if they are bound to a command. The key generated depends on whether a button is being held down by the user, if the user is pressing button 1 then a `mouse−move−1` key is created etc.

mouse−time−1, **mouse−time−2**, **mouse−time−3**, **mouse−time−5**, **mouse−time−5**, **mouse−time**

These key events are generated only when they are bound to a command. They are pseudo keys created when the user hold the mouse buttons done for a period of time, see **Pseudo Keys** below for more information.

mouse−wheel−up, **mouse−wheel−down**

Pilot mouse wheel events, generated when the wheel is spun up or down respectively. **Modifier Keys**

The Shift, Control and Alt modifier keys will also generate key input whenever pressed or released. The keys are however only generated if they are bound to a command. The key names are as follows:

S−pick, **S−drop**

Shift modifier.

C−pick, **C−drop**

Control modifier.

A−pick, **A−drop**

Alt modifier.

Note that the keys are only generated when another key is pressed, i.e. if the user presses and holds only the shift key, no "`S−pick`" key will be generated until another key, such as `down`, is also pressed. If the shift key is released before another key is pressed the event will not be reported.

Pseudo Keys

Pseudo keys events cannot be directly created by the user, they are created internally by MicroEmacs. They are treated like normal keys to allow the user to handle the events properly themselves. Following is a complete list of the system generated pseudo keys:−

bell

The pseudo key is generated when the bell is rung.

callback

The pseudo key when a create−callback(2) macro is executed, this allows the executed macro to know it was executed via a create−callback as @cck(4) will be set to this.

idle−pick, **idle−drop**, **idle−time**

The commands bound to these keys are executed when the system becomes idle for a period of time. See help on $idle−time(5) for more information.

mouse−time−1, **mouse−time−2**, **mouse−time−3**, **mouse−time−4**, **mouse−time−5**, **mouse−time**

The command bound to these keys are executed when mouse button 1, 2, 3, 4, 5 or a combination are held bound for a period of time. See help on $delay−time(5) for more information.

redraw

The command bound to this pseudo key is executed whenever the screen needs redrawing, by default it is bound to screen−update(2). If the user unbinds this key then **screen−update** is still called, but if the user binds it to a function which does not redraw the screen, such as void(2), then the screen will not be up−dated.

The command executed is always given an argument, a non−zero argument indicates a forced complete redraw, an argument of zero indicates that just an up−date is required.

Alt Key

The **Alt Key** has special binding priorities defined as follows:−

- Direct key binding (e.g. **A−b** executes file−browser)
- Main menu hot key (e.g. **A−f** opens the File menu)

- Meta key binding (e.g. **A−space** −> **esc space** −> set−mark)

If the `ALT` key is to be used strictly as the Emacs Meta key then the bindings for the menu should be over−ridden by *Direct Key Bindings* from the user configuration file i.e. to re−map the default MicroEmacs Alt key to equivalent `esc` keys then the following keys should be re−bound.

```
global-bind-key forward-word "A-f"      ; Over-ride File menu binding
:                                       ; For all of the other menu items.
:
global-bind-key backward-word "A-b"     ; Over-ride the file browser.
global-bind-key replace-string "A-r"    ; Over-ride tools binding.
```

This creates a higher priority binding which overrides the underlying default. The commands that are displaced would have to be re−bound to different keys if required.

KEYBOARD MACROS

Keyboard macros do not store the name of keys, instead a more machine oriented format is used (usually in the form "`\s??`") these will work across platforms (assuming the key bindings are the same) but they may not work across different releases.

As a result it is advised that any long term macro should avoid named keys like `up` in favor of using a standard key binding such as `C−p`. See help on execute−string(2) for more information.

kill−line(2)

NAME

kill−line − Delete all characters to the end of the line

SYNOPSIS

n **kill−line** (**C−k**)

DESCRIPTION

kill−line, when used with no argument *n*, deletes all text from the cursor to the end of a line, the end of line character is also deleted if the cursor is in the first column and the line(2m) mode is disabled. The deleted text is placed in the kill buffer, see yank(2) for more information on the kill buffer. When used on a blank line, it always deletes it.

If a +ve argument *n* is supplied the specified number of lines is deleted, the setting of the **line** mode is ignore. If *n* is `0` the command has no effect. If a −ve argument is given, +*n* lines are deleted but the text is NOT added to the kill buffer.

NOTES

If a line is accidentally removed then yank the text back immediately or use undo(2).

The −ve argument is typically used in macro scripts where the yank buffer is more precisely controlled by the script.

SEE ALSO

kill−region(2), line(2m), undo(2), yank(2), forward−kill−word(2).

kill−paragraph(2)

NAME

kill−paragraph − Delete a paragraph

SYNOPSIS

n **kill−paragraph**

DESCRIPTION

kill−paragraph deletes the next *n* paragraphs, if *n* is +ve then the paragraph the cursor is currently in and the next *n*−1 paragraphs are killed. If *n* is −ve then the current paragraph and the previous *n*−1 paragraphs are killed. If *n* is zero the command simply returns. The default value for *n* is 1.

DIAGNOSTICS

The following errors can be generated, in each case the command returns a FALSE status:

[end of buffer]

The given argument *n* was greater that the number of remaining paragraphs, all the remaining paragraphs are still removed.

[top of buffer]

A negative argument *n* was given requesting more paragraphs to be killed then are present before the cursor. All the paragraphs before the cursor are still removed. **NOTES**

A paragraph is terminated by a blank line. All text residing between two blank lines is considered to be a paragraph − regardless of the text layout.

The distinction between killed text and deleted text is that text which is killed is placed into the yank buffer so that it can be pasted into any buffer using yank(2).

SEE ALSO

backward−paragraph(2), forward−paragraph(2), kill−region(2).

kill−rectangle(2)

NAME

kill−rectangle − Delete a column of text
yank−rectangle − Insert a column of text

SYNOPSIS

kill−rectangle (**esc C−w**)
n **yank−rectangle** (**esc C−y**)

DESCRIPTION

kill−rectangle deletes a rectangle (or column) of text defined be the cursor and the set−mark position. The text between the mark column and the cursor column is removed from every line between the mark line and the cursor line inclusive and copied to the kill buffer. The delete text may then be extracted from the kill buffer using yank(2) or **yank−rectangle**.

The mark position may be ahead or behind the current cursor position. If the rectangle column boundary divides a tab character which spans multiple columns, the tab character is replaces with the equivalent number of spaces. Similarly if the boundary divides an unprintable character which is displayed using multiple characters (e.g. '^A' for character 0x01) then spaces are inserted before the character to move it to the right of the boundary.

yank−rectangle inserts the current kill buffer (which may or may not have been generated using **kill−rectangle**) into the current buffer in a column fashion. That is to say that the first line of text in the kill buffer is inserted into the current line of text in the current buffer from the current cursor column, the cursor is then moved the the next line and placed at the same column. The process is then repeated for the second line of text in the kill buffer etc.

NOTES

The command copy−rectangle is not provided by default as this command is rarely required. If this command is required, the following macro definition can be used:

```
define−macro copy−rectangle
    set−alpha−mark "T"
    set−variable #l0 &bmod "view"
    set−variable #l1 &bmod "edit"
    set−variable #l2 &bmod "undo"
    −1 buffer−mode view
    1 buffer−mode undo
    kill−rectangle
    ; undo the kill and restore the buffer state
    undo
```

```
            &cond #l2 1 -1 buffer-mode "undo"
            &cond #l1 1 -1 buffer-mode "edit"
            &cond #l0 1 -1 buffer-mode "view"
            goto-alpha-mark "T"
            ; flag the command to be a copy-region type command
            set-variable @cl copy-region
        !emacro
```

SEE ALSO

set−mark(2), kill−region(2), yank(2), copy−region(2), reyank(2), undo(2).

kill−region(2)

NAME

kill−region − Delete all characters in the marked region

SYNOPSIS

n **kill−region** (**C−w**)

DESCRIPTION

kill−region deletes all characters from the cursor to the mark set with the set−mark(2) command. The characters removed are copied into the kill buffer and may be extracted using yank(2). If a numeric argument of `0` is given the command has no effect. If a −ve argument is given the characters are not placed in the kill buffer, therefore the text is effectively lost (this does not effect the undo(2) operation).

The mark position may be ahead or behind the current cursor position.

USAGE

To move text from one place to another:

- Move to the beginning of the text you want to move.
- Set the mark there with the set−mark (**esc space**) command.
- Move the point (cursor) to the end of the text.
- Use the **kill−region** command to delete the region you just defined. The text will be saved in the kill buffer.
- Move the point to the place you want the text to appear.
- Use the yank (**C−y**) command to copy the text from the kill buffer to the current point.

Repeat the last two steps to insert further copies of the same text.

NOTES

If a region is accidentally removed then yank the text back immediately or use undo(2).

Windowing systems such as X−Windows and Microsoft Windows utilize a global windowing kill buffer allowing data to be moved between windowing applications (*cut buffer* and *clipboard*, respectively). Within these environments MicroEmacs '02 automatically interacts with the windowing systems kill buffer, the last MicroEmacs '02 **kill−region** entry is immediately available for a paste operation into another windowing application.

SEE ALSO

copy−region(2), kill−rectangle(2), reyank(2), set−mark(2), undo(2), yank(2).

languageTemplates(2)

FILE LANGUAGE TEMPLATES

MicroEmacs '02 provides a large range of macros and templates to deal with the most commonly occurring types of ASCII file that may be edited. However, there is a requirement for users to extend this capability to include more obscure file types, in addition to bespoke files found internally within organizations, or devised by the user.

For each file type, MicroEmacs '02 may be tailored to recognize the file and modify it's hilighting, key binding configuration, osd display and indentation to accommodate the file. In addition, new shorthand macros may be introduced to help deal with the contents of the file.

This section outlines the steps to be taken to integrate a new file language template into MicroEmacs '02.

The scope of the File Type

The first step is to decide the scope of the file, this will determine where the file hook should be defined. The options are:−

A standard file type not supported

If this is a standard file type not supported by MicroEmacs '02 then it should be added to `me.emf`, in addition contact us and we will add it to the standard release. Any macro files associated with this file type should be available globally and are added to the MicroEmacs *macro* directory.

Local To your organization

If it is a file type local to your organization then it should be added to your *company*.emf file. Any macro files associated with the file type should be added to your local company MicroEmacs '02 directory.

Local to an individual

If this is a file type that is only used by a limited number of individuals then it should be added to the *user*.emf file. Any files associated with the file type are added to your local user MicroEmacs '02 directory.

Recognizing the File Type

The next step to adding a new file type is to get MicroEmacs '02 to recognize the file as the new type. Recognition is performed by the File Hooks which perform recognition on the file extension and/or the file content. The name of the file type must be determined, this is typically the name of the file prepended by `hk`. e.g. a file with extension *foo* uses the file `hkfoo.emf` for it's language specific definitions.

Using the add−file−hook(2) invocation the file recognition is bound to the file hook macro whenever the file type is loaded. The file hook is added to the appropriate global, company or user start up file as determined in step 1. The file hooks for file *foo* might be defined as follows, depending upon the recognition method:−

Recognizing the extension

To recognize the file extension, then a space separated list of extensions may be defined, including the dot `'.'` (or other) extension separator.

```
add−file−hook ".foo"       fhook−foo
```

Recognizing a magic editor string in the file

If the file type adopts multiple extensions (or does not use a file extension) then an editor specific string may be inserted into the file to enable the editor to recognize it, typically of the form `−!−` *type* `−!−`, if the string is GNU Emacs compatible then the `−*−` convention may be used. The binding is defined as:−

```
−1 add−file−hook "−!−[ \t]*foo.*−!−"          fhook−foo
```

Recognizing a magic string in the file

UNIX files use a "`#!<path>`" notation for executable ASCII files. If the file is this type of file (or uses any other type of common string in the as the first characters of a file) then the binding may be defined as follows, in this case we have assumed *foo* is the UNIX executable variety i.e. `#!/usr/local/bin/foo`:−

```
1 add−file−hook "^#!/.*foo" fhook−foo
```

Any, or all of the above recognition methods may be employed to invoke the language specific macro. Note that the methods are evaluated in a LIFO order, hence it is possible to over−ride an existing method.

Defining the Macro File

Once the hook has been defined, the language specific file must be created. Create the language specific file with the same name as defined in the hooks, removing the **fhook−** prefix and replacing it with **hk**, i.e. `fhook−foo` invokes the language specific file `hkfoo.emf`. Create, the file and add the file hook macro. for example hk*foo*.emf contents may be defined as:

```
define−macro fhook−foo
    ; Temporary comment to make sure that it works.
    ml−write "Loaded a foo file"
!emacro
ml−write "[MicroEmacs foo file hook loaded]"
```

The file hook may be tested by exiting and re−loading MicroEmacs '02, or simply by executing the file containing the `add−file−hook` function. Once the file bindings are installed a *foo* file may be loaded and the hook message should be displayed.

Modifying an Existing file hook

The standard file hooks supplied with MicroEmacs '02 should not be modified, typically a user will want to extend the repertoire of hi−lighting tokens to encompass locally defined programming libraries or syntactical extensions, in addition to extending support macros that are associated with the file type. In this case, an extension to the hook function is required. The hook file **my*XXX*.emf**, allows extensions to be made to the **hk*XXX*.emf**, without editing the original file. This may be considered to be an *include* file and is executed, if it exists, after the **hk** file has been executed. i.e. if the hook file **hkfoo.emf** is already defined and extensions are added to **myfoo.emf**.

Note that the **my*XXX*.emf** files do not typically include any **fhook−XXX** functions, the original *fhook* functions would be used. However, if a different buffer environment is required from the one created be the hook, such as a different setting of tab(2m) mode, the hook function should be copied to **my*XXX*.emf** and altered appropriately.

Adding Hilighting definitions

File specific hilighting is used to pick out key words and tokens used within the file type, it greatly improves readability; the hilighting is also used for printing. The hilighting is defined within the body of the file and is executed once when the hook file is loaded, this occurs when the hook function is executed. During development of the hilighting code, it is usually necessary to execute the hook buffer to view the effects of any changes to the hilighting.

The hilighting is defined using the command hilight(2) which requires a hilighting identifier, used to identify the hilighting scheme. This identifier is dynamically allocated when the hook file is loaded, again using *foo*, the identifier is allocated at the top of the file and is protected such that a value is assigned once only.

```
!if &sequal .hilight.foo "ERROR"
    set−variable .hilight.foo &pinc .hilight.next 1
!endif
```

The variable `.hilight.next` allocates unique hilighting numbers, typically a single hilighting number is consumed, incrementing the `.hilight.next` variable ready for the next allocation. The hilighting color scheme is defined in a macro variable **.hilight.***ext*, where *ext* is the name of the language scheme (i.e. *foo*).

Given a hilighting number, the hilighting scheme may be defined. Each of the tokens in the language is assigned a hilighting color, for our simple *foo* file type:−

```
0 hilight .hilight.foo 1                $global−scheme
hilight .hilight.foo 2 "#"              .scheme.comment
hilight .hilight.foo 4 "\"" "\"" "\\"   .scheme.string
hilight .hilight.foo 0 "'.'"            .scheme.quote
hilight .hilight.foo 0 "'\\\\.'"        .scheme.quote ; '\?' quoted char

hilight .hilight.foo 1 "if"             .scheme.keyword
hilight .hilight.foo 1 "then"           .scheme.keyword
hilight .hilight.foo 1 "else"           .scheme.keyword
hilight .hilight.foo 1 "endif"          .scheme.keyword
```

When the hilighting tokens have been defined, the hilighting scheme is bound to the buffer. This is performed by assigning $buffer−hilight(5) with the hilighting scheme within the *fhook* macro body, e.g.

```
define−macro fhook−foo
    ; Assign the hilighting
    set−variable $buffer−hilight .hilight.foo
    ; Temporary comment to make sure that it works.
    ml−write "Loaded a foo file"
!emacro
```

Putting it all together `hkfoo.emf` now comprises:−

```
!if &sequal .hilight.foo "ERROR"
    ; Allocate a hilighting scheme number
    set−variable .hilight.foo &pinc .hilight.next 1
!endif

; Define the hilighting scheme
0 hilight .hilight.foo 1                $global−scheme
hilight .hilight.foo 2 "#"              .scheme.comment
hilight .hilight.foo 4 "\"" "\"" "\\"   .scheme.string
hilight .hilight.foo 0 "'.'"            .scheme.quote
hilight .hilight.foo 0 "'\\\\.'"        .scheme.quote ; '\?' quoted char

hilight .hilight.foo 1 "if"             .scheme.keyword
hilight .hilight.foo 1 "then"           .scheme.keyword
hilight .hilight.foo 1 "else"           .scheme.keyword
hilight .hilight.foo 1 "endif"          .scheme.keyword

; File hook − called when new file is loaded.
define−macro fhook−foo
    ; Assign the hilighting
    set−variable $buffer−hilight .hilight.foo
    ; Temporary comment to make sure that it works.
    ml−write "Loaded a foo file"
!emacro

; Notification that hook is loaded.
ml−write "[MicroEmacs foo file hook loaded]"
```

Adding a Template

A template inserts initial text into a new file that is created. This mechanism is typically used to insert a standard header into the file on creation. The insertion text is defined within a template file, given the file extension etf(8), which is created in the corresponding global, company or user directory as determined in step 1. The template is named *ext*.etf, so for our example file *foo*, the template file is called `foo.etf`. We shall simply add a file header, our comment is # (as defined by the hilighting tokens). Our example *foo* template file `foo.etf` may be defined as follows:−

```
#-!- foo -!- #################################
#
#  Created By    : $USER_NAME$
#  Created       : $ASCII_TIME$
#  Last Modified : <160495.1521>
#
#  Description
#
#  Notes
#
#  History
#
#  Copyright (c) $YEAR$ $COMPANY_NAME$.
##############################################
```

The template file must be explicitly loaded by the hook file, within the **fhook** function. A new file condition may be tested within the fhook macro by checking the numerical argument, an argument of 0 indicates that this is a new file. The template file is inserted with an invocation of etfinsrt(3). The **fhook** macro checks the argument and inserts the template file as follows:−

```
; File hook - called when new file is loaded.
define-macro fhook-foo
    ; if arg is 0 this is a new file so add template
    !if &not @#
        etfinsrt "foo"
    !endif
    ; Assign the hilighting
    set-variable $buffer-hilight .hilight.foo
    ; Temporary comment to make sure that it works.
    ml-write "Loaded a foo file"
!emacro
```

Adding abbreviations

Abbreviations are short−cut expansions which may be defined for the language specific file. The abbreviations are defined in a eaf(8) file, *ext*.eaf, located in the appropriately defined MicroEmacs directory. The abbreviation file defines the key sequences which may be automatically inserted, under user intervention, using expand−abbrev(2). An abbreviation file for *foo*, `foo.eaf`, may be defined as:−

```
if "if \p\rthen\rendif\P"
el "else\r\p\P"
```

The binding to the hook is defined in the *fhook* macro using buffer−abbrev−file(2). For the example language file *foo* the *fhook* macro becomes:−

```
; File hook - called when new file is loaded.
define-macro fhook-foo
    ; if arg is 0 this is a new file so add template
    !if &not @#
        etfinsrt "foo"
    !endif
    ; Assign the hilighting
    set-variable $buffer-hilight .hilight.foo
```

```
            ; Set the abbreviation file
            buffer-abbrev-file "foo"
            ; Temporary comment to make sure that it works.
            ml-write "Loaded a foo file"
        !emacro
```

Automatic Indentation

Automatic indentation may be applied to the file, such that the indentation is automatically performed when new lines are entered into the file. Indentation also benefits from automatic re−styling operations using restyle−region(3) and restyle−buffer(3).

The indentation style is declared by defining language tokens that constitute positions in the syntax where the indentation is changed. The indentation requires a unique identifier to identify the indentation style, the hilighting identifier is used. If hilighting is not defined, then the language template may still obtain an identifier as described in the hilighting section.

The indention is create with an argument of `0` to the indent(2) command, the subsequent tokens are defined using **indent** with no argument. For our simple *foo* syntax then the indentation might be defined as follows:−

```
0 indent  .hilight.foo 2 10
indent .hilight.foo n "then" 4
indent .hilight.foo s "else" -4
indent .hilight.foo o "endif" -4
```

This provides an indentation of the form:−

```
if condition
then
    XXXX
else
    if condition
    then
        XXXX
    endif
endif
```

The indentation is bound to the buffer in the *fhook* macro by defining $buffer−indent(5). For the example file *foo* then the *fhook* is defined as:−

```
; File hook - called when new file is loaded.
define-macro fhook-foo
    ; if arg is 0 this is a new file so add template
    !if &not @#
        etfinsrt "foo"
    !endif
    ; Assign the hilighting
    set-variable $buffer-hilight .hilight.foo
    ; Assign the buffer indentation
    set-variable $buffer-indent .hilight.foo
    ; Set the abbreviation file
    buffer-abbrev-file "foo"
```

```
        ; Temporary comment to make sure that it works.
        ml-write "Loaded a foo file"
    !emacro
```

Setting Buffer Modes

Buffer modes which are to be adopted (or discarded) by the language specific file are defined in the *fhook* macro. Typical modes that are applied are:−

time

Enables time stamping on the file, modifying the time stamp field with the modification date and time.

indent

Automatic indentation, where the cursor is returned to the same column on entering a new line, rather than to the start of the line.

As an example, the *foo fhook* file becomes:−

```
    ; File hook - called when new file is loaded.
    define-macro fhook-foo
        ; if arg is 0 this is a new file so add template
        !if &not @#
            etfinsrt "foo"
        !endif
        ; Assign the hilighting
        set-variable $buffer-hilight .hilight.foo
        ; Assign the buffer indentation
        set-variable $buffer-indent .hilight.foo
        ; Set the abbreviation file
        buffer-abbrev-file "foo"
        ; Set up the buffer modes
        1 buffer-mode "time"
        1 buffer-mode "indent"
        ; Temporary comment to make sure that it works.
        ml-write "Loaded a foo file"
    !emacro
```

Assigning New Bindings

New bindings and language specific macros may be added to the language specific file. New macros, to extend the repertoire of commands specifically developed for the language file are defined within the macro body using define−macro(2) these are automatically loaded when the hook file is loaded, which in turn is loaded when the file type is identified and loaded.

New bindings, which may be associated with new macros or existing commands, are assigned within the *fhook* macro. As an example, we shall extend the *foo* language file to include a commenting and uncommenting macros, locally binding the macros to the keys "`C−c  C−c`" and "`C−c C−d`" respectively. The macro definitions are defined as follows:−

```
; Macro to comment a line
define−macro foo−comment−line
    !while &gre &pdec @# 1 0
        beginning−of−line
        insert−string "#"
        beginning−of−line
        forward−line
    !done
!emacro

; Macro to remove a comment from a line
define−macro foo−uncomment−line
    !while &gre &pdec @# 1 0
        beginning−of−line
        −1 search−forward "#"
        backward−delete−char
        forward−line
    !done
!emacro
```

The key bindings for the macros are defined for the local buffer ONLY, as such are added using buffer−bind−key(2). The bindings are declared in the *fhook* macro as follows:−

```
; File hook − called when new file is loaded.
define−macro fhook−foo
    ; if arg is 0 this is a new file so add template
    !if &not @#
        etfinsrt "foo"
    !endif
    ; Assign the hilighting
    set−variable $buffer−hilight .hilight.foo
    ; Assign the buffer indentation
    set−variable $buffer−indent .hilight.foo
    ; Set the abbreviation file
    buffer−abbrev−file "foo"
    ; Set up the buffer modes
    1 buffer−mode "time"
    1 buffer−mode "indent"
    ; Set up local bindings
    buffer−bind−key foo−comment−line "C−c C−c"
    buffer−bind−key foo−uncomment−line "C−c C−d"
    ; Temporary comment to make sure that it works.
    ml−write "Loaded a foo file"
!emacro
```

Allowing Other to Modify the Hook

Other users of the file hook may need to modify or extend the file hook, the most common form is the addition of user specific hilight tokens. MicroEmacs uses a simple mechanism of executing a user hook extension file if it exists. The extension file name must be of the form **my*XXX*.emf**, i.e. for our example it must be "`myfoo.emf`". This is performed at the end of the macro file so that anything within the file can be altered, it is executed as follows:−

```
; load in user extensions if found
!force execute-file "myfoo"
```

Note the !force(4) directive is used as the file may not exist.

Summing Up

The previous sections have presented the basic steps involved in setting up a new language file template. They cater for simple file types, for more complex examples then browse the **hk***xxx*.emf files.

The completed files that should have been generated by following the previous examples are now presented:−

file.foo

```
# This is a comment.
if condition
then
    do something
else
    if condition
    then
        do something
    endif
endif
```

hkfoo.emf

```
!if &sequal .hilight.foo "ERROR"
    ; Allocate a hilighting scheme number
    set-variable .hilight.foo &pinc .hilight.next 1
!endif

; Define the hilighting scheme
0 hilight .hilight.foo 1                $global-scheme
hilight .hilight.foo 2 "#"              .scheme.comment
hilight .hilight.foo 4 "\"" "\"" "\\"   .scheme.string
hilight .hilight.foo 0 "'.'"            .scheme.quote
hilight .hilight.foo 0 "'\\\\.'"        .scheme.quote ; '\?' quoted char

hilight .hilight.foo 1 "if"             .scheme.keyword
hilight .hilight.foo 1 "then"           .scheme.keyword
hilight .hilight.foo 1 "else"           .scheme.keyword
hilight .hilight.foo 1 "endif"          .scheme.keyword

; File hook − called when new file is loaded.
define−macro fhook−foo
    ; Assign the hilighting
    set−variable $buffer−hilight .hilight.foo
    ; Temporary comment to make sure that it works.
    ml−write "Loaded a foo file"
!emacro
```

```
; Define the indentation scheme
0 indent  .hilight.foo 2 10
indent .hilight.foo n "then" 4
indent .hilight.foo s "else" -4
indent .hilight.foo o "endif" -4

; Reset the hilighting printer format and define the color bindings.
0 hilight-print .hilight.foo
hilight-print .hilight.foo "i"  .scheme.comment
hilight-print .hilight.foo "b"  .scheme.keyword
hilight-print .hilight.foo "bi" .scheme.string .scheme.quote

; Macro to comment a line
define-macro foo-comment-line
    !while &gre &pdec @# 1 0
        beginning-of-line
        insert-string "#"
        beginning-of-line
        forward-line
    !done
!emacro

; Macro to remove a comment from a line
define-macro foo-uncomment-line
    !while &gre &pdec @# 1 0
        beginning-of-line
        -1 search-forward "#"
        backward-delete-char
        forward-line
    !done
!emacro

; File hook - called when new file is loaded.
define-macro fhook-foo
    ; if arg is 0 this is a new file so add template
    !if &not @#
        etfinsrt "foo"
    !endif
    ; Assign the hilighting
    set-variable $buffer-hilight .hilight.foo
    ; Assign the buffer indentation
    set-variable $buffer-indent .hilight.foo
    ; Set the abbreviation file
    buffer-abbrev-file "foo"
    ; Set up the buffer modes
    1 buffer-mode "time"
    1 buffer-mode "indent"
    ; Set up local bindings
    buffer-bind-key foo-comment-line "C-c C-c"
    buffer-bind-key foo-uncomment-line "C-c C-d"
    ; Temporary comment to make sure that it works.
    ml-write "Loaded a foo file"
!emacro

; Notification that hook is loaded.
ml-write "[MicroEmacs foo file hook loaded]"

; load in user extensions if found
!force execute-file "myfoo"
```

foo.eaf

```
if "if \p\rthen\rendif\P"
el "else\r\p\P"
```

foo.etf

```
#-!- foo -!- #################################
#
#  Created By    : $USER_NAME$
#  Created       : $ASCII_TIME$
#  Last Modified : <160495.1521>
#
#  Description
#
#  Notes
#
#  History
#
#  Copyright (c) $YEAR$ $COMPANY_NAME$.
##############################################
```

SEE ALSO

add−file−hook(2), buffer−abbrev−file(2), etfinsrt(3), execute−buffer(2), expand−abbrev(2), global−abbrev−file(2), hilight(2), scheme−editor(3), indent(2), indent(2m), restyle−buffer(3), restyle−region(3), time(2m), $buffer−hilight(5), $buffer−indent(5), etf(8), eaf(8), File Hooks.

latex(9)

SYNOPSIS

latex − TeX Documentation

FILES

hklatex.emf − Tex File hook definition
latex.etf − Template file

EXTENSIONS

.tex − TeX Documentation

DESCRIPTION

The **latex** file type template handles the hilighting of the TeX files. The hilighting is minimal, hilighting the key words and comments.

General Editing

On creating a new file, a new header is automatically included into the file. time(2m) is by default enabled, allowing the modification time−stamp to be maintained in the header.

Hilighting

The hilighting emphasizes the Tex embedded command strings and comments. No special recognition of the command strings is performed.

Outline Hilighting

The LaTeX content may be viewed with synthetic hilighting such that headers, text in bold and italic are displayed, removing the LaTeX control sequences.

Short Cuts

The short cut keys used within the buffer are:−

C−c C−c − Comment out the current line.
C−c C−d − Uncomment the current line.

The command **latex−compile** is available within the buffer which invokes an external process to build the text.

BUGS

No bugs reported

SEE ALSO

time(2m).

Supported File Types

letter(2m)

NAME

letter − Letter kill policy

SYNOPSIS

letter Mode

l − mode line letter.

DESCRIPTION

By default individually deleted characters are not added to the kill buffer unless an argument is given to the command. This allows the user to delete characters while preserving the kill buffer, at the expense of not being able to yank(2) the character back out. Enabling **letter** mode ensures that all deleted characters are added to the kill buffer.

NOTES

This mode is implemented for backwards compatability only and the use of it is strongly discouraged as this may alter the behaviour of many on the supporting macros. If this feature is required it would be preferable to use a numeric argument with the delete or backspace key binding as follows:

```
1 global-bind-key backward-delete-char "backspace"
```

The use of the numeric argument of 1 has the same effect.

SEE ALSO

buffer−mode(2), global−mode(2), yank(2), line(2m).

line(2m)

NAME

line − Line kill policy

SYNOPSIS

line Mode

L − mode line letter.

DESCRIPTION

By default an invocation of kill−line(2) at the left−hand margin will kill the whole line. If **line** mode is enabled and the line contains text then only the text is killed, leaving an empty line. If the line is empty then it is removed.

SEE ALSO

buffer−mode(2), global−mode(2), letter(2m).

line−scheme−search(3)

NAME

line−scheme−search − Search and annotate the current buffer

SYNOPSIS

line−scheme−search

DESCRIPTION

line−scheme−search provides a method of searching for text patterns within the current buffer and annotating any matches through colored line hilighting. A selection of line colors are provided to allow different search patterns to be assigned their own color.

line−scheme−search is generally used for annotating log files and alike, where indevidual lines are of interest in addition to the context about that line. The hilighting draws attention to the line, by providing a visual cue, allowing the contents of the file to be breifly scanned.

On invocation of **line−scheme−search** a osd(2) dialog is presented to the user, search patterns and their associated hilighting assignment are selected through this interface. The dialog entries are defined as follows:−

Search for

The text dialog entry box allows the search pattern to be entered. This may be a regular expression or plain text.

Color

The **Color** allows the line hilighting color scheme to be selected from a pop−up menu. The color **Remove** is special and allows previously applied line hilighting to be removed.

Case Sensitive

A check box that allows the search to be case sensitive or insensitive. This modifies the exact(2m) mode.

Magic Mode

A check box that enables/disables regular expression pattern matching. This modifies the magic(2m) mode.

Below

Searches and hilights lines matching the search pattern from the current cursor position to the end of the buffer.

Above

Searches and hilights lines matching the search pattern from the current cursor position to the top of the buffer.

All

Searches and hilights lines matching the search pattern for the whole buffer.

Clear All

Removes all line hilighting from the current buffer.

First

Moves to the top of the buffer and hilights the first line that matches the search pattern.

Next

Hilights the next line that matches the search pattern.

Reverse

Hilights the previous line that matches the search pattern.

Exit

Exits the hilighting search dialog. **NOTES**

line−scheme−search is a macro implemented in `hiline.emf`.

SEE ALSO

osd(2), $line−scheme(5).

list−buffers(2)

NAME

list−buffers − List all buffers and show their status

SYNOPSIS

list−buffers (**C−x C−b**)

DESCRIPTION

list−buffers splits the current window and in one half brings up a list of all the buffers currently existing in the editor. The active modes, change flag, and active flag for each buffer is displayed. (The change flag is a ***** character if the buffer has been changed and not written out. The active flag is not an **@** if the file had been specified on the command line, but has not been read in yet since nothing has switched to that buffer.)

The buffer list has some special command keys associated with it which allow the state of the buffers to be edited from the buffer list, the editing allows buffers to be killed and saved to disk. The key codes are defined as follows:−

1 − Switch to buffer

Switch to that buffer and make it the only buffer.

2 − Move to buffer

Switch the buffer list window to that buffer.

D − delete buffer

Flag buffer for deletion. A buffer scheduled for deletion is marked with a '**D**' in first column. The delete status is enacted by the '**X**' command, or may be removed with the '**U**' command.

S − save buffer

Flag buffer for saving. A buffer scheduled from saving is marked with a '**S**' in the second column. Note that a buffer may be marked for saving and deletion, the save operation is performed before the delete.

U − unmark buffer

Unmark the '**D**' and '**S**' flags on current line.

X − execute

Execute all the '**D**' and '**S**' flags currently set. The **S**ave is enacted first.

For all but '**X**', the buffer selected is the buffer noted on the current cursor line. These keys are not remappable.

SEE ALSO

list−variables(2), list−commands(2), split−window−horizontally(2).

list−commands(2)

NAME

list−commands − List available commands

SYNOPSIS

list−commands (**C−h c**)

DESCRIPTION

list−commands constructs a list of all known built in commands and macros that are currently defined by MicroEmacs '02 and presents a list of those commands in the buffer "***commands***". Each entry is formatted as:−

command .... keyCode

Where multiple keys are bound to the same command, then each of the *keyCode*'s is shown.

list−commands is similar to describe−bindings(2) except that the commands are presented in alphabetical order (as opposed to key binding order).

EXAMPLE

The following is an example of the output of **list−commands**:−

```
backward-char ................. "C-b"
                                "left"
backward-delete-char .......... "backspace"
                                "S-backspace"
backward-delete-tab ........... "S-tab"
backward-kill-word ............ "esc backspace"
backward-line ................. "C-p"
                                "up"
                                "C-up"
backward-paragraph ............ "esc ["
                                "esc p"
backward-word ................. "esc b"
                                "C-left"
beginning-of-buffer ........... "esc <"
                                "home"
beginning-of-line ............. "C-a"
buffer-bind-key
buffer-info ................... "C-x ="
buffer-mode ................... "esc ~"
                                "C-x m"
                                "insert"
```

```
buffer-unbind-key
:
:
```

SEE ALSO

describe−bindings(2), list−variables(2).

list−registry(2)

NAME

list−registry − Display the registry in a buffer

SYNOPSIS

list−registry

DESCRIPTION

list−registry lists the contents of the registry in the a buffer in a hierarchical format. The key name and any associated string is shown as a hierarchical tree.

The registry listing is generated in the buffer "`*registry*`".

SEE ALSO

read−registry(2), erf(8).

list−variables(2)

NAME

list−variables − List defined variables

SYNOPSIS

list−variables (**C−h v**)

DESCRIPTION

list−variables pops up a window with a list of all register, buffer, user and global variables with their current setting. The variables are shown for the current buffer from which the command was invoked

list−variables provides a good alternative to describe−variable(2) where the value of multiple variables is to be interrogated.

The output is displayed in four sections:−

Register variables

The current settings of the global register variables ('**#**' prefix).

Buffer Variables

The current setting of the buffer variables ('**:**' prefix). This variables relate to the current buffer from which the command was invoked.

System Variables

The current settings of the system variables ('**$**' prefix).

Global Variables

The current setting of the global variables ('**%**' prefix). **EXAMPLE**

An example output from **list−variables** is shown below:−

```
Register variables:

    #g0 ........................... "29"
    #g1 ........................... ""
    #g2 ........................... "ERROR"
    :
    :
    #g8 ........................... "ERROR"
    #g9 ........................... "ERROR"

Buffer [m2cmd086.2] variables:


System variables:

    $auto-time .................... "300"
    $buffer-bhook ................. "bhook-nroff"
    $buffer-bname ................. "m2cmd086.2"
    $buffer-ehook ................. "ehook-nroff"
    $buffer-fhook ................. "fhook-nroff"
    $buffer-fmod .................. "040"
    $buffer-fname ................. "d:/emacs/doc/m2cmd086.2"
    $buffer-hilight ............... "3"
    :
    :
    $window-width ................. "80"
    $window-x-scroll .............. "0"
    $window-xcl-scroll ............ "0"
    $window-y-scroll .............. "52"

Global variables:

    %black ........................ "0"
    %blue ......................... "4"
    %compile-com .................. "nmake "
    %cyan ......................... "6"
    %green ........................ "2"
    %grep-com ..................... "grep -n "
    :
    :
    %usr1mode ..................... "off"
    %white ........................ "7"
    %yellow ....................... "3"
```

SEE ALSO

describe−variable(2), list−commands(2).

localeSupport(2)

LOCALE SUPPORT

Locale support within MicroEmacs handles the hardware and software configuration with respect to location, including:−

Displayed Character Set
Keyboard Support
Word characters
Spell Support

There are many other locale problems which are not addressed in this help page. Supporting different locale configurations often requires specific hardware (a locale specific keyboard) and knowledge of the language and customs of the region. This makes it a very difficult area for one localized development team to support, as such, JASSPA rely heavily on the user base to report locale issues.

Note on Names and IDs

The language name is not sufficient to identify a locale (Mexican Spanish is different to Spanish Spanish) neither is the country name (two languages are commonly used in Belgium), so before we've really started the first problem of what to call the locale has no standard answer! Call it what you like but please try to call it something meaningful so others may understand and benefit from your work.

In addition, the *internal id* and *data file* names have a length limit of just four characters due to the "`8.3`" naming conversion of MS−DOS. The standard adopted by JASSPA MicroEmacs for the internal locale id is to combine the 2 letter ISO language name (ISO 639−1) with the 2 letter ISO country name (ISO 3166−1). Should the locale encompasses more than one country, then the most appropriate *country id* is selected.

Displayed Character Set

A character set is the mapping of an integer number to a display symbol (i.e. character). The ASCII standard defines a mapping of numbers to the standard English characters, this standard is well defined and accepted, as a result the character set rarely causes a problem for plain English.

Problems occur when displaying characters found outside the ASCII standard, such as letters with accents, letters which are not Latin based (e.g. Greek alphabet) and graphical characters (used for drawing dialog boxes etc.). There are many different character sets to choose between and if the wrong character set is selected then the incorrect character translation is performed resulting in an incorrect character display. If the character display looks incorrect then first try changing the font and character−set setting, these can be configured using the platform page of user−setup(3).

If the problem persists (i.e. because the character set used to write the text is not supported on your current system) use the charset−change(3) command to convert the text to the current character set.

If your character−set is not supported then first make sure that MicroEmacs will draw all of the characters to be used. By default MicroEmacs does not draw some characters directly as the symbol may not be defined. When a character is not defined then there will typically be a gap or space in the text at the unknown character, in some cases there may be no space at all which will make it very hard to use. The symbol(3) command (menu−>symbol) is a good way of looking at which characters can be used with the current character set.

For a character to be rendered (when in main text) or poked (drawn by screen−poke(2) or osd(2)) is defined by the set−char−mask(2) command. The characters that are used when drawing MicroEmacs's window boarders or **osd** dialogs is set via the $box−chars(5) and $window−chars(5) variables.

MicroEmacs attempts to improve the availability of useful graphics characters on Windows and UNIX X−Term interfaces. The characters between 0 and 31 are typically control characters with no graphical representation (e.g. new−line, backspace, tab etc.) if bit 0x10000 of the $system(5) variable is set then MicroEmacs renders its own set of characters. These characters are typically used for drawing boxes and scroll−bars.

With so many character sets, each with their own character mappings, then the problem of spelling dictionary support is also tied to the locale. MicroEmacs uses the ISO standard character sets (ISO 8859) internally for word and spelling support and therefore a mapping between the ISO standard and the user character set is required. This mapping is defined by using the '`M`' flag of the set−char−mask(2) command.

The user may declare the current character set in the platform page of user−setup(3). All the settings required for supporting each character set may be found in the `charset.emf` macro file, so if your character set is not supported, this is the file to edit.

Keyboard Support

The keyboard to character mapping is defined in the Start−Up page of user−setup(3), where the keyboard may be selected from a list of known keyboards. If your keyboard is not present, or is not working correctly, then this section should allow you to fix the problem (please send JASSPA the fix).

Most operating systems seem to handle keyboard mappings with the exception of MS−Windows which requires a helping hand. The root of the problems with MS−Windows is it's own locale character mappings which change the visibility status of the keyboard messages which conflict with Emacs keystroke bindings. To support key−bindings like '`C−tab`' or '`S−return`' a low level keyboard interface is required, but this can lead to strange problems with the more obscure keys, particularly with the '`Alt Gr`' accented letter keys. For example on American keyboards pressing '`C−#`' results in two '`C−#`' key events being generated, this peculiarity only occurs with this one key. On a British keyboard the same key generates a '`C−#`' followed by a '`C−\`'.

This problem can be diagnosed using the $recent−keys(5) variable. Simply type an obvious character, e.g. '`A`' then the offending key followed by another obvious key ('`B`'), then look for this key sequence in the **$recent−keys** variable (use the list−variables(2) or describe−variable(2) command). So for the above British keyboard problem the recent−keys would be:

```
          B C−\\ C−# A
```

($recent−keys lists the keys backwards). Once you have found the key sequence generated by the key, the problem may be fixed using the translate−key(2) to automatically convert the incorrect key sequence into the required key. For the problem above the following line is required:

```
      translate−key "C−# C−\\" "C−#"
```

Note that once a key sequence has been translated everything, including **$recent−keys**, receive only the translated key. So if you a suspected a problem with the existing definition, change the keyboard type in **user−setup** to **Default** so no translations are performed, quit and restart MicroEmacs before attempting to re−diagnose the problem.

All the settings required for supporting each keyboard may be found in the `keyboard.emf` macro file, so if your keyboard is not supported, this is the file you need to edit.

Word characters

Word characters are those characters which are deemed to be part of a word, numbers are usually included. Many MicroEmacs commands use the 'Word' character set such as forward−word(2) and upper−case−word(2). The characters that form the word class are determined by the language being used and this can be set in the Start−Up page of user−setup(3).

If your language is not supported you will need to add it to the list and define the word characters, these settings may be found in the `language.emf` macro file. The '`a`' flag of command set−char−mask(2) is used to specify whether a character is part of a word, you must specify the uppercase letter and then the lowercase equivalent so the case conversion functions work correctly.

A list of characters to be removed from the word character set is stored in the **.set−char−mask.rm−chars** variable. This is done so that the language may be changed many times in the same session of MicroEmacs without any side effects (such as the expansion of the word character set to include all letters of all languages). This makes MicroEmacs ideal for writing multi−language documents.

This may unfortunately be made a little more tricky by the requirement that this list must be specified in the most appropriate ISO standard character set (see **Displayed Character Set** section). When extending the word character set the characters have to be mapped to the current character set which may not support all the required characters. For example in the PC−437 DOS character set there is an e−grave (`` `e ``) but no E−grave so the E−grave is mapped to the normal E. As a result, if trying to write French text the case changing commands will behave oddly, for example:

```
          r`egle −> REGLE −> r`egl`e
```

The conversion of all '`E`'s to '`` `e ``' is an undesirable side effect of '`` `E ``' being mapped to `E`. This can be avoided by redefining the base letter again at the end of the word character list, for example:

```
      set−char−mask "a" "`E`eEe"
```

Spell Support

The current language is set using the Language setting on General page of user−setup(3), if your required language is not listed you must first create the basic language support by following the guide lines in the **Word Character** section above. If you Language is listed, select it and enable it by either pressing **Current** or saving and restarting MicroEmacs. in a suitable test buffer run the spelling checker, one of three things will happen:

The `Spelling Checker` dialog opens and spelling is checked

> The spelling checker is supported by the current language and can be used (the rules and dictionaries have been downloaded and installed).

Dialog opens with the following error message:

```
Rules and dictionaries for language "XXXX"
   are not available, please download.
```

> The spelling checker is supported by the current language but the required rules and dictionaries have not been downloaded. You should be able to download them from the JASSPA website, see Contact Information. Once downloaded they must be placed in the MicroEmacs search path, i.e. where the other macro files (like `me.emf`) are located.

Dialog opens with the following error message:

```
Language "XXXX" not supported!
```

> The spelling checker is not supported by the current language, see the following **Adding Spell Support** section.

Adding Spell Support

To support a language MicroEmacs's spelling checker requires a base word dictionary and a set of rules which define what words can be derived from each base word in the dictionary. The concept and format of the word list and rules are compatible with the **Free Software Foundation** GNU **ispell(1)** package.

The best starting point is to obtain **ispell** rules and word lists in plain text form, the web can usually yield these. Once these have been obtained the rules file (or affix file) must be converted to a MicroEmacs macro file calling the add−spell−rule(2) command to define the rules. The rule file should be named "`lsr`<*lang−id*>`.emf`" where "<*lang−id*>" is the spelling language id, determined by the **.spell.language** variable set in the `language.emf` macro file.

The `spellutl.emf` macro file contains the command **spell−conv−aff−buffer** which will attempt to convert the buffer but due to formatting anomalies this process often goes wrong so using the command **spell−conv−aff−line** (also contained in `spellutl.emf`) to convert a single line is often quicker. See existing spelling rule files (`lsr*.emf`) for examples and help on command add−spell−rule(2).

Note: the character set used by the rules should be the most appropriate ISO standard (see **Displayed Character Set** section), this can make the process much more difficult if the current character set not compatible, if you are having difficulty with this please e−mail JASSPA Support.

Once the rules have been created, create a dictionary for the language from the word lists, see help on command add−dictionary(2). The dictionary file name should be "`lsdm`<*lang−id*>`.edf`", if the dictionary is large and can be split into two sections, a set of common words and a set of more obscure ones, create two dictionaries calling the dictionary containing obscure words "`lsdx`<*lang−id*>`.edf`" and the other as above.

Once the generated word and dictionary files have been place in the MicroEmacs search path, the spelling checker should find and use them. Please submit your generated support to MicroEmacs for others to benefit.

SEE ALSO

user−setup(3), charset−change(3), set−char−mask(2), translate−key(2), $box−chars(5), $window−chars(5), $recent−keys(5).

lock(2m)

NAME

lock − Pipe cursor position lock

SYNOPSIS

lock Mode

k − mode line letter.

DESCRIPTION

This mode can only be used while an incremental pipe (started by ipipe−shell−command(2)) is running in the current buffer, denoted by the pipe(2m) being set. When this mode is enabled and MicroEmacs '02 buffer cursor is at the same location as the process shell cursor, the buffer cursor is automatically moved with the shell cursor.

This mode is automatically enabled for a piped buffer.

SEE ALSO

ipipe−shell−command(2), pipe(2m).

MacroNumericArguments(4)

NAME

@#, @? − Macro numeric arguments

SYNOPSIS

@# − The numerical argument to a macro
@? − The truth of the numerical argument to a macro

DESCRIPTION

All built−in commands and macros are invoked with a numerical argument. The argument is obtained from either the command line when the user invokes a command line such as:

esc 5 esc x forward−char

where the argument is entered after prefix 1 (**esc**). In this case, causing the cursor to be moved forward 5 characters. Within a macro file the same operation is defined as:−

5 forward−char

In both cases the numerical argument 5 is passed to the command requesting that the resultant operation is performed 5 times in succession before returning. The command itself is invoked once, it is the responsibility of the command to iterate if requested.

The command determines how the numerical argument is interpreted, in the case of spell−word the argument identifies the type of word that is being spelled and NOT the number of words to spell.

The invocation of named macros operate in the same way, the macro may use the variables **@?** and **@#** to determine the status of the numerical argument passed to it. The variables are interpreted as follows:

@?

A logical value defined as `TRUE` (1) if a numerical argument has been specified, otherwise `FALSE` (0).

@#

A signed integer value of the supplied numeric argument. If no argument is supplied (i.e. **@?**==`FALSE`) then **@#** is set to 1.

The **@?** and **@#** are only valid for the current macro invocation. Other macros or commands that are invoked have their own values of **@?** and **@#**.

EXAMPLE

Consider the following example, which sorts lines into alphabetical order using the sort−lines(2) function. A new command **sort−lines−ignore−case** is created using a macro to sort lines case insensitively regardless of the current buffer mode. The command **sort−lines** takes an optional argument which determines which column should be used to perform the sort.

```
;
; sort-lines-ignore-case
; Sort lines case insensitively regardless of the current 'exact' mode
; setting.
define-macro sort-lines-ignore-case
    set-variable #l0 &bmod exact
    -1 buffer-mode "exact"
    !if @?
        @# sort-lines
    !else
        sort-lines
    !endif
    &cond #l0 1 -1 buffer-mode "exact"
!emacro
```

@? is used to test the presence of the argument, if it is false **sort−lines** is invoked without an argument. When true the numeric argument is propagated e.g. **@# sort−lines**.

This particular macro highlights an important consideration when passing the numerical argument to other functions, had the macro been implemented as:

```
; INCORRECT IMPLEMENTATION
define-macro sort-lines-ignore-case
    set-variable #l0 &bmod exact
    -1 buffer-mode "exact"
    @# sort-lines
    &cond #l0 1 -1 buffer-mode "exact"
!emacro
```

then when **sort−lines−ignore−case** is invoked with no arguments **@#** is defined as 1, this is would be incorrectly propagated to **sort−lines** causing it to sort on column 1 rather than column 0 as expected.

SEE ALSO

MacroArguments, define−macro(2).

Mahjongg(3)

NAME

Mahjongg − MicroEmacs '02 version of the solitaire Mah Jongg game

SYNOPSIS

Mahjongg

DESCRIPTION

Mah Jongg is an ancient Chinese game usually played by four players with tiles similar to dominos. This is a MicroEmacs '02 version which was inspired by the X−Windows version of the same game. The X−Windows version for the solitaire game originally seen on the PC and later ported to SunView.

Theory Of Play

The object of the game is to remove all the tiles from the board. Tiles are removed by matching two identical tiles which have either an open left edge or open right edge. The only exception to this rule is that any open "*flower*" tile (bamboo `[BAMB]`, orchid `[ORCH]`, plum `[PLUM]`, or chrysanthemum `[CHRY]`) matches any other open "*flower*" tile and any open "*season*" tile (spring, summer, autumn, or winter) matches any other open "*season*" tile.

Tiles are stacked on the board, the height of the tile is indicated by the color coding as follows:−

`Level 5` − White
`Level 4` − Red
`Level 3` − Yellow
`Level 2` − Green
`Level 1` − Cyan

To remove a pair of tiles, click the left mouse button on a tile (which will show in the selection color) and then click the left mouse button on the matching tile. At this point, both tiles will disappear from the board. If after selecting the first tile, you decide that you don't wish to play that tile, simply reclick the left button on the selected tile, alternatively click the right button to deselect any selected tile.

To the right of the board are a number of control buttons. To select an option, click the left mouse button on it.

NEW

Start a new game (keyboard n).

SAME

Start the same game again (keyboard s).

QUIT

Exit the game (keyboard q).

HELP

This help page (keyboard esc h).

The counter shows the number of remaining tiles on the board, at the start of the game there are 144 tiles.

NOTES

Mahjongg is a macro defined in `mahjongg.emf`.

Mah Jongg may only be played with a mouse, there is no keyboard support, with the exception of the re−start keys.

ACKNOWLEDGEMENT

Thanks to Jeff S. Young who (I think) wrote the original X−Windows version, and whose manual page formed the basis of this page.

The tile patterns were inspired from the X−Windows tile patterns. The X−Windows tile patterns themselves are copyright 1988 by Mark A. Holm <tektronix!tessi!exc!markh>.

SEE ALSO

Games, Match−It(3), Patience(3).

MainMenu(3)

NAME

Main Menu − The top main menu

SYNOPSIS

n osd

DESCRIPTION

The main menu is provided to give an easier access to parts of MicroEmacs functionality, the menu is not burnt into MicroEmacs but defined on start−up in `me.emf` and `osd.emf`. The user−setup(3) command can be used to set whether the menu is always visible and if the Alt−Hotkeys are enabled (i.e. '`A−f`' to open the **File** menu).

The main menu is osd(2) dialog number `0` so key bindings can be made which will open the main menu, an argument of `0` will simply open the main menu, an argument of `0x0n0000` will not only open the main menu but also the `n`th sub menu, e.g. to open the edit menu use:

```
0x020000 osd
```

Following is a brief description of the main menu items:

File Menu

`New`

Changes the current buffer to a new buffer.

`Open`

Opens a dialog enabling the user to select files for opening into MicroEmacs. By default the dialog opens the selected file using command find−file(2), but if the view option is selected the view−file(2) command is used. The binary or encrypt options configure whether the files are to be loaded with binary(2m) or crypt(2m) modes enabled.

`Quick Open`

Opens a sub−menu list all user file types (defined in user−setup(3)). Selecting one will open another sub−dialog list all files of that type in the current directory, selecting a file will open it using command find−file(2).

`Favorites`

Opens a sub−menu enabling the user to add new favorite files, edit the existing list of favorite files, or select an existing favorite file in which case the file is opened using command find−file(2). The favorite file using to store the list is "**$MENAME**`.eff`" and is saved in the first path given in the $search−path(5). Each favorite file takes 2 lines in the file, the first is the text displayed in the dialog (note that characters '`\`' and '`&`' must be protected with a '`\`' and the '`&`' can be used to set the Hot key) and the second line is the file name. A line with a single '**−**' character creates a separater line in the dialog.

`Find Tag`

Only visible when a `tags` file is found in the current directory, the command jumps to the current tag or if not on a tag or the tag is not found, opens a dialog enabling the user to select a tag. See command find−tag(2) for more information.

`Find File`

Executes command file−browser(3).

`FTP`

Executes command ftp(3).

`Close`

Executes a dialog form of the command delete−buffer(2).

`Attributes`

Opens a dialog enabling the user to set the current buffers file attributes. See command file−attrib(3) for more information.

`Save`

Executes a dialog form of the command save−buffer(2).

`Save As`

Executes a dialog form of the command write−buffer(2).

`Save All`

Executes a dialog form of the command save−all(3).

`Printer Setup`

Opens a dialog which enables the user to configure the printer driver, output location and page layout (executes command print−setup(3)).

`Print`

Executes command print−buffer(2).

`Buffer`

Opens a sub−menu listing all created buffers, selecting one will change the current buffer to the selected one.

`Exit`

Executes command save−buffers−exit−emacs(2). **Edit Menu**

`Undo`

Undoes the last edit in the current buffer (executes command undo(2)).

`Redo`

Redo the last undo, only available immediately after an undo. This is also done via the undo(2) command.

`Undo All`

Undo all edits in the current buffer until the last save or no more undo history is available. Executes the command undo(2) with a 0 numerical argument.

`Set Mark`

Executes command set−mark(2).

`Cut`

Executes command kill−region(2).

`Copy`

Executes command copy−region(2).

`Paste`

Executes command yank(2).

`Narrow Out`

Executes command narrow−buffer(2) with a numeric argument of 4.

`Narrow To`

Executes command narrow−buffer(2) with a numeric argument of 3.

`Remove Single Narrow`

Executes command narrow−buffer(2) with a numeric argument of `2`.

`Remove All Narrows`

Executes command narrow−buffer(2) with a numeric argument of `1`. **Search Menu**

`Search`

Executes a dialog form of the command isearch−forward(2).

`Replace`

Executes a dialog form of the command query−replace−string(2).

`Hilight Search`

Opens another dialog which can be used to add and remove hilighting of individual lines in the current buffer. Note that setting a line hilight is a temporary change, it will not effect any files etc and will be lost when the buffer is deleted.

`Goto Line`

Executes a dialog form of the command goto−line(2).

`Goto Fence`

Executes command goto−matching−fence(2).

`Set Bookmark`

Executes command set−alpha−mark(2).

`Goto Bookmark`

Executes command goto−alpha−mark(2). **Insert Menu**

`Symbol`

Executes command symbol(3).

`Date & Time`

Opens a dialog with the current date and time in a selection of common formats; selecting one of these will insert the string into the current buffer at the current position. Note that the format text strings depend on the current language (Default and American languages use the order MM−DD−YY

etc whereas the rest use DD−MM−YY). The names used for the day and month names can be defined using the Setup page of Organizer(3).

`File`

Executes command insert−file(2).

`File Name`

Executes command insert−file−name(2).

`Macro...`

Executes command insert−macro(2). **Format Menu**

`Restyle Buffer`

Executes command restyle−buffer(3).

`Restyle Region`

Executes command restyle−region(3).

`Clean Buffer`

Executes command clean(3).

`Change Buffer Char Set`

Executes command charset−change(3).

`IQ Fill Paragraph`

Executes command ifill−paragraph(3).

`Fill Paragraph`

Executes command fill−paragraph(2).

`Fill All Paragraphs`

Executes command fill−paragraph(2) with a very large positive numerical argument. Note that this only effects paragraphs from the current position onwards.

`Paragraph to Line`

Executes command paragraph−to−line(3).

`All Paragraphs to Line`

Executes command paragraph−to−line(3) with a very large positive numerical argument. Note that this only effects paragraphs from the current position onwards.

`Sort Lines`

Executes command sort−lines(2).

`Ignore Case Sort Lines`

Executes command sort−lines−ignore−case(3).

`Capitalize Word`

Executes command capitalize−word(2).

`Lower Case Word`

Executes command lower−case−word(2).

`Lower Case Region`

Executes command lower−case−region(2).

`Upper Case Word`

Executes command upper−case−word(2).

`Upper Case Region`

Executes command upper−case−region(2). **Execute Menu**

`Execute Command`

Executes command execute−named−command(2).

`Execute Buffer`

Executes command execute−buffer(2).

`Execute File`

Executes command execute−file(2).

`Start Kbd Macro`

Executes command start−kbd−macro(2).

`Query Kbd Macro`

Executes command kbd−macro−query(2).

`End Kbd Macro`

Executes command end−kbd−macro(2).

`Execute Kbd Macro`

Executes command execute−kbd−macro(2).

`Name Kbd Macro`

Executes command name−kbd−macro(2).

`Ipipe command`

Executes command ipipe−shell−command(2).

`Shell`

Executes command shell(2). **Tools Menu**

`Current Buffer Tools`

For some file formats MicroEmacs provides a file format specific set of tools, see the file type help page for more specific information.

`Count Words`

Executes command count−words(2).

`Spell Word`

Executes command spell−word(3).

`Spell Buffer`

Executes command spell−buffer(3).

`Word Complete`

Takes the incomplete word to the left of the cursor and attempts to complete the word by using the users current language dictionary. Executes command expand−word(3).

`Compare Windows`

Executes command compare−windows(2).

`Compile`

Executes command compile(3).

`Grep`

Executes command grep(3).

`Graphical Diff`

Executes command gdiff(3).

`Diff`

Executes command diff(3).

`Diff Changes`

Executes command diff−changes(3).

`Organizer`

Executes command organizer(3).

`Mail`

Executes command mail(3).

`View Mail`

Executes command vm(3).

`More...`

Opens a sub−menu with a collection of other useful miscellaneous tools. **Window Menu**

`Split Window V`

Executes command split−window−vertically(2).

`Grow Window V`

Executes command change−window−depth(2) with an argument of 1.

`Shrink Window V`

Executes command change−window−depth(2) with an argument of −1.

`Split Window H`

Executes command split−window−horizontally(2).

`Grow Window H`

Executes command change−window−width(2) with an argument of 1.

`Shrink Window H`

Executes command change−window−width(2) with an argument of −1.

`One Window`

Executes command delete−other−windows(2).

`Delete Window`

Executes command delete−window(2).

`Previous Window`

Executes command previous−window(2).

`Next Window`

Executes command next−window(2).

`Create New Frame`

Create an new external frame, only available on version which support multiple−window frames. Executes command create−frame(2).

`Create New Frame`

Closes the current frame, only available on version which support multiple−window frames. The command will fail if this is the only frame, use File −> Exit to exit MicroEmacs, executes command delete−frame(2).
Help Menu

`Curr Buffer Help`

For some file formats MicroEmacs provides a file format specific help page giving details of key−bindings and tools specific to the current buffers file type.

`General Help`

Executes command osd−help(3).

```
Help on Command
```

Executes command help−command(2).

```
Help on Variable
```

Executes command help−variable(2).

```
Describe Bindings
```

Executes command describe−bindings(2).

```
Describe key
```

Executes command describe−key(2).

```
Describe Variable
```

Executes command describe−variable(2).

```
Describe Word
```

Executes command describe−word(3).

```
List Buffers
```

Executes command list−buffers(2).

```
List Commands
```

Executes command list−commands(2).

```
List Registry
```

Executes command list−registry(2).

```
List Variables
```

Executes command list−variables(2).

```
Command Apropos
```

Executes command command−apropos(2).

```
Buffer Setup
```

Executes command buffer−setup(3).

`User Setup`

Executes command user−setup(3).

`Scheme Editor`

Executes command scheme−editor(3).

`Games`

Opens a sub−menu listing all available games, see Games for more information.

`Product Support`

Opens on−line Contact information.

`About MicroEmacs`

Executes command about(2). **NOTES**

The main menu is defined using osd(2) in macro files me.emf and osd.emf.

General user extensions to the main menu can be added to the user file `myosd.emf` which is executed once when the main menu is first opened. The macro file can add new items to any of the main sub menus and can delete most existing items (some are dynamically added when appropriate, these should not be deleted). See `osd.emf` for examples of how to add items to the menu.

New sub−menus should be added in the company or user setup files as this must be done at start−up. The content on the menu is not required until the main menu is used so populating the new sub−menu can be done in `myosd.emf`.

SEE ALSO

user−setup(3).

Match−It(3)

NAME

Match−It − MicroEmacs '02 version of the Match−It game

SYNOPSIS

Match−It

DESCRIPTION

The object of the game is to score the largest number of points, to do this the player must complete as many sheets as possible. A sheet is completed when all the tiles are removed from the board within the given time limit − ALL sheet are possible. If the player fails to remove all the tiles before the time runs out a life is lost, if all lives have been lost then the game is over.

Tiles are removed from the board by matching two identical tiles which have an 'extraction' path between them. The only exception to this rule is that any open "*flower*" tile (bamboo `[BAMB]`, orchid `[ORCH]`, plum `[PLUM]`, or chrysanthemum `[CHRY]`) matches any other open "*flower*" tile and any open "*season*" tile (spring, summer, autumn, or winter) matches any other open "*season*" tile.

An 'extraction' path is a straight line which uses 2 or less right angles, the following are legal extraction paths, '`*`'s denote the right angles:

```
                A---*        *-----*     A----*
    A----A          A        AXXXXXA     XXXXX|
                                         A----*
```

The following are illegal paths:

```
    *----*          *---*
    AXXXX|          |XXXA
    XXXXA*      A---*XXXX
```

2 points are added to the score whenever a pair is successfully removed, a point is deducted whenever a pair is selected which can not be removed because there is no valid extraction path. There are 2 aids, pressing the right button on a tile when no other tile is selected will hilight all tiles of matching type, this costs 4 points. The other help is activated by a button at the top right of the screen and it removes a random removable pair (or informs the user that there are no removable pairs), there are a limited number of these helps.

At the end of a successful sheet the score is increased be the time left, the number of lives and helps remaining and by the Pedigree and Internal bonuses if they were achieved.

The Pedigree bonus is obtained when only identical tiles are paired, i.e. no differing flowers or seasons were paired, 50 points are awarded when achieved. Its status is indicated by a 'P' to the left of the 'Help' button and the top of the window.

The internal bonus is obtained when the outer 4 margins are not used. If the left or right margins are not used then 10 points are awarded for each, if the top or bottom are not used then 20 points are awarded for each and if none are used then 400 points are awarded! The status on the Internal bonus is indicated by an '`I`' surrounded by '`*`'s, one for each margin. This can be found next to the Pedigree bonus '`P`'.

GAME CONTROLS

To the right of the high score table on the main menu there are a number of control buttons. To select an option, click the left mouse button on it.

NEW

Start a new game.

QUIT

Exit Match−It.

HELP

This help page (keyboard esc h).

During a sheet, to remove a pair of tiles, click the left mouse button on a tile (which will show in the selection color) and then click the left mouse button on the matching tile. At this point, if the tiles can be removed, the extraction path is drawn and both tiles will disappear from the board. If after selecting the first tile, you decide that you don't wish to play that tile, simply reclick the left button on the selected tile, alternatively click the right button to deselect any selected tile.

To the top right of the sheet there are a number of control buttons:−

HELP

Removes a tile pair.

QUIT

Exit the game.

BOSS

Hides Match−It, also acts as a pause. Execute Match−It again to return to the game.

The top left shows the number of remaining lives, the current sheet level, the current score, time remaining for the current sheet and the status of the Internal and Pedigree bonuses.

NOTES

Match−It is a macro defined in `matchit.emf`.

Match−It may only be played with a mouse, there is no keyboard support, with the exception of the re−start keys.

The sheet database file matchit.edf must be accessible for Match−It to work.

SEE ALSO

Games, Mahjongg(3), Metris(3).

MetaFont(9)

SYNOPSIS

MetaFont/MetaPost − Meta Font and Post File.

FILES

hkmeta.emf − MetaFont/MetaPost file hook definition

EXTENSIONS

.mf − MetaFont file
.mp − MetaPost file

DESCRIPTION

The **Meta** file type template provides simple hilighting of **MetaFont** (`.mf`) and **MetaPost** (`.mp`) files, the template provides minimal hilighting. The same hilighting definition is used for both file types.

File recognition is performed using the standard file extensions.

NOTES

JASSPA have no idea as to the state of this file hook definition.

SEE ALSO

Supported File Types

Metris(3)

NAME

Metris − MicroEmacs '02 version of the falling blocks game

SYNOPSIS

Metris

DESCRIPTION

Traditional falling blocks game, make solid horizontal lines out of the falling blocks. The blocks can be rotated and moved left or right by the user as they fall. Once a horizontal line is completely solid it will disappear and everything above it will drop down. A bonus is given if 3 solid rows are made at the same time, i.e. using one block.

Every line you make the game speeds up until it gets too fast!! The game ends when there is no more room to put a block.

The keys used to control Metris are:

left or **j**

Move the block left one character.

right or **l**

Move the block right one character.

down or **k**

Rotate the block counter−clockwise 90 degrees.

space

Drop the current block.

p

Pause the current game.

q

Quit the current game.

C−l

Redraw the display.

return

Start a new game.

esc h

View this help page. **NOTES**

Metris is a macro defined in `metris.emf`.

SEE ALSO

Games, Match−It(3), Patience(3).

m4(9)

SYNOPSIS

me − M4 Macro Processor

FILES

hkm4.emf − M4 macro processor macro file.
m4.etf − M4 macro processor header template file.

EXTENSIONS

.m4

DESCRIPTION

The **M4 macro processor** template performs simple hilighting of **.m4** files. The file type is recognized by the standard extension only.

Hilighting

The hilighting features allows components of the language to be differentiated and rendered in different colors. **NOTES**

The M4 hilighting is minimal, no other features have been implemented.

SEE ALSO

Supported File Types

magic(2m)

NAME

magic − Regular expression search

SYNOPSIS

magic Mode

M − mode line letter.

DESCRIPTION

magic mode enables the regular expression search capability used in the search and the replace commands such as search−forward(2) and query−replace−string(2).

In the magic mode of MicroEmacs '02, certain characters gain special meanings when used in a search pattern. Collectively they are know as regular expressions, and a limited number of them are supported in MicroEmacs '02. They grant greater flexibility when using the search commands (note that they also affect isearch−forward(2) commands).

The symbols that have special meaning in magic mode are `^`, `$`, `.`, `\|`, `*`, `[ ]`, `\( \)`, `\{ \}` and `\`.

The characters `^` and `$` fix the search pattern to the beginning and end of line, respectively. The `^` character must appear at the beginning of the search string, and the `$` must appear at the end, otherwise they loose their meaning and are treated just like any other character. For example, in magic mode, searching for the pattern "`t$`" would put the cursor at the end of any line that ended with the letter '`t`'. Note that this is different than searching for "`t<NL>`", that is, '`t`' followed by a newline character. The character `$` (and `^`, for that matter) matches a position, not a character, so the cursor remains at the end of the line. But a newline is a character that must be matched, just like any other character, which means that the cursor is placed just after it − on the beginning of the next line.

The character '`.`' has a very simple meaning − it matches any single character, except the newline. Thus a search for "`bad.er`" could match "`badger`", "`badder`" (slang), or up to the '`r`' of "`bad error`".

The character `*` is known as closure, and means that zero or more of the preceding character will match. If there is no character preceding, `*` has no special meaning, and since it will not match with a newline, `*` will have no special meaning if preceded by the beginning of line symbol `^` or the literal newline character `<NL>`. The notion of zero or more characters is important. If, for example, your cursor was on the line

```
        This line is missing two vowels.
```

and a search was made for "`a*`", the cursor would not move, because it is guaranteed to match no letter '`a`', which satisfies the search conditions. If you wanted to search for one or more of the letter 'a', you would search for "`aa*`", which would match the letter `a`, then zero or more of them, note that this pattern is better searched using "`a+`".

The character "`+`" is the same as "`*`" except that it searches for one or more occurrences of the preceding character.

The character `[` indicates the beginning of a character class. It is similar to the *any* (`.`) character, but you get to choose which characters you want to match. The character class is ended with the character `]`. So, while a search for "`ba.e`" will match "`bane`", "`bade`", "`bale`", "`bate`", et cetera, you can limit it to matching "`babe`" and "`bake`" by searching for "`ba[bk]e`". Only one of the characters inside the `[` and `]` will match a character. If in fact you want to match any character except those in the character class, you can put a `^` as the first character. It must be the first character of the class, or else it has no special meaning. So, a search for `[^aeiou]` will match any character except a vowel, but a search for `[aeiou^]` will match any vowel or a `^`. If you have a lot of characters in order that you want to put in the character class, you may use a dash (`−`) as a range character. So, `[a−z]` will match any letter (or any lower case letter if exact mode is on), and `[0−9a−f]` will match any digit or any letter '`a`' through '`f`', which happen to be the characters for hexadecimal numbers. If the dash is at the beginning or end of a character class, it is taken to be just a dash.

The `?` character provides a simple zero or one occurrence test of the previous character e.g. "`ca?r`" matches "`cr`" and "`car`", it will not match "`caar`".

Where a previous item has a range of repetitions then the `\{N,M\}` syntax may be used to denote the minimum and maximum iterations of the previous item. Where a set quantity of repetitions is required then the simpler syntax of `\{N\}` may be used. i.e. "`ca\{2\}r`" matches "`caar`", "`ca\{2,3\}r`" matches "`caar`" and "`caaar`".

The escape character `\` is for those times when you want to be in magic mode, but also want to use a regular expression character to be just a character. It turns off the special meaning of the character. So a search for "`it\.`" will search for a line with "`it.`", and not "`it`" followed by any other character. The escape character will also let you put `^`, `−`, or `]` inside a character class with no special side effects.

In search−replace strings the `\( \)` pair may be used to group characters for in the search string for recall in the replacement string. The `\( \)` bracket pair is recalled using `\1−\9` in the replace string, `\1` is the first pair, `\1` the second and so on. Hence to replace `%dgdg%name%dhdh%` with `%dgdg%names%dhdh%` then we could use the following search replace string `\(%[a−z]+%\)\([a−z]*\)\(%[a−z]+%\)` replacing with `\1\2s\3`.

`\0` in the replace string implies the whole string.

A summary of magic mode special characters are defined as follows:−

`^`

Anchor search at beginning of line

$

Anchor search at end of line

.

Match any character except `<NL>`

*****

Match zero or more occurrences of the preceding item.

\|

Match either/or i.e. `car\|bike` matches the work `car` and matches the word `bike`.

+

Match one or more occurrences of the preceding item.

?

Match zero or one occurrences of the preceeding item.

[]

Match a class of characters (`[a-z]` would be all alphabetics)

\

Take next literally

\\{*N***,***M***\\}**

Match a minimum of *N* occurrences and maximum of *M* occurrences of the preceeding item.

\\{*N***\\}**

Match a *N* occurrences of the preceeding item.

\\(...\\)

> Delimit pattern to replicate in replace string. Max of 9 allowed. Called in replace string with `\1`,..,`\9`. 1 being 1st etc. `\0` or `\&` in the replace string is the whole string. i.e.
>
> Search: **\\(ab\\)\\(dc\\)**
> Replace: **\1\2 \1\2**
> on "**abdc**" => "**abdc abdc**"

SEE ALSO

buffer−mode(2), global−mode(2), query−replace−string(2), search−forward(2).
Regular Expressions

vm(3)

NAME

vm − Email viewer
mail−check − Check for new email
stop−mail−check − Disable the check for new email
mail − Compose and send an email

SYNOPSIS

vm
mail−check
stop−mail−check
mail

DESCRIPTION

vm is a simple email manager, it is configured to send and receive emails using the user−setup(3) Mail dialog.

mail−check tests the size of this incoming mail box, a non−zero length indicates that new mail has arrived and **mail−check** informs the user by inserting a '`M`' in the mode−line (2nd character for the left) and ringing the system bell. **mail−check** uses create−callback(2) to check for new mail every 10 minutes, this can be disabled by executing **stop−mail−check**.

When **vm** is executed it checks for new mail, if found it first copies the new mail to a file called "`new_mail`" in the users mail directory. The incoming box is then emptied by truncating the file to zero length. The users main mail box is then loaded and the new mail (if any) is appended. The mail box is then processed after which 2 windows are created the bottom window listing all messages in the box and the top displaying the current message.

vm is capable of:

- Scrolling through the mail box displaying each message (`up`, `p`, `down`, `n`, `return`, `space`).
- Check and get new mail messages (`g`).
- Extract and cut embedded data files (`x`, `C`, `c`).
- Reply to and forward mail messages (`R`, `r`, `z`).
- Delete mail messages (`d`, `u`).
- Archive messages to other mail boxes (`A`, `a`).
- Save changes to the current mail box (`S`, `s`).
- Delete the current mail box (`D`).
- Visit another mail box (`v`).
- Send a mail message (`m`).
- Hide vm windows (`delete`).

Use the vm help page (bound to "`esc h`") for further information.

vm supports two types of embedded data, uuencode and mime encoding and uses ipipe−shell−command(2) to extract the data, the commanding to use must be supplied by the user using the setup dialog, which can contain the following special tokens:

`%i`

Temporary file name, if used, the embedded data is written to the this file first.

`%o`

User supplied output file name, if `%i` is not used, the embedded data is written to this file first.

`%b`

The output base name, i.e. `%o` without the path.

If no command line is supplied then the embedded data is written to the user supplied file name as a text file in the form found in the mail message.

mail can be used to compose and send an email, it can insert embedded data in a similar way to **vm**'s data extraction, the following special tokens can be used:

`%i`

The user supplied data file to be embedded.

`%b`

The input base name, i.e. `%i` without the path.

`%o`

Temporary file name used to output the processed data file, this file is inserted into the mail message using insert−file(2).

mail also uses **ipipe−shell−command** to send the mail message, the following special tokens can be used:

`%f`

The from user name.

`%s`

The email subject.

`%t`

A comma separated list of 'To' recipients.

%c

A comma separated list of 'Cc' recipients.

%o

A file name of the mail message.

Any field not used in the command−line is left at the head of the mail message.

EXAMPLE − UNIX

The following command−line can be used on most UNIX systems to extract uuencoded data:

```
rm −f %o ; uudecode %i ; rm −f %i
```

The following command−line can be used on most UNIX systems to extract mime encoded data:

```
rm −f /tmp/%b ; metamail −B −d −q −w −x −z %i ; mv −f /tmp/%b %o
```

The following command−line can be used on most UNIX systems to uuencode a data file ready for it to be embedded, the original file is not changed:

```
uuencode %b < %i > %o
```

The following command−line can be used on most UNIX systems to send an email:

```
/usr/lib/sendmail −oi −oem −odi −t < %o
```

EXAMPLE − WIN32

Typically the **cygnus(1)** utilities can be used for data insertion and extraction. These have the advantage of being very similar to the unix ones so only minor changes are required, i.e. try the following for data insertion and mime & uuencode extraction respectively:

```
del %o ^ uudecode %i ^ del %i
del c:\tmp\%b ^ metamail −B −d −q −w −x −z %i ^ move c:\tmp\%b %o
uuencode %b < %i > %o
```

This assumes that the shell you are using supports the '^' multiple commands on a single line feature, this is supported by **4dos(1)** and **4nt(1)**. If your shell does not support this feature a simple batch file command could be used instead.

postie(1) is a freely available pop3/smpt e−mail support program, available on the net, which can be used to provide a fully working **vm** on windows systems. As it is typically used in a dial−up connect environment, the **user−setup** 'Queue Outgoing Mail' option will be enabled while the 'Check Mail' and 'VM Gets Mail' will be disabled. This ensures that a connection is only made when the **vm** 'g' command is used which sets all queued outgoing mail and gets any incoming mail.

The following command−line can be used to get mail from your pop server using postie:

```
postie −host:pop−mail−addr −user:user−addr −pass:password −file:inbox
        "−sep:From root Mon Jan 11 20:02:02 1999" −raw −rm
```

Where the `inbox` is the 'Incoming Mail Box' file specified in user−setup. The `−sep` option is used to partition each mail message from the previous message, this string is used as it is in a unix standard form so the resulting mail box could be understood by unix mail systems such as netscape etc.

NOTE: The **−rm** option is used to remove the incoming mail messages from the server. It is strongly recommended that the system is thoroughly tested without this option first.

The following command−line can be used to send mail to your smtp server using postie:

```
postie −host:smtp−mail−addr "−from:user@mail−addr" −use_mime:0
        "−to:%t" "−s:%s" "−cc:%c" "−file:%o"
```

blat(1) is another freely available windows program which can be used to send mail with the following command−line:

```
blat %o −f %f −s \"%s\" −t \"%t\" −c \"%c\"
```

NOTES

vm is a macro defined in `vm.emf`, **mail−check**, **stop−mail−check** and **mail** are macros defined in `mail.emf`.

vm has only been tested in a couple of environments, the author will not except any responsibility for any loss of data, i.e. use at your own peril. You have been warned! Back−up all data files and test **vm** THOROUGHLY before using it.

SEE ALSO

user−setup(3), ipipe−shell−command(2), create−callback(2), **sendmail(1)**.

makefile(9)

SYNOPSIS

makefile − Make file

FILES

hkmake.emf − Make file hook definition
make.etf − Template file

EXTENSIONS

Makefile, **makefile**, **.mak** − Makefiles.

MAGIC STRINGS

−!− makefile −!−

Recognized by MicroEmacs only, defines the file to be a makefile. **DESCRIPTION**

The **make** file type template handles the hilighting of the makefile files.

General Editing

On creating a new file, a new header is automatically included into the file. time(2m) is by default enabled, allowing the modification time−stamp to be maintained in the header.

By default, TAB's are enabled as this is the syntactical feature of the file.

Hilighting

The hilighting emphasizes the keywords and comments within the makefile. No special support for Microsoft **nmake(1)** is provided because of the number of oddities in their implementation of make. **BUGS**

No attempt is made to hilight any embedded shell commands.

SEE ALSO

imakefile(9), time(2m).

Supported File Types

man(3)

NAME

man − UNIX manual page viewer. man−clean − Clean UNIX manual page.

SYNOPSIS

man
man−clean

DESCRIPTION

man provides a mechanism to display a UNIX manual page within the MicroEmacs window. On invoking **man** the user is prompted for the name of the manual page to display:−

```
Man on ?
```

The name of the manual page (and any options) are entered on the command line. The macro invokes the UNIX utility **man(1)** to generate the page and displays the results in a window.

Another manual page can be selected by either moving the cursor to the link and pressing return or double clicking on it with the left mouse button. MicroEmacs will then attempt to load and display the selected manual page.

man−clean removes any man−page formatting codes from the current buffer reducing a manual page to plain text. The formatting codes are used to create the bold and underline fonts. This allows the page to be treated as a normal buffer, i.e. string searches and other similar command will work as expected.

NOTES

man and **man−clean** are macros defined in `hkman.emf`.

man is only made available within UNIX environments, the UNIX start up file `unixterm.emf` links in the macro. If the **man** utility is required on other platforms then the following definition is required in a start−up file.

```
define−macro−file hkman man
```

SEE ALSO

man(9), user−setup(3), spell−buffer(3).

man(9)

SYNOPSIS

man − UNIX Manual page

FILES

hkman.emf − UNIX manual page hook definition

EXTENSIONS

.man − UNIX manual page file

DESCRIPTION

The **man** provides the hilighting of UNIX manual pages, generally acquired through the man(3) command, via a pipe. *man* references within the displayed manual page may be accessed using the mouse in a hypertext fashion.

Hilighting

The hilighting commands recognize the manual page *bold* and *underline* character sequences and transpose these into the appropriate character hilighting. The hilighting sequences are generally unpleasant because they also remove the characters for display.

> The multitude of different platforms causes problems as different vendors produce different character sequences for *bold*/*italic* text, hence on some platforms it may be necessary to add additional hilighting rules to cater for any local variations.

Short Cuts

> Selecting a link node within the manual page using the mouse (i.e. a reference to another manual page) then MicroEmacs '02 attempts to find the manual page in the text and invokes man(3) to render the page. This provides a crude hyper text mechanism simply using the manual page information itself.

The man−clean(3) command can be used to remove all of the hilighting characters from the current manual page. This is the typical method of reducing a manual page to plain text. **SEE ALSO**

man(3), man−clean(3),

Supported File Types

mark−registry(2)

NAME

mark−registry − Modify the operating mode of a registry node

SYNOPSIS

n **mark−registry** "*root*" "*mode*"

DESCRIPTION

mark−registry modifies the *mode* of a registry node *root*. If an argument *n* is supplied then the *n*th register node down from **root** (as viewed from list−registry(2) output) is modified instead. The *mode* is string specifying the modes, each mode is represented by a character. Lower case characters add a mode, upper case characters delete a mode. The modes are defined as:−

? − Query Name

Returns the full name, including path, of the given registry node in the variable $result(5). This does not alter the registry.

! − Hide Value

Hides the value of the given registry node, i.e. its value will not be displayed in the output of list−registry(2). Once set, this mode cannot be removed.

a − Autosave

Automatically saves the registry when it is deleted or unloaded from the registry. The user is not prompted for a save.

b − Backup

Automatically performs a backup of the registry file whenever a save operation is performed.

c − Create

If the registry file cannot be loaded then the *root* node is created and the invocation succeeds. If this mode is omitted then the call fails if the *file* cannot be found.

d − Discard

Marks the registry as discardable. This is typically used for registries that are not saved.

f − File

The registry node is marked as a file root, the value must be set to the registry file name.

g − Get Modes

Returns the list of modes currently set on the given registry node in the variable $result(5). This does not alter the registry.

h − Hidden

The registry node is marked as *Hidden*, i.e. its children will not be shown in list−registry(2) output.

u − Updated

Marks the registry as modified. The modified bit is removed when the registry file is saved. If the modified bit is applied to a registry node the user will be prompted to save the registry when it is deleted (or it will be automatically saved when the *Autosave* mode is used).

Multiple modes may be applied.

EXAMPLE

A history registry can be hidden with the following invocation:−

```
mark-registry "/history" "h"
```

It could then be made visible again using:−

```
mark-registry "/history" "H"
```

BUGS

At exit only registry nodes attached to the root are saved.

DIAGNOSTICS

mark−registry fails if *root* does not exist.

SEE ALSO

get−registry(2), list−registry(2), read−registry(2), set−registry(2), erf(8).

me(1)

NAME

me − MicroEmacs '02 text editor

SYNOPSIS

me [*options*] [*files ...*]

me [*@startupFile*] [**−b**] [**−c**] [**−d**] [**−h**] [**−i**] [**−l***lineNo*] [**−m***command*] [**−n**] [**−0***file*] [**−p**] [**−r**] [**−s***string*] [**−u***username*] [**−v***variable=string*] [**−x**] *files...*

DESCRIPTION

MicroEmacs '02 is a cut down version of the EMACS text editor, based on Danial Lawrences MicroEmacs. **MicroEmacs '02** 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 '02 can be extensively customized for the needs of the individual user.

MicroEmacs '02 allows multiple files to be edited at the same time. The screen may 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 '02** 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 '02** behaves. Also, any and all the commands can be used by any key stroke 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.

The command line options to **MicroEmacs '02** are defined as follows:−

@startFile

Initialize MicroEmacs '02 using *startFile*[**.emf**]. The default when omitted is **me.emf**. See start−up(3) and Command Line Filters for more information.

−b

Load next file as a binary file (binary editor mode, uses binary(2m) buffer mode).

−c

Continuation mode. Load **MicroEmacs '02** last edit session, restoring the buffers to their previous loaded state and position. Note that history mode must be enabled. The **−c** option is generally used with windowing interfaces (X−Windows/Microsoft Windows) as the shortcut icon invocation.

−d

Enable debug mode (for macro files).

−h

Show the help page (does not start the editor).

−i

MS−DOS versions of **MicroEmacs '02** only. Insert the contents of the current screen into the ***scratch*** buffer

−k[*key*]

Load next file as an encrypted file (uses crypt(2m) buffer mode). The optional adjoining argument can be used to specify the decrypting key, if this argument is not specify the user will be prompted for it on start−up.

−l*lineNo*

Go to line *lineNo* in the next given file. Typically used with utilities such a **more(1)** where an external editor may be invoked from other viewer.

−m*command*

Sends a client−server command to an existing MicroEmacs session. The command takes the form "**C:**<*client*>**:**<*command*>" i.e. to write "`Hello World`" on the message line then a client may issue the command:−

```
; launch server
me &
; send message
me −m "C:ME:ml−write \"Hello world\"
```

Note that the <*command*> is a MicroEmacs macro command, the escape sequences must be adhered to. The *client−server* interface is typically used to load a file, this may be performed as follows:−

```
me −m "C:myutility:find−file \"/path/foo.bar\""
```

The absolute path is specified in this type of transaction as the current working directory of the active MicroEmacs session is unknown. The **−m** option de−iconize's the existing editor session and bring it to the foreground.

−n

UNIX X−Windows environments only and MicroSoft Windows NT console versions. Execute **MicroEmacs '02** using termcap rather than X−Windows for UNIX; typically used within an **xterm** shell to fire up **MicroEmacs '02** for a quick edit. For Microsoft Windows, a console window is started as opposed to a GUI window.

−o<*file*>

Use already running version of MicroEmacs '02 to load the <*file*>, if it exists, otherwise start a new editor session. This uses the *client−server* interface to push the new file into the existing editor session. Refer to the Client−Server Interface for details.

−p

Pipe *stdin* into buffer ***stdin***, when saved output to *stdout*, following is a simple example which changes '`a`'s to '`b`'s:

```
define−macro start−up
    find−buffer "*stdin*"
    beginning−of−buffer
    replace−string "a" "b"
    save−buffer
    quick−exit
!emacro
```

This can be used in the following manner:

```
me "@testpipe.emf" < foo.a > foo.b
```

−r

Read−only, all buffers will be in view mode

−s*string*

Search for string "*string*" in the current buffer. e.g. `me −sfoo bar` starts **MicroEmacs '02**, loads file `bar` and initiates a search for *foo*. The cursor is left at the end of the string if located, otherwise at the top of the buffer.

−u*username*

Set the current user name to *username* before MicroEmacs is initialized. This is done by setting the environment variable MENAME(5) to the given value.

−v*variable=string*

Assign the MicroEmacs '02 *variable* with *string*. The assignment is performed before the buffers are loaded. Typically used to change the start−up characteristics of the startup file(s).

−x

UNIX environments. Disable the capture of signals. **MicroEmacs '02** by default captures an handles all illicit signal interrupts. The option is enabled when debugging the source code allowing exception conditions to be trapped within the debugger.

−y

Load next file as a reduced binary file (uses rbin(2m) buffer mode). **ENVIRONMENT**

The following environment variables are used by **MicroEmacs '02**.

DISPLAY

UNIX environments running X−Windows only. The identity of the X−Windows server. Typically set to **unix:0.0**, refer to the X−Windows documentation for details of this environment variable.

MENAME and **LOGNAME**

The identity of the user, **$MENAME** takes precedence over **$LOGNAME**. **$LOGNAME** variable is generally defined within UNIX as part of the login script. The variables are used to determine which start−up configuration to use in the initialization of **MicroEmacs '02** (*$MENAME*.erf).

Non−UNIX platforms usually need to explicitly set the **$MENAME** environment variable to identify the aforementioned files. for MS−DOS and Microsoft Windows this is typically performed in the `AUTOEXEC.BAT` file.

PATH

The **$PATH** environment variable is used on most operating systems as a search path for executable files. This **$PATH** environment variable must be defined with **MicroEmacs '02** on the search path. Under UNIX this is set in the `.login`, `.cshrc` or `.profile` file i.e.

```
export PATH $PATH:/usr/name/me
```

Within MS−DOS or Microsoft Windows environments it is defined in the `AUTOEXEC.BAT` file. e.g.

```
set PATH=%PATH%;c:\me
```

MicroEmacs '02 utilizes information in the **$PATH** environment variable to locate the start−up files, dictionaries etc.

TERM

The terminal identification sting. In UNIX environments the environment variable **$TERM** is set to "`vt...`", in this case it is assumed that the machine is a server, and the host cannot support X (see command line option **−n**).

In MS−DOS the environment variable is usually set to define the graphics adapter mode. **%TERM** is assigned a string, understood by the `me.emf` start−up file, to set the graphics mode. Predefined strings include:−

E80x50

Initiates an 80 column by 50 line screen.

E80x25

Initiates an 80 column by 25 line screen.

userDefined

A user defined string to set an explicit graphics card mode. The operation is dependent upon the support offered by the graphics adapter.

MEPATH

MicroEmacs '02 uses the environment variable `$MEPATH` as the directory(s) used to search for the macro files (see emf(8)). Within the UNIX `$MEPATH` is a semi−colon separated list of directories which are used to search for the MicroEmacs '02 macro files. The path is searched from left to right. The environment variable is typically defined in the in the `.login`, `.cshrc` or `.profile` file i.e.

```
export MEPATH /usr/name/me/macros:/usr/local/microemacs
```

The default when omitted is `/usr/local/microemacs`.

Within MS−DOS or Microsoft Windows environments it is defined in the `AUTOEXEC.BAT` file. e.g.

```
set MEPATH=c:\me\username;\me\macros
```

There is no default location in these environments. For Microsoft Windows environments refer to me32.ini(8) for a method of setting up the `$MEPATH` from the windows configuration file.

INFOPATH

MicroEmacs '02 uses the environment variable `$INFOPATH` as the directory(s) used to search for GNU **Info** files. Within the UNIX `$INFOPATH` is a semi−colon separated list of directories which are used to search for the MicroEmacs '02 macro files. The path is searched from left to right. The environment variable is typically defined in the in the `.login`, `.cshrc` or `.profile` file i.e.

export INFOPATH /usr/local/info:$HOME/info

The default when omitted is `/usr/local/info`.

Within MS−DOS or Microsoft Windows environments it is defined in the `AUTOEXEC.BAT` file. e.g.

set MEPATH=c:\usr\local\info

There is no default location in these environments. For Microsoft Windows environments refer to me32.ini(8) for a method of setting up the `$INFOPATH` from the windows configuration file.

FILES

All of the macro files and dictionaries are located in the **MicroEmacs** home directory. The standard file extensions that are utilized are:−

.eaf

MicroEmacs '02 abbreviation file, defines completion definitions for buffer dependent text expansion.

.edf

A **MicroEmacs '02** spelling dictionary. *<language>***.edf** provide language specific dictionaries; *$LOGNAME***.edf** is personal spelling dictionary.

.ehf

MicroEmacs '02 help file information. On−line help information for emacs, the main file is `me.ehf`.

.emf

A **MicroEmacs '02** macro file. The following classes of macro file exist:

me.emf

The default startup file.

<platform>.emf

A platform specify startup file, these include UNIX generic (`unixterm.emf`), UNIX specific (`irix.emf`, `hpux.emf`, `unixwr1.emf`, `linux.emf`, `sunos.emf` etc), Microsoft Windows (`win32.emf`), MS−DOS (`dos.emf`).

hk*xxxxxx***.emf**

Buffer context specific hook files to initialize a buffer with macros and highlighting appropriate to the contents of the file type. e.g. 'C' language editing (`hkc.emf`), N/Troff typesetting (`hknroff.emf`), UNIX Manual page display (`hkman.emf`), Makefiles (`hkmake.emf`), etc.

.erf

Registry files, used to retain personal information, users history in the file etc.

.etf

Template files used to seed new files. Typically contains standard header information, copyright notices etc. that are placed at the head of files. The 'C' programming language is called `c.etf` **MICROSOFT WINDOWS**

Microsoft Windows environments should refer to me32.ini(8) for a method of setting up the environment variables without editing the `AUTOEXEC.BAT` configuration file.

SEE ALSO

emf(8), erf(8), **emacs(1)** [GNU], **more(1)**, **vi(1)**.
Client−Server Interface.
Command Line Filters.

me32.ini(8)

NAME

me32.ini − Microsoft Window's Initialization (ini) File

SYNOPSIS

[Location]
exe=<*executablePathname*>

[Defaults]
mepath=<*directoryPath*>
userpath=<*directoryPath*>
fontfile=<*fontFileName*>

[<*userName*>**]**
<*environmentVariable*>**=**<*value*>
<*environmentVariable*>**=**<*value*>

[<*userName*>**]**
<*environmentVariable*>**=**<*value*>
<*environmentVariable*>**=**<*value*>

; *Comments commence with a semi−colon*

PLATFORM

Microsoft Windows environments only

DESCRIPTION

me32.ini is the Microsoft Windows configuration file, located in the windows directory (typically **C:\Windows**), the **me32.ini** file is primarily used to counteract the deficiencies of Windows shell environment (as compared with UNIX) with respect to the initialization of environment variables.

The configuration file may be considered to be split into two sections, a **Defaults** section, which defines system settings and a **User** section which allows environment variables to be defined.

User Section

The **User Section** is executed prior to the **Defaults Section**. The **User Section** uses the user name which is defined as follows:−

- ♦ The environment variable [$MENAME(5)](#).
- ♦ The *login name* under Windows '95 or NT if **$MENAME** is not set. If the *login name* is defined then environment variable **$MENAME** is set to this value.
- ♦ The environment variable [$LOGNAME(5)](#) if the *login name* cannot be located.
- ♦ **guest** if none of the above are defined.

A section **[***userName***]** is looked up, and if located each of the entries *<environmentVariable>* = *<value>* is extracted and pushed into the execution environment. The **<environmentVariable>** is automatically promoted to upper case if specified as a lower case entry. The environment variables may be subsequently used within the **.emf** macro files to configure MicroEmacs '02 at start up.

Any value may be inserted into the environment including the `$MENAME` environment variable which is used in the next section.

Defaults Section

The defaults section, labeled **[Defaults]** includes the following keys:−

mepath

The directory (or folder) location of the MicroEmacs '02 default configuration files.

userpath

The directory (or folder) location of the user(s) directories.

> Given that the **userpath** is specified as `c:\me98.5` and the user is called `foo`, then the directory location `c:\me98.5\foo` is considered to be the user path.
>
> If the **userpath** is omitted then the **mepath** entry is used as the user path.

The **userpath** and **mepath** entries are concatenated together to form the environment variable [$MEPATH(5)](#), e.g. *userpath******logname***;***mepath*. If the entries are omitted the then environment variable `$MEPATH` is used as defined. The **mepath** and **userpath** are configured initialy by the **InstallShield** installation process.

fontfile

The name of the font file used to render the text to the screen. The default font file is **dosapp.fon**, this is a fixed mono font as used in the MS−DOS windows. **Location Section**

The location section, labeled **[Location]**, identifies the location of MicroEmacs '02, and is typically used by other components to find and launch MicroEmacs. The section includes the following keys:−

exe

The absolute pathname to the MicroEmacs '02 executable image. **EXAMPLE**

The following is an example of the **me32.ini** file:−

```
; External locater for the executable
[Location]
exe=c:\Program Files\JASSPA\MicroEmacs\me32.exe
;
[Defaults]
; mepath
; The location of the MicroEmacs common files.
;
mepath=d:/me98.4/common
;
; userpath − The location of the users MicroEmacs directory.
; The $MENAME is appended as a directory to userpath
;
userpath=d:/me98.4/common
;
; fontfile − The name of the font file used as default.
fontfile=dosapp.fon
;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;
; Environment settings for a user.
; All settings are pushed into the environment.
;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;
[guest]
term=8x12
;
[jon]
MENAME=jon
FOO=bar
;
[jnaught]
MENAME=jon
FOO=bar
;
[bill]
....
```

Note that multiple users share the same **me32.ini** file, each user may include their own configuration settings which may be interrogated in the configuration files (e.g. `$FOO` is assigned the value `bar`, which may be extracted from the environment context).

SEE ALSO

$MENAME(5), $MEPATH(5), user−setup(3), emf(8).

memsdev(1)

NAME

memsdev − Microsoft Developer Studio Add−in for MicroEmacs '02

SYNOPSIS

memsdev.dll

DESCRIPTION

meMsdev is a Microsoft Visual Studio Add−In that allows MicroEmacs '02 to be integrated as the default text editor. It will be used instead of the Visual Studio built in editor when you double click on a file or press F4.

INSTALLATION

1) Copy mesdev.dll into the MicroEmacs directory i.e.

```
c:/Program Files/JASSPA/MicroEmacs
```

2) Edit the me32.ini(8) file in your Windows directory and identify the location of the MicroEmacs executable. The executable name is used to spawn MicroEmacs if it is not already running. The entry takes the form:−

```
; Identify the location of the MicroEmacs executable so that the
; Developer Studio "Add−In" can locate the executable
[Location]
exe=c:\Program Files\JASSPA\MicroEmacs\me32.exe
```

Change the *exe* entry to match the location and name of your executable.

3) For MS−DEV V5.0 only; from a DOS box, register the DLL using **regsvr32.exe(1)** i.e.

```
> cd c:/Program Files/JASSPA/MicroEmacs
> regsvr32 memsdev.dll
```

For MS−DEV V6.0 it is not necessary to perform this registration step.

4) Start Visual Studio and goto:−

```
Tools
    Customize...
        Add−Ins and Macro Files
```

5) Click on *Browse* and point Visual Studio to your **memsdev.dll** file.

6) Click the check box to indicate that you want to use the Add−In, and close the Customize dialog box.

7) You should notice the MicroEmacs tool bar showing the MicroEmacs Icon. This invokes a dialog that allows you to attach and detach MicroEmacs as the default editor.

USING meMsdev

Clicking on the MicroEmacs Tool bar shows the meMsdev configuration dialog. Check the boxes when MicroEmacs edit session is required as default; uncheck the boxes if you wish to revert to the built−in dialog.

Use Visual Studio as normal, and MicroEmacs should almost always bring MicroEmacs to the foreground to edit the document. If a MicroEmacs is already running then "meMsdev" will attach to an existing session and will load the file. If MicroEmacs is not detected then a new version is spawned off and then an attachment is made.

RUNNING A DEBUG SESSION

meMsdev does not currently provide any debugging capability (but we are working on it !!). To start debugging it is suggested that the Editor is decoupled *(MicroEmacs Dialogue −> Uncheck Boxes)* and work within the Developer studio.

When you have finished debugging and wish to move back to an edit session then re−enable MicroEmacs *(MicroEmacs Dialogue −> Check Boxes)* AND close any windows that are open within the MS−Dev environment (*Window−>Windows...−>Select All* and *Close All Windows*). Then commence editing again.

While MicroEmacs is attached, selecting any of the find file lines, compilation error lines etc within the response panes will take MicroEmacs to the specified line.

BUGS

meMsdev works by hooks exposed by Visual Studio. Most of the functionality works from the OpenDocument (look it up in VS 5) hook. So...If a document is ALREADY open in Visual Studio, and you double click the file in the File Browser...Emacs will NOT come to the foreground. Since the document was open in the Visual Studio editor, the OpenDocument event never occurred.

ACKNOWLEDGEMENTS

meMsdev is based on the initial work from **VisEmacs** performed by **Christopher Payne** *<payneca@sagian.com>* for GNU Emacs. This software comes under the GNU General Public License, as such, meMsdev is covered by the same licensing.

Many thanks to Christopher for putting together this technology, this manual page is derived from the documentation supplied with *VisEmacs*.

LICENSING

SEE ALSO

Microsoft Developer Studio Add−In Documentation

ml−bind−key(2)

NAME

ml−bind−key − Create key binding for message line
ml−unbind−key − Remove key binding from message line

SYNOPSIS

n **ml−bind−key** "*command*" "*key*"
n **ml−unbind−key** "*key*"

DESCRIPTION

ml−bind−key creates a key binding local to the message line input buffer. There are several commands that can be used in message line input, each command is associated with a main buffer editing command and inherits all that commands global bindings, i.e. moving forward 1 character is associated with the command forward−char(2) and thus inherits the binding `C−f` (as well as any other like the right cursor key). The following is a list of available commands, what they do and their associated commands

Cursor Movement

- move backwards 1 character, command: backward−char (**C−b**)
- move forwards 1 character, command: forward−char (**C−f**)
- move backwards 1 word, command: backward−word
- move forwards 1 word, command: forward−word
- move to the beginning of buffer, command: beginning−of−line (**C−a**)
- move to the end of buffer, command: end−of−line (**C−e**)

Input

- Quote a character, command: quote−char (**C−q**)
- Yank kill buffer into message line, command: yank (**C−y**)
- insert current buffers current line into the buffer, command: insert−newline (**C−o**)
- insert current buffers file name into the buffer, command: insert−file−name (**C−x C−y**).
- insert current buffers buffer name into the buffer, command: reyank (**esc y**)

Deletion

- delete backward 1 character, command: backward−delete−char (**C−h**)
- delete forward 1 character, command: forward−delete−char

- kill text from current position to end of line, command: kill−line (**C−k**).
- erase whole line, command kill−region (**C−w**). Note that in incremental searches this is used to add the current word to the search string.

History

MicroEmacs '02 stores the last 20 entries of each kind (command, buffer, file, search and general which is also saved in the history file so the state of the history is retained when next loaded. The following commands can be used to manipulate the history.

- next history list entry (loop through history), command: forward−paragraph (**esc n**)
- previous history list entry, command: forward−paragraph (**esc p**)

Completion

When entering a command, file, buffer or a mode name MicroEmacs '02 creates a list of possible completions the following operations can be performed on this list.

- expand. This completes the given input until the first ambiguous character, command: a space (' ') or tab (**C−i**).
- expand to the previous completion (loops through the completion list, command: backward−line (**C−p**)
- expand to the next completion (loops through the completion list, command: forward−line (**C−n**)
- create a listing of all completions, command: a double expansion, i.e. 2 spaces or tabs. The first expands and the second creates the list.
- page up the completion list buffer, scroll−up (**C−z**)
- page down the completion list buffer, scroll−down (**C−v**)

Miscellaneous

- abort input, returning failure to the input, abort−command (**C−g**)
- re−fresh the message line, command: recenter (**C−l**)
- finish input, command newline (**C−m**, return)
- transpose previous character with current character, command: transpose−chars (**C−t**)
- capitalize the next word, command: capitalize−word (**esc c**)
- Turn the whole of the next word to lower case letters, command: upper−case−word (**esc u**)
- Turn the whole of the next word to upper case letters, command: lower−case−word (**esc l**)

ml−unbind−key unbinds a user created message line key binding, this command effects only the message line key bindings. If a −ve argument is given to **ml−unbind−key** then all message line bindings are removed.

EXAMPLE

If expansion was required on the **esc esc** key binding then use the following:−

```
ml−bind−key tab esc esc
```

NOTES

The prefix commands cannot be rebound with this command.

Command key response time will linearly increase with each local binding.

SEE ALSO

global−bind−key(2), buffer−bind−key(2), describe−bindings(2), osd−bind−key(2), global−unbind−key(2).

ml−clear(2)

NAME

ml−clear − Clear the message line

SYNOPSIS

ml−clear

DESCRIPTION

ml−clear clears the message line during script execution. This is useful so as not to leave a confusing message from the last command(s) in a script.

Callback macros which may interrupt the user at any point in time are handled by **ml−clear**. The callback macro for instance may interrupt the user while entering a new file name, and any ml−write(2) erases the message−line which may currently be in use. MicroEmacs '02 stores the line and when ml−clear(2) is invoked, instead of clearing the message line the current input line is restored.

SEE ALSO

create−callback(2), ml−write(2).

ml−write(2)

NAME

ml−write − Write message on message line

SYNOPSIS

n **ml−write** "*message*"

DESCRIPTION

ml−write writes the given *message* to the message line. If a positive argument *n* is given then there will be an *n* millisecond uninterruptible delay, giving the user time to see the message.

A call to **ml−write** from a callback macro can erase a message line which is currently being used (to enter a buffer name say). A call to ml−clear(2) restores the previous message−line.

EXAMPLE

The following call displays a message on the message−line with a fixed 2 second pause:

```
2000 ml-write "Hello World!"
```

SEE ALSO

ml−clear(2), command−wait(2), create−callback(2).

nact(2m)

NAME

nact − Buffer not active

SYNOPSIS

nact Mode

n − mode line letter.

DESCRIPTION

This mode can not be set and is used to indicate that the buffer has not been activated, i.e. the buffer has not been displayed in a window. If the buffer is linked to a file but has not been displayed, so is not active, the file will not have been loaded into the buffer.

The list−buffers(2) command output denotes active buffers with a '@' character in the left hand column, inactive buffers have a ' '.

This mode can not be tested using the more usual &bmode(4) macro command as it only operates on the current buffer as which point the mode cannot be set. Instead the &nbmode(4) macro command must be used.

SEE ALSO

list−buffers(2), &nbmode(4), &bmode(4).

name−kbd−macro(2)

NAME

name−kbd−macro − Assign a name to the last keyboard macro

SYNOPSIS

name−kbd−macro "*command*"

DESCRIPTION

name−kbd−macro labels the last defined keyboard macro with the given *command* name. The command name must be either unique or the name of an existing macro. A keyboard macro is deleted when another keyboard macro is defined, but when named, it is preserved. A named keyboard macro can also be bound to its own command key sequence, and may be inserted into a buffer enabling it to be saved and thus re−loaded and re−used at a later date.

SEE ALSO

execute−file(2), execute−kbd−macro(2), global−bind−key(2), insert−macro(2), start−kbd−macro(2).

narrow(2m)

NAME

narrow − Buffer contains a narrow

SYNOPSIS

narrow Mode

N − mode line letter.

DESCRIPTION

This mode can not be set and is used to indicate whether the buffer contains a narrow, created by the narrow−buffer(2) command.

SEE ALSO

narrow−buffer(2).

narrow−buffer(2)

NAME

narrow−buffer − Hide buffer lines

SYNOPSIS

n **narrow−buffer**

DESCRIPTION

The effect of **narrow−buffer** depends on the argument given, defined as follows:−

1

Removes all narrows in the current buffer (Default).

2

Removes the current line's narrow.

3

Narrow to region. Hides all but the lines of test in the current buffer from the mark position to the current cursor position, effectively 'narrowing' the buffer to the remaining text.

4

Narrow out region. Hides the lines of test in the current buffer from the mark position to the current cursor position, opposite to argument **3**.

When a narrow is created the buffer mode narrow(2m) is automatically set, when the last is removed this mode is deleted.

For example, if the buffer contains the following text:

```
1 Richmond
2 Lafayette
3 Bloomington
4 Indianapolis
5 Gary
6
```

If the mark is on line 2 and the current point is on line 4, executing:−

```
4 narrow−buffer
```

Creates one narrow, narrowing out line 2 and 3. Line 4 becomes the narrow anchor line, when the narrow is removed lines 2 and 3 will be inserted before line 4. The buffer will become:−

```
1 Richmond
4 Indianapolis
5 Gary
```

If instead the following was executed:−

```
3 narrow−buffer
```

Two narrows are created, the first narrowing out line 4 and 5 (line 6, the last line, being the anchor line) the second narrowing out line 1 (line 2 being the anchor line). The buffer will become:−

```
2 Lafayette
3 Bloomington
6
```

Executing **narrow−buffer** with an argument of **2** will only work on the anchor lines, namely 4 in the first example and 2 and 6 in the second.

NOTES

Alpha mark set by set−alpha−mark(2) in text which is subsequently narrowed out will automatically remove the narrow if the user returns to the mark using goto−alpha−mark(2).

get−next−line(2) operates on the unnarrowed buffer and will remove any narrows hiding the 'next' line.

EXAMPLE

c−hash−eval(3) macro defined in cmacros.emf uses narrow−buffer to hide regions of source code which has been #defined out, improving readability.

vm(3) defined in vm.emf uses narrow−buffer with appropriate arguments to append−buffer(2) and write−buffer(2) to write out only parts of the current buffer.

SEE ALSO

narrow(2m), set−mark(2), set−alpha−mark(2), get−next−line(2), c−hash−eval(3), vm(3).

newline(2)

NAME

newline − Insert a new line

SYNOPSIS

n **newline** (**return**)

DESCRIPTION

newline inserts *n* new lines into the text, move the cursor down to the beginning of the next physical line, carrying any text that was after it with it. The next line may automatically be indented depending on the current buffer mode, see cmode(2m), indent(2m), and wrap(2m).

SEE ALSO

cmode(2m), indent(2m), wrap(2m), buffer−mode(2).

next−frame(2)

NAME

next−frame − Change the focus to the next frame

SYNOPSIS

n **next−frame**

DESCRIPTION

next−frame changes the focus to the next frame. The numerical argument *n* can be used to select the type of frame to change to, it is a bit based flag defined as follows:

0x01

Allow the selection of an external frame.

0x02

Allow the selection of an internal frame. The default operation when *n* is omitted is to allow the selection of either type of frame (equivalent to an argument of 3). **SEE ALSO**

create−frame(2), delete−frame(2).

next−window(2)

NAME

next−window − Move the cursor to the next window
previous−window − Move the cursor to the previous window

SYNOPSIS

n **next−window** (**C−x o**)
n **previous−window** (**C−x p**)

DESCRIPTION

next−window makes the next window down the current window, if the current window is the last one in the frame the first one is selected. The numeric argument *n* can be used to modify this default behaviour, it is a bitwise flag where the bits are defined as follows:

0x01

If there is no 'next' window because this is the last then if this bit is set the search for the next window is allow to continue with the first window of the frame. As the default argument *n* is `1` this is the default behaviour.

0x02

When this bit is set windows whose $window−flags(5) are set to be ignored by this command are not skipped. The setting of bit `0x010` of a windows **$window−flags** will make the default action of this command skip it which means the the command may not simply select the next window but the next window without this flag set. Setting this bit of the numeric argument will force the command to always select the next window.

0x04

When set the search for the next window starts at the first window instead of the current window, this can be used to find the first window in the current frame.

previous−window makes the next window up the current window. The numeric argument *n* has the same effect on this command as for **next−window** except bit **0x04** starts the search at the last window of the frame.

EXAMPLE

The following example visits every window in the current frame printing the buffer it displays with a second pause between each one:

```
; go to the first window
!force 6 next-window
!while $status
    1000 ml-write $buffer-bname
    ; go to the next window - fail if this is the last
    !force 2 next-window
!done
```

NOTES

Both commands fail if a suitable window cannot be for, see the example on how this can be used.

SEE ALSO

next−window−find−buffer(2), next−window−find−file(2), set−position(2), goto−position(2), $window−flags(5).

next−window−find−buffer(2)

NAME

next−window−find−buffer − Split the current window and show new buffer

SYNOPSIS

next−window−find−buffer "*buffer*" (**C−x 3**)

DESCRIPTION

next−window−find−buffer splits the current window into two near equal windows, and swaps the current windows buffer to the given *buffer*. It is effectively a split−window−vertically(2) command followed by a find−buffer(2). When there is insufficient space in the current window to perform the split, then the current window is replaced. The requested *buffer* is always displayed, if the buffer does not already exist it is created.

SEE ALSO

find−buffer(2), split−window−vertically(2), next−window−find−file(2).

next−window−find−file(2)

NAME

next−window−find−file − Split the current window and find file

SYNOPSIS

next−window−find−file "*file*" (**C−x 4**)

DESCRIPTION

next−window−find−file splits the current window into two near equal windows, and loads the given *file* into the current window. It is effectively a split−window−vertically(2) command followed by a find−file(2).

When there is insufficient space in the current window to perform the split, then the current window is replaced. The requested *file* is always displayed, if the file does not already exist it is effectively created within MicroEmacs (although it will not exist on the disk until a save operation is performed).

The numeric argument *n* can be used to modify the default behaviour of the command, where the bits are defined as follows:

0x01

If the file does not exist and this bit is not set the command fails at this point. If the file does not exist and this bit is set (or no argument is specified as the default argument is `1`) then a new empty buffer is created with the given file name, saving the buffer subsequently creates a new file.

0x02

If this bit is set the file will be loaded with binary(2m) mode enabled. See help on **binary** mode for more information on editing binary data files.

0x04

If this bit is set the file will be loaded with crypt(2m) mode enabled. See help on **crypt** mode for more information on editing encrypted files.

0x08

If this bit is set the file will be loaded with rbin(2m) mode enabled. See help on **rbin** mode for more information on efficient editing of binary data files. **SEE ALSO** find−file(2), next−window−find−buffer(2), split−window−vertically(2), binary(2m), crypt(2m), rbin(2m).

normal−tab(3)

NAME

normal−tab − Insert a normal tab

SYNOPSIS

n **normal−tab**

DESCRIPTION

normal−tab insert a tab into the current buffer by temporarily disabling any auto indentation schemes. The macro first disables any indentation rules by setting $buffer−indent(5) to 0 and disabling the cmode(2m), the command tab(2) is then called with the given argument *n*. This means that the buffer's tab(2m) mode setting will be respected, i.e. whether a tab character or spaces are inserted. The initial indentation rules are restored on exit.

NOTES

normal−tab is a macro implemented in `format.emf`.

SEE ALSO

tab(2), insert−tab(2), tab(2m).

ntags(3f)

NAME

ntags − Generate a nroff tags file

SYNOPSIS

me "@ntags" <*files*>

DESCRIPTION

The start−up file `ntags.emf` may be invoked from the command line to generate a **tags** file for nroff files.

Given a list of *files* a tags file `tags` is generated in the current directory, which may be used by the find−tag(2) command. If no *files* are specified the default file list is "`./`", i.e. process the current directory. If a directory name is given (such as the default "`./`") all nroff files within the directory will be processed.

The value of variable **%tag−option** is used to control the tag generation process, its value <*flags*> can contain any number of the following flags:

`a`

Append new tags to the existing tag file, note that if also using flag '`m`' multiple 'tags' to the same item may be created.

`m`

Enable multiple tags. This enables the existence of 2 tags with the same tag name, but typically with different locations. See help on find−tag(2) for more information on multiple tag support.

`r`

Enables recursive mode, any sub−directory found within any given directories will also be processed.

NOTES

This function is invoked from menu

Tools −> Nroff Tools −> Create Tags File

when the user requests a tags file to be generated.

The tags are generated from the nroff macro:−

```
        .XI <name> ......
```

which indicates an index entry, where <*name*> is the tag name. <*name*> may be delimited by double quotes if any whitespace is present in the string.

This is the macro definition used in the MicroEmacs documentation system. The `ntags.emf` file should be edited and shadowed in the user directory if some other search criteria is used for nroff files. This macro file should provide a good starting point for any other search.

The user setup file "`myntags.emf`" is executed by ntags during start−up, this file can be used to over−ride any of the ntags configuration variables (see below).

The following variables are set within "`ntags.emf`" and are used to control the process:−

%tag−option

Tags options flag, default value is "". See above for more information.

%tag−filemask

A list of source file masks to be processed when a directory is given, default value is "`:*.nrs:*.[1−9]:*.n:`".

%tag−ignoredir

A list of directories to be ignored when recursive option is used, default value is "`:SCCS/:CVS/:`".

These variables can be changed using the −v command−line option or via the "`myntags.emf`" file

SEE ALSO

find−tag(2), start−up(3), nroff(9).

occur(3)

NAME

SYNOPSIS

occur

DESCRIPTION

occur performs a regular expression search for a string in the current buffer; generating a list of every occurrence of the regular expression in the buffer.

On invocation the user is prompted for a Regular Expressions the buffer is searched for the expression and the results are presented in the `*item-list*` window appearing at the left−hand side of the window.

The user may interact with the `*item-list*` buffer using the mouse or `<RETURN>`, on selecting a line then the user is moved to the corresponding line in the original buffer.

NOTES

The `*item-list*` window may be closed with the command item−list−close(3) typically bound to `esc-F7`.

occur is a macro defined in `itemlist.emf`.

SEE ALSO

item−list(3), item−list−close(3), search−forward(2), Regular Expressions

organizer(3)

NAME

organizer − Calendar and address organizer

SYNOPSIS

organizer

DESCRIPTION

organizer is a calendar and address organizer, enabling notes to be stored against the calendar days; addresses may be archived into an address book.

organizer uses the MicroEmacs '02 in−built registry to store information within a registry file called *<username>*`.eof`. **organizer** may be entered directly from the command line, or via the menu (via **Tools**).

organizer is displayed within a single osd dialog box, tab selections at the top of the window enable the different forms of information to be displayed. Navigation is typically performed using the mouse, where the mouse is absent then the `TAB` key may be used to move between the fields. The information presented is defined as follows:−

Month

Shows the calendar month, starting with the current month, the current day is hi−lighted and any notes that have been entered are displayed in the **Notes** entry box at the bottom of the page.

The default mode of operation is note entries for the current month, however specifying the *<year>* as the wild card '*' (star) enables annual events to be entered into the organizer. Annual events are automatically inserted into the calendar each year, typically used for birthdays etc.

The entry controls to the dialog are defined as follows:−

<−

Advances to the previous month.

−>

Advances to the next month.

<Month>

A pull down dialog enabling month selection.

<year>

A text entry field specifying the current year as a 4 digit number. A value of ***** is the wild card year for specifying annual events.

Notes

A free form text entry box allowing a note to be attached to the currently selected day.

Save

Saves the entry back to file.

Month To Buffer

Dumps a view of the month to the currently active buffer, any notes are also dumped to the buffer.

Exit

Exits the **organizer**. **Week**

Shows the calendar week in the current buffer, the days of the week are shown in a column ordering. Note that selection of the week is typically performed from the **Month** view, moving to the **Week** view (via the tab) selects the week appropriate to the previously selected day within the month view.

The entry controls on the dialog are defined as follows:−

<−

Advances to the previous week.

−>

Advances to the next week.

<year>

A text entry field specifying the current year as a 4 digit number. The value of ***** for viewing and setting annual events is not valid in this view.

Notes

A free form text entry box allowing a note to be attached to the currently selected day.

<day>

Selecting a date in the *day* column changes the view to the **Day** view.

Save

Saves the entry back to file.

Week To Buffer

Dumps a view of the week to the currently active buffer, any notes are also dumped to the buffer.

Exit

Exits the **organizer**.

Note: The start day in the week view may be configured to commence on a day other than Sunday from the **Setup** tab.

Day

Shows an extended view of the notes attached to the current day, day selection is typically performed from the **Month** or **Week** views. The entry controls on the dialog are defined as follows:−

<−

Advances to the previous day.

−>

Advances to the next day.

<year>

A text entry field specifying the current year as a 4 digit number. A value of ***** is the wild card year for specifying annual events.

<month>

A pull down dialog enabling month selection.

<day>

A text entry enabling the current day to be entered.

Notes

A free form text entry box allowing a note to be attached to the currently selected day.

Save

Saves the entry back to file.

Day To Buffer

Dumps a view of the day to the currently active buffer, any notes are printed in the buffer.

Exit

Exits the **organizer**. **Lists**

The lists pane provides support for multiple list generation and manipulation. Each list consists of zero or more ordered items each of which has a text field in which the user can enter information.

Entry to the dialog is defined as follows:−

List

Selects a list.

New

Creates a new list.

Lines Per Item

Sets the number of lines to use when displaying a list item.

New

Creates a new list item at the end of the current list.

Up

Moves the currently selected item (left click on the item number) up the list.

Down

Moves the currently selected item down the list.

Insert

Inserts a new list item before the currently selected item.

Delete

Deletes the currently selected item.

Save

Saves the entry back to file.

List To Buffer

Dumps a view of the list to the currently active buffer.

Exit

Exits the **organizer**. **Address**

The address pane provides entry to the address book, enabling personal and business details to be retained against a single name, tabbed selection of **Work** or **Home** selects the information that is displayed. A search engine is provide to locate names within the database, and provision is made to save some text against a name. Entries in the database are, by default, organized by record number, sorting may be explicitly performed from the **Sort** button.

Entry to the dialog is defined as follows:−

<Record No>

The identity number of the record, a value of ***** denotes that this is a new record that is being inserted.

<<

Moves to the start of the database.

>>

Moves to the end of the database, showing record *****, a new entry may be entered.

<

Moves to the previous record.

>

Moves to the next record.

Name

The name of the individual, entered as *fore−name* and *surname*.

Nickname

A pseudo name assigned to an individual.

Partner

Shown in the **Home** view only. The *forename* and *surname* of any partner.

Chld

Shown in the **Home** view only in the **Extended Address Book Mode**. The names of any children (up to 3).

DOB

Date of Birth, shown in the **Home** view only in the **Extended Address Book Mode**. The dates of birth of the parents, any children in addition to an anniversary date.

Company

Shown in the **Work** view only. The name of the company.

Address

The address of the individual/company.

Tel/Fax/Mobile

Telecommunication information.

Email/WWW/FTP

Electronic communication information.

Notes

Notes associated with the individual.

Save

Saves the address information to file.

Dup

Duplicates the currently selected address entry, creating a new record card. Typically used to construct a similar entry for a different individual.

Delete

Deletes the currently selected entry.

Addr to Buffer

Dumps the currently selected address to the current buffer.

Exit

Exits the organizer.

Find

find provides access to a search engine, enabling addresses to be located in the address book.

Search For

The string to search for.

In Field

Pull−down menu allowing the selection of the field to be searched in.

Match

Selects how strict the search should be; typically **Any Part** is used as this is the least in−exact search. The default mode is configured in the **Setup** tab.

Case/magic

Selects the search criteria. The default mode is configured in the **Setup** tab.

First

Finds the first record that matches the search criteria

Next

Finds the next record that matches the search criteria, from the currently displayed record.

Reverse

Searches in reverse order.

Exit

Exits the search

Sort

sort provides a mechanism to re−sort the data base into a different order. The sort is performed on up to 3 different keys enabling conflicting primary sort fields to be resolved by the secondary sort criteria. The default sort order is *<Record No>*, *<None>*, *<None>*.

Sort Keys

The *Primary*, *Secondary* and *Tertiary* sort fields are selected by a pull down menu. The fields to be used for sorting are selected from the list.

Sort

Performs the sort, based on the settings of the *Sort Keys*.

Exit

Exits the sort dialog. **Setup**

The **setup** pane configures a number of general settings of the organizer.

Current Organizer File

The full pathname of the organizer file. By default this is set to *<userpath><userName>*.**eof** and can be altered using user−setup(3).

Change Name

Allows the displayed name of the month and the day to be modified.

First Day of the week

Selects the first day of the week, this sets the first day to be displayed in the **Week** view and the first column in the **Month** view.

Min New Year Days

The number of days that must appear in the first week of the New Year for the week to be considered week 1. Modifying the value of this field modifies the week number.

The **Calendar** section allows the wordy representation of the calendar date to be modified. Typically used to modify the names to the native language.

Change Month Name

Select the existing month representation from the left−hand box and type in a new selection into the right−hand box.

Change Week Day Name

As *Change Month Name*, enables the day of the week representation to be modified.

First Day Of The Week

Selects the first day that appears in the **Week** view.

Minimum Days of New Year in first week

Specifies the number of days that must appear in the first week of the New Year for the week to be designated as week 1. This value allows the week number to be aligned with the calender weeks of standard diaries. The default value is 7 days; but may be reduced to 5 or 6 for typical alignment.

The **Address Book** section allows the operation of the address book to be modified.

Use Extended Address Book

The extended address book allows additional information to be added to the personal address book. The extended information is limited to the amount of personal information attributed to an individual, including *Date of Birth* and *Child* information.

Import From File

The **Import** from file allows the address book to be imported from a file. The import data format is a single line per entry, comma `,` separated. The field order is defined as follows, the ***** entries indicate the **Extended Address Book** fields:−

Record No, *First Name*, *Surname*, *Nick Name*, *Selected*, *Notes*, *Partner First Name*, *Partner Surname*, *Home Address*, *Home Telephone*, *Home Fax*, *Home Mobile*, *Home E−Mail*, *Home WWW Page*, *Home FTP Site*, *Work Company*, *Work Address*, *Work Telephone*, *Work Fax*, *Work Mobile*, *Work E−Mail*, *Work WWW Page*, *Work FTP Site*, *Date−Of−Birth**, *Partner DOB**, *Date−Of−Marriage**, *Child1 Name**, *Child1 DOB**, *Child2 Name**, *Child2 DOB**, *Child3 Name**, *Child3 DOB**.

Export To File

Exports the address book to a file, the address book is exported in the current sort order, with the fields defined as above. The exported address book may then be imported into a 3rd party package i.e. Microsoft Access, etc.

The **Default Address Find Settings** section defines the default search criteria used in the address book search function.

Whole/Start/Any Part

Radio buttons determine how the search is performed on the string.

· **Whole** matches the whole string exactly.
· **Start** matches the first part of the string only (i.e. `Ab*`).
· **Any Part** finds entries that include the search string at any position within the data base search field.

Case Insensitive

Checked, matches the strings regardless of case. (default).

Magic Mode

Allows magic strings to be included in the search string. **NOTES**

organizer is a macro that is implemented in `organiz*.emf` files. Organizer uses osd(2) to create and manage the dialogs.

The maximum size of a text note is 1024 characters.

With an new address is created it is added to the end of the address list regardless of the current sort criteria.

Organizer replaces the original **Calendar** utility.

SEE ALSO

user−setup(3), osd(2).

osd(2)

NAME

osd − Manage the On−Screen Display

SYNOPSIS

osd
−1 **osd**
−2 **osd**
n **osd**
−1 osd *n*
osd −1 *flag*
osd *n* **0** *flags* ["*scheme*"] ["*x−pos*" "*y−pos*"] ["*min−width*" "*min−depth*" "*max−wid*" "*max−dep*"] ["*default*"] [["*title−bar−scheme*"] ["*Text*"]] ["*resize−command*"] ["*control−command*"] ["*init−command*"]
osd *n i flags* ["*tab−no*"] ["*item−scheme*"] ["*width*" "*depth*"] ["*text*"] ["*argument*" "*command*"]

DESCRIPTION

The **osd** command manages the On−Screen Display, menu and dialogs. The command takes various forms as defined by the arguments. Each of the argument configurations is defined as follows:−

Main Menu−Bar Status

osd −1 *flag*

This invocation determines the state of the top main menu bar. The state is set by the argument *flag* defined as:−

```
 1 − enable.
 0 − disable.
−1 − disable and destroy.
```

Dialog Creation and Redefinition

osd *n* **0** *flags* ["*scheme*"] ["*x−pos*" "*y−pos*"] ["*min−width*" "*min−depth*" "*max−wid*" "*max−dep*"] ["*default*"] [["*title−bar−scheme*"] ["*Text*"]] ["*resize−command*"] ["*control−command*"] ["*init−command*"]

This invocation creates or resets the base properties of dialog *n*. The *flags* argument determines the arguments and are defined as follows:

A

Defines dialog as an alpha type dialog, items are added according to their string text value. Alpha dialogs may not have separator or child items.

i

Used with the **A** flag, sets the alpha ordering to be case insensitive.

G

Create a Grid dialog. Every item in the dialog is given a single character boarder around it. If one of the dialogs items is also given a '**G**' flag, the boarder is drawn as a box around it, otherwise spaces are used.

N

Create a Note−Book (or tabs) dialog. The dialog can only contain one dialog inclusion item ('**I**') and Note−Book pages ('**P**'). Pages added before the Inclusion item (page item number is lass than the inclusion page item number) will be drawn at the top of the note−book, those added after will be drawn at the bottom.

b

Draw boarder, draws a boarder around the outside of the dialog. See also *flag* **t** (title) as flag effects the boarder.

a

Defines the absolute start−up position of the dialog in the arguments *x−pos* and *y−pos*, which are the column and row positions respectively of the dialog from the top left−hand corner of the display. The arguments must be specified. e.g. the main menu is defined with an absolute position of (0,0). If the dialog can not be fully drawn on the screen at the given position it will be moved to a position which shows the most.

o

Specifies an offset to the dialog position calculated by MicroEmacs in the arguments *x−pos* and *y−pos*, which are the column and row offsets. This flag is ignored when flag **a** is also specified. If the dialog can not be fully drawn on the screen at the new position it will be moved to a position which shows the most.

s

Sets the size of the dialog. **osd** automatically resizes a dialog to fit the contents, this flag should be considered as a size hint for **osd**, and is not guaranteed to be honored. If the dialog has a boarder (flag **b**) the size given should include the boarder size.

The arguments, *min−width*, *min−depth*, *max−width* and *max−depth* must be specified, as

+ve

The actual size of the dialog, minimum and maximum sizes.

0

min value should be specified as desired window size, *max* may be 0 which specifies the screen size.

−ve

min defines the maximum size. *max* is unlimited.

The following table shows possible combination of the sing parameters and their effect:−

min=0, *max*=0

Default setting, makes dialog as small as possible, with a maximum size of the screen.

min=0, *max*=50

Make dialog as small as possible with a max of 50 characters.

min=50, *max*=0

Make dialog as small as possible, but make it at least 50 characters big and no larger than the screen.

min=30, *max*=−1

Make dialog at least 30 characters big with no upper limit, very useful for dialogs being used as scrolled children.

min=−1, *max*=50

Make dialog 50 characters big.

min=−1, *max*=0

Make dialog the same size as the screen.

min=−1, *max*=−1

Make dialog as big as possible (do not do this unless you have a large amount of memory to throw away).

S

Sets the main dialog scheme, The default scheme when not specified is $osd−scheme(5) See macro file `fileopen.emf` for an example.

d

Sets default item to select to "*default*". This item is selected when the dialog is first opened, if this item is an automatically opened sub−menu then the child menu will also be opened.

t

Title bar is present − draws the title bar. The *text* argument is optional Also see flags **H**, **c** and **r**.

H

Defines the title bar color scheme if flag **t** is specified. If *t* is absent the option is ignored.

c

Centers the title bar text if specified. Option **t** must be specified, otherwise the option is ignored.

r

Right justifies the title bar text if specified. Option **t** must be specified, otherwise the option is ignored.

R

Defines the dialog as re−sizable. The *resize−command* argument must be specified and the command should resize the dialog to the sizes given in $result(5) in the format "`wwwwdddd`", where `w` is width and `d` the depth. If the *resize−command* is aborted then that resize operation is abandoned.

M

Identifies the dialog as the main menu dialog.

C

Binds a command to the dialog, which is automatically executed when the dialog is opened. When the dialog with a **C** attribute is opened, it is rendered on the screen and then a command, defined by *control−command* is invoked, when the command completes the dialog is closed.

The command dialog is typically used to create status messages. e.g. a "`Busy − Please Wait`" dialog box, such a dialog may be implemented when saving the current buffer then create the simple dialog and sent the *control−command* to save−buffer(2). The dialog would be defined as:−

```
osd 200 0 "btcHC" %osd−title−scheme "Saving Buffer" save−buffer
osd 200 1 ""
osd 200 2 "" "Busy − Please Wait"
osd 200 3 ""
200 osd
```

If the dialog has buttons which need to become active then control can be returned to **osd** by calling **osd** with no arguments, e.g. in the above example the dialog can be made to stay on the screen until the user selects `okay` by:

```
define−macro test−osd
    save−buffer
    osd 200 2 "" "Save Complete"
    osd 200 4 "BcfH" %osd−ebtt−scheme "  &Okay  " f void
    osd
!emacro

osd 200 0 "btcHC" %osd−title−scheme "Saving Buffer" test−osd
osd 200 1 ""
osd 200 2 "" "Busy − Please Wait"
osd 200 3 ""
osd 200 4 "BcfHS" %osd−dbtt−scheme "  Okay  "

200 osd
```

The above mechanism is how spell−buffer(3) operates.

k

Disables hot−keys for the dialog. All text strings are copied literally. This is useful for dialogs like the tags child dialog as many tags have '&'s in them.

B

Makes the mouse right Button have the same behaviour as the left, by default the right mouse button simply closes the dialog. This is useful for some dialogs which are opened using the right mouse button.

f

Automatically uses the first letter of an item's test as the hot key. Unlike the normal hot keys, the letter is not hi−lighted and when typed by the user the item is only selected, not executed. This flag also disables the normal hot−keys for the dialog, so all text strings are copied literally.

n

Disables '\n' characters in text fields leading to multi lines. By default a text item of "Hello\nWorld" will create an item 5 by 2 characters big.

If "*init−command*" is given then this function is always called just prior to the dialog being displayed so it can be used to configure the dialog.

Dialog Destruction

−1 osd *n*

This invocation destructs a dialog *n*. The dialog is only destroyed if it is not currently being displayed.

Dialog Item Creation and Redefinition

osd *n i flags* ["*tab−no*"] ["*item−scheme*"] ["*width*" "*depth*"] ["*text*"] ["*argument*" "*command*"]

This invocation type adds a new item *i* to a dialog *n*, the operation of the invocation is controlled by the *flags* as follows:−

D

Disable item *i*, the item is ignored and is not rendered in the dialog.

I

Include dialog "*argument*" into this dialog. If "*command*" is specified then it is called prior to the child being constructed and can be used to define the child. This is similar to the **M**s command. See also flag **b**.

P

Item is a Note−Book page, the item must have text and have an argument which is the osd dialog to be show when the page is activated.

M

Item is a sub−menu, The argument "*argument*" specifies the sub−menus osd dialog number. A "*command*" may also be specified which is executed first, this can actually re−define the item and set the dialog number, e.g.

```
; To start with the dialog number is unknown
osd 1 1 "M" f submenu−setup

define−macro submenu−setup
    osd 200 0 ....
    ....
    ; Now the sub−menu number is known redefine parent item,
    ; note the setup  command  is not given as we have now set
    ; it up!
```

```
            osd 1 1 "M" 200
        !emacro
```

See also options **m**, **n**, **e**, **s**, **w** and **d**.

m

Sub−menu must be manually opened, using hot−key, the return key or the left mouse button.

n, **e**, **s**, **w**

Specify where a sub−menu is to be placed relative to the parent item. The letter indicates the direction as points on a compass, North, East, South and West, respectively. The default when omitted is East.

d

Display sub−menu type, i.e. "`..`" for auto opening and " >" for a manual opening sub−menu.

−

Fill a non−defined chars with '**−**'s instead of ' 's, used to draw the lines across menus, typically with no text given, e.g.

```
osd 200 5 "−"
```

But could also be specified as:

```
osd 200 5 "−c" "Lined"
```

C

Item is a check−box. The setting of the check−box is evaluated when the dialog is first drawn, re−draw and whenever any item is executed. A "*command*" must be specified which must both return the current setting when the given argument (of 1) is given (!abort if false, !return if true) and change the value if the argument value is negated. The text string must also be specified, the first 6 characters are used in the drawing of the check box. The format can be shown as follows:−

```
String\State       Off          On
"123456"            "12356"      "12456"
" (−+)^"            " (−)"       " (+)"
"^[ *] "            "[ ] "       "[*] "
"^^NY^^"            "N"          "Y"
"^^^^^^"            ""           ""
```

Note that no character is rendered when a '^' character is used. See also **p** for prepending the check−box.

p

Prepend the check−box box. By default a check box is drawn as:

```
"Check box12?56"
```

This option changes it to:

```
"12?56Check box"
```

x

When the item is executed do not exit the dialog. Often used with Check−boxes.

i

The command given is a command line string which is executed in a similar fashion to execute−line(2). Note that if an argument is required it is usually specified in the string, i.e.

```
osd "i" "text" 5 "1000 ml−write @#"
```

writes the argument (i.e. 5) for 1 second.

```
osd "i" "text" 5 "my−command"
```

in this case *my−command* will not be given an argument,

```
osd "i" "text" 5 "10 my−command"
```

in this case *my−command* will be given an argument of 10,

```
osd "i" "text" 5 "@# my−command"
```

in this case *my−command* will be given an argument of 5.

h

Horizontally add the next item, e.g.

```
osd "h" "1st on line "
osd ""  "2nd on line"
```

Will produce "`1st on line 2nd on line`". If there is not enough room on a single dialog line to display all the horizontally added items then the line is split and as many lines as needed are used.

c

Center the text for the item in the middle of the dialog.

r

Right hand justify the text for the item.

t

Set the items tab order in the dialog.

b

Child inclusion is a scroll box type. By default a child inclusion simply draws the whole child dialog at the position. If this flag is specified then arguments "*width*" and "*depth*" must also be supplied and a window displaying "*width*" by "*depth*" of the child is created. The size of this item will be "*width*"+1+*ss* by "*depth*"+1+*ss* where *ss* is the scroll bar size which is 1 or 2 depending on the setting of $scroll−bar(5). It is up to the user to ensure that the child dialog being displayed is at least "*width*" by "*depth*" characters in size, if this is not true then the effect is undefined, (a crash dump is not out of the question).

f

Fix the item size to the given "*size*", by default an item is expanded to the width of the dialog.

E

Item is an entry box type. Use a string of #'s to set the position and size of the entry text box. Similar to Check−boxes, the command given must both return and set the value depending on value of the argument given. The value must be returned in $result(5) if the given argument (or 1 for '`f`') is given, and the value must be set (usually using @ml(4) or @mc(4)) if the argument is negated. The absolute value of the argument is maintained.

```
set-variable %entry-value "Hello world"

define-macro my-entry-set
    !if &equ @# -1
        set-variable %entry-value @ml "" %entry-value
    !else
        set-variable $result %entry-value
    !endif
!emacro

osd 200 1  "S" " &Enter text" 2
osd 200 2  "ExHf" %osd-entry-scheme "########" 1 my-entry-set
```

B

Item is a Button type. Add the last 2 characters of $window−chars(5) to the text string given, one on each side, i.e. if the last two chars are "[]" then:

```
osd "B" " Okay "
```

will be drawn as "`[ Okay ]`". See also flag **T**.

T

Item is a repeat type, this is typically used with buttons, altering their execution behavior. By default an item is only executed when the left mouse button is released while over the item. However when this flag is specified the item is executed as soon as the left mouse button is pressed and is repeatedly executed until the button is release or the mouse moves off the item. The delay between repeated executions is determined by the variables $delay−time(5) and $repeat−time(5).

S

Item is a separator type. This is not often required as any item without anything to execute is automatically set to be a separator. Occasionally a mouse−insensitive item which can be executed is required, typically a text string with a hot key, e.g.

```
osd 200 1  "S" " &Enter text" 2
osd 200 2  "ExHf" %osd−entry−scheme "########" 1 my−entry−set
```

will be drawn as "`[ Okay ]`"

Item 1 will have a hot−key which executes item 2 (as no command is given), but it will not hi−light if the mouse is placed over it.

R

Redraw dialog. Forces a redraw of the dialog when the item is executed. This is not usually required as **osd** generally works out for itself whether a redraw is needed, however, sometimes it does not, most notably when the item sets a variable that is displayed by another item as an entry, e.g.

```
set−variable %entry−value "Hello world"

define−macro my−entry−set
    !if &equ @# −1
        set−variable %entry−value @ml "" %entry−value
    !else
        set−variable $result %entry−value
    !endif
!emacro

osd 200 1  "S" " &Enter text" 2
osd 200 2  "ExHf" %osd−entry−scheme "########" 1 my−entry−set
osd 200 3  "BxHcfiR" %osd−ebtt−scheme  " &Reset " f "set−variable %er
```

If item 3 did not have flag **R** set when executed, **osd** would not realize that the change to value `%entry−value` affects the display and the button would not appear to operate.

H

Sets the item color scheme. Note that for scrolled child items this only sets the scroll−box color scheme, the dialog scheme is used for the rest of the boarder.

G

This flag is only applicable in grid dialogs (see flag **G** in dialog creation). The current item will be drawn with a box around it using $box−chars(5).

z

Sets the item size, arguments "*width*" and "*depth*" must be given.

N

This flag only has an effect on entry item types, it selects 'New−line' style text entry which allows the user to enter multiple line of text using the return key and to end the input using the tab key.

Note that for a non−sub−menu item type, if an argument is given with no command then it is assumed that the number given is the item number to be executed, see flag **S** for an example.

Dialog Exacution

n **osd**

This invocation with a single positive numeric argument executes the *n*th dialog.

Returning Command Control

osd

An invocation of **osd** with no arguments returns control back to the **osd** from a *control−command*. Refer to the **C** flag in the create/reset dialog property for information and an example.

Current Dialog Redraw

−1 **osd**

Calling osd with an argument of −1 forces the complete redrawing of current dialog and any sub−dialogs. This is very useful when the execution of one item may effect the appearance of another.

Redraw All Active Dialogs

−2 **osd**

Calling osd with an argument of −2 forces the complete redrawing of all currently active osd dialogs. This is better than calling screen−update(2) when only the osd dialogs need updating as it suffers less from flickering.

EXAMPLE

Refer to `osd.emf`, `userstp.emf`, `search.emf`, `spell.emf` and `organize.emf` for examples of the OSD.

SEE ALSO

$osd−scheme(5), $result(5), $scroll−bar(5), $window−chars(5).

osd−bind−key(2)

NAME

osd−bind−key − Create key binding for OSD dialog
osd−unbind−key − Remove key binding from OSD dialog

SYNOPSIS

osd−bind−key *n* "*command*" "*key*"
osd−unbind−key *n* "*key*"

DESCRIPTION

osd−bind−key creates a local key binding for a given osd dialog, binding the command *command* to the keyboard input *key*. Only the current root dialog's local bindings are used, local bindings of included dialogs or other root dialogs currently displayed are ignored.

Osd local bindings take priority over default osd bindings, local bindings created using ml−bind−key(2) are also used, but any current buffer local bindings created using buffer−bind−key(2) are ignored.

NOTES

The prefix commands cannot be rebound with this command.

Key response time linearly increases with each osd binding added.

As only the root dialog's bindings are used, creating note−book page specific bindings can be awkward. Typically all required keys are bound to the same command which, depending on the page that is currently being displayed, checks if the key pressed is bound on the current page and if so calls the required command. See organizer(3), defined in `organize.emf` for an example of this operation.

SEE ALSO

osd(2), global−bind−key(2), ml−bind−key(2), buffer−bind−key(2), global−unbind−key(2).

osd−dialog(3)

NAME

osd−dialog − OSD dialog box
osd−xdialog − OSD Extended dialog box
osd−entry − OSD entry dialog box

SYNOPSIS

n **osd−dialog** "*title*" "*prompt*" ["*x−pos*" "*y−pos*"] "*but1*"
n **osd−xdialog** "*title*" "*prompt*" *default* ["*x−pos*" "*y−pos*"]

"*but1*" "*but2*" ...
n **osd−entry** "*title*" "*prompt*" *variable* ["*x−pos*" "*y−pos*"]

[["*entry−xsize*" | "*entry−xsizexentry−ysize*"] ["*type*"]] **DESCRIPTION**

osd−dialog constructs a OSD dialog prompt with a title string *title*, a prompt string within the dialog of *prompt*. A single button, with text rendering *but1*, is placed within the dialog. The dialog remains on the screen until the button is selected or the user aborts.

osd−xdialog creates an extended dialog with multiple buttons similar to **osd−dialog**, the number of buttons created (*#*) is determined from the number of *but* arguments. The *default* integer argument specifies the default button (1..*#*), a value of 0 specifies that there is no default button.

The commands return the button pressed in the variable $result(5).

osd−entry constructs a simple OSD entry dialog which prompts the user to type in a value. The value of the supplied variable is used as an initial entry value, the variable is set to the entered value when the user presses the "`Okay`" button but remains unchanged if the user Cancel or aborts.

The size of the entry defaults to 30 characters if not specified by the user, when a size parameter is given it can take one of two forms, either simply "`w`" specifying the width, the height defaulting to 1, or "`wxh`" (i.e. "`40x5`") specifying both. The last optional argument *type* sets the type of value being entered (e.g. file name, buffer name, etc) see flag **h** on the help page for @ml(4) for a list of entry types and the numerical value to be supplied.

The argument *n* can be used to change the default behavior of the commands described above, *n* is a bit based flag where:−

0x01

Enables command abort (default), except **osd−entry** which ignores the setting of this bit. When enabled, if the user abort by either closing the dialog (top right button) or using the **abort−command** the dialog command will also abort. If bit 0x01 is not set the command will not abort and **$result** will be set to −1.

0x02

When set, flags that a dialog position has also been provided, extra arguments **x−pos** and **y−pos** must also be given. By default the dialog is placed under the mouse. **EXAMPLE**

A simple query dialog is typically constructed using **osd−dialog**, as follows:−

```
!if &seq %osd-search-str ""
    osd-dialog "Replace" "Error: Search string is empty!" "  &OK  "
    !return
!endif
```

The following example uses multiple buttons within a single dialog, using **osd−xdialog**, as follows:−

```
0 define-macro osd-close
    !if &bmod "edit"
        set-variable #l0 &spr "Buffer \"%s\" changed" $buffer-bname
        osd-xdialog "Buffer Close" #l0 1 "&Save First" \
                                        "&Loose Changes" "&Cancel"
        !if &equ $result 3
            !abort
        !elif &equ $result 2
            -1 buffer-mode "edit"
        !else
            !if &seq $buffer-fname ""
                !nma write-buffer
            !else
                !nma save-buffer
            !endif
        !endif
    !endif
    delete-buffer $buffer-bname @mna
!emacro
```

The next example macro can be used to change the value of a user variable to a user supplied file name:

```
set-variable %source-root "~/"

define-macro set-source-root
    osd-entry "Source Root" "&Path : " %source-root 35 1
!emacro
```

NOTES

osd−dialog, **osd−xdialog** and **osd−entry** are macros defined in `osd.emf`, using osd(2) to create the dialog.

SEE ALSO

$result(5), osd(2).

osd−help(3)

NAME

osd−help − GUI based on−line help

SYNOPSIS

osd−help

DESCRIPTION

osd−help provides a GUI front end to the on−line help manual, the dialog consists of 3 pages which are defined as follows:−

Contents

The contents page displays a list on contents similar to the help(2) high level help page. Selecting an item will display the help page in a buffer, selecting **Exit** will exit the dialog.

Index

The index page gives a list of help items, the **Scope** menu can be used to narrow the index list to the required item type.

Search

The search page provides a way of searching the on−line help for a given topic. Similarly to the Index page, the **Scope** menu is provided to narrow the search to the required area.

The search strings is considered to be made up of items separated by spaces, an item can be enclosed in quotes ('**"**') so that the item can include a space. If the first letter of an item is a '**+**' the given item must be found in a page for it to match, if the character is a '**−**' the item must NOT be found on a page for it to match, or other items are considered optional. At least one item must be found on a page for it to be a match, the numbers to the right of each found page is the number of items found.

NOTES

See Help! for help on the on−line help pages.

osd−help is a macro using osd(2), defined in osdhelp.emf.

SEE ALSO

help(2).

over(2m)

NAME

over − Over−strike Mode

SYNOPSIS

over Mode

O − mode line letter.

DESCRIPTION

over mode, when enabled, over writes existing text in a buffer as opposed to inserting text. over maintains the position of text aligned with `tab` characters.

SEE ALSO

buffer−mode(2), global−mode(2).

Patience(3)

NAME

Patience − MicroEmacs '02 version of Patience (or Solitaire)

SYNOPSIS

Patience

DESCRIPTION

Patience (or Solitaire) is a solitaire game using a standard set of playing cards. The object of the game is to use all of the cards in the deck to build up four suit stacks from Ace to King.

The board is laid out with the dealer pile at the top right hand corner, to the left are four suit stacks onto which cards of the same suit are placed, in ascending order from the Ace. Below these two areas of the board are seven row stacks, organized in a triangular shape with zero to six downward facing cards.

Cards may be moved around the playing area by stacking alternative red and black cards in descending order on the row stacks. When a row stack has no upturned cards on the stack then the top card may be turned over and may be played. If a stack becomes empty then only a King may be moved into the vacant position. Cards may be removed from the dealer, they are over−turned in sets of three cards, the underlying 2 cards are visible, but are not accessible, only the top card may be removed and played from the dealer.

Cards are moved around the board using the mouse. Cards may be moved from the dealer or between the row stacks by placing the mouse over the card to be moved and pressing the left mouse button. Move the cursor to the new card position and release the left mouse button. If the move is legal then the card(s) are moved to the new stack. Multiple cards may be moved from the row stacks, the appropriate card(s) to be moved is automatically determined.

Cards may be moved onto the suit stacks by a single left mouse press and release on the same card, the card is moved to the appropriate suit stack. The same technique is used to turn cards over in the suit stacks, and to deal the next set of cards by the dealer. To deal, then click on the down−turned card stack, if there are no further cards at the dealer then click on the empty position and the dealer will turn over the dealer stack and deal from the top again.

Note that once a card is played onto the suit stacks then it cannot be removed.

To the right of the board are a number of control buttons. To select an option, click the left mouse button on it, the buttons are labeled:

DEAL

Start a new game by dealing new cards.

QUIT

Exit the game

HELP

This help page

Note that the screen may be updated at any time using "*C−l*".

NOTES

Patience is a macro defined in `patience.emf`.

The game is best played with a mouse, it is possible to play with the keyboard, as follows:−

"*esc h*" for help

To move a card between stacks enter the source and destination column number ("*1*","*2*",.."*7*"). To move from the dealer pile then the source is the "*space*" key.

"*tab*" deals the next cards.

To overturn a card on the row stacks then enter the card column twice i.e. source and destination are the same.

To move a card from the row to the suit stacks then either enter the card column twice, or enter the destination as "*h*","*d*","*c*","*s*" (i.e. "*2 2*" or "*2 s*" to move the card in column 2 to the spades stack).

"*C−c C−c*" to deal the cards again.

"*C−l*" redraw the screen.

"*q*" to quit the game.

SEE ALSO

Games, Triangle(3), Mahjongg(3).

p(9)

SYNOPSIS

p, pas − Pascal files

FILES

hkpascal.emf − Pascal hook definition
pascal.etf − Pascal template file.

EXTENSIONS

.p, **.pas** − Pascal file

DESCRIPTION

The **pascal** file type template provides simple hilighting of Pascal files, the template provides minimal hilighting.

BUGS

None reported.

SEE ALSO

Supported File Types

paragraph−to−line(3)

NAME

paragraph−to−line − Convert a paragraph to a single line

SYNOPSIS

n **paragraph−to−line**

DESCRIPTION

paragraph−to−line is a variation of fill−paragraph(2). **paragraph−to−line** reduces each of the next *n* paragraphs of text to single lines. This command is typically used to prepare text for import into a word processor such as **Microsoft Word** or **Word Perfect**. Reduction of text to a single line allows the word processor to import the raw text file and keep the text within paragraph blocks. If the text is not prepared then all of the line−feeds have to be manually deleted.

paragraph−to−line allows text based documents to be prepared in MicroEmacs '02 and imported into the word processor at the final stage for formatting and layout.

NOTES

paragraph−to−line is a macro defined in `format.emf`.

SEE ALSO

fill−paragraph(2).

perl(9)

SYNOPSIS

perl − Practical Extraction and Report Language File.

FILES

hkperl.emf − Practical Extraction and Report Language file hook definition
perl.etf − Practical Extraction and Report Language header template file.
perl.eaf − Practical Extraction and Report Language abbreviation file.

EXTENSIONS

.pl, **.pm** − Perl file

MAGIC STRINGS

#![\t]*/.*perl

MicroEmacs '02 recognizes the magic string on the first line of the file used to locate the executable. The Perl files may be extension−less and are still recognized.

# −*− perl −*−

MicroEmacs '02 recognizes the standard GNU Emacs magic string on the first line of the file. The Perl files may be extension−less and are still recognized. **DESCRIPTION**

The **perl** file type template provides the hilighting, indentation and tools definitions for a perl file.

File recognition is performed using the standard file extension **.pl**, **.pm** or by the magic string.

General Editing

On creating a new file, a new header is automatically included into the file. time(2m) is by default enabled, allowing the modification time−stamp to be maintained in the header.

Hilighting

The hilighting features allow commands, variables, logical, comments, strings and characters of the language to be differentiated and rendered in different colors.

Auto Layout

The indentation mechanism is enabled which performs automatic layout of the text. restyle−region(3) and restyle−buffer(3) are available to reformat (re−layout) selected sections of the buffer, or the whole buffer, respectively.

Folding and Information Hiding

Generic folding is enabled within perl files. The folds occur about *sub*...**}** located on the left−hand margin. fold−all(3) (un)folds all regions in the file, fold−current(3) (un)folds the current region.

Short Cuts

The short cut keys used within the buffer are:−

C−c C−c − Comment out the current line.
C−c C−d − Uncomment the current line.
C−c esc esc − Command complete.
A−C−i − Restyle the current region.
f2 − (un)fold the current region
f3 − (un)fold all regions

Debugging

Debugging a perl script can be done inside MicroEmacs by using the perldb(3) command. **BUGS**

The flexibility of the perl language does cause some hilighting anomalies from time to time, typically with unbalanced quote characters. Most of the common exceptions have been caught, however there are a few syntax sequences that involve quotation marks that can cause problems.

SEE ALSO

perldb(3), fold−all(3), fold−current(3), indent(2), restyle−buffer(3), restyle−region(3), time(2m).

Supported File Types

perldb(3)

NAME

perldb − Perl Debugger

SYNOPSIS

n **perldb** ["*script−name*"] "*script−args*"

DESCRIPTION

perldb provides an editor interface to the Perl debugger, on running the command an interactive shell window is opened to the debugger command line interface. MicroEmacs then interprets the information from the debugger interface and opens files and hilights the current line as required. The current line is maintained while single stepping through the script.

Buffers opened and referenced by the debugger have the key `F9` bound to setting a break point, this only works if the buffer contains the current execution point.

If an argument *n* of 2 is given to **perldb** the command assumes that the current buffer is the script file to debug, the "`script−name`" argument is not prompted for.

NOTES

perldb is a macro defined in file `hkipipe.emf`.

BUGS

The '**R**' rerun command does not work correctly on Windows platforms, perldb is rerun in a newly created external dos command window instead of inside the MicroEmacs ipipe buffer.

SEE ALSO

gdb(3), ipipe−shell−command(2).

pipe(2m)

NAME

pipe − Incremental Pipe running

SYNOPSIS

pipe Mode

P − mode line letter.

DESCRIPTION

This mode indicates whether an incremental pipe (started by ipipe−shell−command(2)) is running in the current buffer. This mode is automatically set and can not be changed by the user. pipe modes main use is in macros.

Modes lock(2m) and wrap(2m) effect the output behavior of a piped command.

SEE ALSO

ipipe−shell−command(2), lock(2m), wrap(2m).

popup−window(2)

NAME

popup−window − Pop−up a window on the screen

SYNOPSIS

n **popup−window** "*name*"

DESCRIPTION

popup−window manages the display of a new window on the screen. If only one window exists then it will be split else the current window will changed to one of the other existing visible windows. If the given buffer name "*name*" is not null ("") then the buffer is created, if it does not exist, and swapped in.

If an argument *n* of zero is given then the command only succeeds if the given buffer is already being displayed in an existing window, this window is made current. If an non−zero argument is given to the command and the given buffer is not visible then a window displaying a system buffer is chosen in preference. A system buffer is one who's name starts with a `'*'` character, e.g. `"*help*"`. window used to display

SEE ALSO

find−buffer(2).

prefix(2)

NAME

prefix − Key prefix command
prefix2 − Control(2) prefix
prefix3 − Control(3) prefix
prefix4 − Control(4) prefix

SYNOPSIS

n **prefix**

Default prefix bindings:

prefix 1 (**esc**)
prefix 2 (**C−x**)
prefix 3 (**C−h**)
prefix 4 (**C−c**)

DESCRIPTION

prefix sets up to 8 prefix key sequences, allowing two stoke key bindings. The command does not do anything, it is used to create double barrel key bindings such as such as goto−line(2) (**esc g**). This binding may be redefined, redefining ALL meta bindings. If the meta bindings are not required the command should first be unbound using the global−unbind−key(2).

The prefix key can only be defined using the global−bind−key(2), passing the command the prefix number required, for example:

```
1 global-bind-key "prefix" "esc "
2 global-bind-key "prefix" "C-x"
```

Binds the first prefix to the Escape key and the second prefix to Control−x.

The first prefix key (**prefix 1**) differs from the other prefixes since it permits entry of the numeric argument at the message line, e.g. "`esc 1 0 C-f`" will move forward 10 characters.

NOTES

Invocating this command via execute−named−command(2) or by a macro has no effect. It can be bound to only one key sequence which must be a single key stroke such as **C−x** etc. Re−binding the command to another key will not only unbind the new key but also the current **prefix ?** key bindings.

SEE ALSO

global−bind−key(2), global−unbind−key(2).

print−buffer(2)

NAME

print−buffer − Print buffer, with formatting
print−region − Print region, with formatting

SYNOPSIS

n **print−buffer**
n **print−region**

DESCRIPTION

print−buffer and **print−region** print the current buffer or region, respectively, using high−lighting where appropriate. The hilighting assigned to a buffer is defined by the variable $buffer−hilight(5) the print scheme is defined with print−scheme(2), the scheme−editor(3) should be used to create printer schemes.

The printing is typically configured using print−setup(3), which can be found in the main menu under **File−>Printer Setup**.

The numerical argument *n* is generally used for macro development, it changes the default behaviour of these commands as follows:

−2

Configures the printer and, on win32 platforms, opens a Windows printing dialog box enabling the user to configure the printer, font and page layout. The configuration is stored in the "`/print`" registry.

−1

Configures the printer, the configuration is stored in the "`/print`" registry.

0

Configures the printer and, on win32 platforms, using the Windows printer, opens a Windows printing dialog box enabling the user to configure the printer, font and page layout. The required printing is then performed.

1

Configures the printer and performs the required printing. **Printing Process**

When either of these commands are executed the macro file `print.emf` is executed to configure the printer (in a same vain as `me.emf` is executed to configure MicroEmacs for general usage). After the macro file has been executed the "`/print`" registry must contain the information required for printing. Following is a list of registry entries and their use:

flags (*integer*)

The setup flags, defined as a bit mask as follows:−

`0x0f` − Destination of the printer output.

`0x00` − Buffer only.
`0x01` − Internal queue.
`0x02` − To file only.
`0x03` − To file and command line.
`0x10` − Bit set, header enabled.
`0x20` − Bit set, footer
`0x40` − Bit set, enable line numbers.
`0x80` − Bit set, Enable truncated line character (typically \).

paper−x (*integer*)

Paper page width in character cells.

paper−y (*integer*)

Paper page depth in character cells.

page−x (*integer*)

The logical page width in character cells.

page−y (*integer*)

The logical paper depth in character cells.

specifier−x (*integer*)

Windows only.

specifier−y (*integer*)

Windows only.

font−face (*string*)

The name of the font face (Windows only).

rows (*integer*)

Number rows per output page.

cols (*integer*)

Number of columns per output page.

mtop (*integer*)

The size of the top margin in character cells (i.e. where printing may commence).

mbottom (*integer*)

The size of the bottom margin in character cells (i.e. where printing stops).

mleft (*integer*)

The number of characters of space forming the left magin of the physical page.

mright (*integer*)

The number of characters of space forming the right magin of the physical page.

header (*string*)

The ASCII text string for the header line.

footer (*string*)

The ASCII text string for the footer line.

port (*string*)

Printer port identity.

buffer (*string*)

The name of the destination buffer.

file (*string*)

The name of the destination file.

strip (*integer*)

If *integer* value strip spaces from eol.

device (*string*)

The ASCII name of the device (i.e. `/dev/lp`).

eof (*string*)

The printer codes for the end of the file, may be the empty string if not reqired.

eol (*string*)

The printer codes for the end of line character.

eop (*string*)

The printer codes for the end of a page.

sof (*string*)

The printer codes for the start of a file, may be the empty string if not required.

sol (*string*)

The printer codes for the start of a line.

sop (*string*)

The printer codes for the start of a page.

scont (*string*)

The printer codes for a start of row continuation.

econt (*string*)

The printer codes for the end of row continuation.

hsep (*string*)

The horizonal logical page separator character.

vsep (*string*)

The vertical logical page separator character.

wsep (*string*)

The depth in character cells of the vertical logical page separator.

xsep (*string*)

The width in character cells of the logical horizontal separator.

bg−color (*integer*)

The background colour number.

command−line (*string*)

The command line to perform a print operation. **Printing Under Microsoft Windows Environments**

Printing under Microsoft Windows Environments automatically invokes a dialog box to assign and configure the printer page characteristics. The dialog box allows the printer to be selected, enables line numbering, headers and footers.

The dialog allows the user to select the font size, by defining the number of characters that appear on a logical page, and the number of logical pages that appear on a physical page. Selecting the logical and physical page characteristics determine the size of the font. For dense pages with a small typeface then a point size of 6 is appropriate. For clarity, a larger typeface of 10 or 12 points is advised.

NOTES

The last printer configuration selected by the user is held in the registry file "`print.erf`" which is loaded into the */print−history* registry section. This feature is implemented in the macro file `print.emf`.

BUGS

Landscape printing under Microsoft Windows environments is temperamental.

Font selection under Microsoft Windows environments does not always determine the most appropriate font size.

The printer interface does not support native postscript generation. (In progress).

SEE ALSO

print−setup(3), scheme−editor(3), print−scheme(2), hilight(2), printall(3f), $buffer−hilight(5).

print−color(2)

NAME

print−color − Create a new printer color
print−scheme − Create a new printer color and font scheme

SYNOPSIS

n **print−color** "*col−no*" "*red*" "*green*" "*blue*"
n **print−scheme** "*schemeNum*" "*fore*" "*back*" "*font−mask*"

DESCRIPTION

print−color and **print−scheme** are similar to add−color(2) and add−color−scheme(2) except they configure MicroEmacs's printer scheme.

print−color creates a new printer color and inserts it into the printer color table, where *red*, *green* and *blue* are the color components and *col−no* is the printer color index. The printer color table contains 256 entries indexed by *col−no* in the range 0−255. **print−color** may also be used to modify an existing *col−no* index by re−assignment, the existing color definition is over−written with the new color definition.

An argument *n* of 0 to **print−color** resets the printer color table, removing all currently defined colors.

print−scheme creates a new printer scheme. A printer scheme maps the hilight(2) buffer's text into a print scheme. For example key words could be printed in *bold* or in *blue* etc. **print−scheme** arguments comprise an identifying index number "*schemeNum*", two color values, "*fore*" and "*back*" (defined by **print−color**) and a font setting "*font−mask*". The *font−mask* is a bit mask where each bit is defined as follows:

`0x01`	Enable bold font.
`0x02`	Enable italic font.
`0x04`	Enable light font.
`0x08`	Enable reverse font.
`0x10`	Enable underlining.

An argument *n* of 0 to **print−scheme** resets the printer scheme table, removing all currently defined printer schemes.

NOTES

Printer schemes may be created and altered using the scheme−editor(3) dialog, the created printer scheme may then be used directly in the print−setup(3) dialog. Therefore direct use of these commands is largely redundant.

SEE ALSO

scheme−editor(3), print−setup(3), print−buffer(2), hilight(2), $buffer−hilight(5).

print−setup(3)

NAME

print−setup − Configure MicroEmacs's printer interface

SYNOPSIS

print−setup

DESCRIPTION

print−setup provides a dialog interface for configuring MicroEmacs's printing interface. **print−setup** may be invoked from the main *File* menu or directly from the command line using execute−named−command(2).

The **print−setup** dialog is broken down into three pages of configuration options, on all pages the following buttons are available at the bottom of the dialog:−

`Print`

Prints the current buffer using the current configuration.

`Exit`

Quits **print−setup**, changes made to the configuration will be saved.

The following pages appear in the dialog:−

Printer

The **Printer** page is used to configure the type, style and location of the printer, the items on this page are defined as follows:−

`Driver`

Sets the printer type to be used, selecting this item creates a drop down list of available printer drivers. The drivers inform MicroEmacs which fonts and colors are available and how to enable/disable them, these are usually special character sequences. The following special drivers are defined:−

Default Plain Text

This driver does not use any special character sequences so the output it produces is plain text. This should work with most printers, but it does not support any colors or fonts.

HTML

This is a virtual printer driver as no printer uses HTML directly. However the files produced by this driver can be loaded by a web−browser and rendered with full color and font support so provides an efficient way of testing printer schemes. In addition may be used to convert the text rendered in MicroEmacs into HTML content.

Windows

This utilizes MicroEmacs's built−in Windows printer interface (Windows platforms only). When selected MicroEmacs communicates directly to the MS Printer Manager.

`Print Scheme`

Sets the color and font scheme to be used, selecting this item creates a drop down list of available printer schemes − choose one appropriated for your printer. The Default Plain Text scheme does not use any color or fonts so should work for all drivers. see the next item for scheme creation and editing.

`Edit`

Opens the scheme−editor(3) dialog box to edit the currently selected printer scheme, the editor may also be used to create and install new printer schemes.

`Destination`

Specifies the resultant print output, when selected a drop down menu appears containing the following items:

To buffer only

Creates a "`*printer*`" buffer and prints to the buffer.

To file only

Creates a new temporary file and prints to it.

To file & print

Prints to a temporary file and then executes the command−line (see next item) to print the resultant file (option not available when using the Windows printer driver).

Direct to printer

Output is sent directly to the printer, option only available when using the Windows driver.

`Command−line`

Sets the command−line required to print a generated print file (option not available when the Windows driver is selected as printing is done by talking to MS Print Manager directly). The command−line should be a single shell command using "`%f`" whenever the name of the file to be printed is required, e.g. on UNIX systems **lp(1)** or **lpr(1)** can usually be used as follows:−

```
lp −s %f
```

On MS−DOS machines this can usually be achieved by copying the file to the `PRN` device, as follows:

```
copy %f PRN
```

`Page Size`

Displays the currently configured page size in the form:

*Columns***x***Rows Chars−Wide***x***Chars−High*

the field cannot be edited directly, the settings **Page Setup** affect these values.

Page Setup

`Paper Size`

Sets the size of the printer paper, selecting this item will produce a pop down menu listing all available paper sizes unless the Windows printer driver is being used in which case this field cannot be selected and the **Edit** button must be used.

`Character Size`

Sets the size of a character within the page, expressed in terms of the number of characters which will fit on the paper (*width***x***height*). When selected a drop down menu lists all available sizes for the current paper size unless the Windows driver is selected in which case this field cannot be selected and the **Edit** button must be used.

`Edit` (Windows only)

Opens a Windows printer dialog box allowing the user to specify the windows printer, paper size and character size etc.

`No. of Columns` and `Rows`

Sets the number of sub−columns and rows to divide the page into, creating pages within a page.

`Line Numbers`

When enabled, prints the line number at the left hand edge for each line.

`Split Line ID`

When enabled the last right hand text column is reserved for a split identifier. Whenever a line is too long to fit on a single line it is split over two or more lines, if this option is enabled the right edge will be set to the split character (usually a '\' char) to clearly indicate that the line is split.

`Page Size`

As with the **Printer Page Size** it displays the current page size, the field cannot be edited. **Layout**

`Margins`

Configures the top, bottom, left and right margins in characters.

`Header`

Sets whether a header should be printed and if so what it should be, the following special strings can be used:

`%%`

Print a '`%`' character.

`%b`

Print the current buffer's name.

`%D`

Print the current day of the month.

`%f`

Print the current buffer's file name.

`%h`

Print the current hour.

`%M`

Print the current month of the year.

`%m`

Print the current minute of the hour.

`%p`

Print the current page number.

`%s`

Print the current seconds.

`%Y`

Print the current year as a 2 digit number.

`%y`

Print the current year as a 4 digit number.

`Footer`

Sets whether a footer should be printed and if so what it should be, the same special strings can be used as for the header. **NOTES**

user−setup is a macro using osd(2), defined in `printstp.emf`.

The list of available printer drivers and print schemes is stored in the macro file `printers.emf`. Using the **Install** option of the scheme−editor(3) automatically adds the new scheme to the print schemes list. To create a new printer driver a new configuration registry file (`erf` file − see `print*.erf` for examples) must be created and added to the printer driver lists within `printer.emf`.

SEE ALSO

print−buffer(2), scheme−editor(3), osd(2).

printall(3f)

NAME

printall − Formatted print job

SYNOPSIS

me "@printall" <*files*>

DESCRIPTION

The start−up file `printalls.emf` may be invoked from the command line to generate a print job for each file specified on the command line.

Given a list of <*files*>, the files are loaded into the editor, and then printed through MicroEmacs printing formatter. This is an alternative to **cgrind(1)** or some other syntax smart *pretty print* filter.

The operation of this macro assumes that the printer is functioning correctly.

BUGS

As a guess, I would probably bet that this does not work very well on Windows as a dialog is invoked for the print.

SEE ALSO

start−up(3).

python(9)

SYNOPSIS

python − Python Language File.

FILES

hkpython.emf − Python Language file hook definition

EXTENSIONS

.py − Python file

MAGIC STRINGS

^#![\t]*/.*env[\t]+python

MicroEmacs '02 recognizes the magic string on the first line of the file used to locate the executable. The Python files may be extension−less and are still recognized. **DESCRIPTION**

The **python** file type template provides simple hilighting of Python files, the template provides minimal hilighting.

File recognition is performed using the standard file extension **.py**, or by the magic string.

BUGS

There would appear to be too much applied hilighting in this file, it could probably do with rationalizing.

SEE ALSO

Supported File Types

query−replace−all−string(3)

NAME

query−replace−all−string − Query replace string in a list of files

SYNOPSIS

n **query−replace−all−string** "*from*" "*to*" "*files*" ["*grep−from*"]

DESCRIPTION

query−replace−all−string, similar to query−replace−string(2), replaces all occurrences of "*from*" to "*to*" in the given list of files prompting the user before replacing each occurrence.

The command finds all occurrences of "*from*" by calling the command grep(3) to search for string "*from*" in files "*files*". Thus all relevant edited files must be saved or **grep** may return the wrong line numbers. This is achieved by a call to save−some−buffers(2) which prompts the user to save any changed buffers one at a time.

Each occurrence of "*from*" is jumped to using get−next−line(2) and the string is replaced by the call:

```
−1 query−replace−string "from" "to"
```

This query−replaces all occurrences of "*from*" to "*to*" on the current line only, hence the line numbers must be correct. This also means that the "*from*" search string must be correctly formatted for both grep and query−replace−string, unless bit 0x02 is set (see below).

The given argument *n* is a bit based flag which changes the default behavior described above. The bits have the following effect:−

0x01

Prompt before saving any changed buffer, enabled by default. If this bit is not set then any changed buffer is automatically saved before the **grep** is performed.

0x02

If set then a fourth argument "*grep−from*" must also be given. This string is used in place of the "*from*" string for the **grep** only. **NOTES**

query−replace−all−string is a macro defined in `search.emf`.

The **grep** command must be working before this command can function properly.

It is not recommended to use a "from" or "to" string which uses more that one line as the results may be unpredictable.

As the change is likely to be over several files a single call to undo(2) at the end of execution will not undo all the changes made. To undo all the changes made, use get−next−line(2) to loop through all the occurrences and call **undo** for each occurrence

SEE ALSO

query−replace−string(2), save−some−buffers(2), grep(3), get−next−line(2), undo(2), replace−all−string(3), search−forward(2).
Regular Expressions

query−replace−string(2)

NAME

query−replace−string − Search and replace a string − with query

SYNOPSIS

query−replace−string (**esc C−r**)

DESCRIPTION

query−replace−string operates like the replace−string(2) command. replacing one string with another. However, it allows you to step through each string and ask you if you wish to make the replacement. The user is prompted for a replacement response as follows:−

Y

Make the replacement and continue on to the next string.

N

Do not make the replacement, and continue.

!

Replace the rest of the strings without asking.

^G

Stop the command.

.

Go back to place the command started

u

Undo last replacement.

l

Last replacement, do next and stop.

?

Help − get a list of options. **SEE ALSO**

Refer to search−forward(2) for a description of the magic mode search characters.

replace−string(2).
Regular Expressions

quick−exit(2)

NAME

quick−exit − Exit the editor writing changes
save−buffers−exit−emacs − Exit the editor prompt user to write changes

SYNOPSIS

quick−exit (**esc z**)
save−buffers−exit−emacs (**C−x C−c**)

DESCRIPTION

quick−exit writes out all changed buffers to the files they were read from, saves all changed dictionaries, killing any running commands and exits the editor.

save−buffers−exit−emacs operates a **quick−exit** only prompts the user before saving any files.

NOTES

All buffers with a name starting with a '*****' are assumed to be system buffer (i.e. ***scratch***) and are not saved.

SEE ALSO

exit−emacs(2), save−buffer(2).

quiet(2m)

NAME

quiet − Quiet mode

SYNOPSIS

quiet Mode

DESCRIPTION

When **quiet** mode is enabled, visual warnings are given instead of the default audible warning. This mode can only be globally changed, an error will occur if an attempt is made to change the mode for a buffer.

The default state is on, so users of MicroEmacs '02 can relax in the knowledge that they won't annoy other people when things go wrong.

When disabled the system bell is rung when required, otherwise the usual visual warning is the "*[BELL]*" string, printed on the bottom right hand side.

SEE ALSO

global−mode(2).

quote−char(2)

NAME

quote−char − Insert literal character

SYNOPSIS

n **quote−char** "*key*" (**C−q**)

DESCRIPTION

quote−char inserts the next typed character *n* times, default is 1, ignoring the fact that it may be a command character. **quote−char** obeys the current buffer setting of over(2m) mode.

SEE ALSO

insert−string(2), Symbol(3).

RegularExpressions(2)

REGULAR EXPRESSIONS

Regular Expressions are used in the search (and replace) operations. The following notes are applicable when magic(2m) mode is enabled.

Overview

A "*regular expression*" (or "*regex*", or "*pattern*") is a text string that describes some (mathematical) set of strings. A regex **R** "*matches*" a string **S** if **S** is in the set of strings described by **R**.

MicroEmacs '02 includes the GNU **reg**ular **exp**ression pattern matcher library, **regex** which provides a powerful search engine, using the search engine you can:

- see if a string matches a specified pattern as a whole, and
- search within a string for a substring matching a specified pattern.

Some regular expressions match only one string, i.e., the set they describe has only one member. For example, the regular expression '`foo`' matches the string '`foo`' and no others. Other regular expressions match more than one string, i.e., the set they describe has more than one member. For example, the regular expression '`f*`' matches the set of strings made up of any number (including zero) of '`f`'s. As you can see, some characters in regular expressions match themselves (such as '`f`') and some don't (such as '`*`'); the ones that do not match themselves instead let you specify patterns that describe many different strings.

Syntax of Regular Expressions

Regular expressions have a syntax in which a few characters are special constructs and the rest are "*ordinary*". An ordinary character is a simple regular expression which matches that same character and nothing else. The special characters are '`$`', '`^`', '`.`', '`*`', '`+`', '`?`', '`[`', '`]`' and '`\`'. Any other character appearing in a regular expression is ordinary, unless a '`\`' precedes it.

For example, '`f`' is not a special character, so it is ordinary, and therefore '`f`' is a regular expression that matches the string '`f`' and no other string. (It does **not** match the string '`ff`'.) Likewise, '`o`' is a regular expression that matches only '`o`'. (When case distinctions are being ignored, these regexs also match '`F`' and '`O`', but we consider this a generalization of "*the same string*", rather than an exception.)

Any two regular expressions `A` and `B` can be concatenated. The result is a regular expression which matches a string if `A` matches some amount of the beginning of that string and `B` matches the rest of the string.

As a simple example, we can concatenate the regular expressions '`f`' and '`o`' to get the regular expression '`fo`', which matches only the string '`fo`'. Still trivial. To do something nontrivial, you need to use one of the special characters. Here is a list of them.

. (Period)

is a special character that matches any single character except a newline. Using concatenation, we can make regular expressions like '`a.b`', which matches any three−character string that begins with '`a`' and ends with '`b`'.

***** (asterisk)

is not a construct by itself; it is a postfix operator that means to match the preceding regular expression repetitively as many times as possible. Thus, '`o*`' matches any number of '`o`'s (including no '`o`'s).

> '`*`' always applies to the **smallest** possible preceding expression. Thus, '`fo*`' has a repeating '`o`', not a repeating '`fo`'. It matches '`f`', '`fo`', '`foo`', and so on.
>
> The matcher processes a '`*`' construct by matching, immediately, as many repetitions as can be found. Then it continues with the rest of the pattern. If that fails, backtracking occurs, discarding some of the matches of the '`*`'−modified construct in case that makes it possible to match the rest of the pattern. For example, in matching '`ca*ar`' against the string '`caaar`', the '`a*`' first tries to match all three '`a`'s; but the rest of the pattern is '`ar`' and there is only '`r`' left to match, so this try fails. The next alternative is for '`a*`' to match only two '`a`'s. With this choice, the rest of the regex matches successfully.
>
> **+** (plus) is a postfix operator, similar to '`*`' except that it must match the preceding expression at least once. So, for example, '`ca+r`' matches the strings '`car`' and '`caaaar`' but not the string '`cr`', whereas '`ca*r`' matches all three strings.

'**?**' (question mark)

is a postfix operator, similar to '`*`' except that it can match the preceding expression either once or not at all. For example, '`ca?r`' matches '`car`' or '`cr`'; nothing else.

[... **]**

is a "character set", which begins with '`[`' and is terminated by '`]`'. In the simplest case, the characters between the two brackets are what this set can match.

> Thus, '`[ad]`' matches either one '`a`' or one '`d`', and '`[ad]*`' matches any string composed of just '`a`'s and '`d`'s (including the empty string), from which it follows that '`c[ad]*r`' matches '`cr`', '`car`', '`cdr`', '`caddaar`', etc.
>
> You can also include character ranges in a character set, by writing the starting and ending characters with a '`−`' between them. Thus, '`[a−z]`' matches any lower−case ASCII letter. Ranges may be intermixed freely with individual characters, as in '`[a−z$%.]`', which matches any lower−case ASCII letter or '`$`', '`%`' or period.
>
> Note that the usual regex special characters are not special inside a character set. A completely different set of special characters exists inside character sets: '`]`', '`−`' and '`^`'.

To include a '`]`' in a character set, you must make it the first character. For example, '`[]a]`' matches '`]`' or '`a`'. To include a '`−`', write '`−`' as the first or last character of the set, or put it after a range. Thus, '`[]−]`' matches both '`]`' and '`−`'.

To include '`^`' in a set, put it anywhere but at the beginning of the set.

When you use a range in case−insensitive search, you should write both ends of the range in upper case, or both in lower case, or both should be non−letters. The behavior of a mixed−case range such as '`A−z`' is somewhat ill−defined, and it may change in future Emacs versions.

[^ ...]

'`[^`' begins a "*complemented character set*", which matches any character except the ones specified. Thus, '`[^a−z0−9A−Z]`' matches all characters ***except*** letters and digits.

'`^`' is not special in a character set unless it is the first character. The character following the '`^`' is treated as if it were first (in other words, '`−`' and '`]`' are not special there).

A complemented character set can match a newline, unless newline is mentioned as one of the characters not to match. This is in contrast to the handling of regexs in programs such as **grep(1)**.

^ (caret)

is a special character that matches the empty string, but only at the beginning of a line in the text being matched. Otherwise it fails to match anything. Thus, '`^foo`' matches a '`foo`' that occurs at the beginning of a line.

$ (dollar)

is similar to '`^`' but matches only at the end of a line. Thus, '`x+$`' matches a string of one '`x`' or more at the end of a line.

\ (backslash)

has two functions: it quotes the special characters (including '`\`'), and it introduces additional special constructs.

Because '`\`' quotes special characters, '`\$`' is a regular expression that matches only '`$`', and '`\[`' is a regular expression that matches only '`[`', and so on.

Note: for historical compatibility, special characters are treated as ordinary ones if they are in contexts where their special meanings make no sense. For example, '`*foo`' treats '`*`' as ordinary since there is no preceding expression on which the '`*`' can act. It is poor practice to depend on this behavior; it is better to quote the special character anyway, regardless of where it appears.

For the most part, '\' followed by any character matches only that character. However, there are several exceptions: two−character sequences starting with '\' that have special meanings. The second character in the sequence is always an ordinary character when used on its own. Here is a table of '\' constructs.

\| (bar)

specifies an alternative. Two regular expressions `A` and `B` with '`\|`' in between form an expression that matches some text if either `A` matches it or `B` matches it. It works by trying to match `A`, and if that fails, by trying to match `B`.

Thus, '`foo\|bar`' matches either 'foo' or 'bar' but no other string.

'`\|`' applies to the largest possible surrounding expressions. Only a surrounding '`\( ... \)`' grouping can limit the grouping power of '`\|`'.

Full backtracking capability exists to handle multiple uses of '`\|`'.

\(... **\)**

is a grouping construct that serves three purposes:

- To enclose a set of '`\|`' alternatives for other operations. Thus, '`\(foo\|bar\)x`' matches either '`foox`' or '`barx`'.
- To enclose a complicated expression for the postfix operators '`*`', '`+`' and '`?`' to operate on. Thus, '`ba\(na\)*`' matches '`bananana`', etc., with any (zero or more) number of '`na`' strings.
- To record a matched substring for future reference. This last application is not a consequence of the idea of a parenthetical grouping; it is a separate feature that is assigned as a second meaning to the same '`\( ... \)`' construct. In practice there is no conflict between the two meanings.

'**\D**'

matches the same text that matched the Dth occurrence of a `` `\( `` ... `\)`' construct.

After the end of a '`\( ... \)`' construct, the matcher remembers the beginning and end of the text matched by that construct. Then, later on in the regular expression, you can use '\' followed by the digit `D` to mean "match the same text matched the `D`th time by the '`\(` ... `\)`' construct."

The strings matching the first nine '`\(` ... `\)`' constructs appearing in a regular expression are assigned numbers 1 through 9 in the order that the open−parentheses appear in the regular expression. So you can use '`\1`' through '`\9`' to refer to the text matched by the corresponding '`\(` ... `\)`' constructs.

For example, '`\(.*\)\1`' matches any newline−free string that is composed of two identical halves. The '`\(.*\)`' matches the first half, which may be anything, but the

'`\1`' that follows must match the same exact text.

If a particular '`\( ... \)`' construct matches more than once (which can easily happen if it is followed by '`*`'), only the last match is recorded.

`` \` ``

matches the empty string, but only at the beginning of the buffer or string being matched against.

NOTE: This currently only matches the start of the current line − it does not match the start of the buffer.

`\'`

matches the empty string, but only at the end of the buffer or string being matched against.

NOTE: This currently only matches the end of the current line − it does not match the end of the buffer.

`\=`

matches the empty string, but only at point.

`\b`

matches the empty string, but only at the beginning or end of a word. Thus, '`\bfoo\b`' matches any occurrence of '`foo`' as a separate word. '`\bballs?\b`' matches '`ball`' or '`balls`' as a separate word.

'`\b`' matches at the beginning or end of the buffer regardless of what text appears next to it.

`\B` matches the empty string, but *not* at the beginning or end of a word.

`\<`

matches the empty string, but only at the beginning of a word. '`\<`' matches at the beginning of the buffer only if a word−constituent character follows.

`\>`

matches the empty string, but only at the end of a word. '`\>`' matches at the end of the buffer only if the contents end with a word−constituent character.

`\w`

matches any word−constituent character. The syntax table determines which characters these are.

`\W`

matches any character that is not a word−constituent.

`\sC`

matches any character whose syntax is `C`. Here `C` is a character that represents a syntax code: thus, '`w`' for word constituent, '`−`' for whitespace, '`(`' for open parenthesis, etc. Represent a character of whitespace (which can be a newline) by either '`−`' or a space character.

`\SC`

matches any character whose syntax is not `C`.

`\{N,M\}`

Matches an integer number of the previous item, where `N` and `M` are integer constants interpreted as follows:−

`\{N\}`

The preceeding item is matched exactly `N` times.

`\{N,\}`

The preceeding item is matched `N` or more times.

`\{N,M\}`

The preceeding item is matched at least `N` times, but no more than `M` times.

`\{,M\}`

The preceeding item is optional and is matched at most `M` times.

The constructs that pertain to words and syntax are controlled by the setting of the syntax table.

Syntax of Replacement Expressions

A regular expression replacement, query−replace−string(2) command (with magic(2m) mode enabled), replaces exact matches for a single string or pattern. The replacement pattern may be a constant; it may also refer to all or part of what is matched by the regular expression search string.

\&

In the replacement pattern, **\&** stands for the entire match being replaced. (as does `\0`).

\D

In the replacement pattern, where **D** is a digit 1−9, stands for whatever matched the Dth parenthesized grouping (`\( .. \)`) in search pattern. To include a '\' in the text to replace with, you must enter '`\\`'. For example,

```
M−x query−replace−string<RET> c[ad]+r <RET> \&−safe <RET>
```

replaces (for example) "`cadr`" with "`cadr−safe`" and "`cddr`" with "`cddr−safe`".

```
M−x query−replace−string<RET> \(c[ad]+r\)−safe <RET> \1 <RET>
```

performs the inverse transformation.

\0 is a special case, this represents the whole of the search pattern, it is equivalent to **\&**.

Searching and Case

Searching may be either case sensitive or case insensitive, and is controlled by the exact(2m) mode. When *exact* mode is enabled (default) the then searches are case sensitive; disabled then case is ignored. The exact(2m) mode is set on a per−buffer basis.

NOTES

The search engine searches for the longest string that matches a given pattern, the longest pattern is sometimes the pattern that is not actually required. For instance, consider searching for an HTML bracket set. The simplest search is:−

```
M−x search−forward "<.*>"
```

Unfortunately, this pattern is not specific enough, given an HTML line:−

```
<a href="www.jasspa.com">Jasspa Site</a>
```

Then the pattern matched is actually the whole line as the `.*` matches everything to the last `>`, this is the longest string. To rectify the pattern then we must be more specific, the correct search pattern to use in this instance is:−

```
M−x search−forward "<[^>]*>"
```

In this case we match any character excluding the closing character, this guarantees that we always find the shortest string match. A search of our HTML line locates two separate instances of the regular expression `<a href="www.jasspa.com">` and `</a>`.

SEE ALSO

search−forward(2), search−backward(2), buffer−mode(2), exact(2m), hunt−backward(2), hunt−forward(2), isearch−forward(2), magic(2m), replace−string(2).

rbin(2m)

NAME

rbin − Reduced binary editor mode

SYNOPSIS

rbin Mode

r − mode line letter.

DESCRIPTION

rbin mode is enabled when a file is edited in reduced binary mode. The mode is automatically enabled when a file is loaded as a binary data file via find−file(2).

When a file is loaded using **rbin** mode, every 256 bytes is converted into a line of text, the line is a single list of hex numbers 512 characters long, 2 bytes for each character. This format is not very user friend unlike binary(2m) mode, but is much more memory efficient (requiring approximately 2 times more memory than the file size).

When writing out a file which has rbin mode enabled the format of each line must be correct, namely an even number of hex numbers with no other characters.

EXAMPLE

Given a single line MSDOS file:−

```
Live long and prosper.
```

When loaded in using **binary** mode the following 2 line buffer will be produced:−

```
4C697665206C6F6E6720616E642070726F737065722E0D0A1A
```

Note the "`0D 0A 1A`" at the end, this is due to MSDOS's "`\n\r`" carriage returns and `^Z` file termination. The given file could be made UNIX compatible by editing the buffer to:−

```
4C697665206C6F6E6720616E642070726F737065722E0D
```

NOTES

rbin and **binary** modes are mutually exclusive, i.e. they cannot both be enabled at the same time.

SEE ALSO

find−file(2), binary(2m).

rcs−file(2)

NAME

rcs−file − Handle Revision Control System (RCS) files

SYNOPSIS

n **rcs−file** (**C−x C−q**)

DESCRIPTION

MicroEmacs '02 RCS support command. The action of this command depends on the current buffer view(2m) mode state, the argument *n*, and the existence of an RCS file.

view−mode ON; RCS file does not exist

Removes buffer view mode to enable the user to edit the file.

view−mode ON; RCS file exists

MicroEmacs attempts to check out the file using the command line given by the variable $rcs−cou−com(5) (co unlock). The file is then reloaded and the view mode status re−evaluated.

view−mode OFF; RCS file does not exist

MicroEmacs attempts to check−in the file into RCS for the first time using the command−line given by the variable $rcs−cif−com(5) (ci first). The file is then reload.

view−mode OFF; RCS file exists

MicroEmacs attempts to check−in the file into RCS using the command−line given by the variable $rcs−ci−com(5). The file is then reload.

−ve argument given

MicroEmacs attempts to unedit any changes made to the file using the command−line given by the variable $rcs−ue−com(5). The file is then reload. **SEE ALSO**

rcs(1). $rcs−file(5), buffer−mode(2), find−file(2), view(2m).

read−file(2)

NAME

read−file − Find and load file replacing current buffer

SYNOPSIS

n **read−file** "*file−name*" (**C−x C−r**)

DESCRIPTION

read−file operates like find−file(2), this command either finds the file in a buffer, or creates a new buffer and reads the file in. The command destroys the current buffer before the new buffer is created making this command ideal to use when the wrong file was entered on a find−file(2). This command is also useful for re−loading files that have changed on disk.

The numeric argument *n* can be used to modify the default behaviour of the command, where the bits are defined as follows:

0x01

If the file does not exist and this bit is not set the command fails at this point. If the file does not exist and this bit is set (or no argument is specified as the default argument is `1`) then a new empty buffer is created with the given file name, saving the buffer subsequently creates a new file.

0x02

If this bit is set the file will be loaded with binary(2m) mode enabled. See help on **binary** mode for more information on editing binary data files.

0x04

If this bit is set the file will be loaded with crypt(2m) mode enabled. See help on **crypt** mode for more information on editing encrypted files.

0x08

If this bit is set the file will be loaded with rbin(2m) mode enabled. See help on **rbin** mode for more information on efficient editing of binary data files. **SEE ALSO**

reread−file(3), find−file(2), view−file(2), binary(2m), crypt(2m), rbin(2m).

read−history(2)

NAME

read−history − Read in session history information

SYNOPSIS

n **read−history** ["*hist−file*"]

DESCRIPTION

read−history reads in a MicroEmacs '02 history file, setting the current history information. If argument **n** is not given then the given "*hist−file*" is simply read in. If a non−zero argument is specified then default history is set to the given file−name and the file is read. If an argument of zero is given then the default history is re−read. Information read in (and saved) from the history file includes:−

- Searching and replacing history.
- Buffer name history.
- Command name history.
- File name history.
- General (all the rest) history.
- Buffer and file list with line numbers.

MicroEmacs '02's environment may be retained almost intact by the use of the default history and using the **−c** (continue) command−line option to re−load all files that were being edited in a previous session.

NOTES

When running multiple MicroEmacs '02 sessions on the same work−station (or different workstations sharing the same home directory), the default history is saved when MicroEmacs '02 exits. As a result the last MicroEmacs '02 sessions that terminates writes the history information used next time.

The history information is saved in a registry format file (see erf(8)). Reference should be made to the notes included in erf(8) as to how the history file may be edited and effected in the same MicroEmacs '02 session.

SEE ALSO

erf(8), save−history(2).

read−registry(2)

NAME

read−registry − Read in a registry definition file

SYNOPSIS

read−registry "*root*" "*file*" "*mode*"

DESCRIPTION

read−registry loads a registry file erf(8) into the internal registry memory, where the information may be queried via the registry macro commands. The arguments are defined as follows:−

root

The root node in the registry to into which the registry contents are attached. The root name is limited to 32 characters in length and is specified without a leading forward slash '`/`'. The node *root* is created at the root of the registry.

file

The name of the registry file erf(8) to load. This may be an absolute, relative or `$MEPATH` specified file; typically it is located on `$MEPATH`.

mode

The *mode* is string specifying the registry node loading and saving modes, each mode is represented by a character. Lower case characters add a mode, upper case characters delete a mode. The modes are defined as follows:−

a − Autosave

Automatically saves the registry when it is deleted or unloaded from the registry. The user is not prompted for a save.

b − Backup

Automatically performs a backup of the registry file whenever a save operation is performed.

c − Create

If the registry file cannot be loaded then the *root* node is created and the invocation succeeds. If this mode is omitted then the call fails if the *file* cannot be found.

d − Discard

Marks the registry as discardable. This is typically used for registries that are not saved.

r − Reload

If the registry node already exists then it is deleted and reloaded, see also the merge flag (**m**). By default, when both the **r** and **m** flags are omitted and the registry node already exists the read operation is not performed and the existing node is used.

m − Merge

The registry file is merged with the contents of any existing registry node. (i.e. the existing registry tree nodes are not deleted if they already exist). See also the reload flag (**r**).

h − Hidden

The registry node is created in the *Hidden* state. (i.e. children will not be shown in list−registry(2) output).

u − Updated

Marks the registry as modified. The modified bit is removed when the registry file is saved. If the modified bit is applied to a registry node the user will be prompted to save the registry when it is deleted (or it will be automatically saved when the *Autosave* mode is used).

Multiple modes may be applied.

EXAMPLE

The following example is a typical call made from a macro using a registry file where the user may edit the registry file. In this case this a reload of the registry is forced to ensure that the most up−to−date contents are retrieved. Note that the name of the registry file is actually retrieved from the *history* registry.

```
set-variable #l1 &reg "/history" "address" $MENAME
!if &seq &set #l0 &find #l1 ".ab" "ERROR"
    set-variable #l0 &reg "/" "history" ""
    set-variable #l0 &spr "%s%s.ab" &lef #l0 &rsin "/" #l0 #l1
!endif
read-registry "AddressBook" #l0 "rc"
```

BUGS

At exit only registry nodes attached to the root are saved.

SEE ALSO

save−registry(2), list−registry(2), mark−registry(2), erf(8).

recenter(2)

NAME

recenter − Recenter the window (refresh the screen)

SYNOPSIS

n **recenter** (**C−l**)

DESCRIPTION

recenter scrolls the current window so that the cursor position is at the center of the window and redraws the whole screen. If *n* is given then scrolls the window so that the cursor is *n* lines from the top if *n* is positive or from the bottom if negative.

recenter is typically used to refresh the screen if it is out of date (i.e. needs to be redrawn).

SEE ALSO

screen−update(2).

regex−forward(3)

NAME

regex−forward − Search for a magic string in the forward direction
regex−backward − Search for a magic string in the backward direction

SYNOPSIS

n **regex−forward** "*string*"
n **regex−backward** "*string*"

DESCRIPTION

regex−forward searches for a regular expression string from the current cursor position to the end of the file. A case insensitive regular expression search is performed regardless of the magic(2m) and exact(2m) mode settings.

The numeric argument *n* is interpreted as follows:−

n > **0**

The *n*th occurrence of the *string* is located.

n < **0**

The first occurrence of the *string* is located in the next *n* lines.

regex−backward searches backwards in the file. In all other ways it is like **regex−forward**.

DIAGNOSTICS

The command returns a status of `FALSE` if the *string* could not be located (or *n*th *string* where *n* occurrences are requested). If the *string* is found within the given search criteria the return status is `TRUE`.

NOTES

The **regex−forward** and **regex−backward** commands are not publically available from the command line, but may be used within macros to perform regular expression searches regardless of the user mode settings.

These commands are implemented as macros in `utils.emf`.

SEE ALSO

buffer−mode(2), exact(2m), isearch−forward(2), magic(2m), replace−string(2), search−backward(2), search−forward(2).
Regular Expressions

replace−all−pairs(3)

NAME

replace−all−pairs − Replace string pairs in a list of files

SYNOPSIS

n **replace−all−pairs** "*files*"

DESCRIPTION

replace−all−pairs uses the current buffer to extract "*from*" and "*to*" pairs and then replaces all occurrences of "*from*" to "*to*" in the given list of files without prompting the user. An optional third argument "*grep*" can be given which will be used as the grep string, if not given the "*from*" string is used. The format of the current buffer must be:

```
/from1/to1/
Xfrom2Xto2X
?from3?to3?
/from4/to4/grep4/
  .
  .
/fromN/toN/
```

For each pair the command finds all occurrences of "*from*" (or "*grep*" if specified) by calling the command grep(3) to search for string "*from*" in files "*files*". Thus all relevant edited files must be saved or **grep** may return the wrong line numbers. This is achieved by a call to save−some−buffers(2) between each replace pair, it is called with an argument of 0 to ensure that any changed buffers are automatically saved.

Each occurrence of "*from*" is jumped to using get−next−line(2) and the string is replaced by the call:

```
−1 replace−string "from" "to"
```

This replaces all occurrences of "*from*" to "*to*" on the current line only, hence the line numbers must be correct. This also means that the "*from*" search string must be correctly formatted for both grep and replace−string.

The given argument *n* is a bit based flag which changes the default behavior described above. The bits have the following effect:−

0x01

Prompt before saving any changed buffers FIRST time ONLY, enabled by default. If set then the user is also prompted to continue before any changes are made. If this bit is not set then the command executes without any user input. **NOTES**

replace−all−pairs is a macro defined in `search.emf`.

The **grep** command must be working before this command can function properly.

It is not recommended to use a "from" or "to" string which uses more that one line as the results may be unpredictable.

As the change is likely to be several pair strings with each changed buffer being saved between pairs undo(2) cannot be used to undo the changes. Neither can the backups be relied on as a buffer may be saved more than once in this process, therefore it is strongly recommend that a backup of the files is made before commencing with this command.

SEE ALSO

replace−all−string(3), replace−string(2), save−some−buffers(2), grep(3), get−next−line(2), undo(2), query−replace−all−string(3), search−forward(2).
Regular Expressions

replace−all−string(3)

NAME

replace−all−string − Replace string with new string in a list of files

SYNOPSIS

n **replace−all−string** "*from*" "*to*" "*files*" ["*grep−from*"]

DESCRIPTION

replace−all−string, similar to replace−string(2), replaces all occurrences of "*from*" to "*to*" in the given list of files without prompting the user.

The command finds all occurrences of "*from*" by calling the command grep(3) to search for string "*from*" in files "*files*". Thus all relevant edited files must be saved or **grep** may return the wrong line numbers. This is achieved by a call to save−some−buffers(2) which prompts the user to save any changed buffers one at a time.

Each occurrence of "*from*" is jumped to using get−next−line(2) and the string is replaced by the call:

```
−1 replace-string "from" "to"
```

This replaces all occurrences of "*from*" to "*to*" on the current line only, hence the line numbers must be correct. This also means that the "*from*" search string must be correctly formatted for both grep and replace−string, unless bit 0x02 is set (see below).

The given argument *n* is a bit based flag which changes the default behavior described above. The bits have the following effect:−

0x01

Prompt before saving any changed buffer, enabled by default. If this bit is not set then any changed buffer is automatically saved before the **grep** is performed.

0x02

If set then a fourth argument "*grep−from*" must also be given. This string is used in place of the "*from*" string for the **grep** only. **NOTES**

replace−all−string is a macro defined in `search.emf`.

The **grep** command must be working before this command can function properly.

It is not recommended to use a "from" or "to" string which uses more that one line as the results may be unpredictable.

As the change is likely to be over several files a single call to undo(2) at the end of execution will not undo all the changes made. To undo all the changes made, use get−next−line(2) to loop through all the occurrences and call **undo** for each occurrence

SEE ALSO

replace−string(2), save−some−buffers(2), grep(3), get−next−line(2), undo(2), query−replace−all−string(3), replace−all−pairs(3), search−forward(2).

replace−string(2)

NAME

replace−string − Replace string with new string

SYNOPSIS

n **replace−string** (**esc r**)

DESCRIPTION

replace−string replaces all occurrences of one string with another string. The replacement starts at the current location of the cursor and goes to the end of the current buffer.

A numeric argument positive *n* limits the number of strings replaced to *n*. A negative argument *n* limits the number of lines in which the replacement may take place, e.g. a value of −15 restricts the replacement of the string to the next 15 lines from the current cursor position.

SEE ALSO

See Operating Modes for a description of the magic(2m) and exact(2m) modes which change the search space.

buffer−mode(2), query−replace−string(2), search−forward(2).
Regular Expressions

reread−file(3)

NAME

reread−file − Reload the current buffer's file

SYNOPSIS

reread−file

DESCRIPTION

reread−file reloads from disk the file associated with the current buffer, this command is particularly useful when the file is continually updated by an external program. If the buffer has been edited and its name does not start with a `'*'` then the user is prompted as to whether the changes should be discarded. Also if the buffer has an active process running in it then confirmation is sort from the user before the process is killed.

NOTES

reread−file is a macro implemented in `tool.emf`.

SEE ALSO

find−file(2), read−file(2), view−file(2).

resize−all−windows(2)

NAME

resize−all−windows − Automatically resize the windows

SYNOPSIS

n **resize−all−windows**

DESCRIPTION

resize−all−windows performs an automatic layout of the windows on the screen, reorganizing the windows such that each window has an equal amount of space. The argument *n* determines which axes reorganization is performed in.

- A +ve argument reorganizes the windows vertically, leaving the horizontal arrangement as is.
- A −ve argument rearranges the windows horizontally, leaving the vertical arrangement as is.
- An argument of zero performs no vertical or horizontal arrangement.
- No argument re−arranges both the vertical and horizontal window layout.

SEE ALSO

resize−window−vertically(2), resize−window−horizontally(2), split−window−vertically(2).

restyle−buffer(3)

NAME

restyle−buffer − Automatically reformat a buffer's indentation.
restyle−region − Automatically reformat a regions indentation.

SYNOPSIS

restyle−buffer
restyle−region

DESCRIPTION

restyle−buffer automatically re−formats the indentation of a buffer. The indentation only operates if the indentation method is defined with cmode(2m) or $buffer−indent(5), otherwise the command has no effect.

restyle−region modifies the indentation between *point* and *mark*.

NOTES

restyle−buffer and **restyle−region** are macros defined in `format.emf`.

SEE ALSO

cmode(2m), indent(2), $buffer−indent(5).

reyank(2)

NAME

reyank − Restore next yank buffer

SYNOPSIS

n **reyank** (**esc y**)

DESCRIPTION

Every region killed goes onto a stack, with the most recent at the top. Immediately after yanking text out into the current buffer using yank(2), the user may **reyank** which deletes the region just yanked and replaces it with *n* insertions of the next region on the kill stack. Another call to reyank deletes that region and replaces it with the next in the stack etc.

The last 15 kills are stored.

SEE ALSO

copy−region(2), kill−region(2), set−mark(2), yank(2).

rul(9)

SYNOPSIS

rul − Install Shield Rules

FILES

hkrul.emf − Install Shield hook definition
rul.etf − Install Shield template file.

EXTENSIONS

.rul − Install Shield Rules file

DESCRIPTION

The **rul** file type template provides simple hilighting of Install Shield Rules files.

Hilighting

The hilighting features allow commands, variables, logical, comments, strings and characters of the language to be differentiated and rendered in different colors.

Auto Layout

The indentation mechanism is enabled which performs automatic layout of the text. restyle−region(3) and restyle−buffer(3) are available to reformat (re−layout) selected sections of the buffer, or the whole buffer, respectively.

Folding and Information Hiding

Generic folding is enabled within the rul file. The folds occur about the keywords **function**...**end** located on the left−hand margin. fold−all(3) (un)folds all regions in the file, fold−current(3) (un)folds the current region.

Short Cuts

The short cut keys used within the buffer are:−

C−c C−c − Comment out the current line.
C−c C−d − Uncomment the current line.
C−c C−e − Comment to the end of the line with stars (*).
f2 − (un)fold the current region

f3 − (un)fold all regions

BUGS

None reported.

SEE ALSO

fold−current(3), fold−all(3), indent(2), restyle−region(3) restyle−buffer(3)

Supported File Types

save(2m)

NAME

save − Flag buffer to be saved

SYNOPSIS

save Mode

S − mode line letter.

DESCRIPTION

This mode cannot be set globally and can only be set on a buffer which needs saving. The mode is used to flag that the buffer is to be saved, the state of the mode is displayed in the output of list−buffers(2). If the second column is an 'S' the mode is set, otherwise it is not. Only the execute command in list−buffers(2) (bound to 'x') uses this flag to actually save the buffer and the flag is automatically removed as soon as the buffer is saved.

SEE ALSO

list−buffers(2), del(2m).

save−all(3)

NAME

save−all − Save all modified files (with query)

SYNOPSIS

n **save−all**

DESCRIPTION

save−all cycles through all buffers, dictionaries and registry files writing back any changes made. For each buffer, dictionary or registry file which has been modified the user is prompted before the changes are saved, a value of **y** initiates the save, **n** skips the save.

The argument *n* can be used to change the default behavior of save−all described above, *n* is a bit based flag where:−

0x01

Enables the user prompt before the file is saved (default). If this flag is not supplied then all modified files will automatically be written. **NOTES**

save−all is a macro defined in `me.emf`, using commands save−some−buffers(2), save−dictionary(2) and save−registry(2).

SEE ALSO

save−some−buffers(2), save−dictionary(2), save−registry(2).

save−buffer(2)

NAME

save−buffer − Save contents of changed buffer to file

SYNOPSIS

n **save−buffer** (**C−x C−s**)

DESCRIPTION

save−buffer saves the contents of the current buffer if the contents have been changed, writing the buffer back to the file it was read from.

On saving the file, if time(2m) mode is enabled then the time stamp string is searched for in the file and modified if located, to reflect the modification date and time.

If backup(2m) mode is enabled then a backup copy of the file existing is created and the contents of the buffer are written to the file. Any automatic save copies of the file are deleted.

If the buffer contains a narrow(2m) it will automatically be removed before saving so that the whole buffer is saved and restored when saving is complete

If auto(2m) mode is enabled the the file is written out in the style indicated by modes crlf(2m) and ctrlz(2m). Otherwise the file is written out in the style on the current platform.

The argument *n* can be used to change the default behavior of save−buffer described above, *n* is a bit based flag where:−

0x01

Enables validity checks (default). These include check that the buffer has been modified, if not an error occurs. Also the time stamp of the file to be written is checked, if the file systems file exists and is newer the confirmation of writing is requested from the user. If this flag is not supplied then the buffer is written whenever possible and without any prompts to the user.

0x02

Disables the expansion of any narrows (see narrow−buffer(2)) before saving the buffer. **NOTES**

- undo(2) information is discarded when the file is saved.
- Refer to $auto−time(5) for a description of the file extensions used by MicroEmacs '02 for backup and temporary files.

- Buffers may also be saved via the list−buffers(2) command.

SEE ALSO

$auto−time(5), $timestamp(5), buffer−mode(2), find−file(2), narrow−buffer(2), save−some−buffers(2), undo(2), backup(2m), time(2m), undo(2m), narrow(2m), auto(2m), crlf(2m), ctrlz(2m), write−buffer(2), append−buffer(2).

save−dictionary(2)

NAME

save−dictionary − Save changed spelling dictionaries

SYNOPSIS

n **save−dictionary** ["*dictionary*"]

DESCRIPTION

save−dictionary may be used to save one, or all changed, dictionaries back to disk. By default **save−dictionary** prompts for a single dictionary, which is then saved. If the dictionary to be saved has been created within the session (rather than read from disk) the user is always prompted to save and enter a full dictionary file name (pathname) to save to. If the dictionary was not created then the user is only prompted to save if,

- a non−zero argument is supplied
- and the users history registry node "*/history/spell/autosave*" does not exist or its value is zero.

Otherwise the dictionary is automatically saved.

The argument *n* may be used to control the effect of the command, *n* is a bit based flag defined as follows:−

0x01

Enables prompting before saving, only used when saving all dictionaries.

0x02

Save all changed dictionaries. **NOTES**

This command is called to save all dictionary changes whenever MicroEmacs is exited.

The dictionary auto−save registry value can be changed via the user−setup(3) dialog.

SEE ALSO

add−dictionary(2), delete−dictionary(2), spell(2).

save−history(2)

NAME

save−history − Write history information to history file

SYNOPSIS

n **save−history** "*hist−file*"

DESCRIPTION

save−history writes out MicroEmacs '02's current history information into the given history file.

The command read−history(2) can set a default history file in which case the history is automatically written out to this file if an argument of zero is given; the user is not prompted for a file. MicroEmacs '02 automatically tries to write the default history whenever it is exited.

NOTES

The history information is saved in a registry format file (see erf(8)). Reference should be made to the notes included in erf(8) as to how the history file may be edited and effected in the same MicroEmacs '02 session.

SEE ALSO

erf(8), read−history(2).

save−registry(2)

NAME

save−registry − Write a registry definition file

SYNOPSIS

n **save−registry** ["*root*" "*file*"]

DESCRIPTION

save−registry saves a registry tree, defined by *root*, to a registry file *file* in the erf(8) format. By default the user is prompted for the registry *root* to save, which must already exist. If the *file* given is the empty string **""**, the registry node *root* must be a root node with an associated file name stored, this file name is used.

The argument *n* may be used to control the effect of the command, *n* is a bit based flag defined as follows:−

0x01

Enables prompting before saving, only used when saving all registries.

0x02

Save all changed registries except the history node which should be saved using the command save−history(2). **NOTES**

This command is called to save all registry changes whenever MicroEmacs is exited.

SEE ALSO

read−registry(2), save−history(2), erf(8).

save−some−buffers(2)

NAME

save−some−buffers − Save contents of all changed buffers to file (with query)

SYNOPSIS

n **save−some−buffers**

DESCRIPTION

save−some−buffers cycles through all visible buffers (buffers without mode hide(2m) set) and attempts to save all modified ones, writing the contents back to the file from where it was read. For each buffer that has been modified the user is prompted to save the buffer, a value of **y** initiates a save for the buffer, **n** skips the buffer.

The argument *n* can be used to change the default behavior of save−some−buffers described above, *n* is a bit based flag where:−

0x01

Enables the user prompt before the buffer is saved (default). If this flag is not supplied then all modified visible buffers will be written. **SEE ALSO**

save−buffer(2), save−buffers−exit−emacs(2), write−buffer(2), hide(2m).

scheme(9)

SYNOPSIS

scheme − Scheme File.

FILES

hkscheme.emf − Scheme file hook definition

EXTENSIONS

.scm, **.sch** − Scheme file

DESCRIPTION

The **scheme** file type template provides simple hilighting of Scheme files, the template provides minimal hilighting.

File recognition is performed using the standard file extensions **.scm** or **.sch**.

NOTES

JASSPA have no idea as to the state of this file hook definition.

SEE ALSO

Supported File Types

scheme−editor(3)

NAME

scheme−editor − Color Scheme Editor

SYNOPSIS

scheme−editor

DESCRIPTION

scheme−editor is a color and font scheme editor that provides a dialog interface to configure the display schemes used by the editor. The schemes may be created or modified within the scheme editor and then committed to the configuration files for general use.

The editor can be used to create both screen and printer color/font schemes, they are typically stored in the `macros` directory and are executed as macro files at start up or when printing. The standard screen schemes are called `scheme`*X*`.emf` and printer ones `print`*X*`.emf`.

The **scheme−editor** is displayed within a single dialog box, tab selections at the top of the dialog box enable **color** and **scheme** creation and/or modification. Navigation is typically performed using the mouse, where the mouse is absent then the `TAB` key may be used to move between the fields. The information presented is defined as follows:−

File Name

The name of the color scheme to be modified. This is the name of the **scheme*X*.emf** file, omitting the file extension. See the **FILES** section below for a list of standard screen and printer scheme supplied with MicroEmacs '02.

Type

Defines whether the scheme is a screen or printer type.

Description

An ASCII description of the color scheme, used to identify the color scheme.

Buffer Hilight

Available when scheme is a screen type. Defines whether buffer hilighting should be enabled, when *Completely Disable* all buffers are displayed character for character in the standard text scheme, this will ensure maximum update performance but some file formats such as the on−line help will become unreadable so this option is really selected. Similarly *Reformat Only* disables the majority of buffers, hilighting is only enabled when the file would be unreadable without it, such as the on−line help or man page files. The default *Fully Enabled* setting enables all buffer hilighting.

Print Option

Available when scheme is a printer type. Defines what components of a scheme is to be used when printing. **Colors**

The **colors** tab allows the basic palette colors of the editor to be created and modified. The left−hand side of the dialog contains a scrolling window containing the existing color entries. The right−hand side of the dialog provides the controls to add and change the color assignment. The controls operate on the currently selected palette entry.

Add

Creates and adds a new color entry into the palette. The new palette entry is created with a default color that may be subsequently modified.

Change

Commits the current selection color to the palette.

Red/**Green**/**Blue**

The color entries allow the currently selected palette color entry to be modified. The color values may be changed by direct numeric entry (0..255) or via the ^/v controls; the color is committed to the palette using the **Add** or **Change** button. **Schemes**

The **schemes** tab allows the schemes to be edited. The left−hand side of the dialog contains a scrolling window of the available color palette (created from the **Colors** tab). The right−hand side of the window shows the variants of the scheme.

Selection

The **selection** item provides a pull−down menu containing gross scheme categories used by the editor.

Scheme

A pull−down menu containing the schemes of the selection, modifying this entry shows the variants of the scheme in the **Normal**, **Current**, **Select** and **Sel−Cur** dialogs.

There are 4 variants, or styles, for a single scheme; each style is comprised of a foreground and background color, and a row of toggle button to enable/disable fonts, defined as follows.

`B` − Bold.
`I` − Italic.
`L` − Light (typically not supported).

`R` − Reverse video (fore/back−ground swapped).
`U` − Underline.
`V` − Toggle reverse video when inverted.

The last mode **V** needs a little more explanation; commands such as screen−poke(2) are able to invert the color scheme, i.e. use the fore color for the background etc. Enabling this mode will toggle the reverse video mode (**R**) when this feature is used.

The style displayed by a particular scheme depends upon the selection/current status of the text:

Normal

The normal style, when the text object is not selected or current (i.e. out of focus).

Current

The style used when the text object is current (i.e. in focus)

Select

The style used when the text object is selected (i.e. by the mouse) and is not current.

Sel−Cur

The style used when the text object is selected and is current.

Note that a printer scheme only uses the Normal style.

Setting of the **selection** and **scheme** shows the current scheme in the **Normal**, **Current**, **Select** and **Sel−Cur** dialogs. New colors are assigned by selecting a color in the palette area and making it current. The current color is applied by selecting the **Fore** / **Back** boxes of the scheme dialog. The assigned color is displayed in the text box *The big brown fox...*.

Controls

The controls at the bottom of the dialog apply the edits to the configuration files.

Current

Makes the changes to the palette and schemes current, they are applied to the current editing session but are not committed to file. This allows the palette changes to be used prior to commitment. Note that all modifications are lost if they are not saved and the editing session is terminated.

Save

Saves the scheme modifications to file, effectively making the changes permanent. Note however that the scheme macro file will be saved in the first directory in the $search−path(5), regardless of the location of the original. For network systems this typically means that the changes will only effect the current user.

Install

Installs the current color scheme into the configuration files, making the color scheme accessible to the user−setup(3) dialog.

Exit

Quits the scheme editor without modifying the settings. **FILES**

`scheme.emf` − Defines the standard scheme variables, including the available scheme list, and associated text.
`schemed.emf` − Default white on black color scheme.
`schemej.emf` − Black on cream color scheme.
`schemevi` − Sandy shores.
`schemesf` − Sherwood Forest.
`schemebh` − Blue Hue.
`schemepd` − Plain Black on Cream.
`schemepl` − Plain White on Black.
`schemel` − Black on grey.
`schememd` − Microsoft Developer Studio Colors.
`printers.emf` − Defines the list of available printer schemes and drivers.
`printd` − Default plain print−out.
`printf` − Print using fonts.
`printepc` − Print using Epson base colors and fonts.

NOTES

scheme−editor is a macro that is implemented in file `schemosd.emf`. The scheme editor uses osd(2) to create and manage the dialogs.

Only the Normal scheme style is used by printer schemes.

The setting of **Buffer Hilight** can effect the way buffer hooks are load so changing from one scheme to another with differing Buffer Hilight settings may not fully work. This can be rectified by restart MicroEmacs with the new scheme as default.

The current screen scheme can effect the printing due to the **Buffer Hilight** setting, e.g. if the screen scheme is set to completely disable hilighting then any print−out will also have no hilighting.

SEE ALSO

user−setup(3), add−color−scheme(2), print−scheme(2), osd(2).

screen−poke(2)

NAME

screen−poke − Immediate write string to the screen

SYNOPSIS

n **screen−poke** *row column colorScheme* "*string*"

DESCRIPTION

screen−poke writes a *string* to the screen at position (*row*, *column*) using the given color scheme. The screen coordinates are defined with (0,0) at the top left of the screen.

screen−poke by−passes the conventional buffer update and writes directly to the screen buffer. The command has no effect on buffers already showing on the screen and is erased on the next screen update. The *string* is clipped to the screen area hence the caller need not continually check on the size of the client area.

The numeric argument *n* is a bitwise flag which has the following meaning
`0x01` Don't mark the poke area for update.
`0x02` Don't flush poke to screen.
`0x04` colorScheme is an array of values, one for each letter.
`0xf0` colorScheme pair offset to use.

If the **0x01** flag is absent then the parts of the screen over written by **screen−poke** are marked and refreshed on the next **screen−update** operation, thereby erasing the poked information. If the flag is present the poked information remains on the screen until a forced refresh is performed (i.e. **1 screen−update**) or the window information under the poked screen data is modified.

In macros using many consecutive screen−pokes (e.g. Patience(3) to display a pack of cards) most pokes use the 'No flush' flag to improve performance and look on some platforms.

The use of **screen−poke** has largely been reduced to games such as Metris(3) since the introduction of osd(2) to create dialogs.

NOTES

Some platforms do not allow all character values to be poked, illegal characters are replaced with a '`.`'.

SEE ALSO

osd(2), screen−update(2), Mahjongg(3), Metris(3).

screen−update(2)

NAME

screen−update − Force screen update

SYNOPSIS

n **screen−update** (**redraw**)

DESCRIPTION

screen−update updates the current screen, usually used in macros. The argument *n* can be used to change the behaviour of this command as follows:

`−ve`

Disables the next −*n* screen updates, i.e. if *n* is `−1` then the next time the screen needs to be redrawn nothing will happen.

`0`

Resets the screen update disable count to zero, useful to remember when the the disable feature has been used incorrectly.

`1`

Full screen update (default), the screen is completely cleared and redrawn (as if garbled).

`2`

Partial screen update, only the parts of the screen which require updating are redrawn.

`3`

No screen redraw, only window variables are up−dated. This feature is provided for macros which manipulate the screen view and need to know where the cursor is in the window without redrawing the screen (which may cause unwanted flickering). Note that as the screen is not redrawn not all variables may have the correct value, for example the frame store variable @fs(4) could be out of date. **EXAMPLES**

The following macro demonstrates the problems encountered when trying to use screen variables in macros after the current position has changed. The first value printed is the starting cursor Y position and the next value should be one less than the first value due to the call to backward−line(2). But it is the same as the first because the screen (and its variables) have not been updated. The subsequent call to screen−update ensures that the third value is the correct one although by giving it an argument of 3 the screen is not visibly updated thus avoiding any annoying screen flicker:

```
define−macro test−screen−update
    set−variable #l0 $cursor−y
    backward−line
    set−variable #l1 $cursor−y
    3 screen−update
    set−variable #l2 $cursor−y
    forward−line
    ml−write &spr "%d %d %d" #l0 #l1 #l2
!emacro
```

NOTES

Every time the screen requires updating, MicroEmacs executes the *redraw* key, it is similar in mechanism to the user pressing *C−l* to refresh the screen. The user can therefore re−bind the *redraw* key to another command or macro, thereby allowing the user complete control of what is displayed. For example if *redraw* was bound to void(2) the screen would not be up−dated (**Note**: this is difficult to get out of and may require MicroEmacs to be killed).

This feature is often exploited by macros which take control of the input and output, such macros include gdiff(3), Metris(3), and Mahjongg(3).

SEE ALSO

recenter(2), screen−poke(2).

scroll−down(2)

NAME

scroll−down − Move the window down (scrolling)
scroll−up − Move the window up (scrolling)

SYNOPSIS

n **scroll−down** (**C−n**)
n **scroll−up** (**C−p**)

DESCRIPTION

scroll−down moves the window in the current buffer down by *n* lines, the default when *n* is omitted is 1 windows worth of lines i.e. a next page operation. A −ve value of *n* causes the window to move up.

scroll−up moves the window in the current buffer up by *n* lines, default when *n* is omitted is 1 windows worth of lines, i.e. a previous page operation. A −ve value of *n* causes the window to move down.

SEE ALSO

scroll−left(2), scroll−right(2), $window−y−scroll(5).

scroll−left(2)

NAME

scroll−left − Move the window left (scrolling)
scroll−right − Move the window right (scrolling)

SYNOPSIS

n **scroll−left** (**C−x** <)
n **scroll−right** (**C−x** >)

DESCRIPTION

scroll−left moves the window in current buffer left by 1 screen width. If an argument *n* is supplied then the resolution of movement is specified in characters relative to the current displacement. Moving the window in the current buffer left by *n* characters (that is if the current left−hand margin of the screen is column 0, the left hand margin becomes column *n*).

scroll−right moves the window in current buffer right by 1 screen width. If an argument *n* is supplied then the resolution of movement is specified in characters relative to the current displacement.

The ends of the lines of a scrolled screen are delimited with a dollar (**$**) character indicating that the text continues. When no scroll is in effect the left hand margin of the screen does not show the **$** symbol. i.e. The line `This text is scrolled on this line` with a current scroll offset of 2 in a 22 column window would appear as follows:

```
         22
|<-------------------->|

|$s text is scrolled $|
```

The amount of scroll (*n*) is effectively unlimited, it is possible to scroll all of the text in a buffer out of the window, when only **$**'s appear in the left margin, in the last highlighting color of the line (blank lines always remain blank and are not delimited with a **$**). Text on the current line is handled according to the value of $scroll(5) as follows:

$scroll 0

The current line ONLY is scrolled (about the current scroll position) to enable the current buffers cursor position to be viewed. To enable the user to determine where the current line is in relation to the scrolled lines then the first character of the current line is interpreted as follows:−

All of user text appears

```
|$f line of te$|
|At start of l$|
|$f line of te$|
```

Surrounding lines commence with "$" indicates at the start of the line.

$ in column 0

```
|$f line of te$|
|$f line of te$|
|$f line of te$|
```

Text column is the same as the surrounding text i.e. the line and window scroll are the same.

> Left of scroll position

```
|$f line of te$|
|>f line of te$|
|$f line of te$|
```

The current line is to the left of the scrolled position. forward−char (i.e. interpret as **−−>** indicating the direction of travel) moves the cursor, and therefore the line, towards the natural scroll position (**$** in column).

< Right of scroll position

```
|$f line of te$|
|<f line of te$|
|$f line of te$|
```

The current line is to the right of the scrolled position. backward−char (i.e. interpret as **<−−** indicating the direction of travel) moves the cursor, and therefore the line, towards the natural scroll position (**$** in column).

$scroll 1

The position of the cursor on the line determines the scrolled position. In this case all lines in the window are scrolled to ensure that the cursor is always visible. This mode is only useful when dealing with large blocks of text whose line lengths do not vary. **NOTES**

The scrolling is an attribute of the WINDOW and not the BUFFER. If the window is closed, or contents swapped to a different buffer then the scroll setting is reset for the next buffer. A return to the previous buffer does not restore the scroll setting. The only case where scrolling is inherited is when a window is split (see split−window−vertically(2)).

When binding **scroll−left** to the keyboard then it is important to note that when no argument is specified the resolution is *frame−width*'s. A key binding would operate on character multiples, hence the command should be bound with a numeric argument to perform the perform the keyboard action. e.g.

```
1 global−bind−key scroll−left  "A−left"
1 global−bind−key scroll−right "A−right"
```

To move 5 columns on a key stroke, for an accelerated scroll, then the binding may be re−written as:−

```
5 global−bind−key scroll−left  "A−left"
5 global−bind−key scroll−right "A−right"
```

SEE ALSO

$scroll(5), scroll−up(2), scroll−down(2), $window−x−scroll(5).

scroll−next−window−down(2)

NAME

scroll−next−window−down − Scroll next window down
scroll−next−window−up − Scroll next window up

SYNOPSIS

n **scroll−next−window−down** (**esc C−v**)
n **scroll−next−window−up** (**esc C−z**)

DESCRIPTION

scroll−next−window−down scrolls the next window down *n* lines, if *n* is omitted then the next window is scrolled by *window* number of lines (i.e. next screen page).

scroll−next−window−up scrolls the next window up *n* lines, as **scroll−next−window−down**.

These commands are useful in macros to control other windows.

SEE ALSO

scroll−up(2), scroll−down(2).

search−forward(2)

NAME

search−forward − Search for a string in the forward direction
search−backward − Search for a string in the backward direction

SYNOPSIS

n **search−forward** "*string*" (**C−x s**)
n **search−backward** "*string*" (**C−x r**)

DESCRIPTION

search−forward searches for a string from the current cursor position to the end of the file. The string is typed on the bottom line of the screen, and terminated with the `<ESC>` key. Special characters can be typed in by preceding them with a `^Q`. A single `^Q` indicates a null string. On successive searches, hitting `<ESC>` alone causes the last search string to be reused.

Searching is affected by magic(2m) mode, which allows regular expression pattern matching, and exact(2m) mode which makes the search case sensitive.

The numeric argument *n* is interpreted as follows:−

n > **0**

The *n*th occurrence of the *string* is located.

n < **0**

The first occurrence of the *string* is located in the next *n* lines.

search−backward searches backwards in the file. In all other ways it is like **search−forward**.

DIAGNOSTICS

The command returns a status of `FALSE` if the *string* could not be located (or *n*th *string* where *n* occurrences are requested). If the *string* is found within the given search criteria the return status is `TRUE`.

SEE ALSO

buffer−mode(2), exact(2m), hunt−backward(2), hunt−forward(2), isearch−forward(2), magic(2m),

replace−string(2).
Regular Expressions

set−alpha−mark(2)

NAME

set−alpha−mark − Place an alphabetic marker in the buffer

SYNOPSIS

set−alpha−mark "*?*" (**C−x C−a**)

DESCRIPTION

set−alpha−mark places an alpha mark at the current location in the buffer which can be returned to from anywhere in the buffer using the command goto−alpha−mark(2). The user is prompted for a mark name which can be any alphabetic character. the mark is destroyed if the line is deleted.

SEE ALSO

goto−alpha−mark(2).

set−char−mask(2)

NAME

set−char−mask − Set character word mask

SYNOPSIS

n **set−char−mask** "*flags*" ["*value*"]

DESCRIPTION

set−char−mask returns or modifies the setting of MicroEmacs internal character tables. The argument *n* defines the action to be taken, as follows:−

−1

Removes characters from the given set.

0

Returns characters in the given set in $result(5).

1

Adds characters to the given set.

The first argument "*flags*" determines the required character set as follows:−

d

Is Displayable. Characters in this set can be directly displayed to the screen (as a single character) when occurring in a buffer. When a character not in this set is to be displayed it is performed using more than one character. Characters in the range 1−31 are displayed as "`^?`" where `?` is the ASCII character plus 64, (e.g. 0x01 −> 65, i.e. "`^A`") otherwise the character is displayed in the form "`\xhh`" where `hh` is the hex form of the ASCII value. One notable exception is the tab character (0x09), by default this character is not displayable, instead it is displayed as a sequence of one or more spaces up to the next tab stop.

p

Is Pokable. Similar to **d**, characters in this set can be poked to the screen when using screen−poke(2). When found in a binary file the character is displayed in the right hand column. Unlike **d**, any character outside this set will be displayed as a single period '`.`', indicating that it cannot be displayed.

P

Is Printable. Similar to **d**, characters in this set may be printed as a single character when using print−buffer(2) or print−region(2). Any character not in this set is printed in a similar fashion to **d**.

M

Character font Map. Internally MicroEmacs uses ISO−8859−1 (Latin 1) to configure alphabetic classes and the spell−checker, however the system font being by the native platform may not be the same, for example a small 'e' acute is character 0xe9 in ISO−8859−1 but character 0x82 in Windows OEM fonts. To change the characteristics of the 'e' acute character (such as making it an alphabetic character), the ISO−8859−1 character should always be used, but a correct mapping of ISO−8859−1 to the display font (such as Windows OEM) must also be supplied.

> Unlike other sets, this set cannot be incrementally altered, any calls to alter this set leads to the resetting of all the character tables so the character mapping must be performed first and in a single call. No other set may be altered in the same call. When setting, the "*value*" must supply pairs of characters, an ISO−8859−1 character followed by its system font equivalent.

L

ISO−8859−1 (Latin 1) character map list. This set cannot be altered using this flag, character mappings must be set up using flag **M**. The order of the characters in the returned **$result** string is the same as the order for flag **U**.

U

User font character map list. This set cannot be altered using this flag, character mappings must be set up using flag **M**. The order of the characters in **$result** when returned is the same as the order for flag **L**.

a

Is Alphabetic letter. Characters in this set are alphabetical characters, used by many MicroEmacs commands such as forward−word(2). When setting, the "*value*" must specify pairs of ISO−8859−1 (Latin 1) characters, an Upper−case character followed by its lower−case equivalent. This enables commands such as lower−case−word(2) to operate correctly regardless of the font and language being used. Some fonts may not have all the characters available for rendering, for instance PC Code page 437 does not have an upper−case 'e' grave. In this case an ordinary 'E' should be used as a sensible replacement, i.e. "E`e" (where `e is an 'e' grave). However, this will lead to all upper−case 'E's to map to a lower−case 'e' grave in a case changing operation, this may be corrected by adding a further mapping of 'E' to 'e' to over−ride the 'e' grave mapping, i.e. "E`eEe". This technique does fail when changing the case more than once, when all lower case 'e' graves will be lost.

> Note that the returned character list will pair all lower−case characters with their upper−case equivalent letters first.

l

Is Lower case letter. This set cannot be altered using this flag, alterations to the alphabetic set must be performed using flag **a**. Characters in this set are all the lower−case letters, typically the characters '`a`' to '`z`'. The order may not be the same as returned by flag **u**.

u

Is Upper case letter. This set cannot be altered using this flag, alterations to the alphabetic set must be performed using flag **a**. Characters in this set are all the upper−case letters, typically the characters '`A`' to '`Z`'. The order may not be the same as returned by

h

Is Hex−decimal Digit. The set is rarely used as it is invariably the digits '`0`' to '`9`' and the letters '`a`' to '`f`' in upper and lower case. It is often used in the setting of $buffer−mask(5).

A

Is Alpha−numeric. This set cannot be altered using this flag, alterations to the alphabetic set must be performed using flag **a**. Characters in this set are either alphabetic characters or the digits 0−9.

s

Is Spell extended word character. The characters in this set are recognized by the spell checker as characters which may be considered part of a word, for example the period '`.`'s in e.g. or the hyphen '`−`' in hyphenated−words. Typically this set contains the characters '`'`', '`−`' and '`.`'.

1, **2**, **3** & **4**

Is in Word. These user definable sets are used to add characters to a buffer's word character set, affecting the operation of commands like forward−word(2). Many different file types operate better with a different word character set, e.g. it is preferable to include the '_' character when editing C files. See variable $buffer−mask(5).

Unless stated otherwise, multiple flags may be specified at the same time returning a combined character set or setting multiple properties for the given "*value*" characters.

EXAMPLE

For many UNIX XTerm fonts the best characters to use for $box−chars(5) (used in drawing osd(2) dialogs) lie in the range 0x0B to 0x19. For example the vertical bar is '`\x19`', the top left hand corner is '`\x0D`' etc. These characters are by default set to be not displayable or pokable which renders them useless. They can be made displayable and pokable as follows:−

```
set−char−mask "dp" "\x19\x0D\x0C\x0E\x0B\x18\x15\x0F\x16\x17\x12"
```

MicroEmacs variables have either '`$`', '`#`', '`%`', '`:`' or a '`.`' character prepended to their name, they may also contain a '`−`' character in the body of their name. It is preferable for these characters to be part of the variable 'word' so commands like forward−kill−word(2) can work correctly. This may be achieved by adding these characters to user set **2** and setting the **buffer−mask** variable to include set **2**, as follows:

```
set−char−mask "2" "$#%:.−"

define−macro fhook−emf
    set−variable $buffer−mask "luh2"
    .
    .
!emacro
```

For the examples below only the following subset of characters will be used:−

```
Character                ISO−8859−1    Windows OEM    PC Page 437

Capital A (A)            A             A              A
Capital A grave (`A)     \xC0          \xB7           No equivalent
Capital A acute ('A)     \xC1          \x90           No equivalent
Small a (a)              a             a              a
Small A grave (`a)       \xE0          \x85           \x85
Small A acute ('a)       \xE1          \xA0           \xA0
```

As the spell checker only operates in ISO−8859−1 (Latin 1), the character font mapping (flag **M**) must be correctly setup for spell checking to operate correctly. For ISO−8859−1 (ISO) this is an empty string as the default mapping is correct, but for both Windows OEM (OEM) and PC Code Page 437 (PC−437) the mappings should be set as follows:−

```
; OEM font mapping setup
set−char−mask "M" "\xC0\xB7\xC1\x90\xE0\x85\xE1\xA0"
; PC−437 font mapping setup
set−char−mask "M" "\xC0A\xC1AAA\xE0\x85\xE1\xA0"
```

As all the characters in ISO have equivalents in OEM, the mapping for OEM is a simple ISO to OEM character list. However the missing capital **A**'s in PC−437 cause problems, for the command charset−iso−to−user(3) it is preferable for a mapping of **`A** to be given, otherwise the document being converted may remain unreadable. Therefore a mapping of **`A** to **A** is given to alleviate this problem, similarly **'A** is also mapped to **A**.

This leads to a similar problem with the conversion of PC−437 back to ISO (the operation of command charset−user−to−iso(3)). If only the mapping of "`\xC0A\xC1A`" was given, the last mapping (**'A** to **A**) would also be the back conversion for **A**, i.e. ALL **A**'s would be converted back to **'A**'s. To solve this problem, a further seemingly pointless mapping of **A** to **A** is given to correct the back conversion.

For languages which use these characters, the alphabetic character set must be extended to include these characters for letter based commands like forward−word(2) and upper−case−word(2) to operate correctly. The addition of extra letters must achieve two goals, firstly to define whether a character is a letter, enabling commands like **forward−word** to work correctly. The second is to provide an upper case to lower case character mapping, enabling commands like **upper−case−word** to work correctly. This is achieved with a single call to **set−char−mask** using the **a** flag as follows:−

```
set−char−mask "a" "\xC0\xE0\xC1\xE1"
```

Note that this flag always expects a ISO−8859−1 character, this allows the same map character list to be used regardless of the font set being used, i.e. the above line can be used for ISO, OEM and PC−437 fonts. But it does mean that the ISO to user font character mapping (flag **M**) must already have been performed.

Similar problems are encountered with the **M** flag with font PC−437. This problem is not immediately obvious because the mapping is given in ISO, but when this is converted to PC−437, the mapping string becomes "`A\x85A\xA0`". As can be seen, **A** is mapped last to **'a** so an upper to lower character operation will convert a **A** to **'a**. A similar solution is used, a further mapping of **A** to **a** is given to correct the default case mapping for both **A** and **a**, i.e. the following line should always be used instead:−

```
set−char−mask "a" "\xC0\xE0\xC1\xE1Aa"
```

SEE ALSO

forward−word(2), $buffer−mask(5), screen−poke(2), spell(2), $tabwidth(5).

set−cursor−to−mouse(2)

NAME

set−cursor−to−mouse − Move the cursor to the current mouse position

SYNOPSIS

n **set−cursor−to−mouse**

DESCRIPTION

set−cursor−to−mouse sets the current window and cursor position to the location of the mouse on it's last event (button press or release). This command may change the current window. If the line on which the mouse was located was the message line then the no action is taken, if the line was a window mode line the that window is made the current window but the cursor location within the window remains the same. This is usually used in user defined macros that control the functionality of the mouse.

An argument *n* determines if the command is permitted to change windows, when omitted a window change is permitted on **set−cursor−to−mouse**. When specified, the mouse is not permitted to change windows and returns an error condition in $mouse−pos(5) indicating that the mouse is not within the current window.

Invocation of this command sets the variable $mouse−pos(5) which determines where the mouse is within the window. Interrogation of the variable following the command may be used to determine if the mouse is located on one of the more specialized window or screen regions.

When writing macros to cut and paste using the mouse, care should be taken to ensure that the window at the button release is the same is at the button press. If this is not undertaken, undesired effects could result. The use of set−position(2) and goto−position(2) are most usefully used with this command to restore existing window context.

SEE ALSO

$mouse−pos(5), $mouse−x(5), $mouse−y(5), $window−mode−line(5), $window−scroll−bar(5), set−scroll−with−mouse(2), set−position(2), goto−position(2).

set−encryption−key(2)

NAME

set−encryption−key − Define the encryption key

SYNOPSIS

set−encryption−key (**esc e**)

DESCRIPTION

set−encryption−key sets the encryption key for files loaded or saved with crypt(2m) mode enabled. This must be performed for each file, key is not entered into the history. The key can be set for each file on the command line using the **−k** flag. When saving a buffer in encryption mode the key will be prompted for if not already set.

SEE ALSO

buffer−mode(2), crypt(2m), find−file(2), find−cfile(3).

set−mark(2)

NAME

set−mark − Set starting point of region

SYNOPSIS

set−mark (**esc space**)

DESCRIPTION

set−mark is used to delimit the beginning of a marked region. Many commands are effective for a region of text. A region is defined as the text between the mark and the current cursor position. To delete a section of text, for example, one moves the cursor to the beginning of the text to be deleted, issues the **set−mark** command by typing **esc space**, moves the cursor to the end of the text to be deleted, and then deletes it by using the kill−region(2) (**C−w**) command. Only one mark can be set in one window or one buffer at a time, and MicroEmacs '02 will try to remember a mark set in an off screen buffer when it is called back on screen.

A region is a block of text to be acted upon by some MicroEmacs '02 commands. It is demarcated by the **POINT** on one end and the **MARK** at the other. The point is the primary location identifier where most of the action takes place and is always between two characters. The point is indicated by the cursor position in that it is just behind the cursor. The point is also significant in that it defines one end of the region. The mark, on the other hand, is invisible, and is used to demarcate the other end of the region and is set through **set−mark**.

SEE ALSO

copy−region(2), exchange−point−and−mark(2), kill−region(2). reyank(2), yank(2),

set−scroll−with−mouse(2)

NAME

set−scroll−with−mouse − Scroll the window with the mouse

SYNOPSIS

n **set−scroll−with−mouse**

DESCRIPTION

The **set−scroll−with−mouse** command controls the scrolling of a window by the mouse. This is a two stage process, the first stage locks the cursor to the mouse, the second stage scrolls the screen.

The first stage (locking) is performed when the mouse is located on the scroll−box (typically when the left button is depressed i.e. **pick−mouse−1**). **set−scroll−with−mouse** is invoked with an argument *n*, this causes the mouse position to be recorded ready for a scroll. Depending on the scroll method, the blank lines present at the end of the buffer are scrolled off the screen.

Subsequent calls to the **set−scroll−with−mouse** are made with no argument, the window is scrolled by the relative displacement of the mouse from it's locked position, motion is limited at the end of the scrolling region. Scrolling is proportional to the buffer length. The command is typically bound to **move−mouse−1** which results in an update whenever the mouse is moved by the user.

When the button is released **drop−mouse−1** then the scrolling is stopped by unbinding **move−mouse−1**, thereby breaking the binding between the mouse moving and the scroll command.

The scrolling utilizes fractional mouse positional information (i.e. units smaller than a character cell), if available, resulting in a smoother scrolling motion.

EXAMPLE

The following example shows how the command is used.

```
0 define−macro mouse−scroll−pick
    1 set−scroll−with−mouse         ; Lock mouse position to scroller
    global−bind−key set−scroll−with−mouse "mouse−move−1"
!emacro

0 define−macro mouse−scroll−drop
    global−unbind−key "mouse−move−1"
!emacro

global−bind−key mouse−scroll−pick "mouse−pick−1"
global−bind−key mouse−scroll−drop "mouse−drop−1"
```

When the left button is 'picked', **mouse−scroll−pick** lock the cursor to the mouse and binds mouse movement to **set−scroll−with−mouse** so that whenever the mouse is moved the cursor will be repositioned appropriately. When the button is 'dropped', the mouse movement is unbound so that the cursor will no longer be locked to the mouse.

SEE ALSO

$mouse−pos(5), $scroll−bar(5), set−cursor−to−mouse(2).

set−variable(2)

NAME

set−variable − Assign a new value to a variable
unset−variable − Delete a variable

SYNOPSIS

set−variable "*variable*" "*value*" (**C−x v**)
unset−variable "*variable*"

DESCRIPTION

set−variable sets the given register (**#** name), system (**$** name), global (**%** name), buffer (**:** name) or command (**.** name) variable to the given value, erasing its current value. The returned value of an undefined variable is the string "`ERROR`", this maybe used to determine whether a variable has been set.

unset−variable unsets the given variable so that it no longer exists. The variable must be a global (**%**), buffer (**:**) or command (**.**) variable, system (**$**) variables cannot be unset.

The *value* may be quoted or unquoted, if there are any white space characters, or characters open to other interpretation (e.g. **@wc**) in *value* then quotes should be used.

value may contain control characters which are delimited by a back slash (\) which include:−

`\n` newline
`\t` tab
`\\` backslash

Confusion sometimes arises in macros with the back slash, as the back slashes are dereferenced when set. Commands such as replace−string(2) where the command itself utilizes back slashes. In this case the number of back slashes should be doubled as the variable contents under go two stages of dereferencing.

SEE ALSO

describe−variable(2), list−variables(2), &set(4).

Variables
Introduction to Variable Functions
Register Variables

shell(2)

NAME

shell − Create a new command processor or shell

SYNOPSIS

shell (**C−x c**)

DESCRIPTION

shell−command creates a new command processor or shell. Upon exiting the shell, MicroEmacs '02 redraws its screen and continues editing. The exceptions to this are as follows:

X−Windows

A new **xterm** is spawned off and editing control is returned to MicroEmacs '02 once the **xterm** has initialized.

Microsoft Windows

A new MS−DOS shell is created and control is returned to MicroEmacs '02 once the DOS console window has initialized. The shell created is determined by the MS−DOS environment variable `COMSPEC`, this may be a replacement shell e.g. `4DOS`. **SEE ALSO**

ipipe−shell−command(2), pipe−shell−command(2), suspend−emacs(2).

shell−command(2)

NAME

shell−command − Perform an operating system command

SYNOPSIS

shell−command "*string*"

DESCRIPTION

shell−command performs an operating system call with the given *string* as its argument. The command only fails if the shell−command call returns −1. The $result(5) variable is set the return value and can be used to test the result.

SEE ALSO

$result(5), ipipe−shell−command(2), pipe−shell−command(2), suspend−emacs(2).

show−cursor(2)

NAME

show−cursor − Change the visibility of the cursor

SYNOPSIS

n **show−cursor**

DESCRIPTION

show−cursor hides the cursor if a negative argument is given and restores it if a positive or no argument is given. Note that this is not supported on all platforms.

show−cursor internally performs a counting operation, if the cursor is hidden *m* times then it must also be shown *m* times before the cursor becomes visible again, giving no argument will restore the count ensuring it is visible.

show−region(2)

NAME

show−region − Show the current copy region

SYNOPSIS

n **show−region**

DESCRIPTION

show−region manipulates the currently defined region, it can be used to inquire the state of the current region, if any. It can also be used to define a region, enable and disable the region hilighting, as well as move the cursor to the start or end of the region.

Region hilighting occurs between the *mark* (see set−mark(2)) and *point* (current cursor) positions within the current buffer. A region is defined when text is copied to the kill buffer, by using any of the kill commands such as kill−region(2), or copy−region(2). However, the kill region is only visible after a copy−region(2) or a yank(2) operation. A hilight region is also created on a successful search using commands like search−forward(2), the region encloses the search matching string. Spell(2) also creates a hilight region around the current spell word. The user can also define their own region using the numeric argument to **show−region**.

The argument *n* supplied to the command indicates the require functionality and can take the following values:−

`-3` − Set the start position of the region.
`-2` − Move the cursor the Mark position.
`-1` − Disable the hilighting of the current region.
 `0` − Return the current status of the region in $result(5).
 `1` − Enable the hilighting of the current region.
 `2` − Move the cursor the Dot position.
 `3` − Set the end position of the region.
 `4` − Reactivate the current region.

Where an argument of 0 is used to return the current state the value of $result is a bit based flag where:−

0x01

Indicates a region is currently active (visible).

0x02

Indicates a region has been fixed (may not visible).

0x04

Indicates the region is in the current buffer.

0x08

Indicates the cursor is in the current region.

The color of the selection hilight is defined by add−color−scheme(2) and is determined by $buffer−scheme(5), $global−scheme(5) or $buffer−hilight(5).

DIAGNOSTICS

The following errors can be generated, in each case the command returns a FALSE status:

[No current region]

There is no current defined region on which to operate.

[Current region not in this buffer]

An argument of 2 or −2 was used and the defined region isn't in the current window so the cursor can not be moved to it. **NOTES**

If no argument is given to the command it hilights the current region, similar to an argument of 1. But the properties of the hilight, namely how long it will be hilighted for, are inherited from the setting of $show−region(5), whereas if an argument of 1 is passed in then the hilighting is set to be kept until the region becomes invalid (i.e. as if $show−region(5) is set to 3).

SEE ALSO

$show−region(5), $buffer−hilight(5), $buffer−scheme(5), $global−scheme(5), add−color−scheme(2), copy−region(2), yank(2), search−forward(2), spell(2), set−mark(2).

start−up(3)

NAME

start−up − Editor startup callback command
shut−down − Editor exit callback command

SYNOPSIS

start−up
shut−down

DESCRIPTION

By default **start−up** is not defined, if the command is defined (via a user macro) then it is executed immediately after MicroEmacs '02 has completed its initialization.

This command may initially seem redundant as the user may execute any command at start−up by editing the "`me.emf`" file or using the '**@**' command−line argument. At the point of "`me.emf`" file execution none of the files specified on the command−line will be loaded, thus any actions required on the given command−line files will not work (the only buffer present will be the "***scratch***" buffer).

The **start−up** command is executed AFTER the execution of "`me.emf`" and initialization of buffers, but before MicroEmacs '02 waits for user input.

The **shut−down** command is also not defined by default, but if it is defined during the running of MicroEmacs the command will be called when MicroEmacs exits. The command is not called if MicroEmacs has to perform an emergency exit (due to the system being shut down or process being killed etc).

SEE ALSO

me(1).

sort−lines(2)

NAME

sort−lines − Alphabetically sort lines

SYNOPSIS

n **sort−lines**

DESCRIPTION

sort−lines alphabetically sorts lines of text in the current buffer from the mark position to the current cursor position. If the buffer mode exact(2m) is enabled then the sort is case sensitive, otherwise the sort is case insensitive. By default the text is compared from left to right from column 0 (the left hand edge), if a positive argument *n* is given then the text is compared left to right from the *n*th column, any lines shorter than *n* characters are moved to the top and sorted from column 0.

If a negative argument *n* is given then the text is sorted in reverse order. The comparison starts at column `−1−n`, i.e. an argument of −1 sorts in reverse order from column 0.

EXAMPLE

The following table gives the results of **sort−lines** for different exact modes and values of *n*.

	Original	Sorted Lines					
exact	−	n	n	y	y	n	n
n	−	−	1	−	1	−1	−2
	B	a2	B	Aa	B	CA	Aa
	CA	Aa	c	B	c	c	CA
	b1	B	b1	CA	b1	b1	a2
	Aa	b1	a2	a2	a2	B	b1
	c	c	CA	b1	CA	Aa	c
	a2	CA	Aa	c	Aa	a2	B

NOTES

Typically MicroEmacs is executed with exact(2m) mode enabled, the macro command **sort−lines−ignore−case** provides a command to sort lines case insensitively while **exact** mode is enabled. The macro is defined as follows:−

```
define−macro sort−lines−ignore−case
    set−variable #l0 &bmod exact
    −1 buffer−mode "exact"
    !if @?
        @# sort−lines
    !else
        sort−lines
    !endif
    &cond #l0 1 −1 buffer−mode "exact"
!emacro
```

sort−lines−ignore−case(3) is a macro defined in format.emf.

SEE ALSO

buffer−mode(2), exact(2m), sort−lines−ignore−case(3), transpose−lines(2), uniq(3).

sort−lines−ignore−case(3)

NAME

sort−lines−ignore−case − Alphabetically sort lines ignoring case"

SYNOPSIS

n **sort−lines−ignore−case**

DESCRIPTION

sort−lines−ignore−case forces the current buffers exact(2m) mode to off and then calls sort−lines(2) which will perform a case insensitive alphabetical line sort from the mark position to the current cursor position. The state of the current buffers **exact** mode is restored on completion.

NOTES

sort−lines−ignore−case is a macro defined in `format.emf`, see help on command sort−lines(2) for a complete definition.

SEE ALSO

sort−lines(2), buffer−mode(2), exact(2m), transpose−lines(2).

spell(2)

NAME

spell − Spell checker service provider

SYNOPSIS

n **spell** ["*word*"] ["*rules*"] ["*correction*"] ["*rule*"]

DESCRIPTION

spell is a low level command which provides spell checking capabilities for MicroEmacs '02, it is not designed to be used directly. The action of **spell** depends on the argument given, which is a bitwise flag defined as follows:−

0x001

If set then gets the input word from the user, i.e. "*word*" must be supplied. Otherwise the word input is taken from the current buffer.

0x002

If set then keeps getting words from the current buffer until either the end of the buffer is reached or an error is found. If the end of the buffer is reached then the command succeeds setting $result(5) to the value "*F*". This bit is ignored if bit 0x001 is set. **spell** sets the current show−region to enclose the problematical word and the command show−region(2) can be used to move around the word.

0x004

Adds the given word to a dictionary determined by the state of bit 0x008. If the word is flagged as erroneous (see bit 0x010) then a "*correction*" word must be supplied, otherwise a list of "*rules*" which can be applied to the word must be given, this list can be empty. Note that if the word is not flagged as erroneous and it already exists in the dictionary, the word is not removed, instead a combined rule list is created.

0x008

When set flags that word additions (bit 0x004) and deletions (bit 0x200) should be made to the ignore dictionary. Otherwise word additions are made the last added dictionary and deletions are made to all main dictionaries.

0x010

When set flags that the given word is erroneous, used solely by word additions to create auto−corrections.

0x020

Returns a '|' separated guest guess list for the given word in **$result**.

0x040

> If bit **0x100** is also set a complete list of valid words derivable from the given word are inserted into the current buffer. Otherwise spell returns $result(5) set to the derivative word created when the given "*rule*" is applied to "*word*". The rule applied is the first found of the given rule letter with a matching base ending (see add−spell−rule(2)). The word need not exist as not tests for the legality of the resultant word is used, for example in American, executing
>
> ```
> 65 spell "spelling" "V"
> ```
>
> returns "`spellingive`" in **$result**. Returns the empty string if no rule could be applied.

0x080

Used with bit 0x002 to enable double word checking.

0x100

Return information in **$result** about the given word, or the word which is used to derive the given word. The information consists of the spell status, the word as stored in the dictionary, and either the list of valid rules, or the correction word. See also bit **0x040**.

0x200

Delete the given word from a dictionary determined by bit 0x008

If none of the main functions are used (bits 0x004, 0x020, 0x040 & 0x200) then the status flag is returned in the first column of **$result**. These are defined as follows:−

A

Auto−replace. The word was found and flagged as erroneous. The correction word is given in **$result**, either next to the flag, or if bit 0x100 is set then after the '>' character.

D

Double word. Indicates that the first problem found is a double occurrence of the same word one after the other.

E

Erroneous. The word was not found, so is Erroneous

N

Not a word. The current word found contains no alphabetic characters so is not deemed to be a word, e.g. 3.141593.

O

Okay. The word was found and is not an erroneous word. **SEE ALSO**

add−dictionary(2), add−spell−rule(2), delete−dictionary(2), save−dictionary(2), show−region(2), spell−buffer(3), spell−word(3), Locale Support.

spell−add−word(3)

NAME

spell−add−word − Add a word to the main dictionary

SYNOPSIS

n **spell−add−word** ["*word*"]

DESCRIPTION

spell−add−word adds words to the last dictionary added using the command add−dictionary(2). If no argument is supplied the user is prompted for the word and rule flags, only a 'Good' word can be added (see below). If an argument *n* is given then the next *n* words from the current buffer are added. The words must take one of the following three forms:

`xxxx` − Good word `xxxx` with no spell rules allowed
`xxxx/abc` − Good word `xxxx` with spell rules `abc` allowed
`xxxx>yyyy` − Erroneous word with an auto−replace to `yyyy`

NOTES

spell−add−word is a macro defined in file `spellutl.emf`. It is not defined by default so `spellutl.emf` must be executed first using execute−file(2).

SEE ALSO

add−dictionary(2), edit−dictionary(3), save−dictionary(2), delete−dictionary(2).

split−window−horizontally(2)

NAME

split−window−horizontally − Split current window into two (horizontally)

SYNOPSIS

n **split−window−horizontally** (**C−x 5**)

DESCRIPTION

split−window−horizontally splits the current window horizontally into two near equal windows, each displaying the buffer displayed by the original window.

A numeric argument *n* of 1 forces the left window to be the new current window, and an argument of 2 forces the right window to be the new current window. The default when omitted is the left window.

SEE ALSO

$scroll−bar(5), $scroll−bar−scheme(5), $window−chars(5), grow−window−horizontally(2), split−window−vertically(2).

split−window−vertically(2)

NAME

split−window−vertically − Split the current window into two

SYNOPSIS

n **split−window−vertically** (**C−x 2**)

DESCRIPTION

split−window−vertically splits the current window vertically into two near equal windows, each displaying the buffer displayed by the original window. A numeric argument *n* of 1 forces the upper window to be the new current window (default), and an argument of 2 forces the lower window to be the new current window.

SEE ALSO

grow−window−vertically(2), next−window−find−buffer(2), next−window−find−file(2), resize−window−vertically(2), split−window−horizontally(2).

sql(9)

SYNOPSIS

sql − SQL files

FILES

hksql.emf − SQL hook definition
sql.etf − SQL template file.

EXTENSIONS

.sql − SQL file

DESCRIPTION

The **sql** file type template provides simple hilighting of SQL files, the template provides minimal hilighting.

BUGS

None reported.

SEE ALSO

Supported File Types

suspend−emacs(2)

NAME

suspend−emacs − Suspend editor and place in background

SYNOPSIS

n **suspend−emacs**

PLATFORM

Supported on UNIX platforms − *irix*, *hpux*, *sunos*, *freebsd* or *linux*.

DESCRIPTION

suspend−emacs suspends the editing processor and puts it into the background. The "*fg*" command restarts MicroEmacs. The prompt to suspend is disabled if a `0` numeric argument *n* is given to the command.

SEE ALSO

shell(2).

symbol(3)

NAME

symbol − Insert an ASCII character

SYNOPSIS

symbol

DESCRIPTION

symbol draws the ASCII character table to the screen, displaying decimal, hexadecimal and character notations in a tabular form. A character is selected using the mouse or cursor characters inserting the selected character into the current buffer at the current position.

NOTES

symbol is a macro defined in `misc.emf`.

The dialog is created using osd(2).

SEE ALSO

insert−string(2), &atoi(4), osd(2).

Triangle(3)

NAME

Triangle − MicroEmacs '02 version of Triangle patience game

SYNOPSIS

Triangle

DESCRIPTION

Triangle is a solitaire game using a standard set of playing cards. The object of the game is to use all of the cards in the deck to build up four suit stacks from Ace to King.

The board is laid out so that every card is used to create a triangle shape. In the first column there is one up−turned card, in the second column there is one down−turned card and 2 up−turned, third has 2 down 3 up etc. The only break form this pattern is in the last 3 columns where there is an extra up−turned card so that all the deck is used.

Cards may be moved around the playing area by stacking the same suit cards in descending order on the row stacks. When a row stack has no up−turned cards on the stack then the top card may be turned over and may be played. If a stack becomes empty then only a King may be moved into the vacant position.

If the last card in a stack is an Ace then it can be moved to its suit stack, then the 2 of that suit etc. until finally the King is removed.

Cards are moved around the board using the mouse. Cards may be moved from one row stack to another row stack by placing the mouse over the 'from' stack and pressing the left mouse button. Move the cursor to the 'to' stack and release the left mouse button. If the move is legal then the card(s) are moved to the new stack. Multiple cards may be moved from the row stacks, the appropriate card(s) to be moved is automatically determined.

Cards may be moved onto the suit stacks by a single left mouse press and release on the same card, the card is moved to the appropriate suit stack. The same technique is used to turn cards over in the suit stacks.

Note that once a card is played onto the suit stacks then it cannot be removed.

To the right of the board are a number of control buttons. To select an option, click the left mouse button on it, the buttons are labeled:

DEAL

Start a new game by dealing new cards.

QUIT

Exit the game

HELP

This help page

Note that the screen may be updated at any time using "*C−l*".

NOTES

Triangle is a macro defined in `triangle.emf`.

The game is best played with a mouse, it is possible to play with the keyboard, as follows:−

"*esc h*" for help

To move a card between stacks enter the source and destination column number ("*1*","*2*",.."*7*").

To overturn a card on the row stacks then enter the card column twice i.e. source and destination are the same.

To move a card from the row to the suit stacks then either enter the card column twice, or enter the destination as "*h*","*d*","*c*","*s*" (i.e. "*2 2*" or "*2 s*" to move the card in column 2 to the spades stack).

"*C−c C−c*" to deal the cards again.

"*C−l*" redraw the screen.

"*q*" to quit the game.

SEE ALSO

Games, Patience(3), Mahjongg(3).

tab(2)

NAME

tab − Handle the tab key

SYNOPSIS

n **tab** (**tab**)

DESCRIPTION

tab manages the `tab` key, typically inserts *n* tabs. The effect of the command is determined by:

$buffer−indent

If $buffer−indent(5), is non−zero then the effect of tab is defined by the setting of bit `0x1000` of variable $system(5), typically it resets the current line indentation or inserts a tab.

cmode

If cmode is enabled then the effect of tab is defined by the setting of bit `0x1000` of variable $system(5), typically it resets the current line indentation or inserts a tab.

tab

If a tab is to be inserted and this mode is enabled then multiple spaces are used instead of tab characters, see tab(2m) mode. **SEE ALSO**

cmode(2m), $buffer−indent(5), tab(2m), backward−delete−tab(2), insert−tab(2), normal−tab(3), $tabsize(5), $tabwidth(5).

tab(2m)

NAME

tab − Tabulation mode

SYNOPSIS

tab Mode

T − mode line letter.

DESCRIPTION

tab mode, when enabled, simulates all tab stops with spaces. This allows '*variable*' tab sizes (see variable $tabsize(5)) and fixes indentation. If **tab** mode is not enabled literal tab characters are inserted, their displayed width may be controlled with the variable $tabwidth(5).

SEE ALSO

buffer−mode(2), global−mode(2), $tabsize(5), $tabwidth(5), tabs−to−spaces(3).

tabs−to−spaces(3)

NAME

tabs−to−spaces − Converts all tabs to spaces

SYNOPSIS

tabs−to−spaces

DESCRIPTION

tabs−to−spaces converts all tab characters found in the current buffer with spaces. The number of spaces a tab is replaced with depends on the column of the tab character and the setting of $tabwidth(5).

The cursor is restored to the start of the current line after completion.

NOTES

tabs−to−spaces is a macro defined in `format.emf`.

SEE ALSO

$tabwidth(5), tab(2), tab(2m), clean(3).

tcl(9)

SYNOPSIS

tcl, tk − TCL Programming language templates

FILES

hktcl.emf − TCL/TK programming language hook definition
tcl.etf − TCL/TK programming language template file

EXTENSIONS

.tcl, **.tk** − TCL/TK file

MAGIC STRINGS

^#![\t]*/.*wish

MicroEmacs '02 recognizes the magic string on the first line of the file used to locate the executable. The tcl files may be extension less and are still recognized. **DESCRIPTION**

The **tcl** provides hilighting and automatic formatting features, in addition to a number of tools to handle the file type.

General Editing

On creating a new file, a new header is automatically included into the file. time(2m) is by default enabled, allowing the modification time−stamp to be maintained in the header.

Hilighting

The hilighting features allow commands, variables, logical, preprocessor definitions, comments, strings and characters of the language to be differentiated and rendered in different colors.

Auto Layout

The indentation mechanism is enabled which performs automatic layout of the text. restyle−region(3) and restyle−buffer(3) are available to reformat (re−layout) selected sections of the buffer, or the whole buffer, respectively.

Tags

A C−tags file may be generated within the editor using the **Tools** −> **Tcl−Tools** −> **Create Tag File**. find−tag(2) takes the user to the file using the tag information.

Folding and Information Hiding

Generic folding is enabled within the C and C++ files. The folds occur about braces **{...}** located on the left−hand margin. fold−all(3) (un)folds all regions in the file, fold−current(3) (un)folds the current region. Note that folding does not operate on K&R style code.

Short Cuts

The short cut keys used within the buffer are:−

C−c C−c − Comment out the current line.
C−c C−d − Uncomment the current line.
C−c C−e − Comment to the end of the line with stars (*).
A−C−i − Restyle the current region.
f2 − (un)fold the current region
f3 − (un)fold all regions

SEE ALSO

indent(2), find−tag(2), fold−all(3), fold−current(3), restyle−buffer(3), restyle−region(3), tcltags(3f), time(2m).

Supported File Types

tcltags(3f)

NAME

tcltags − Generate a Tcl/Tk tags file

SYNOPSIS

me "@tcltags" <*files*>

DESCRIPTION

The start−up file `tcltags.emf` may be invoked from the command line to generate a **tags** file for Tcl/Tk files.

Given a list of *files* a tags file `tags` is generated in the current directory, which may be used by the find−tag(2) command. If no *files* are specified the default file list is "`./`", i.e. process the current directory. If a directory name is given (such as the default "`./`") all Tcl/Tk files within the directory will be processed.

The value of variable **%tag−option** is used to control the tag generation process, its value <*flags*> can contain any number of the following flags:

`a`

Append new tags to the existing tag file, note that if also using flag '`m`' multiple 'tags' to the same item may be created.

`m`

Enable multiple tags. This enables the existence of 2 tags with the same tag name, but typically with different locations. See help on find−tag(2) for more information on multiple tag support.

`r`

Enables recursive mode, any sub−directory found within any given directories will also be processed.

NOTES

This function is invoked from menu

Tools −> Tcl Tools −> Create Tags File

when the user requests a tags file to be generated.

The following variables are set within "`tcltags.emf`" and are used to control the process:−

%tag−option

Tags options flag, default value is "". See above for more information.

%tag−filemask

A list of source file masks to be processed when a directory is given, default value is "`:*.tcl:*.tk:`".

%tag−ignoredir

A list of directories to be ignored when recursive option is used, default value is "`:SCCS/:CVS/:`".

These variables can be changed using the −v command−line option or via the "`mytcltags.emf`" file

SEE ALSO

find−tag(2), start−up(3), tcl(9).

texinfo(9)

SYNOPSIS

texinfo − GNU Texinfo documentation file.

FILES

hktexi.emf − Texinfo file hook definition

EXTENSIONS

.texi − Texinfo file

MAGIC STRINGS

−*− texinfo −*−

Recognized by GNU Emacs and MicroEmacs. **DESCRIPTION**

The **texinfo** file type template provides simple hilighting of GNU **Texinfo** files (`.texi`), the template provides minimal hilighting.

File recognition is performed using the standard file extensions, or the magic string.

NOTES

This template file could benefit from some of the `hklatex.emf` technology for generating the *info* file.

SEE ALSO

Supported File Types

textags(3f)

NAME

textags − Generate a LaTeX/BibTeX tags file

SYNOPSIS

me "@textags" <*files*>

DESCRIPTION

The start−up file `textags.emf` may be invoked from the command line to generate a **tags** file for LaTeX and BibTeX files.

Given a list of *files* a tags file `tags` is generated in the current directory, which may be used by the find−tag(2) command. If no *files* are specified the default file list is "`./`", i.e. process the current directory. If a directory name is given (such as the default "`./`") all LaTeX files within the directory will be processed.

The value of variable **%tag−option** is used to control the tag generation process, its value <*flags*> can contain any number of the following flags:

`a`

Append new tags to the existing tag file, note that if also using flag '`m`' multiple 'tags' to the same item may be created.

`m`

Enable multiple tags. This enables the existence of 2 tags with the same tag name, but typically with different locations. See help on find−tag(2) for more information on multiple tag support.

`r`

Enables recursive mode, any sub−directory found within any given directories will also be processed.

NOTES

This function is invoked from menu

Tools −> LaTeX Tools −> Create Tags File

when the user requests a tags file to be generated.

The following variables are set within "`textags.emf`" and are used to control the process:−

%tag−option

Tags options flag, default value is "". See above for more information.

%tag−filemask

A list of source file masks to be processed when a directory is given, default value is "`:*.tex:*.bib:`".

%tag−ignoredir

A list of directories to be ignored when recursive option is used, default value is "`:SCCS/:CVS/:`".

These variables can be changed using the −v command−line option or via the "`mytextags.emf`" file

SEE ALSO

find−tag(2), start−up(3), tex(9).

time(2m)

NAME

time − File time stamping

SYNOPSIS

time Mode

t − mode line letter.

DESCRIPTION

time mode, when enabled, performs automatic time stamping of files on file write operations. A time stamp string, defined by $timestamp(5) is searched for in the file and updated with the current data and time information, providing a record in the file of the last edit.

SEE ALSO

buffer−mode(2), global−mode(2), $timestamp(5).

time(3)

NAME

time − Command time evaluator

SYNOPSIS

time "*string*"

DESCRIPTION

time evaluates the time take to execute line "*string*". **time** uses command execute−line(2) to execute the given string.

EXAMPLE

The following example simply times the time take to save the current buffer:−

```
time "save-buffer"
```

NOTES

time is a macro defined in `misc.emf`.

On multi−task systems like UNIX **time** cannot take into account the number of other processes running at the same time, it can only return the actual time elapse. This leads to inaccuracies and variation in results.

SEE ALSO

execute−line(2).

translate−key(2)

NAME

translate−key − Translate key

SYNOPSIS

n **translate−key** ["*from*" ["*to*"]]

DESCRIPTION

translate−key may be used to convert any given input key sequence to another single key. **translate−key** operates at a very low level, before MicroEmacs attempts to evaluate keyboard bindings, so it may be used to solve a variety of keyboard problems such as special language characters and UNIX termcap key sequence bindings (see below).

If a +ve numeric argument *n* is given it is used to set the time in milliseconds MicroEmacs waits for another key to be pressed before continuing, the default time use when no argument is supplied is 250ms.

If a numeric argument *n* of −1 is specified then the "*to*" argument is not required and the "*from*" character sequence is removed from the translate key table.

If a numeric argument *n* of 0 is specified then no arguments are required; the current translation table is dumped to buffer "**tcap−keys**". Following is a sample output:−

```
"C−h" ......................... "backspace"
"C−[" ......................... "esc"
"C−[ [ 1 ~" ................... "delete"
"C−[ [ 1 1 ~" ................. "f1"
"C−[ [ 1 2 ~" ................. "f2"
"C−[ [ 1 3 ~" ................. "f3"
"C−[ [ 1 4 ~" ................. "f4"
"C−[ [ B" ..................... "down"
"C−[ [ 4 ~" ................... "end"
"C−[ [ 2 ~" ................... "insert"
"C−[ [ 3 ~" ................... "home"
"C−[ [ D" ..................... "left"
"C−[ [ 6 ~" ................... "page−down"
"C−[ [ 5 ~" ................... "page−up"
"C−[ [ C" ..................... "right"
"C−[ [ A" ..................... "up"
"C−[ [ V" ..................... "page−up"
"C−[ [ U" ..................... "page−down"
"C−m" ......................... "return"
"C−i" ......................... "tab"
"\x7F" ........................ "backspace"
```

FOREIGN KEYBOARDS

Foreign keyboards (non−US/UK) use a variety of key sequences, not recognized by MicroEmacs, to expand the keyboard character range to cope with accented characters. For example, on a German keyboard '`AltGr−m`' (recognized as '`A−C−m`') is used to insert a Greek mu (or micro sign). On a Belgian keyboard '`AltGr−9`' inserts a '{' character.

Many foreign keyboards are already directly supported by MicroEmacs and the keyboard specifics of a country have been understood and resolved. In these cases the **Keyboard** configuration in user−setup(3) may be used for the country location.

If MicroEmacs does not support your keyboard, **translate−key** may be used to fix any key input problems. For the aforementioned examples the following **translate−key** commands would be required:

```
; translate AltGr−m to a Greek mu (char 0xb5)
translate−key "A−C−m" "\xB5"
; translate AltGr−9 to a '{'
translate−key "A−C−9" "{"
```

The problem is complicated further on Microsoft Window's platforms by the simultaneous generation of 2 keys for some Alt−Gr key combinations (this is a side effect of endeavoring to capture all key combinations in this environment). For the Belgian keyboard example, on Win32 platforms an '`AltGr−9`' generates an '`A−C−9`' key first followed immediately by an '`A−C−{`'. As both keys are generated in quick succession this is unexpected and confusing.

When the key is first pressed on a poorly configured system the error "*[Key not bound "A−C−{"]*" is given even when using the command describe−key(2) as the key described will be '`A−C−9`' and then the '`A−C−{`' key is generated and interpreted creating the error message.

The variable $recent−keys(5) can be used to diagnose this problem and to obtain the 2 keys generated; alternatively use the macro below:

```
define−macro report−2−keys
    ml−write "Press key 1"
    set−variable #l0 @cgk
    ml−write "Press key 2"
    set−variable #l1 @cgk
    ml−write &spr "[The following keys where pressed: \"%s\" \"%s\"]" #l0 #l1
!emacro
```

When executed the user is prompted for the first key; press the required key sequence (in this case '`AltGr−9`'), if you are not prompted for the second key and the result is immediately returned then the key you pressed has generated 2 keys, both of which will be given in the print out, i.e.:

```
"[The following keys where pressed: "A−C−9" "A−C−{"]"
```

The translate−key required to fix this type of problem would be:

```
translate−key "A−C−9 A−C−{" "{"
```

If your keyboard is not directly supported by MicroEmacs, please submit the keyboard name and platform with a working translate−key configuration to JASSPA as a **BUG**.

UNIX TERMCAP

translate−key may also be used to interpret non−standard key sequences for UNIX termcap platforms to standard MicroEmacs keys. Non−standard keys, such as the cursor keys, have system dependent key sequences. The output from these keys usually take the form:

`^[[X` or `^[[DX` or `^[[DDX` or `^[[DDD`

where **^[** is the escape key (27), **D** is a digit and **X** is any character. These keys may be bound to the standard keys, for example the typical output of the cursor keys may be translated as follows:−

`^[[A` = **up**, `^[[B` = **down**, `^[[C` = **right** and `^[[D` = **left**

The "*from*" string is specified as this key sequence and the "*to*" string is simply the key it is to be bound to, see global−bind−key(2) for a guide to the string format. For the above example the following set of translations are required:−

```
translate−key "esc [ A" "up"
translate−key "esc [ B" "down"
translate−key "esc [ C" "right"
translate−key "esc [ D" "left"
```

Note that MicroEmacs interprets `\e` as an escape key. More obscure keys tend to be very platform specific, following are some examples:

```
translate−key "esc [ 2 ~" "insert"
translate−key "esc [ 5 ~" "page−up"
translate−key "esc [ 5 ^" "C−page−up"
```

EXAMPLE

Using the +ve numeric argument it is possible to reduce the delay and there by increase usability is some features. For instance, in the Mouse configuration of **user−setup** there is an option to 'Simulate 3 Buttons' which translates a rapid left and right button press into a middle button press. This is implemented using **translate−key** as follows:

```
10 translate−key "mouse−pick−1 mouse−pick−3" "mouse−pick−2"
10 translate−key "mouse−pick−3 mouse−pick−1" "mouse−pick−2"
10 translate−key "mouse−drop−1 mouse−drop−3" "mouse−drop−2"
10 translate−key "mouse−drop−3 mouse−drop−1" "mouse−drop−2"
```

When a `mouse−pick−1` key is generated MicroEmacs must wait to see if a `mouse−pick−3` key is next and therefore translate both to a single `mouse−pick−2` key. This wait time is usually a quarter of a second but this makes the left button unusable for dragging regions etc as the delay is too long. By giving a argument of 10ms the delay is long enough for a simultaneous left and right button press but short enough for the left button to still be usable on its own.

The +ve numeric argument can be very useful for delaying MicroEmacs as well, for example, the character string "`'e`" can be converted to e−accute using expand−iso−accents(3). This could be performed automatically using translate−key as follows:

```
1000 translate-key "' e" "\xE9"
```

The larger 1 second delay give the user enough time to type the '`e`' after the '`'`' character.

NOTES

The concept of standardized key−bindings is very important for cross platform use and maintenance.

Refer to global−bind−key(2) for a list of standard bindings.

One of the easiest ways of obtaining a key sequence is to run **sh(1)** which does not attempt to interpret these keys so when a key is pressed (followed by `<RETURN>`) the following type of error message is usually generated:−

```
sh: ^[[2~:  not found.
```

where `^[[2~` is the required key sequence. Another method of obtaining these key sequences is to start MicroEmacs '02, use start−kbd−macro(2) to start a macro definition, press the required keys and then use end−kbd−macro(2) followed by name−kbd−macro(2) and insert−macro(2) to display the keys pressed.

The key sequences generated for these keys are dependent on the machine displaying MicroEmacs '02 as opposed to the machine running it. Often they are the same machine, but when they are not there is no easy method of determining the displaying machine and therefore correctly configuring MicroEmacs '02.

A better way of obtaining this cross platform consistency is to create an XTerm app−defaults setup file with the correct VT100 key translations, e.g. the setup file could contain the following

```
*vt100.translations: #override \
        Shift<Key>Tab:          string("\033[Z") \n\
        <Key>BackSpace:         string("\177") \n\
        <Key>Delete:            string("\033[1~") \n\
        <Key>Insert:            string("\033[2~") \n\
        <Key>Home:              string("\033[3~") \n\
        <Key>End:               string("\033[4~") \n\
        <Key>Prior:             string("\033[5~") \n\
        <Key>Next:              string("\033[6~") \n\
        Ctrl<Key>Up:            string("\033Oa") \n\
        Ctrl<Key>Down:          string("\033Ob") \n\
        Ctrl<Key>Right:         string("\033Oc") \n\
        Ctrl<Key>Left:          string("\033Od") \n\
        Shift<Key>Up:           string("\033[a") \n\
        Shift<Key>Down:         string("\033[b") \n\
        Shift<Key>Right:        string("\033[c") \n\
        Shift<Key>Left:         string("\033[d") \n
```

By using the environment variable *XUSERFILESEARCHPATH* to ensure that this configuration file is found instead of the system one (found in `/usr/lib/X11/app-defaults`), the key sequences will then be the same across all platforms. See manual page on **xterm(1)** for more information.

SEE ALSO

expand−iso−accents(3), user−setup(3), describe−key(2), global−bind−key(2), start−kbd−macro(2), **xterm(1)**, **sh(1)**.

transpose−chars(2)

NAME

transpose−chars − Exchange (swap) adjacent characters transpose−lines − Exchange (swap) adjacent lines

SYNOPSIS

transpose−chars (**C−t**)
n **transpose−lines** (**C−x C−t**)

DESCRIPTION

transpose−chars exchanges (swaps) the current character under the cursor with the previous character. **transpose−characters** does not operate in column 0 (since there is no previous character). If the cursor is at the end of a line when the command is initiated then the cursor is moved to the previous character and the operation performed from the new position.

transpose−lines swaps the next line for the current line and moves to the next line, effectively retaining the same text position. Repeating this *n* times moves the current line *n* lines down.

EXAMPLE

transpose−character performs the following operations (cursor at ^):−

```
abcde  => acbde         [Middle of line]
  ^         ^

abcde  => abced         [End of line]
     ^         ^
```

SEE ALSO

sort−lines(2).

User Profiles(2)

USER PROFILES

This section describes how a user profile should be incorporated into MicroEmacs '02. A user profile defines a set of extensions to MicroEmacs which encapsulates settings which are used by an individual user.

The user profile allows:−

- Saving of the last session (history), allowing the next invocation of MicroEmacs '02 to restore your previous session.
- Personalized spelling dictionaries.
- Redefinition of MicroEmacs '02, allowing the editor to be tailored to an individual's requirements. Including the re−binding of keys, modification of the screen colors. Definition of personal macros etc.

Identification

In order to identify a user MicroEmacs '02 uses information in the system to determine the name of the user, and in turn the configuration to use. On all systems the value of the environment variable $MENAME(5) takes priority over any other means of user identification. If this variable is not defined then the host system typically provides a mechanism to determine the current user. DOS and *Windows* systems present problems where a login prompt is not supplied.

Each of the supported platforms are now described.

UNIX

The environment variable `$LOGNAME` is defined. This is the user name used by the system.

DOS

MS−DOS typically has no concept of the user name. The user name should be defined in the `autoexec.bat` file, choose a name of 8 characters or less, i.e. to fix the user name to `fred` then add the following line:−

```
SET MENAME=fred
```

Remember to re−boot the system before the new command takes effect. (see the next step, there is another change to `autoexec.bat`).

Microsoft Windows

Microsoft windows environments may, or may not, have logging enabled. If you have to log into your system then a login identification has been supplied and will be recognized by MicroEmacs, setting the environment variable $MENAME(5) to this value.

If login is not enabled then the me32.ini(8) file may be modified to provide a default login name. To add the user **fred** then add the following lines to the *ini* file:−

```
[guest]
MENAME=fred
```

If login is subsequently enabled on the system then these lines should be removed. These lines force the user identification to be **fred**.

The above technique may be used within the windows environment to modify your login name. Assuming that the system administrator has assigned **fred** a user login name of **fwhite**, and *fred* requires all of his configuration files to be the same name as his UNIX login which is **fred**. Then *fred* may force his user name to *fred* from the `me32.ini` file as follows:−

```
[fwhite]
MENAME=fred
```

Once *fred* has entered MicroEmacs he will adopt his new login name which will be used to identify his own files etc. The action of this statement is to force the environment variable `$MENAME` to a new value. Any other environment variables may be forced in this way i.e. `$HOSTNAME` is a good candidate here as the `me32.ini` is local to the machine.

Shared Platforms

Platforms may share the same set of configuration files. Consider a system which may boot under MS−DOS, Windows '98, NT and Linux. Provided that the macro files are located on a file system that may be mounted by all of the other operating systems and the $MEPATH is set appropriately, then a single set of MicroEmacs macro files may be shared across all platforms. **Personal MicroEmacs Directory**

The private user profile is stored in a separate directory. The directory that MicroEmacs uses must be created by the user, create the directory in your local file system. In addition, the MicroEmacs search path $MEPATH(5) should be modified to include your new MicroEmacs personal directory.

UNIX

Create in your local directory, typically called `microemacs` or `.microemacs` (if it is to be hidden).

Add/modify the $MEPATH(5) environment variable to include your personal directory in your `.login`, `.chsrc` or `.profile` file, the file and exact syntax will depend upon your shell. For a Korn shell the following line would be added to the `.profile` file:−

```
export MEPATH=$HOME/.microemacs:/usr/local/microemacs
```

Where `$HOME` is assumed to be the users login home directory, or use the directory location of your new directory.

DOS

For MS−DOS environments, there is typically no user directory, it is suggested that the user directory is created in the MicroEmacs directory, use the `$MENAME` defined in the previous step i.e.

```
mkdir c:\me\fred
```

Change the $MEPATH(5) in the **autoexec.bat** to include the new directory i.e.

```
SET MEPATH=c:\me\fred;c:\me
```

Windows

Windows environments, the me32.ini(8) **userPath** entry defines the location of the user profile directories, within the **Install Shield** installation, the `me32.ini` is typically defined as:−

```
userPath=C:\Program Files\JASSPA\MicroEmacs
```

Create your MicroEmacs personal directory in this folder, the name of the folder should be your login name or `$MENAME`, depending upon how your name is identified.

Creating Your Profile

Once you have created a new directory to store your user profile, create a default profile for yourself from MicroEmacs using the user−setup(3) dialog:−

```
Help => User Setup
```

Fill in the entries in the dialog, and ensure that **Save** is depressed on exit to write the files.

The dictionaries often present difficulties the first time, a prompt to save the dictionary requires the full pathname and the name of the file, the pathname is the path to your personal folder, the filename is typically your *username*.edf. Once the file is created you will not have a problem in the future.

The User Profile

Files created in the user directory include:−

- Setup registry and previous session history *username*.erf, see erf(8)). This stores the **user−setup** settings and also the context from your previous edit session.
- Users start−up file *username*.emf, see emf(8) the user may make local changes to MicroEmacs in this file, this may include changing key bindings, defining new hook functions etc. You should over−ride the standard MicroEmacs settings from your start−up file rather than modifying the standard MicroEmacs files.
- Personal spelling dictionary *username.edf*, see edf(8). This file contains your personal spelling modifications, any words that are added to the spelling dictionary are added to this file.

In addition to the above, if new file hooks are defined then they should be added to this directory (if they are not global to the company).

EXAMPLE

The following are examples of some individuals start−up files:−

```
; Jon's special settings
;
; Last Modified <190698.2226>
;
; Macro to delete the whitespace, or if an a word all of the
; word until the next word is reached.
define−macro super−delete
    set−variable #l0 0
    !while &not &sin @wc " \t\n"
        forward−char
        set−variable #l0 &add #l0 1
    !done
    !repeat
        !force forward−char
        !if $status
            set−variable #l0 &add #l0 1
        !endif
    !until &or &seq @wc "" &not &sin @wc " \t\n"
    #l0 backward−delete−char
    !return
!emacro
; Make a previous−buffer command.
define−macro previous−buffer
    &neg @# next−buffer
!emacro
; spotless; Perform a clean and remove any multi−blank lines.
define−macro spotless
    −1 clean
!emacro
; comment−adjust; Used for comments in electric−c mode (and the other
; electic modes. Moves to the comment fill position, saves having to mess
; around with comments at the end of the line.
0 define−macro comment−adjust
    ; delete all spaces up until the next character
    !while &sin @wc " \t"
        forward−delete−char
    !done
    ; Fill the line to the current $c−margin. We use this as
    ; this is the only variable that tells us where the margin
    ; should be.
    !if &gre $window−acol 0
        backward−char
        !if &sin @wc " \t"
            forward−delete−char
            !jump −4
        !else
            forward−char
        !endif
    !endif
    ; Now fill to the $c-margin
    &sub $c-margin $window-acol insert-string " "
!emacro
; Macro to force buffer to compile buffer for C-x '
define-macro compile-error-buffer
    !force delete-buffer *compile*
    change-buffer-name "*compile*"
!emacro
;
; Set up the bindings.
;
global-bind-key super-delete            "C-delete"
global-bind-key beginning-of-line       "home"
global-bind-key end-of-line             "end"
global-bind-key undo                    "f4"
!if &seq %emulate "ERROR"
    global-bind-key comment-adjust      "esc tab"
    global-bind-key comment-adjust      "C-insert"
    ; Like a korn shell please.
    ml-bind-key tab "esc esc"
!endif
;
; Setup for windows and UNIX.
;
; Define my hilighting colour for Windows and UNIX.
!if &equ &band $system 0x001 0
    !if &not &seq $platform "win32"
        ; Small bold font is better for me.
        change-font "-*-clean-medium-r-*-*-*-130-*-*-*-*-*-*"
        ; Small non-bold font.
        ; change-font "-misc-fixed-medium-r-normal--13-*-*-*-c-70-iso8859-1"
        ; Change the size of the screen
        82 change-frame-width
        50 change-frame-depth
    !endif
!endif
; Change the default diff command-line for GNU diff utility all platforms
set-variable %diff-com "diff --context --minimal --ignore-space-change --report-id
set-variable %gdiff-com "diff --context --ignore-space-change -w"
; Setup for cygnus
!if &seq $platform "win32"
    set-variable %cygnus-bin-path "c:/cygwin/bin"
    set-variable %cygnus-hilight 1
    set-variable %cygnus-prompt "$"
!endif
; Set up the ftp flags. The letters have the following meaning:
; c   - Create a console (*ftp-console* for ftp, *http-console* for http)
; s   - Show the console
; p   - Show download progress ('#' every 2Kb downloaded)
set-variable %ftp-flags "csp"
; Info files
;To hilight the .info and also the dir file
add-file-hook ".info dir"                                   fhook-info   ; Info-fi
;To hilight all info files without the extension .info
;but starting with the text "This is info file..
-2 add-file-hook "This is Info file"                        fhook-info

; Finished
ml-write "Configured to Jon's requirements"
```

SEE ALSO

$MEPATH(5), $MENAME(5), user−setup(3), Company Profiles, File Hooks, File Language Templates, Installation.

undo(2)

NAME

undo − Undo the last edit

SYNOPSIS

n **undo** (**C−x u**)

DESCRIPTION

undo removes the last *n* edits made to the current buffer. The undo(2m) buffer mode must be enabled for this command to operate.

The undo information is retained up until the next save operation, at which point the undo information is discarded. When editing large files with gross changes then it is advisable to either disable undo mode, or save frequently to flush the undo buffer, thereby keeping MicroEmacs '02 memory requirements reasonable (most UNIX users have restrictions on the amount of memory that may be consumed by a single process. Windows is restricted by the amount of virtual memory (or swap space)).

SEE ALSO

buffer−mode(2), save−buffer(2), undo(2m).

undo(2m)

NAME

undo − Retain edit modifications

SYNOPSIS

undo Mode

U − mode line letter.

DESCRIPTION

undo mode, when enabled, stores a history of all user edits so that the command undo(2) may be used to undo the last *n* edits to a buffer. If this mode is not enabled the **undo** command has no effect.

Obviously memory is required to store this information, particularly storing deleted, reformed or replaced text, users editing large files or operating in restricted memory environments may wish to use this mode selectively.

NOTES

The **undo** information is flushed, and is effectively lost, when a save operation is performed on the buffer.

SEE ALSO

buffer−mode(2), global−mode(2), undo(2).

uniq(3)

NAME

uniq − Make lines in a sorted list unique

SYNOPSIS

uniq

DESCRIPTION

uniq reduces a sorted lines of text in the current buffer to a unique list such that no entries are repeated. The list is made unique from the mark position to the current cursor position (point). The operation is case sensitive.

NOTES

uniq is a macro implemented in `tools.emf`.

For **uniq** to operate correctly then the list must have been previously sorted, see sort−lines(2).

SEE ALSO

sort−lines(2), sort−lines−ignore−case(3), transpose−lines(2),

universal−argument(2)

NAME

universal−argument − Set the command argument count

SYNOPSIS

universal−argument (**C−u**)

DESCRIPTION

universal−argument sets the argument number passed to a command to 4^*n* (4 to the power of *n*) where *n* is the number of calls to **universal−argument**, e.g. the key sequence "`C−uC−n`" moves down 4 lines, "`C−uC−uC−uC−n`" moves down 4*4*4 = 64 lines.

After invoking the **universal−command** a '−' character can be pressed to negate the argument value, and an alternative numeric argument can be entered using the '*0*' to '*9*' keys.

Invoking this command via execute−named−command(2) or by a macro has no effect. The command should be treated as a command key prefix (like prefix(2)) in that it may be bound to only one key sequence which must be a single key stroke. Re−binding this command to another key unbinds the new key and also the current **universal−argument** key.

The **prefix 1** key (by default bound to `esc`) may also be used to enter a numeric argument at the message line, e.g. "`esc 1 0 C−f`" will move forward 10 characters.

SEE ALSO

prefix(2).

user−setup(3)

NAME

user−setup − Configure MicroEmacs for a specific user

SYNOPSIS

user−setup

DESCRIPTION

user−setup provides a dialog interface to enable the user to configure the editor. **user−setup** may be invoked from the main *Help* menu or directly from the command line using execute−named−command(2). **user−setup** configures the user's setup registry file, "<*logname*>`.erf`" which is used by MicroEmacs to initialize the environment to a user's preference.

Note, if your screen is too small to display the whole dialog, it may be moved using any key bound to the scroll commands such as **scroll−up**, e.g. `A−up`, `C−z`, `A−down`, `C−v`, `A−left` etc. For systems without mouse support, the `tab` key may be used to move between fields.

On all pages the following buttons are available at the bottom of the dialog and have the following effect:

`Save`

Saves the changes made to the users registry file, i.e. "<*Log−Name*>`.erf`" but does not re−initialize MicroEmacs. Some changes, such as color scheme changes, only take effect when the **Current** button is used or when MicroEmacs is restarted.

`Current`

Makes the current user and the changes made Current to this MicroEmacs session, dismissing the **user−setup** dialog and reinitializing MicroEmacs. This also saves the registry file out!

`Exit`

Quits user−setup, if changes where not **Save**d or made **Current** they will be lost.

The following pages, which appear in the dialog, are defined as follows:−

Start−up

`Log Name`

Sets the name of the current user to setup, this can be set to any valid file base name (no extension) which need not be the current user. The rest of the **user−setup** entries are then initialized to the settings defined for the given user (or standard defaults if not defined).

`Default User`

Creates a small macro file, "`default.emf`", setting $MENAME(5) to the current setting of **Log Name**. This may be executed at start−up to determine the current user. See $MENAME(5) for more information.

`Setup Path`

Sets the location of the user files, the files are searched for and created in this directory. $MEPATH(5) should be defined to include this path.

`Setup File`

Sets the personal user setup macro file name which is executed at start−up. A user macro file should contain all personal settings such as preferred key bindings etc. See Setting Up A User Profile for more information. The **Edit** check box can be used to enable/disable the automatic loading of the setup file ready for editing when the **Current** button is used.

`Company File`

Sets the company setup macro file name which is executed at start−up. A company macro file should contain all company wide standard settings such as `%company−name`, No `.emf` extension is supplied. See Setting Up a Company Profile for more information.

`Emulate`

Sets an emulation mode which changes the behaviour on MicroEmacs to emulate another editor/program; this is done by executing a macro file at start−up. An emulation macro file should contain the macro code required to simulate the environment of the other editor. MicroEmacs '02 is released with two emulation modes, MicroEmacs v3.8 which executes macro file `meme3_8.emf` (See Compatibility for more information) and NEdit v5 which is at best a demonstration of what can be achieved, this executes macro file `menedit.emf`.

`MS Friendly Keys`

When enabled the following key bindings are created to ease frustration for MS users:

`home`

Bound to beginning−of−line instead of beginning−of−buffer.

`end`

Bound to end−of−line instead of end−of−buffer.

`C−home`

Bound to beginning−of−buffer.

`C−end`

Bound to end−of−buffer.

`C−v`

Bound to yank (paste).

`esc−v`

Bound to reyank.

Note that the "`C−x`" and "`C−c`" keys are just to intrinsic to MicroEmacs to rebind (sorry).

`MS Shift Region`

Enables/disables cursor key manipulation with the shift key similar to the conventional Microsoft region selection. When enabled, pressing the shift key in conjunction with the cursor movement keys selects a region which is hilighted. Once the region is selected then the `<DELETE>` or `<BACKSPACE>` key erases the selected region. This also enables a similar behaviour with the Mouse **Drag region** driver, see below. **Locale Setup**

`Keyboard`

Configures MicroEmacs to the user's keyboard. Accent character generation keys present on foreign keyboards cannot be automatically supported on Windows platforms. MicroEmacs must be informed of the keyboard being used to correctly interpret the keys. If a required keyboard is not supported please see FAQ38 on how to setup the keyboard, also see Locale Support.

`Language`

Sets the user language, this sets the word (or letter) characters and if available sets up spell(2) with appropriate spelling rules and dictionaries. For more information on adding support for a language see Locale Support.

NOTES

Earlier versions MicroEmacs had "`(Ext)`" languages which use extended language dictionaries, vastly increasing the word list. New versions automatically test for and use these dictionaries if available.

In earlier versions a personal dictionary name could be set in the next field, this option was removed on Oct 2001. Instead a personal dictionary for each language is automatically created for you, any words or auto−corrected words will be added to the current languages personal dictionary. The name of dictionary is "`lsdp`*<lang−id>*`.edf`" where "*<lang−id>*"

is the 4 letter MicroEmacs language name (e.g. "`enus`" for American), simply rename any existing personal dictionary to this new name.

`Auto Spell`

Enables Auto Spell Checking in file types which support this feature (usually text based files such as txt(9) or nroff(9) files etc). Auto spell detects word breaks as you type and checks the spelling of every completed word hilighting any erroneous words in the error color scheme (usually red). The feature can be manually enabled and disabled by invoking the auto−spell(3) command (usually bound to "`f5`").

`Auto Save Dics`

Enables auto−saving of any changed dictionaries on exit. If this is disabled the user is prompted to save for each changed dictionary. **General**

`Full Name`

This should be set to the user's name and is used in a variety of places, e.g. by etfinsrt(3) to set the "Created By" field in a template.

`Organizer`

Sets the organizer file base name, defaults to the **Log Name**. When notes and addresses are stored using organizer(3) the file "<*Organizer*>`.eof`" is used.

`Auto−Save Time`

Sets the length of time in seconds between buffer auto−saves, a setting of `0` or an empty string disables auto−saving. The default setting is 300 seconds or 5 minutes. This indirectly sets the auto−time(5) variable and the autosv(2m) global mode.

`Global Modes`

Sets the initial state of the global quiet(2m) mode. This indirectly executes global−mode(2) to set the required modes.

`Buffer Modes`

Sets the initial state of the global modes auto(2m), backup(2m), tab(2m) and undo(2m), any buffers created will inherit the state of these modes. However, as changing these modes directly effects only the global modes, any existing buffers (including ones re−created using the `−c` command−line option, see me(1)) will not be effect by the setting of these modes. For them to take effect, the buffers should be reloaded. These modes can be changed on a per file type basis using the command buffer−setup(3), also some file hooks override these global settings, such as the makefile(9) hook which overrides the **tab** mode. This indirectly executes global−mode(2) to set the required modes.

`Search Modes`

Sets the initial state of the global search modes exact(2m) and magic(2m). This indirectly executes global−mode(2) to set the required modes.

`Keep Undo History`

If this is enabled the undo history is kept after a save allowing the undo(2) command to back−up changes beyond the last save. When clear the undo history is discarded after the buffer is saved. This indirectly sets bit 0x8000 of the $system(5) variable.

`Hide Backups`

Enables hiding MicroEmacs generated backup files. On Windows and Dos platforms the Hidden file attribute is used to hide the file, whereas on UNIX the backup file name is prepended with a '`.`'. This indirectly sets bit 0x100000 of the $system(5) variable.

`Main Menu`

Enables the top main menu bar.

`Alt -> Main Menu`

If enabled the main menu Alt hot−key bindings are enabled. These are dynamic bindings automatically generated from the main menu. Typically the first item in the main menu is "`File`" with a hot key of '**F**', with this enabled '`A−f`' will open this menu item. Note that global and local key bindings override these. This indirectly sets bit 0x2000 of the $system(5) variable.

`Alt -> Esc Prefx`

If enabled the Alt key acts as a prefix 1 modifier key. By default '`A−n`' is not bound, with this bit set the key is inferred to '`esc n`' which is bound to **forward−paragraph**. Note that global, local and menu hot−key bindings override these. This indirectly sets bit 0x4000 of the $system(5) variable.

`Abbrev Expansion`

Configures which expansion methods are enabled by default when the expand−abbrev−handle(3) is executed. **Accent** enables expand−iso−accents(3), **Lookbk** enables expand−look−back(3) and **Dict'n** enables expand−word(3).

`Tab To Indent`

Sets the tab(2) behavior in a buffer which has cmode(2m) enabled or an indentation method. This indirectly sets bits `0x1000` and `0x200000` of the $system(5) variable.

`Show Modes`

Selects which modes are to be displayed on the mode−line whenever a "`%e`" token is used in the $mode−line(5) variable. This indirectly sets the $show−modes(5) variable. **Platform − UNIX Setup**

Only present on UNIX platforms using the X interface, see below for the Console setup.

`Font`

Sets the X font name to be used. This indirectly executes change−font(2) with the given font name. e.g.

```
      "-misc-fixed-bold-r-normal--13-*-*-*-c-70-iso8859-1"
```

`Display Char Set`

Selects the display character set being used by the system to render the MicroEmacs window, dependent on the **Font** being used. The setting of this option effects the configuration of MicroEmacs's internal character maps (using command set−char−mask(2)) enabling the character sets of foreign languages to be correctly supported. It also changes the definition of variables $box−chars(5) and $window−chars(5) to their best values for the given font.

`Extend Char Set`

When enabled MicroEmacs replaces the display of characters `0x00` to `0x1f` with forms which are useful for variables $box−chars(5) and $window−chars(5) greatly improving the look of osd(2) dialogs, the scroll bars etc.

`Use Fonts`

When enabled the bold, italic, light and underline characteristics of the font will be used depending on their availability and the Color Scheme being used. This indirectly sets bit `0x10` of the $system(5) variable.

`Draw White Spaces`

Enables the drawing of visible white spaces, i.e. space, tab and new−line characters. This indirectly sets bit `0x80000` of the $system(5) variable.

`Enable Toolbar`

Enables the Toolbar − configurable, managed windows giving easy access to many features and tools.

`Client Server`

The client/server enables the file based external macro command driver to be enabled − see Client−Server. This by default is disabled, when enabled it is used by command−line options **−m** and **−o**.

`DOS File Names`

DOS has a restricted 8.3 file naming system (i.e. "`BBBBBBBB.XXX`"), if this option is enabled the MicroEmacs '02 will adhere to this system for auto−save and backup file names whenever possible. See $auto−time(5) for more information on the naming convention used. This indirectly sets bit

0x400 of the $system(5) variable.

`# Backups`

This option only has an effect when **DOS File Names** is disabled. Setting this to a number greater than zero enables multiple backup files to be created, the number determined by this value. If set to zero (or less) then only a single backup file is created. This indirectly sets the $kept−versions(5) variable.

`Ignore Files`

Sets a list extensions of files to be ignored in file completion, e.g. MicroEmacs backup files (~). This indirectly sets the $file−ignore(5) variable.

`Cursor Blink Rate`

Sets the cursor blink period in millisecond. The first entry box sets the cursor visible time, a setting of zero disables blinking. The second box sets the hidden time. A visible time of 600 and hidden time of 200 gives a reasonable blink cycle. This indirectly sets the $cursor−blink(5) variable.

`Fence Display`

Sets the preferred method of displaying a matching fence, a fence is one of the following brackets:

```
{...}   (...)   [...]
```

Jumping to the opening fence only occurs when the closing brace is typed, whereas the drawing of matching fences occurs whenever the cursor is on an open fence or one character past the close fence. When this option is set to "`Never Display`" the buffer−setup(3) setting is ignored.

`Scroll Bars`

Selects the scroll bar support required. When Splitter is enabled, the first character of the scroll bar and mode−line is a split character used for splitting the window into two using the mouse. This indirectly sets the $scroll−bar(5) variable.

`Horizontal Scroll`

Selects the horizontal scrolling method used with the scroll−left(2) and scroll−right(2) commands. This indirectly sets the $scroll(5) variable.

`Vertical Scroll`

Selects the vertical scrolling method used with the forward−line(2) and backward−line(2) commands. This indirectly sets the $scroll(5) variable.

`Color Scheme`

Sets the color scheme setup macro file name which is executed at start−up. MicroEmacs by default comes with 4 color schemes. Color schemes can be created and altered using the scheme−editor(3) dialog. **Platform − UNIX Console Setup**

Only present on UNIX platforms when using the termcap interface, all the Console platform settings are kept independent of the X interface settings.

`Termcap Color`

This option determines whether Termcap based colors should be used. These are typically the standard eight colors and may not be supported on all terminals. If this option is disabled Termcap fonts (such as bold) are used instead to create a primitive hi−lighting. This indirectly sets bit 0x004 of the $system(5) variable.

`Use Fonts`

See **Platform UNIX Setup** above.

`Display Char Set`

See **Platform UNIX Setup** above.

`Draw White Spaces`

See **Platform UNIX Setup** above.

`Client Server`

See **Platform UNIX Setup** above.

`DOS File Names`

See **Platform UNIX Setup** above.

`# Backups`

See **Platform UNIX Setup** above.

`Ignore Files`

See **Platform UNIX Setup** above.

`Cursor Blink Rate`

See **Platform UNIX Setup** above.

`Scroll Bars`

See **Platform UNIX Setup** above.

`Horizontal Scroll`

See **Platform UNIX Setup** above.

`Vertical Scroll`

See **Platform UNIX Setup** above.

`Color Scheme`

See **Platform UNIX Setup** above. **Platform − Win32 Setup**

Only present on Microsoft Windows based machines.

`Font Name`

Sets the windows font name and size. This indirectly executes change−font(2) with the given font name. MicroEmacs may only use a Fixed Mono Font, either an OEM font as used by the MS−DOS command line, or the more conventional ANSI fonts. The fonts are selected using the **Change Font** button which invokes a dialog to allow the available fonts to be selected. True−Type mono fonts such as `Courier New` or `Lucida Console` are typically used.

`Weight & Size`

Allows the size and weight of the font to be selected, specified as *weight*, *width* and *height*. The *weight* is typically 4, this corresponds to a regular weighting, 7 is bold. *width* is the width of the font in pixels, this may be 0 when the height is specified as −ve. *height* is the height of the font, typically a −ve value (where the *width* is 0), which produces a proportionally sized font, values of in the range −11 .. −14 generally produce reasonably sized fonts. The *hight* and *width* may be specified as +ve values and allow explicit font dimensions to be specified, generally used to achieve a precise font size requirement.

`Use Fonts`

See **Platform UNIX Setup** above.

`Display Char Set`

See **Platform UNIX Setup** above.

`Extend Char Set`

See **Platform UNIX Setup** above.

`Choose Font`

Opens a windows dialog allowing the user to select a font, the selection is used to configure the above font fields.

`Draw White Spaces`

See **Platform UNIX Setup** above.

`Capture Alt Space`

Used to enable/disable the capture and interpretation of the '`A-space`' key sequence. If this key sequence is not captured by MicroEmacs it is passed back to Windows which opens the top left window menu, allow keyboard access to Window commands like Maximize.

`Client Server`

See **Platform UNIX Setup** above. Note that on windows based systems the client/server is also used by memsdev(1) to drive the editor from the Microsoft Developer environment.

`DOS File Names`

See **Platform UNIX Setup** above. Note that some early version of Windows '95 have problems with ~ extensions. Service release 2 fixed these problems − if you experience problems then return to 8.3 filename mode − note that MicroEmacs will still store longer file names, only the backup naming convention changes.

`# Backups`

See **Platform UNIX Setup** above.

`Ignore Files`

See **Platform UNIX Setup** above.

`Cursor Blink Rate`

See **Platform UNIX Setup** above.

`Scroll Bars`

See **Platform UNIX Setup** above.

`Horizontal Scroll`

See **Platform UNIX Setup** above.

`Vertical Scroll`

See **Platform UNIX Setup** above.

```
Color Scheme
```

See **Platform UNIX Setup** above. **Platform − Win32 Console Setup**

Only present on Windows NT and Win95+ platforms when using the console interface, all the Console platform settings are kept independent of the Window interface settings.

```
Display Char Set
```

See **Platform UNIX Setup** above.

```
Draw White Spaces
```

See **Platform UNIX Setup** above.

```
Client Server
```

See **Platform Win32 Setup** above.

```
DOS File Names
```

See **Platform Win32 Setup** above.

```
# Backups
```

See **Platform UNIX Setup** above.

```
Ignore Files
```

See **Platform UNIX Setup** above.

```
Cursor Blink Rate
```

See **Platform UNIX Setup** above.

```
Scroll Bars
```

See **Platform UNIX Setup** above.

```
Horizontal Scroll
```

See **Platform UNIX Setup** above.

```
Vertical Scroll
```

See **Platform UNIX Setup** above.

```
Color Scheme
```

See **Platform UNIX Setup** above. **Platform − DOS Setup**

Only present on DOS machines.

`Graphic Mode #` and `Double Lines`

Sets the DOS graphics mode number and whether the number of text lines can be doubled. This indirectly executes change−font(2) with the given font name.

`Display Char Set`

See **Platform UNIX Setup** above.

`Draw White Spaces`

See **Platform UNIX Setup** above.

`Ignore Files`

See **Platform UNIX Setup** above.

`Cursor Blink Rate`

See **Platform UNIX Setup** above.

`Scroll Bars`

See **Platform UNIX Setup** above.

`Horizontal Scroll`

See **Platform UNIX Setup** above.

`Vertical Scroll`

See **Platform UNIX Setup** above.

`Color Scheme`

See **Platform UNIX Setup** above. **Mouse**

The mouse device creates keys in a similar way to regular keyboard keys and, like keyboard keysm they must be bound before they are used. MicroEmacs '02 does not have the mouse functionality hard coded into the editor, it provides a macro interface to the mouse for ultimate flexibility and a set of default functionality which can be bound to the mouse in a variety of ways.

All the mouse controlling macros are stored in `mouse.emf` and `mouseosd.emf` although some buffers have local functionality over−rides, such as file−browser(3). The user can expand the range of mouse functionality but how this is achieved is beyond the scope of this documentation.

The **user−setup** dialog allows the user to configure the mouse to use the default functionality, as follows:−

`Enable Mouse`

Enables or disables the mouse, when disabled the mouse can not be used and will not generate any key events. This does not apply to UNIX Termcap systems as the mouse cut and paste operation is performed by the Xterm. This indirectly sets bit 0x010 of the $mouse(5) variable.

`Number Buttons`

Sets the number of buttons on the mouse, may be 1, 2 or 3. MicroEmacs usually obtains the correct number for the system, but sometimes this can be wrong. This entry can be used to correct this problem. For one button mice, the button is considered to be the `left` mouse button, two button mice have an `left` and `right` button. This indirectly sets the $mouse(5) variable.

`Swap Buttons`

If enabled then the `left` and `right` buttons are swapped, i.e. when the left button is pressed it executes the right button bindings. This indirectly sets bit 0x020 of the $mouse(5) variable.

`Simulate 3 Buttons`

If enabled then pressing the `left` and `right` buttons together with generate a middle button press event, this feature is for people with a 2 button mouse who want more. The two buttons must be pressed or release within 10 millisecond of each other.

The following four fields determine which mouse button binding the user wishes to view and change:−

`Button`

The mouse button, `Left`, `Right` or `Middle` for the normal buttons and `Whell Up` or `Whell Down` for the pilot wheel events.

`Shift Pressed`

The action of the mouse can be different for every modifier key setting, if this is enabled then the binding being modified is for the **Button** being pressed with the **Shift** key held down.

`Control Pressed`

If enabled then modifying the action when the **Button** is pressed with the **Control** key held down.

`Alt Pressed`

If enabled then modifying the action when the **Button** is pressed with the **Alt** key held down.

The following two fields determine the functionality of the button defined by the previous four fields:−

`Handle Scroll`

When enabled, if the button is pressed with the mouse on the main menu, a scroll bar or mode−line the standard action is performed, such as opening the main menu or scrolling up or down the window etc. The **bound To** command is only called if the mouse is in a main window. If disabled, the **Bound To** command is always called.

`Bound To`

The function to be performed. The functions available depend on the type of button being bound, the following is a list of functions available for normal buttons:−

`Not bound`

The Button is not bound.

`Drag region`

set−mark(2) is called at the pick location, until the button is dropped, the area of text between this point and the current mouse position is hi−lighted. When the mouse button is dropped, if the drop position is the same as the pick then the double click is tested for, if a double click is entered then the **Select Word** function is executed, otherwise the cursor is simply moved to the drop position. If the pick and drop position are different then the enclosed text is copied to the kill buffer using copy−region(2). Note this behaviour is altered by the setting of **MS Shift Region** on the **Start−Up** page.

`Select Word`

Also executed from a double click bound to **Drag Region**, **Select Word** copies the word under the mouse into the kill buffer using copy−region(2), unless a double click is entered in which case the whole line is copied.

`Default Pan`

While the mouse button is pressed the current buffer pans with any mouse movement.

`MS Pan`

MicroSoft style Pan; while the mouse button is pressed the current buffer pans vertically according to the mouse position relative to the point where the button was pressed.

`Find Tag`

Executes find−tag(2) with the word currently under the mouse.

`Find ME Help`

Executes help−item(2) with the word currently under the mouse.

`Undo`

Simply executes undo(2) without moving the cursor to the position of the mouse. Subsequent calls to this binding will undo multiple edits.

`No move yank`

Simply executes yank(2) without moving the cursor to the position of the mouse.

`Replace yank`

Simplar to "`No move yank`" except when the is a current region (typically defined by "`Drag region`" above), in which case the region is first deleted.

`Move to yank`

Moves the cursor to the current position of the mouse and executes yank(2).

`Reyank`

Executes reyank(2) without moving the cursor. Note, to enable this functionality some sanity checks have had to be removed, as a result it should not be misused as seeming bizarre things can occur.

`Fold current`

Toggles the fold status of the current block, only applicable in buffers supporting fold−current(3), such as c and emf files.

`Fold all`

Toggles the fold status of the whole buffer, opening or closing all found blocks. Only applicable in buffers supporting fold−all(3), such as c and emf files.

`Main menu`

Simply opens the main menu from any where on the screen.

`Multi−Menu`

Opens a context sensitive menu dependent on the position of the mouse, i.e. opens the main menu if over it, opens a different menu when executed on the mode−line etc.

The following is a list of functions available for pilot wheel events:−

`Not bound`

The Button is not bound.

`Scroll Up 1 Line` ....

Scrolls the current buffer by the specified amount.

`Defaults`

Rests the mouse configuration to the default settings. **File Types**

The file type list is used in two places, the main menu's `File => Quick Open` sub−menu list and the `File => Open => File Type` list. In each case the file type "`All Files`" is automatically added. The user can add, remove and change the list of file types by using this dialog. An entry can be selected for editing or deletion by simply selecting it with the left mouse button. A new entry may be added by simply filling in the 3 entry boxes and selecting Add. Items in the Dialog are as follows.

`No.`

The file type entry number. A new entry is always added to the end of the list, ignoring this value. The position of an existing entry can be changed by altering this field to the desired position and selecting the `Change` button to move it to its new position.

`Name`

The file type name, the string printed in the sub−menus.

`File Mask List`

A comma ('`,`') separated list of file masks which match the file type, e.g. for C and C++ source files use "`*.c,*.cc,*.cpp`".

`Add`

Adds a new entry to the list, only the **Name** and **FileMask List** fields are used, the **No.** field is ignored as the new entry is always added to the end of the list. The position can be altered by using the **Change** button.

`Change`

Alters an existing file type entry, all 3 fields must be set.

`Delete`

Deletes the current entry number, only the **No.** entry is used. **Tools**

The Tools dialog allows the user to configure up to 10 system commands, or tools, which can be executed via MicroEmacs Main Tools Menu. The dialog configures the user's registry for the command execute−tool(3) to be used. The execution of a tool can also be bound to a key, see **execute−tool** for more information.

The top half of the dialog consists of the 10 Tools (`0−9`) configuration buttons. Selecting one of these selects the current tool to be configured, the current tool is shown by the title in the middle of the dialog.

The lower half of the dialog configures the currently selected tool, as follows:−

`Tool Name`

Sets the displayed name of the tool. The tool name is used in the buttons in the top half of this dialog and in the MicroEmacs Main Tools Menu.

`Tool Command Line`

> Sets the system command−line to be launched whenever the tool is executed, the following special tokens may be used in the command−line which are substituted at execution:−
>
> **%ff**
>
> The current buffer's full file name, including the path.
>
> **%fp**
>
> The current buffer's file path.
>
> **%fn**
>
> The current buffer's file name without the path.
>
> **%fb**
>
> The current buffer's file base name, i.e. the file name without the path or the extension.
>
> **%fe**

The current buffer's file extension with the '.' (e.g. "*.emf*"), set to the empty string if the file name does not have an extension.

> Note that "**%ff**" is always the same as "**%fp%fn**" and "**%fp%fb%fe**". If any of these tokens are used, the tool will fail to execute if the current buffer does not have a file name.

`Save Current Buffer` and `Prompt`

If the current buffer has been edited, enabling `Save Current Buffer` will automatically save the current buffer before executing the tool. This is particularly useful when the tool operates on the current buffer's file (e.g. compiles the file). If `Prompt` is also enabled the user will be prompted before the file is saved.

`Save All Buffers` and `Prompt`

If `Save All Buffers` is enabled, all edited buffers will be automatically saved before executing the tool. This is particularly useful when the tool may operate on multiple files (e.g. compilation of a project). If `Prompt` is also enabled the user will be prompted before each file is saved.

`Capture Output`

If enabled any output produced from the execution of the tool will be captured and inserted into a new buffer. When enabled the following two items, `Buffer` and `Hide`, may be specified. When disabled the command used to execute the tool is shell−command(2), otherwise the command used is either pipe−shell−command(2) or ipipe−shell−command(2) depending on the setting of `Run Concurrently`.

`Buffer`

Specifies the buffer name the captured output should be dumped to, this option is only visible when `Capture Output` is enabled. The following special tokens may be used in the buffer name which are substituted at execution:−

%fn

The current buffer's file name without the path, set to the buffer name if the current buffer does not have a file name.

%fb

The current buffer's file base name, i.e. the file name without the path or the extension. Set to the buffer name if the current buffer does not have a file name.

%fe

The current buffer's file extension with the '.' (e.g. ".*emf*"), set to the empty string if the current buffer does not have a file name or it does not have an extension.Note that "**%fn**" is always the same as "**%fb%fe**". Default buffer name when this field is left empty is "**command**", or "**icommand**" if `Run Concurrently` is enabled.

`Hide`

When enabled the tool output capture buffer is hidden, this option is only visible when `Capture Output` is enabled.

`Run Concurrently`

If enabled, when the tool is executed the command is launched and run concurrently, allowing the user to continue working in MicroEmacs during the tools execution. This option is not available for all versions on

MicroEmacs and forces the output to be captured. Enabling this option will force the use of command ipipe−shell−command(2) to launch the tool. **E−Mail**

MicroEmacs '02 provides a simple E−Mail manager, see vm(3) for more information and example entries. It must be stressed that **vm** has only been tested in one environment, caution should be used as system differences may cause problems, such as loss of data, which the author does not except any responsibility for.

The **E−Mail Setup** dialog configures a user to use part or all of the **vm** E−Mail manager, as follows:−

Platform ALL Mail Setup

The following field is used for both sending and receiving mail:

`User Mail Dir`

Sets the user mail−box directory where all files are to be found and stored (except usually the **Incoming Mail box**). The value of this field is platform independent and must be setup for each one.

The following fields are used for sending mail:

`Send Mail Signature`

Sets the signature file name which is inserted at the bottom of every out−going email message, if empty the no signature is inserted. The value of this field is platform independent, is value use by all. The file must be located in the **User Mail Dir** and no path entered for it to work across platforms.

`Carbon−Copy File`

Sets the sent−mail carbon−copy file, creating the "`Fcc:`" line of the mail buffer. All out−going emails are appended to the end of this file if the "`Fcc:`" line is not altered. If this field is left empty then no "`Fcc:`" line is created. The value of this field is platform independent, the file must be located in the **User Mail Dir**.

`Insert Data (^C^I)`

Sets the first embedded data command line, bound to "`C−c C−I`". The value of this field is platform dependent.

`Insert Data (^C^Z)`

Sets the second embedded data command line, bound to "`C−c C−z`". The value of this field is platform dependent.

`Send Mail Command`

Sets the command−line used for sending email messages. The value of this field is platform dependent.

The following fields are used for receiving mail:

`Check for mail`

Sets the time interval between the automatic checking for incoming mail in seconds, when set to 0 the automatic checking is disabled. When enabled, the check is performed by mail−check(3) which also sends any queued mail and gets any new mail if the **Get Mail Command** is used. The value of this field is platform dependent.

`Get Mail Command`

The command used to get new mail from the server, if empty it is assumed the **Incoming Mail Box** is automatically updated by the system. If used the command must append new mail to the end of the **Incoming Mail Box** specified below. The value of this field is platform dependent.

`Incoming Mail Box`

Sets the incoming mail box file which new incoming mail is appended to, either automatically by the system or by the **Get Mail Command**. The value of this field is platform dependent.

`VM Main In Box`

Sets the main current mail box, or inbox. The value of this field is platform independent, the file must be located in the **User Mail Dir**.

`VM Gets Mail`

When enabled, executing the command vm will not only create the mail box windows, it will also get and process any new mail. When disabled only the vm '`g`' command can be used to get and process new mail.

`Mime Data Extract`

Sets the command−line used for extracting Mime encoded embedded data. The value of this field is platform dependent.

`Uuencode Extract`

Sets the command−line used for extracting Uuencoded embedded data. The value of this field is platform dependent.

`Auto−Archive Setup`

Sets up the auto−archive of messages in the current inbox to other mail boxes. **NOTES**

user−setup is a macro using osd(2), defined in `userstp.emf`.

SEE ALSO

User Profiles, Company Profiles, Installation, buffer−setup(3), scheme−editor(3).

usr(2m)

NAME

usr1 − usr8 − User buffer modes

SYNOPSIS

usr1−usr8 Mode

1−8 − mode line letters.

DESCRIPTION

usr1 through **usr8** modes have no predefined purpose, they are present to provide the user with the ability to store some buffer state. All of these modes are off by default. For example, the user may wish to have two commands bound to the same key, with another command to toggle which one is currently active.

NOTES

The toolbar 'Buffer File Info' tool uses **usr8** mode to track the status of the buffer, when using this tool the mode should not be used.

SEE ALSO

buffer−mode(2), global−mode(2).

vhdl(9)

SYNOPSIS

vhdl − VHDL hardware simulation files

FILES

hkvhdl.emf − VHDL hook definition
vhdl.etf − VHDL template file.

EXTENSIONS

.vhdl, **.vhd** − VHDL file

DESCRIPTION

The **vhdl** file type template provides simple hilighting of VHDL files, the template provides minimal hilighting.

BUGS

None reported. Template could probably benifit from some form of auto indentation.

SEE ALSO

Supported File Types

view(2m)

NAME

view − Read only

SYNOPSIS

view Mode

V − mode line letters.

DESCRIPTION

view mode sets the buffer to read−only, disabling the ability to alter the contents of the buffer. This mode is automatically set for any files attributed with a read−only status on the file system when read into MicroEmacs '02. Files loaded via view−file(2) are also assigned **view** mode.

While in **view** mode, any attempt to alter the buffer contents results in the following message:−

```
[Key Illegal in view Mode]
```

SEE ALSO

buffer−mode(2), global−mode(2), view−file(2).

view−file(2)

NAME

view−file − Load a file read only

SYNOPSIS

n **view−file** "*file−name*" (**C−x C−v**)

DESCRIPTION

view−file is like find−file(2), and either finds the file in a buffer, or creates a new buffer and reads the file in. A new file is left in view(2m) mode if the file was found (i.e. cannot be edited).

The numeric argument *n* can be used to modify the default behaviour of the command, where the bits are defined as follows:

0x01

If the file does not exist and this bit is not set the command fails at this point. If the file does not exist and this bit is set (or no argument is specified as the default argument is `1`) then a new empty buffer is created with the given file name, saving the buffer subsequently creates a new file.

0x02

If this bit is set the file will be loaded with binary(2m) mode enabled. See help on **binary** mode for more information on editing binary data files.

0x04

If this bit is set the file will be loaded with crypt(2m) mode enabled. See help on **crypt** mode for more information on editing encrypted files.

0x08

If this bit is set the file will be loaded with rbin(2m) mode enabled. See help on **rbin** mode for more information on efficient editing of binary data files. **SEE ALSO**

buffer−mode(2), find−file(2), read−file(2), view(2m), binary(2m), crypt(2m), rbin(2m).

void(2)

NAME

void − Null command

SYNOPSIS

n **void**

DESCRIPTION

void does nothing except return FALSE if the given argument *n* is zero, TRUE otherwise. Used to bind any frequently miss hit keys to something harmless.

SEE ALSO

global−bind−key(2).

vrml(9)

SYNOPSIS

vrml − VRML File

FILES

hkvrml.emf − VRML File hook definition

EXTENSIONS

<none> − Uses the *Magic String* only.

MAGIC STRINGS

#VRML

A generic tag that appears on the first line at the top of a **VRML** (or **wrl**) file. MicroEmacs automatically recognises the tag and adopts the appropriate mode. **DESCRIPTION**

The **vrml** file type template handles the hilighting of **VRML** files.

Hilighting

The hilighting features allow commands, variables, logical, comments, strings and characters of the language to be differentiated and rendered in different colors.

Auto Layout

The indentation mechanism is enabled which performs automatic layout of the text. restyle−region(3) and restyle−buffer(3) are available to reformat (re−layout) selected sections of the buffer, or the whole buffer, respectively.

Short Cuts

The short cut keys used within the buffer are:−

A−C−tab − Restyle a region.

BUGS

No bugs reported

SEE ALSO

indent(2), restyle−buffer(3), restyle−region(3).

Supported File Types

wrap(2m)

NAME

wrap − Line wrap entered text

SYNOPSIS

wrap Mode

W − mode line letters.

DESCRIPTION

wrap mode causes automatic text wrapping when text passes then fill column (see $fill−col(5)), allowing text to be entered non−stop on a standard screen without bothering to use the `RETURN` key.

wrap mode is usually used in conjunction with the justify(2m) and indent(2m) modes for editing text documents.

wrap mode also automatically wraps long lines in the output of an ipipe−shell−command(2) to the width of the MicroEmacs window.

SEE ALSO

buffer−mode(2), global−mode(2), ipipe−shell−command(2) justify(2m), indent(2m), pipe(2m).

wrap−word(2)

NAME

wrap−word − Wrap word onto next line

SYNOPSIS

wrap−word

DESCRIPTION

wrap−word wraps the current word onto the next line, justifying the current line if the justify(2m) mode is enabled. The justification method is defined by $fill−mode(5).

SEE ALSO

buffer−mode(2), fill−paragraph(2), $fill−mode(5), justify(2m).

write−buffer(2)

NAME

write−buffer − Write contents of buffer to named (new) file

SYNOPSIS

n **write−buffer** "*file−name*" (**C−x C−w**)

DESCRIPTION

write−buffer is used to write the contents of the buffer to a NEW file, use save−buffer(2) if the buffer is to be written to the existing file already associated with the buffer.

write−buffer writes the contents of the current buffer to the named file *file−name*. The action of the write also changes the file name associated with the current buffer to the new file name.

Unlike append−buffer(2), **write−buffer** always replaces an existing file and the new file inherits the buffers file characteristics instead of the old file's.

On writing the file, if time(2m) mode is enabled then the time stamp string is searched for in the file and modified if located, to reflect the modification date and time.

If the buffer contains a narrow(2m) it will automatically be removed before saving so that the whole buffer is saved and restored when saving is complete

If backup(2m) mode is enabled and the buffer is associated with a different file (compared with *file−name*) then any automatic save copies of the file associated with the *buffer* are deleted.

The argument *n* can be used to change the default behavior of write−buffer described above, *n* is a bit based flag where:−

0x01

Enables validity checks (default). These include a check that the proposed file does not already exist, if so confirmation of writing is requested from the user. Also MicroEmacs '02 checks all other current buffers for one with the proposed file name, if found, again confirmation is requested. Without this flag the command will always succeed wherever possible.

0x02

Disables the expansion of any narrows (see narrow−buffer(2)) before saving the buffer. **NOTES** undo(2) information is discarded when the file is written.

SEE ALSO

$auto−time(5), backup(2m), time(2m), buffer−mode(2), file−attrib(3), change−file−name(2), save−buffer(2), append−buffer(2).

x86(9)

SYNOPSIS

x86 − Intel .x86 Assembler File

FILES

hkasmx86.emf − Intel .x86 Assembler hook definition
asmx86.etf − Intel .x86 Assembler template file.

EXTENSIONS

.x86 − Intel .x86 Assembler File

MAGIC STRINGS

−!− asmx86 −!−

Recognized by MicroEmacs only, defines the file to be a Intel x86 assembler file. **DESCRIPTION**

The **x86** file type template provides simple hilighting of Intel x86 assembler files.

Hilighting

The hilighting features allow commands, variables, logical, comments, strings and characters of the language to be differentiated and rendered in different colors.

Auto Layout

The indentation mechanism is enabled which performs automatic layout of the text. restyle−region(3) and restyle−buffer(3) are available to reformat (re−layout) selected sections of the buffer, or the whole buffer, respectively.

Short Cuts

The short cut keys used within the buffer are:−

C−c C−c − Comment out the current line.
C−c C−d − Uncomment the current line.

BUGS

None reported.

SEE ALSO

indent(2), restyle−region(3) restyle−buffer(3) asm(9)

Supported File Types

yank(2)

NAME

yank − Paste (copy) kill buffer contents into buffer

SYNOPSIS

n **yank** (**C−y**)

DESCRIPTION

When a non negative argument is supplied to **yank**, the command copies the contents of the kill buffer *n* times into the current buffer at the current cursor position. This does not clear the kill buffer, and therefore may be used to make multiple copies of a section of text. On windowing systems which support clip−boards, such as windows and X−terms, MicroEmacs will also cut to and paste from the global clip−board.

If *yank* is IMMEDIATELY followed by a reyank(2) then the *yanked* text is replaced by text of the next entry in the kill ring. (another **reyank** replaces the text with the previous reyank text and so on).

If an −ve argument is given, **yank** removes the last 0−*n* items from the kill ring.

Text is inserted into the kill buffer by one of the following commands:−

> backward−kill−word(2), copy−region(2), forward−kill−word(2), kill−line(2), kill−paragraph(2), kill−region(2), forward−delete−char(2), backward−delete−char(2).

All the above commands (except **copy−region**) cut text out of the buffer, the last 2 commands require the letter(2m) mode enabled to add the text to the kill buffer. If any of these commands are executed immediately after any other (including itself) or the @cl(4) variable is set to one of these command, the new kill text is appended to the last kill buffer text.

NOTES

Windowing systems such as X−Windows and Microsoft Windows utilize a global windowing kill buffer allowing data to be moved between windowing applications (*cut buffer* and *clipboard*, respectively). Within these environments MicroEmacs '02 automatically interacts with the windowing systems kill buffer, the last MicroEmacs '02 kill buffer entry is immediately available for a *paste* operation into another application (regardless of how it was inserted into the kill buffer). Conversely, data placed in the windowing kill buffer is available to MicroEmacs '02, via **yank**, until a new item has been inserted into the kill buffer (the data may still be available via reyank(2)).

EXAMPLE

The following example is a basic macro code implementation of the transpose−lines(2) command,

```
beginning-of-line
kill-line
forward-line
yank
-1 yank
backward-line
```

Note that similar to **transpose−lines** it does not leave the moved line in the kill buffer, effectively tidying up after itself.

SEE ALSO

yank−rectangle(2), copy−region(2), kill−region(2), letter(2m), reyank(2), @y(4), @cc(4).

Frequently Asked Questions

FAQ

This page contains frequently asked questions submitted to JASSPA.

FAQs − FAQ Contact information
faq00 − New functionality; what is useful to me as an old MicroEmacs user ??
faq01 − Languages; Are any foreign languages supported other than English ??
faq02 − C++ is not default, C is − how do I change this ??
faq03 − GNU Emacs; are there any GNU Emacs bindings. ?
faq04 − Icons are not displayed correctly in Microsoft Windows environments !!
faq05 − ipipes not working on Microsoft Windows network drives ?
faq06 − Language not supported − will it be ??
faq07 − Language file is incomplete
faq08 − Input locked up and not accepting keys; how do I unlock ?
faq09 − MicroEmacs Bindings; How do I get the original MicroEmacs bindings ?
faq10 − Microsoft Windows Locks up after killing an ipipe.
faq11 − Mouse support under Microsoft windows is strange !!
faq12 − Scroll bars too narrow !!
faq13 − Tab key; Why does the tab key not operate in some windows ??
faq14 − Termcap; On a color terminal why is there no color ??
faq15 − Termcap; Some of the keys do not work − how can I bind them ?
faq16 − Timestamp; Format incorrect, how can I change to MMDDYY.hhmm ?
faq17 − Windows; Component characters rendered incorrectly, how do I fix ?
faq18 − Windows Autosave and Backup files; are these potentially a problem ?
faq19 − Printing; Why in Windows does the output come out in a buffer ??
faq20 − Printing; On Windows which font should I use ??
faq21 − Printing; My printer is not supported ?
faq22 − Alt key maps to the Menu, how do I change ?
faq23 − me32.ini − Where does it go, how do I know it's being processed ??
faq24 − Windows − Where is app850.fon ?
faq25 − Time; mode line is showing the date in DD/MM/YY format how do I change ?
faq26 − C or C++ indentation and effects; how can I turn off ?
faq27 − fill−paragraph function does not fill ??
faq28 − Key modifier which acts as the ESC key; what is it ?
faq29 − find−file start location; where is it ?
faq30 − Re−using a MicroEmacs session; how to ??
faq31 − Microsoft Drag and Drop; is it supported ??
faq32 − Cut and Paste to/from other applications; is it supported ??
faq33 − Fonts; how can I change the font ??
faq34 − Colors; how can I change screen colors ??
faq35 − File Types; how do I interchange between UNIX, Windows and DOS files ??
faq36 − Non−English Languages; What font should I select ??
faq37 − MicroEmacs '99; How do I up−grade from MicroEmacs'98 ??
faq38 − Some keys on my foreign keyboard do not work properly, how do I get them working ??
faq39 − Tabs; How to change the tab width ??

faq40 − Windows/DOS; Where do I get grep/diff etc. ??
faq41 − Home/End Keys; How do I change the default bindings ??
faq42 − tags; How do I generate a MicroEmacs compatible tags file ??

FAQs(0f)

FREQUENTLY ASKED QUESTIONS − Contact Information

This document contains frequently asked questions submitted to JASSPA. Use the E−Mail reflector and associated logs, described in the Contact Information section, alternatively questions may be submitted to:−

```
Email:support@jasspa.com
```

We cannot promise to resolve all questions, but will endeavor to answer most. We would also appreciate comments on how to improve the readability of the documentation or suggestions for improvements where you think the documentation is deficient.

FAQ(00) − New functionality; what is useful to me as an old MicroEmacs user ??

QUESTION (00)

New functionality; what is useful to me as an old MicroEmacs user ??

ANSWER

There are a lot of new features in this distribution. Assuming that you just want to use the editor (and have cottoned onto the fact that there are now scroll bars etc.) then the most frequent commands that we use are:−

- grep(3) − May need to set up in *<user>.emf*.
- diff(3) − May need to set up in *<user>.emf*.
- compile(3) − May need to set up in *<user>.emf*.
- clean(3) − cleans a buffer, removing spaces etc.
- restyle−buffer(3) − Reformats 'C' + known languages.
- spell−buffer(3) − For documentation work, spells the buffer.
- **C−s** − isearch−forward(2) incremental search.
- **C−x u** or `C−_` − undo(2) undoes edits.
- **F10** − file−browser(3) allows the file system to be browsed

Other useful macros include:−

- tabs−to−spaces(3) − Good for sorting out the mess made by Microsoft Developer Studio.
- sort−lines(2) − Two versions of this, allows marked lines to be sorted alphabetically.

Be wary of:−

- **esc−o** − fill−paragraph(2). The default mode is an automatic mode which attempts to guess at the format required. It works most of the time. Also works in 'C'.

Most of the other new features are in the background, such as the macro processor, indentation control, color hilighting, indentation control, auto−saving etc.

FAQ(01) − Languages; Are any foreign languages supported other than English ??

QUESTION (01)

Languages; Are any foreign languages supported other than English ??

ANSWER

Unfortunately as we started with V3.8 as a base many years ago, we missed the distribution with foreign language extensions. We have not incorporated them into the release.

The May 1999 release improves the language support by supporting the ISO−Latin character sets.

We do have spelling dictionaries for French, Spanish, Portuguese and German. Other languages may be supported by transforming native **ispell(1)** dictionaries.

If there is enough interest in this release from people with foreign languages then we may consider including support for foreign language(s). However we would be very much reliant on external help for local testing and translation. We would be open to suggestions.

FAQ(02) − C++ is not default, C is − how do I change this ??

QUESTION (02)

C++ is not default, C is − how do I change this ??

ANSWER

If your main programming language is C++, then you will require the `.def` and `.h` files to be loaded in C++ mode by default, rather than 'C'. To modify this hen the order of the file hooks has to be re−defined.

Within your *<user>.emf*, over−ride the default ordering by including the line:−

```
add−file−hook ".h .def"                                     fhook−cpp
```

This adds a newer binding for "`.h`" and "`.def`" to C++, over−riding the existing 'C' binding.

FAQ(03) − GNU Emacs; are there any GNU Emacs bindings. ?

QUESTION (03)

GNU Emacs; are there any GNU Emacs bindings. ?

ANSWER

No not at the moment. The GNU Emacs bindings would be added as a compatibility file (meemacs.emf) in much the same way that the me3.8 bindings are added, see meme3_8.emf.

From the user−setup(3), the user would then ask for "gnu" compatibility.

We would welcome submissions for a gnu compatibility file, gnu.emf, to add to the release.

The **Meta** key (typically `Alt`) may be bound to key strokes, as opposed to the menu short−cut from the user−setup(3) as follows:−

Help −> **User Setup** −> **General** −> **Alt −> Main Menu** = N
Help −> **User Setup** −> **General** −> **Alt −> Esc Prefix** = Y

FAQ(04) − Icons are not displayed correctly in Microsoft Windows environments !!

QUESTION (04)

Icons are not displayed correctly in Microsoft Windows environments !!

ANSWER

After installing on Microsoft platforms, the Icons in the Explorer window may not be showing correctly. To remedy the situation then the following steps may be taken.

Windows '95

Try re−starting the system first. If the icons are still incorrect then re−start in Safe mode and delete the file:

```
c:\windows\ShellIconCache
```

Restart and the Icons should be correct.

Window '98

Try re−starting the system first. If the Icons are still incorrect then re−start in Safe mode, this should re−generate the Icon cache. Restart windows.

NT

Restart the system.

FAQ(05) − ipipes not working on Microsoft Windows network drives ?

QUESTION (05)

ipipes not working on Microsoft Windows network drives ?

ANSWER

We are aware of a problem with the ipipe commands with '95 and '98 (not sure about NT) when the current drive is a Novel network drive.

Although we have not been ably to fully characterize the problem, we know that:−

- Old Novel Clients prior to 2.2 − Does not work
- Novel Intranetware Client 2.2 − Does not work.
- Novel Network Client 2.5 − Does work.
- Novel Client 3.01 − Does work.

Any other information in this area would be appreciated to fully characterize the problem.

To get around the problem then disable ipipes using $system(5). From within your <*user*>.*emf* knock off bit 0x800 from $system(5). This will enable regular pipes, which will work, albeit not in the background.

FAQ(06) − Language not supported − will it be ??

QUESTION (06)

Language not supported − will it be ??

ANSWER

We only support the (programming) languages that we have come into contact with. If you are using a language that we are not supporting then you will need to write a new `hk<language>.emf` file. See Language Templates on how to map a new programming language. The list of currently supported file types is defined in Supported File Types.

Jasspa would appreciate any new templates that people define for standard file types so that we can add them to the distribution.

For Microsoft Windows, any associated "me" icons types would also be appreciated.

FAQ(07) − Language file is incomplete

QUESTION (07)

Language file is incomplete

ANSWER

For a number of the (programming) language templates we have only provided a sub−set of the commands, this is typically because we only use a sub−set ourselves.

For a number of templates, there is no indent support (see indent(2) and Supported File Types).

Note that when extending the template then only standard words should be added. Words which are local extensions should be added to a `myXXX.emf`.

Jasspa would appreciate completed template definitions.

SEE ALSO

FAQ06

FAQ(08) − Input locked up and not accepting keys; how do I unlock ?

QUESTION (08)

Input locked up and not accepting keys; how do I unlock ?

ANSWER

This sometimes happens if a macro has been aborted badly. Typically a few "Ctrl−G"s (see abort−command(2)) will terminate the macro and return control back to the caller.

FAQ(09) − MicroEmacs Bindings; How do I get the original MicroEmacs bindings ?

QUESTION (09)

MicroEmacs Bindings; How do I get the original MicroEmacs bindings ?

ANSWER

From user−setup(3) set the Emulation to "`MicroEmacs v3.8`". On re−starting (or Current) the macro file `meme3_8.emf` is executed and the bindings loaded. This file should restore your familiar execution set.

FAQ(10) − Microsoft Windows Locks up after killing an ipipe.

QUESTION (10)

Microsoft Windows Locks up after killing an ipipe.

ANSWER

This is a known problem for '95/'98 (not NT), on killing an ipipe. Sometimes the "Winoldapp" locks up, if this is the case use "Alt−Ctrl−Del" to bring up the "**Close Program**" dialogue, kill off the "`WinOldApp`" if it is not responding.

MicroEmacs will then come back. We are looking for ways around this problem at the moment. From the programming perspective Windows is just not as nice as UNIX − which just works !!

FAQ(11) − Mouse support under Microsoft windows is strange !!

QUESTION (11)

Mouse support under Microsoft windows is strange !!

ANSWER

The mouse operation under Microsoft windows (and DOS) is biased towards a 3−button mouse operation (Logitech is ideal !!), operating in a similar way to UNIX. i.e. <select> operation gets text <Middle button> yanks text back.

This stems from the fact that we all came from UNIX backgrounds. We have had a number of comments about this already and do plan to address this issue.

Note:− Those of you that have already had a little dip into the operation of the mouse will have probably worked out that the whole of the visible mouse/screen interaction is driven through macros, so this functionality is actually a macro change.

FAQ(12) − Scroll bars too narrow !!

QUESTION (12)

Scroll bars too narrow !!

ANSWER

You can change the width of the scroll bars to double width from user−setup(3) "Wide Scroll Bars". Alternatively, you may do this yourself from `<user>.emf` by:−

```
set-variable $scroll-bar &bor $scroll-bar 1
```

See $scroll−bar(5).

Remember if you have enabled wide scroll bars, under windows, or X−Windows, you may want to change your start−up screen width to `82` characters rather than `80` − see change−frame−width(2).

FAQ(13) − Tab key; Why does the tab key not operate in some windows ??

QUESTION (13)

Tab key; Why does the tab key not operate in some windows ??

ANSWER

In buffers with indentation information the tab key re−computes the indentation of the line. This behavior may be changed from the user−setup.

Refer to documentation for $system(5) and user−setup(3).

FAQ(14) − Termcap; On a color terminal why is there no color ??

QUESTION (14)

Termcap; On a color terminal why is there no color ??

ANSWER

MicroEmacs has to be enabled to show color by default. From user−setup(3) enable "Termcap Color". This will give you basic colors.

You may also try enabling "With Bold" − this may increase the range of colors.

FAQ(15) − Termcap; Some of the keys do not work − how can I bind them ?

QUESTION (15)

Termcap; Some of the keys do not work − how can I bind them ?

ANSWER

In your user setup <user>.emf add the new keys. You have to be careful as to the environment and probably need to do something like the following:−

```
; First check we are not an Xterm
!if &not $use-x
    ; Quick check on the terminal type. We probably need to
    ; distinguish between terminal types for different bindings
    !if &seq $TERM "myterm"
        translate-key "<from>" "<to>"
        ...
        translate-key "<from>" "<to>"
    !endif
!endif
```

See translate−key(2) for details of translating termcap keys. See describe−key(2) to help identify the key.

FAQ(16) − Timestamp; Format incorrect, how can I change to MMDDYY.hhmm ?

QUESTION (16)

Timestamp; Format incorrect, how can I change to MMDDYY.hhmm ?

ANSWER

From within your <*user.emf*> set the time stamp default format i.e.

```
set-variable $timestamp "<%M%D%Y.%h%m>"
```

See $timestamp(5).

FAQ(17) − Windows; Component characters rendered incorrectly, how do I fix ?

QUESTION (17)

Windows; Component characters rendered incorrectly, how do I fix ?

ANSWER

If some of the components of the windows are rendered incorrectly, typically caused by local variations of character sets, then new window component characters may be defined. See $window−chars(5) for details on how to define new character replacements.

FAQ(18) − Windows Autosave and Backup files; are these potentially a problem ?

QUESTION (18)

Windows Autosave and Backup files; are these potentially a problem ?

ANSWER

For windows '95 up until OEM service release 2, the OS could not distinguish the difference between the files:−

```
.xxx
.xxx~
```

on a read, we have managed to find a work around for this, however we would advise that the 3 letter extension is adhered to for these releases. For releases of '95 OEM service release 2 and greater, '98 and NT we have not found a problem with any of the auto save and backup naming.

Obviously, the backup naming will depend on the native file system. For instance if your system administrator has not enabled long file names on your Novel server.

FAQ(19) − Printing; Why in Windows does the output come out in a buffer ??

QUESTION (19)

Printing; Why in Windows does the output come out in a buffer ??

ANSWER

Use the **File** −> **Printer Setup** dialog and change the destination to the "**Direct to printer**".

FAQ(20) − Printing; On Windows which font should I use ??

QUESTION (20)

Printing; On Windows which font should I use ??

ANSWER

We suggest that "**Courier New**" is used as the print font. This scales well and supports the full character set. Problems have been reported with networked postscript printers when used in conjunction with fixed fonts.

FAQ(21) − Printing; My printer is not supported ?

QUESTION (21)

Printing; My printer is not supported ?

ANSWER

We are in the process of providing native postscript generation − UNIX users can stream their output through "a2ps" and then into their standard printer queues.

Windows, the support is already built in.

For DOS then you need to get your printer manual out and sort out how to map the printer codes onto fonts. The printer codes are added to "`printer.erf`". We have already provided support for the HP DeskJet printer (PCL), look at this printer definition for some help as to the type of information that you need to set up. It's all a bit fiddley, but you do not get much choice if you want more than plain ASCII out.

FAQ(22) − Alt key maps to the Menu, how do I change ?

QUESTION (22)

Alt key maps to the Menu, how do I change ?

A−f opens the main File menu instead of executing forward−word (esc f). How do I make the Alt key act like the Meta key all the time?

ANSWER

The **Meta** key (typically `Alt`) may be bound to key strokes, as opposed to the menu short−cut from the user−setup(3) as follows:−

Help −> **User Setup** −> **General** −> **Alt −> Main Menu** = N
Help −> **User Setup** −> **General** −> **Alt −> Esc Prefix** = Y

FAQ(23) − me32.ini − Where does it go, how do I know it's being processed ??

QUESTION (23)

me32.ini − Where does it go, how do I know it's being processed ??

ANSWER

Question posed as:−

```
> 1)  Am I right in assuming that for NT the file me32.ini goes
>     into  %windir%, i.e. into c:
```

Yes, this is where the other .ini files are.

```
> 2)  How do I know me32.ini is being processed?  Creating one,
>     as described in the readme.txt file doesn't seem to have
>     any visible effect.
```

From within the editor, if you show the variable $MEPATH(5), then it should echo the paths that you have defined in the me32.ini file.

```
esc−x describe−variable
$MEPATH
```

See: me32.ini(8), Installation Information, Setting Up A User Profile.

```
> 3) What does the "fontfile" statement do ?
```

For releases after May 1999 then the *fontfile* statement may be omitted as typically **Lucida Console** or **Courier New** is used. If you want to use the fixed OEM fonts then *fontfile* should be defined as **app860.fon** (or local language equivalent), this forces the font to be loaded as a resource, prior to use.

FAQ(24) − Windows − Where is app850.fon ?

QUESTION (24)

Windows − Where is app850.fon ?

ANSWER

"app850.fon" is the font file used for the DOS window under '95/'95/NT. You should find it in your c: hidden. If you search from the Explorer−>Tools−>Find "app850.fon" it should be found in the fonts directory. There is nothing to be done − the file exists and is in the correct location.

If you do not have this file then, you might have "appXXX.fon", or some other fixed font. You can locate the file that you want as follows:−

Start−>Settings−>Control Panel−>Fonts

Display the font list as 'details'. Within that list you should find a "MS−DOS CPXXX" entry. It will be a red font (if you are in monochrome then it will have a 'A' in the box rather than a 'Tt'). This is a fixed font and will be a good alternative to "app850.fon", you can also try the "Fixedsys" font file which has some weird name.

To be honest I do not know what Microsoft are currently shipping. Most of the Windows platforms that I have used have been upgrades or been abused by so many people you never know what is original !!

We would be interested in any details of other fixed fonts, which support the full OEM character set that are better alternatives to the DOS ones.

FAQ(25) − Time; mode line is showing the date in DD/MM/YY format how do I change ?

QUESTION (25)

Time; mode line is showing the date in DD/MM/YY format how do I change ?

ANSWER

From within your user setup, over−ride the default mode line setting with the modifications you require. i.e. to change the date format to MM/DD/YY use:−

```
set−variable $mode−line "%s%r%u%k %b %l − %h:%m %M/%D/%Y (%e) − (%f) "
```

SEE ALSO

$mode−line(5).

FAQ(26) − C or C++ indentation and effects; how can I turn off ?

QUESTION (26)

C or C++ indentation and effects; how can I turn off ?

ANSWER

The cmode(2m) is supposed to make editing 'C' easier, by forcing the user to follow a preset editing convection. The layout is pretty standard, following a 4 space indent, writing either K&R or standard 'Pascal' type layout, with braces aligning vertically.

The problem most new users have is the inability of the tab key to function, or more simply do not want to be 'forced' to write in a particular style (GNU writers will probably not like this either − conversely they will be using GNU emacs !!). However the constrained layout can be configured to create most styles and does help in a project situation, whereby most of the authored code roughly adheres to the same sort of layout conventions. For C++ users then edit "`hkcpp.emf`" rather than "`hkc.emf`".

To turn off all automatic 'C' layout

To disable ALL automatic layout then edit "hkc.emf" and turn "cmode" off. It is probably quite useful to apply "indent", this will return the cursor to the same indentation column whenever a new line is inserted. i.e. in "fhook−c" of hkc.emf:−

```
0 buffer−mode "cmode"
1 buffer−mode "indent"
```

If you want proper tabs then you may also want to add:−

```
0 buffer−mode "tab"
```

This inserts the <tab> character into the text, rather than translating to spaces. Alternatively disable **tab** mode for all file type using user−setup(3) i.e.

Help −> **User Setup** −> **General** −> **Tab**

To re−enable the <tab> key

To retain the 'C' layout aid, but re−enable the tab key operation then disable the **Tab To Indent** option in user−setup(3) i.e.

Help −> **User Setup** −> **General** −> **Tab To Indent**

This enables the use of the TAB key in all column positions with the exception of column 0. A <tab> in Column 0 will still enable the automatic line re−formatting.

If you want real <Tabs> then disable the **tab** mode using user−setup(3) i.e.

Help −> **User Setup** −> **General** −> **Tab**

To change the 'C' Indentation Layout

The 'C' layout indentation is controlled from the system variables:−

$c−case(5), $c−contcomm(5), $c−continue(5), $c−margin(5), $c−brace(5), $c−statement(5).

These settings may be defined in your <user>.emf to change the default layout. Refer to the on−line documentation for details.

FAQ(27) − fill−paragraph function does not fill ??

QUESTION (27)

fill−paragraph function does not fill ??

I can't seem to get the fill−paragraph function to fill the following paragraph:

```
This is a very
poorly formed paragraph
which refuses to fill
properly!
```

ANSWER

The default justification mode is Auto which tries to work out the mode required for each paragraph. Its fairly smart at maintaining a documents indentation, e.g. consider the example right hand justified:

```
                                             This is a very
                                        poorly formed paragraph
                                          which refuses to fill
                                                     properly!
```

It will maintain this indentation. The problem comes when the detected form is not the required form as in the example. The detected paragraph justification to be used is "none" because the lines are short. There are 2 ways to solve this problem:

- You can change the $fill−mode(5) to left or both (in fhook−doc mode use C−c l or C−c b) and then use fill−paragraph as normal.
- Manually concatenate the first few lines into one to create a longer first line and then use the fill−paragraph a normal, i.e. change the paragraph to:

  ```
  This is a very poorly formed paragraph which refuses to fill
  properly !
  ```

  and then fill. This works because the longer line will lead to a different assessment of what's required.

FAQ(28) − Key modifier which acts as the ESC key; what is it ?

QUESTION (28)

Key modifier which acts as the ESC key; what is it ?

What is the modifier key which acts as the ESC key ? Having to type ESCAPE and then f to move one word forward is very boring.

With Gnu Emacs (on Unix systems), there is a "meta" modifier key which is a shortcut for pressing ESCAPE followed by the command key. The "meta" key should be the "Alt" key.

ANSWER

The "meta" key is the "Alt" key. But 'F' is the Main menu hot−key for the 'File' sub−menu so by default 'A−f' will open the File sub−menu. This can be disabled by clearing bit 0x2000 in the $system variable. This option can now be set using user−setup (Alt −> Main Menu).

FAQ(29) − find−file start location; where is it ?

QUESTION (29)

find−file start location; where is it ?

ANSWER

The find−file(2) start location is defined as follows:−

- `*scratch*` is current buffer; the current working directory.
- `file` is current buffer; the directory location containing *file*.

Running under Microsoft Windows or UNIX, using an icon launch, then it may become frustrating that the start location is always `C:\Program Files\JASSPA\MicroEmacs` (Microsoft windows) or `/usr/local/bin` (UNIX) this is simply resolved by starting the executable with the **−c** option, as defined by me(1). The **−c** option starts the editor with the last editing session, this is typically where a user will want to commence an editing session.

If the **−c** approach is not acceptable, then it is worth defining the environment variable `$HOME` within the start up script, or in the users environment. Using **find−file** with tilde (~) implies that the directory start path is `$HOME`.

FAQ(30) − Re−using a MicroEmacs session; how to ??

QUESTION (30)

Re−using a MicroEmacs session; how to ??

ANSWER

A MicroEmacs editing session may be re−used, such that the current editor is prompted to load a new file externally. This is typically invoked from a short−cut launch from a file manager i.e. **Explorer(1)**, **Tkdesk(1)** etc.

In order to facilitate the re−use of the session, then me(1) is invoked with the **−o** option, this locates the active editor session and passes the file load request. If an existing session does not exist then a new session is started.

In order for this mechanism to operate, then the Client−Server Interface must be enabled from the user−setup(3) i.e.

[**Help** −> **User Setup** −> **Platform** −> **Client Server** = Y]

FAQ(31) − Microsoft Drag and Drop; is it supported ??

QUESTION (31)

Microsoft Drag and Drop; is it supported ??

ANSWER

MicroEmacs '02 supports Microsoft *drag and drop* interaction. Multiple files and directories may be dragged from Microsoft Explorer (or other application) and dropped into a buffer window. The destination buffer window is the window in which the dropped file(s) are displayed.

Note if the user is currently on the command line, then the command line operation is aborted in order to facilitate the *dropped* files.

FAQ(32) − Cut and Paste to/from other applications; is it supported ??

QUESTION (32)

Cut and Paste to/from other applications; is it supported ??

ANSWER

MicroEmacs '02 supports *cut and paste* operations on all platforms.

To copy a region from MicroEmacs '02 to another application

Select a region (with the mouse or keys) − there is no need to invoke a copy operation. All selected text is immediately available to other applications.

Move to the new application and paste, as dictated by the platform.

To copy a region from another application to MicroEmacs '02

Select the region in the application into the clipboard, as dictated by the platform.

Move to MicroEmacs '02, position the cursor and yank(2) (`C−y` or typically the middle mouse button) the clipboard text.

FAQ(33) − Fonts; how can I change the font ??

QUESTION (33)

Fonts; how can I change the font ??

ANSWER

The currently selected font may be modified from the user−setup(3).

Help −> **User Setup** −> **Platform**

The font selection depends upon the platform, in all cases a fixed font should be selected, otherwise rendering anomalies will result.

If you are running on Microsoft platforms ensure that the **OEM/ANSI** flag matches the settings of the **Display Font Set** entry.

FAQ(34) − Colors; how can I change screen colors ??

QUESTION (34)

Colors; how can I change screen colors ??

ANSWER

The screen colors are selected from the user−setup(3).

Help −> **User Setup** −> **Platform** −> **Color Scheme**

The default setting is *White on Black*, the *Black on Cream* is the most popular setting.

FAQ(35) − File Types; how do I interchange between UNIX, Windows and DOS files ??

QUESTION (35)

File Types; how do I interchange between UNIX, Windows and DOS files ??

ANSWER

MicroEmacs '02 facilitates the editing of the standard file types on all platforms. All files retain their line ending type through edits. i.e. if a DOS file is edited on a UNIX system, the file is still written as a DOS file. When new files are created, they are created with the standard attributes of the host O/S.

The line ending of the file may be modified from the menu

file −> **attributes**

This brings up a dialog that allows the file type and attributes to be modified.

Note that the only ending that is NOT preserved are files whose lines end in `<CR>`'s only. The line format is correctly interpreted on reading, but is not retained on the write.

FAQ(36) − Non−English Languages; What font should I select ??

QUESTION (36)

Non−English Languages; What font should I select ??

ANSWER

MicroEmacs '02 has only been tested with Western Lanuguages only. Within the Microsoft Windows environment an ANSI type font should be selected, assuming of course that the characters required are in the ISO−Latin character set. UNIX typically supports ISO−Latin character sets.

FAQ(37) − MicroEmacs '99; How do I up−grade from MicroEmacs'98 ??

QUESTION (37)

MicroEmacs '99; How do I up−grade from MicroEmacs'98 ??

ANSWER

Backup your current version!

Follow the MicroEmacs'99 installation procedure to install and get MicroEmacs'99 running.

Due to the great improvement to user−setup(3) it is advised that the user creates a new setup using **user−setup** and then migrates required macro code changes from the old release into the new.

FAQ(38) − Some keys on my foreign keyboard do not work properly, how do I get them working ??

QUESTION (38)

Some keys on my foreign keyboard do not work properly, how do I get them working ??

ANSWER

The most common problem are with foreign keyboards where the <AltGr> key is used to generate some characters in a similar fashion to the <Shift> key. For example, on a Belgian keyboard the '9' key produces a '{' character when the <AltGr> key is also pressed.

The quickest and best solution is to use the **Keyboard** setup on the Start−Up page of user−setup(3). This however may not provide the solution as not all keyboards are currently supported. If you are using an unsupported keyboard please send configuration information back to JASSPA for inclusion in the next release. The keyboard configuration information is stored in the macro file `keyboard.emf`.

If **user−setup** does not currently support your keyboard, or you wish to remap some keys, then the command translate−key(2) should be used. **translate−key** remaps generated key stroke(s) into another key at a low level so the mapping is supported in all areas. If a macro and key binding were used instead, while they would work in the main text windows, they would not work in the message line. See help on **translate−key** for more information.

Note that some <AltGr> keys can produce 2 keys, for example on a Belgian keyboard '<AltGr>−9' produces the key 'A−C−9' first, immediately followed by 'A−C−{'. This is an unfortunate side effect of windows, it is better to have two keys rather than none. But this does add confusion to the problem! Again, see **translate−key** for more information.

FAQ(39) − Tabs; How to change the tab width ??

QUESTION (39)

Tabs; How to change the tab width ??

ANSWER

There are two variables that change the width of the tab $tabwidth(5) and $tabsize(5) they control the size of a displayed tab character (number of spaces) and the simulated tab character size, where the user entered tab character is replaced by a number of space characters. The latter is only used when tab(2m) mode is enabled (it is typically enabled by default).

To change the tab character width then the set−variable(2) command is used:

```
esc x set−variable
```

You will then be prompted for the remaining arguments. <TAB> is the completion so:−

```
esc x set−v<TAB>
$tabw<TAB>
2
```

If this is the setting that you always want to use then it is easier if you put this in your <user.emf> as:−

```
set−variable $tabwidth 2
```

then whenever you start a new session you will always have the $tabwidth defined as you want it.

We would recommend that $tabwidth(5) is not modified because it turns the all tab's to 2 so when you read it into something like Microsoft notepad the indentation is not as you like it because it displays tabs as 8 characters.

Instead, set the **$tabsize** to 2, and run with tab(2m) enabled (this is the default). This turns <TAB>'s to spaces, hence the layout is retained. This makes the file slightly larger, but the presentation is maintained.

If you are reading in a file with TAB's embedded then you can convert all of the <TAB>'s to spaces using tabs−to−spaces(3):

```
esc x tabs−to−spaces
```

If these TAB's are 8 characters, and they should be displayed as 2, then prior to conversion change the tabwidth, convert and then restore.

- Change the $tabwidth to 2
- tabs−to−spaces

- Restore the $tabwidth to 8

FAQ(40) − Windows/DOS; Where do I get grep/diff etc. ??

QUESTION (40)

Windows/DOS; Where do I get grep/diff etc. ??

ANSWER

For windows and DOS users the UNIX tools may be obtained from:−

```
ftp://ftp.cdrom.com/pub/garbo/garbo_pc/unix/uxutl23a.zip (238 Kb)
ftp://ftp.cdrom.com/pub/garbo/garbo_pc/unix/uxutl23b.zip (227 Kb)
ftp://ftp.cdrom.com/pub/garbo/garbo_pc/unix/uxutl23c.zip (221 Kb)
ftp://ftp.cdrom.com/pub/garbo/garbo_pc/unix/uxutl23d.zip (160 Kb)
```

comments for this at:

```
http://www.geocities.com/SiliconValley/Lakes/1401/softlib1.htm
```

One awk−port; the Gnuish project has 16 bit and 32 bit versions of **gawk(1)** in:

```
ftp://mirrors.aol.com/pub/simtelnet/gnu/gnuish/gawk303x.zip (1997, 495K)
```

Acknowledgment: **DG** − 99/07/02

FAQ(41) − Home/End Keys; How do I change the default bindings ??

QUESTION (41)

Home/End Keys; How do I change the default bindings ??

ANSWER

Some users prefer the `HOME` and `END` keys to map to the beginning and end of the line, rather than beginning/end of the buffer, respectively. Within the *<user>.emf* the following global bindings may be applied to re−assign the key mappings:−

```
global-bind-key "beginning-of-buffer" "C-home"
global-bind-key "end-of-buffer"       "C-end"
global-bind-key "end-of-line"         "end"
global-bind-key "beginning-of-line"   "home"
```

Acknowledgment: **DG** − 99/07/02

FAQ(42) − tags; How do I generate a MicroEmacs compatible tags file ??

QUESTION (42)

tags; How do I generate a MicroEmacs compatible tags file ??

ANSWER

A **tags** file is used by the find−tag(2) command. This is used to hypertext to the *tagged* definition or variable. The standard **ctags(1)** format is used by MicroEmacs. The **tags** file itself may be generated by MicroEmacs '02 from the menu (*Tools−>XX Tools−>Create Tags File*). Alternatively a **tags** file may be generated by the **ctags(1)** utility. This is typically standard on UNIX platforms. For Windows and DOS platforms then the **Exuberant Ctags** is recommended, this is available from:−

```
http://darren.hiebert.com
```

A MicroEmacs '02 compatible tags file may be generated using the command line "`ctags −N --format=1 .`" cataloging the current directory. To generate **tags** for a directory tree then use "`ctags −NR --format=1 .`". Refer to the **Exuberant Ctags** documentation for a more detailed description of the utility.

The user variable %tag−option(5) may be used to enable find−tag(2) to locate a recursivelly generated **tags** file from a parent directory.