MicroEmacs Reference Card

for JASSPA's version 2006.09
Abbreviations: C = Ctrl (Control), shortcuts are case insensitive: C-a = C-A.

Starting and leaving

To enter MicroEmacs, just type me (= ne + macros). You can activate the menu line with f1 and leave it with C-g. Exit the program through C-x C-c (or esc z without any questions).

Help

General help	esc ?
command-apropos	C-h a
describe-key	C-x ? or C-h k
describe-variable	C-h v
describe-bindings	C-h b

Error Recovery

abort-command partially typed or executed	C-g
undo an unwanted change (undo)	C-x u or C-_
redraw a garbaged screen (recenter)	C-l

File

view file read-only (view-file)	C-x C-v
open file read-write (find-file)	C-x C-f
insert contents of another file into this buffer (insert-file)	C-x C-i
replace this file with the one you really want (read-file)	C-x C-r
save this buffer to a file on disk (save-buffer)	C-x C-s
write this buffer to a specified file (write-buffer)	C-x C-w

File-browser (£10)

directory	*files*
space	update
return fold/unfold	update
C-return recursive fold/unfold	update
tab toggle buffers	tab toggle buffers
enter directory	return open file
—	space select current file
—	+ select files by pattern
—	x execute shell command on selected files (%f)

Capitalisation and transposition

word-commands do not apply to characters left of the cursor!

capitalize-word (Example)	esc c
upper-case-word (EXAMPLE)	esc u
lower-case-word (example)	esc l
lower-case-region	C-x C-l
upper-case-region	C-x C-u
swap character with previous (transpose-chars)	C-t
current line sinks down (transpose-lines)	C-x C-t

Multiple Windows

delete all other windows (delete-other-windows)	C-x 1
delete this window (delete-window)	C-x 0
split window in two: upper / lower (split-window-vertically)	C-x 2
split window in two: left / right (split-window-horizontally)	C-x 3
switch cursor to next window (next-window)	C-x o
switch cursor to previous window (previous-window)	C-x p

Buffers

select another buffer (find-buffer)	C-x b
go to next buffer (next-buffer)	C-x x
list all buffers (list-buffers)	C-x C-b
remove a buffer (delete-buffer)	C-x k
change buffer-mode	C-x m

Motion

entity to move over	backward	forward
-char (-acter)	C-b	C-f
-word	esc b	esc f
-line	C-p	C-n
beginning-/end-of-line	C-a	C-e
-paragraph	esc p	esc n
beginning-/end-of-buffer	esc <	esc >

recenter and redraw (recenter)	C-l
go to line (goto-line)	esc g
set-alpha-mark	C-x C-a ⟨one character⟩
goto-alpha-mark	C-x a ⟨one character⟩
browsing mode (browse)	f3

Killing and Deleting

entity to kill	backward	forward
-delete-char (-acter)	backspace	C-d
-kill-word	esc backspace	esc d
line (to end of)	—	C-k (kill-line)

start region (set-mark)	esc space or C-space
exchange-point-and-mark	C-x C-x
kill-region	C-w
copy-region	esc w
paste (yank)	C-y
paste previously cut (reyank)	esc y
copy-rectangle	—
kill-rectangle	esc C-w
yank-rectangle	esc C-y
toggle overwrite / insert mode	ins
insert-tab	C-i
insert certain character (quote-char)	C-q

Shell

execute a shell command (pipe-shell-command)	esc @
run new interactive shell (quit to return)	C-x c

Regular Expressions

If magic mode (letter M on the mode line) is on, then the following regular expressions can be used in search or replace operations.

any single character except a newline	. (dot)
zero or more repeats	*
one or more repeats	+
zero or one repeat	?
between *M* and *N* repeats	\{*M*,*N*\}
any character in the set	[...]
any character not in the set	[^ ...]
beginning of line	^
end of line	$
quote a special character *c* (not { or })	*c*
alternative (or)	|
grouping	\(... \)
*n*th group	*n*
word break	\b
not beginning or end of word	\B
beginning of word	\<
end of word	\>
(non-) digit	(\D) \d
(non-) hexadecimal character	(\X) \x
(not) word-constituent character	(\W) \w
(not) lower-/upper-case word-constituent char.	(\L) \l / (\U) \u
(not) white-space	(\S) \s

Searching and replacing

incrementally search forward (isearch-forward)	C-s
incrementally search backward (isearch-backward)	C-r
cancel search (abort-command)	C-g
continue last search forward (hunt-forward)	C-x h
continue last search backward (hunt-backward)	C-x C-h
repl. string on all following occurr. (replace-string)	esc r
interact. repl. a text string (query-replace-string)	esc C-r or esc %

Macro commands

execute-named-command	esc x
optional numerical argument *n* (prefix)	esc *n* or C-u *n*
start macro recording (start-kbd-macro)	C-x (
stop macro recording (end-kbd-macro)	C-x)
execute last-defined macro (execute-kbd-macro)	C-x e
name the last macro (name-kbd-macro)	—
insert macro commands in buffer (insert-macro	—
run macro from buffer (execute-buffer)	—
run script from file (execute-file)	esc /

Key bindings

globally bind key (global-bind-key)	esc k
globally unbind key (global-unbind-key)	esc C-k
locally bind key (buffer-bind-key)	—
locally unbind key (buffer-unbind-key)	—

Developping MicroEmacs macros

Define a macro by writing:

```
define-macro <name>
    ...
!emacro
```

Macros are written in an interpreted scripting language

- prefix (polish) notation for functions (`function arg arg arg ...`)
- one command per line and one line per command
- `;` starts comment lines
- `\` is escape character that needs to be doubled in command arguments

Datatypes

String every value is a string and can be quoted "`string`" (necessary if containing whitespace); maximum length is 1024 characters; C-like `\t`, `\n` and `\\` works

Integer implicitly converted to/from decimal string representation; C-like notations (`123, 076, 0x2f`) work

Boolean true (integer $\neq 0$, e.g. `1`) or false (otherwise, e.g. `0` or non numeric string), compatible with C

List well formed string: first and last character is delimiter; positive integer index (1, 2, ...)

Types of variables

Name	1st char	scope	example
environment v.	`$`	OS environment	`$PATH`
system v.	`$`	MicroEmacs	?
user v. (deprecated)	`%`	MicroEmacs	
buffer v.	`:`	buffer	`:buffer:var`
command v.	`.`	command or macro	`.command.var`
register v.	`#`	macro execution	`#l, #p, #g`
macro v.	`@`	macro	?
directive	`!`		`!force`
function (operator)	`&`		`&add`

Directives

```
!if <condition>
    ...
!elif <condition>
    ...
!else
    ...
!endif
```

```
!while <condition>
    ...
    [!continue]
!done

!repeat
    ...
!until <condition>
```


`!abort`	exits with failure
`!return`	exits with success
`!force`	ignores return status
`!bell`	rings the bell

Debugging

Set debugging on with `set-variable $debug 1`
Options then:

help	`?`
abort macro	`C-g`
redraw screen	`C-l`
continue normally	`!`
step (through all called macros)	`s`
display value of variable	`v`
step through this macro only	any other key

If necessary set debugging back off: `set-variable $debug 0`

Variable functions

First three chracters of name after `&` are significant.
Show value in message line `describe-variable`

Numeric functions

		return value
add /subtract	`&add / &sub`	$a1 + a2$ / $a1 - a2$
multiply / divide	`&mul / &div`	$a1 \cdot a2$ / $a1 : a2$
pre- / post-increment	`&inc / &pinc`	$a1 + a2$ / $a1$
pre- / post-decrement	`&dec / &pdec`	$a1 - a2$ / $a1$
negate a number	`&negate`	$-a1$
absolute value	`&abs`	$\|a1\|$
division remainder	`&mod`	$a1 \bmod a2$

Bitwise operators

AND / OR / XOR	`&band / &bor / &bxor`
NOT	`&bnot`

Logical operators and Comparisons

returning `1` for true and `0` for false.

AND / OR	`&and / &or` (strict evaluation)
NOT	`&not`
integer equality	`&equal`
less than	`&less`
greater than	`&great`
case (in-) sensitive string equality	`&sequal` (`&isequal`)
does string match regular expression	`&xsequal`

String functions

length of string	`&len`
concatenation	`&cat`
divide string and return left / right part	`&left` / `&right` ⟨index before cut⟩
cut out a part	`&mid` ⟨string⟩ ⟨start index⟩ ⟨length⟩
convert to upper / lower case	`&supper` / `&slower`
trim whitespace on left / right end	`&trleft` / `&trright`
trim whitespace on both ends	`&trboth`
search and replace	`&rep` ⟨string⟩ ⟨search⟩ ⟨replace⟩
search and replace regular expression	`&xrep` ⟨string⟩ ⟨search⟩ ⟨replace⟩

Test functions

Is character word constituent?	`&inword`
Does variable/command/macro exist?	`&exist`
Is string contained in another one?	`&sin` (returns start of first occurr.)
case insensitive variant	`&isin` (returns start of first occurr.)

List functions

return entry	`&lget` ⟨list⟩ ⟨index⟩
index of a certain element	`&lfind` ⟨list⟩ ⟨entry⟩

return list after ...

inserting an entry	`&linsert` ⟨list⟩ ⟨index⟩ ⟨entry⟩
	index 0: at the beginning
	index -1: at the end
	index -2/-3: alphabetically (case s./i.)
updating an entry	`&lset` ⟨list⟩ ⟨index⟩ ⟨entry⟩
deleting an entry	`&ldel` ⟨list⟩ ⟨index⟩

Output

to the message line	`ml-write` ⟨message⟩
into the buffer	`insert-string`
	`yank` and `reyank`)

Message line input

read **string**	`@ml` ⟨prompt⟩
ditto with editable default value	`@ml2` ⟨prompt⟩ ⟨default⟩
ditto with non-editable default value	`@ml1` ⟨prompt⟩ ⟨default⟩
(append to `@ml`.)	`0` general input / history (default)
	`1` absolute filename
	`9` user supplied completion list
	`a` completion list in buffer
read **character**	`@mc` ⟨prompt⟩
ditto with list of accepted characters	`@mc1` ⟨prompt⟩ ⟨valid chars⟩
ditto with help message	`@mc4` ⟨prompt⟩ ⟨help message⟩