PCS/Geneva Reference

Version 4.02—©1993 Larry Bartholdi, Marc Vuilleumier
E-Mail: `schemege@cui.unige.ch` — FTP: `cui.unige.ch:pub/pcs`

Edwin: the Traditional Editor

`Edwin` is a sophisticated editor using Emacs' key sequences

`(edwin)`	starts the editor
`(remove-edwin)`	forgets the editor
`(edwin-reset-lines)`	resets the console to full-screen

Meta- can be Escape or Ctrl-Z.

	Next	Previous
Character	Ctrl-F	Ctrl-B
Word	Meta-F	Meta-B
Line	Ctrl-N	Ctrl-P
Sentence	Meta-A	Meta-E
Paragraph	Meta-]	Meta-[
Screen	Ctrl-V	Meta-V
List	Meta-Ctrl-N	Meta-Ctrl-P
S-Expr	Meta-Ctrl-F	Meta-Ctrl-B
	Last	**First**
Line	Meta->	Meta-<

Mark Commands

Ctrl-@	set a mark
Meta-@	mark a word
Meta-Ctrl-@	mark a Scheme expression
Meta-H	mark the whole paragraph
Ctrl-X H	mark the whole buffer
Ctrl-X Ctrl-X	exchange current position and mark

Kill/Unkill Commands

Backspace	delete the character before the cursor
Ctrl-D	delete the character at the cursor
Meta-\	delete all spaces & tabs around the point
Meta-Space	delete all spaces & tabs except one
Meta-D	kill the next word
Meta-Backspace	kill the previous word
Ctrl-K	kill till end of line
Meta-K	kill till end of sentence
Ctrl-X Backspace	kill backward to begin of sentence
Meta-Ctrl-K	kill next Scheme expression
Ctrl-W	kill a region
Meta-W	copy a region
Ctrl-Y	yank back a kill
Meta-Y	unkill using previous kill ring entry
Meta-Ctrl-W	append next kill to preceding kill
Ctrl-X Ctrl-K	expunge kill ring entry

File Commands

Ctrl-X Ctrl-V	visit a file
Ctrl-X Ctrl-S	save the buffer
Ctrl-X Ctrl-W	write the buffer
Ctrl-X Ctrl-I	insert a file
Ctrl-X Ctrl-P	put a region to a file
Ctrl-X Ctrl-Q	toggle read-only flag
Meta-~	ignore changes made to the buffer

Breaking & Indenting

Ctrl-O	open the line
Return	insert a line break
Ctrl-J	insert a line break and indent
Tab	indent the line (according to Scheme syntax)
Meta-Ctrl-Q	indent the next Scheme expression

Miscellaneous

Ctrl-S	search forward incrementally
Ctrl-R	search backward incrementally
Ctrl-X Ctrl-M	toggle Fundamental/Scheme mode
Ctrl-X !	toggle full/split screen
Meta-Ctrl-Z	evaluate the mark
Meta-Ctrl-X	evaluate the next Scheme expression
Meta-O	evaluate the buffer
Ctrl-X Ctrl-Z	suspend `edwin`
Ctrl-X Ctrl-C	exit the editor
Ctrl-L	redraw the screen
Ctrl-U	repeat a command
Ctrl-G	abort the current command
Ctrl-Q	quote the next character
Ctrl-T	transpose (swap the last two characters)

Inside `(%system-file-name "EDWIN.INI")`

`(set-edwin-key` *key handler*`)`	define a new sequence
`(remap-edwin-key` *new-key old-key*`)`	define an alias sequence

Here *key* can be a character or a list of characters.
Predefined chars are `meta-char`, `alt-char`, `ctrl-x-char` and `ctrl-z-char`.

Ed: a Generic Editor

`Ed` is an editor toolkit using a subset of Brief's key sequences

`(make-editor` [*window*] [`'EXIT-FREELY`]`)`	⟹	a basic editor object
`(make-enhanced-editor ...)`	⟹	ditto, with additional features
`(make-color-editor ...)`	⟹	ditto, with block highlight
`(make-scheme-editor ...)`	⟹	ditto, with scheme-specific rules

Examples:

```
(define ed (make-[...-]editor))      ; create a full-screen editor
(ed 'REMAP-KEY 27 '@GOTO-LINE)       ; remap Esc to "go to line"
(ed [filename])                      ; open the editor
```

To create a multi-window editor, define two or more editors in different windows; scrapbook and keystroke assignements will be shared by all.

Basic Editor Commands

Step ↑ ← ↓ →

Jump PgUp; Ctrl-← Ctrl-→; PgDn

∞ Ctrl-PgUp; Home End; Ctrl-PgDn

Delete Backspace Del

Alt-I	toggle insert/overwrite mode
Alt-G	go to line
Alt-K	kill to end of line
Alt-D	delete current line
Ctrl-L	refresh display
F7	record a keystroke sequence
F8	replay a keystroke sequence
F10	execute a command by name
Alt-E	edit another file
Alt-W	write file to disk
Alt-O	choose new output name
Alt-R	read a file into current
Alt-X	exit editor

Enhanced Editor Commands

Use a Color Editor if you want block highlight features.

Alt-M	mark block
Alt-L	line block
Alt-C	column block
⟨Keypad −⟩	cut block/line to scrap
⟨Keypad +⟩	copy block/line to scrap
⟨Keypad /⟩	invert block bounds (swap anchor)
Ins	insert scrap into text
Del	delete block
Alt-W	write block/file to disk
Alt-[1...3]	drop a bookmark
Alt-J [1...3]	jump to bookmark
Ctrl-R	replicate a command
Ctrl-F5	toggle case sensivity
Alt-S or F5	search string
Shift-F5	repeat previous search
Alt-T or F6	translate string
Shift-F6	repeat previous translate

Scheme Editor Commands

Autoindent is active and color changes on each parenthesis level.

Ctrl-A	enlarge mark around Scheme expression
Ctrl-Z	mark the biggest Scheme expression
Ctrl-F10	evaluate marked expression
Alt-F10	evaluate current file
Tab	complete symbol or reindent line/block
Shift-Tab	prepare to add a comment
Alt-Q ⟨*key*⟩	use *key*'s basic definition

Driving an Editor

`Ed` is assumed to be an instance of an appropriate editor.

`(ed` [*message*] [*arguments*]`)` — *message* defaults to `'OPEN`

`'OPEN` [*filename*]	pop-up the editor, handle commands
`'SAFE`	ensure current buffer is saved
`'CLEAR`	clear buffer
`'NAME` [*new-value*] ⇒	name of buffer
`'BUFFER` [*new-value*] ⇒	a list of strings
`'POSITION` [*new-value*] ⇒	cursor position
`'INSERT` [*new-value*] ⇒	`#F` when in overwrite mode
`'TAB` [*new-value*] ⇒	tab expansion width
`'TABULIZE-MODE` [*new-value*] ⇒	`'COMPRESS`, `'NORMAL` or `'EXPAND`
`'SEPARATORS` [*new-value*] ⇒	string (used for word move)
`'CASE-SENSIVITY` [*new-value*] ⇒	`#F` when disabled
`'COLORS` [*new-value*] ⇒	associative color list
`'COMMENT-COLUMN` [*new-value*] ⇒	column #
`'INDENT-TOKENS` [*new-value*] ⇒	list of special tokens
`'INPUT-PORT` [*new-value*] ⇒	port used for input
`'DO-STRING` *string*	feed editor with keystrokes
`'READ-ACTION` ⇒	an *action*: the next event
`'HANDLE-ACTION` *action*	process one command

`'REMAP-KEY` *key function*

key is `(list*` [... ASCII_2] ASCII_1`)`
function if `(list*` [(*context action*) ...] *default-action*`)`
context is $\lambda(ed) \longrightarrow$ boolean
action is a character, a string, or a symbol:

`'@LEFT`	`'@WORD-LEFT`	`'@HOME`
`'@RIGHT`	`'@WORD-RIGHT`	`'@END`
`'@UP`	`'@PAGE-UP`	`'@TOP-OF-BUFFER`
`'@DOWN`	`'@PAGE-DOWN`	`'@END-OF-BUFFER`
`'@DEL`	`'@DELETE-TO-EOL`	`'@DELETE-LINE`
`'@BACKSPACE`	`'@TAB`	`'@ENTER`
`'@QUOTE`		
`'@INSERT-MODE`	`'@GOTO-LINE`	`'@REFRESH`
`'@RECORD`	`'@PLAY`	`'@EXECUTE`
`'@LOAD`	`'@READ-INTO`	`'@WRITE`
`'@RENAME`	`'@EXIT`	
`'@SEARCH`	`'@REPEAT-SEARCH`	`'@CASE-SENSITIVITY`
`'@TRANSLATE`	`'@REPEAT-TRANSLATE`	`'@REPLICATE`
`'@BOOKMARK-[1...3]`	`'@JUMP-TO-[1...3]`	
`'@MARK-BLOCK`	`'@LINE-BLOCK`	`'@COLUMN-BLOCK`
`'@CUT-BLOCK`	`'@COPY-BLOCK`	`'@INSERT-BLOCK`
`'@CANCEL-BLOCK`	`'@SWAP-ANCHOR`	
`'@MARK-EXPR`	`'@MARK-DEF`	`'@SCHEME-PARENTHESIS`
`'@COMPLETION`	`'@INDENT`	`'@COMMENT`
`'@EVAL`	`'@EVAL-BLOCK`	`'@SCHEME-ENTER`

BGI: the Graphics Package

Points and distances are represented as pairs.
The X– and Y–components can be arbitrary numbers.

`(set-world!` *upper-left-point lower-right-point*`)`

Control System

`(init-graph` [*driver* [*mode* [*BGI-path*]]]`)`
driver is `'DETECT`, `'CGA`, `'MCGA`, `'EGA`, `'EGA64`, `'EGAMONO`, `'IBM8514`, `'HERCMONO`, `'ATT400`, `'VGA` or `'PC3270`
`(set-write-mode` *wmode*`)` *wmode* is `'COPY` or `'XOR`
`(restore-crt-mode)`
`(set-graph-mode` [*mode*]`)`
`(close-graph)`
`(graph-defaults)`
`(detect-graph)`
`(get-mode-range` [*driver*]`)`
`(get-graph-mode)`
`(install-user-driver` *name*`)` ⟹ a symbol (*driver*)
name is a string (filename without extension)
`(install-user-font` *name*`)` a symbol (*font*)

Drawing

`(line` *start-point end-point*`)`
`(rectangle` *upper-left-point lower-right-point*`)`
`(draw-poly` *list-of-points*`)`
`(circle` *center-point radius*`)`
`(arc` *center-point start-angle end-angle radius*`)`
`(ellipse` *center-point start-angle end-angle distances*`)`
`(get-arc-coords)` ⟹ (*center-point start-point end-point*)
`(get-aspect-ratio)`
`(set-aspect-ratio` *factor*`)` sets circles' and arcs' x–y aspect
factor is a fraction, passed as a pair
`(get-line-settings)`
`(set-line-style` *line-style user-pattern thickness*`)`
line-style is `'SOLID`, `'CENTER`, `'DOTTED`, `'DASHED` or `'USER-BIT`
thickness is `'NORMAL` (1), `'THICK` (3) or an integer
`(move-to` *point*`)` `(move-rel` *distances*`)`
`(line-to` *point*`)` `(line-rel` *distances*`)`

Filling

`(flood-fill` *start-point stop-color*`)`
`(bar` *upper-left-point lower-right-point*`)`
`(bar-3d` *upper-left-point lower-right-point depth draw-a-top?*`)`
`(fill-poly` *list-of-points*`)`
`(fill-ellipse` *center-point distances*`)`
`(pie-slice` *center-point start-angle end-angle radius*`)`
`(sector` *center-point start-angle end-angle distances*`)`
`(get-fill-settings)` ⟹ (*fill-style color*)
`(set-fill-style` *fill-style color*`)`
fill-style is `'EMPTY`, `'SOLID`, `'LINE`, `'[LT][BK]SLASH`, `'HATCH`, `'XHATCH`, `'INTERLEAVE`, `'CLOSE-DOT` or `'WIDE-DOT`
`(get-fill-pattern)` ⟹ (*fill-pattern color*)
`(set-fill-pattern` *fill-pattern color*`)`
fill-pattern is a list of integers

Bitmapping

`(get-pixel` *point*`)` ⟹ *color*
`(put-pixel` *point color*`)`
`(get-image` *upper-left-point lower-right-point*`)` ⟹ a binary string
`(put-image` *new-upper-left-point image-string put-mode*`)`
put-mode is `'COPY`, `'XOR`, `'OR`, `'AND`, or `'NOT`
`(image-size` *upper-left-point lower-right-point*`)`
`(get-view-settings)` ⟹ (*ul-point lr-point clip?*)
`(set-viewport` *upper-left-point lower-right-point clip?*`)`
`(clear-viewport)`
`(clear-device)`
`(set-active-page` *page*`)`
`(set-visual-page` *page*`)`

Writing Text

`(out-text-xy` *start-point text-string*`)`
`(out-text` *text-string*`)`
`(get-text-settings)` ⟹ (*font direction size horiz-just vert-just*)
`(set-text-style` *font direction size*`)`
font is `'DEFAULT`, `'TRIPLEX`, `'SMALL`, `'SANS-SERIF`, `'GOTHIC`, `'SCRIPT`, `'SIMPLEX`, `'TRIPLEX-SCR`, `'COMPLEX`, `'EUROPEAN` or `'BOLD`
direction is `'HORIZ` or `'VERT`
size is an integer between 1 and 10, or 0 for default
`(set-text-justify` *horiz-just vert-just*`)`
horiz-just is `'LEFT`, `'CENTER` or `'RIGHT`
vert-just is `'BOTTOM`, `'CENTER` or `'TOP`
`(set-user-char-size` *x-ratio y-ratio*`)`
`(text-size` *text-string*`)` ⟹ a distance

Using Color

Predefined colors are:

`'BLACK` (0)	`'BROWN` (6)	`'LIGHT-CYAN` (11)
`'BLUE` (1)	`'LIGHT-GRAY` (7)	`'LIGHT-RED` (12)
`'GREEN` (2)	`'DARK-GRAY` (8)	`'LIGHT-MAGENTA` (13)
`'CYAN` (3)	`'LIGHT-BLUE` (9)	`'YELLOW` (14)
`'RED` (4)	`'LIGHT-GREEN` (10)	`'WHITE` (15)
`'MAGENTA` (5)		

`(get-color)` `(get-bk-color)`
`(set-color` *color*`)` `(set-bk-color` *color*`)`
`(get-max-color)`
`(get-palette-size)`
`(get-palette)` `(get-default-palette)`
`(set-palette` *entry color*`)`
`(set-rgb-palette` *entry red green blue*`)`
`(set-all-palette` *color-list*`)`

Miscellaneous

`(get-xy)` `(get-max-xy)`
`(get-driver-name)` `(get-mode-name` *mode*`)`
`(get-max-mode)`
`(graph-error-msg` *error-id*`)`
`(graph-result)`

Hershey Fonts

This package is an enhanced alternative to `out-text`, providing various special symbols, types and alphabets as well as a subset of TEX.

`(load (%system-file-name "HERSHEY.FSL"))`
`(out-hershey` *string* [*position*]`)`
position is a *point*
string is a sequence of any of:

- `\small`, `\large`, `\horiz`, `\vert`,
- `\`⟨*font*⟩ or `\`⟨*font*⟩`@`⟨*scale*⟩, where ⟨*font*⟩ is:

small	light	bold	
RM10	RM2L	RM2B	Roman family
SL10	SL2L	SL2B	*Slanted family*
SS10	SS2L	SS2B	Sans-serif family
GR10	GR2L	GR2B	Γρεεκ family
AN20	GO20	OE20	Antique, Gothic & Old English
SY10	SY20	CY20	Symbols & Cyrillic

  and ⟨*scale*⟩ is *integer* or (*integer*$_x$ *integer*$_y$),
- plain text or symbols (e.g. "See `note\\dagger`"),
- grouping {} to save/restore settings and delimit items,
- `^`⟨*item*⟩ or `_`⟨*item*⟩ for super- or sub-scripts

Beware that \'s in strings must be doubled!

Debugging Tools

In this section, *proc* is a procedure to debug.

`(break[-entry,-exit,-both]` *proc*`)` `break` ≡ `break-entry`
primes `inspect`; binds `*ARGS*`, `*PROC*` and `*RESULT*`
`(unbreak[-entry,-exit]` *proc*`)` `unbreak` kills both breakpoints
`(trace[-entry,-exit,-both]` *proc*`)` `trace` ≡ `trace-entry`
`(untrace[-entry,-exit]` *proc*`)`
`(advise-entry` *proc spy*`)` *spy* is λ(*proc args env*)
`(advise-exit` *proc spy*`)` *spy* is λ(*proc args env retval*)

The Inspector

`(inspect)` starts the inspector

?	display the command summary
!	reset `inspect`
Ctrl-A	display all environment frame bindings
Ctrl-B	display prodecure call backtrace
Ctrl-C	display the current environment binding
Ctrl-D	move down to callee's stack frame
Ctrl-E	edit a variable's binding
Ctrl-G	go (resume execution)
Ctrl-I	evaluate an expression & inspect the result
Ctrl-L	list the current procedure
Ctrl-M	repeat the breakpoint message
Ctrl-P	move to the parent environment's frame
Ctrl-Q	quit (reset to top-level)
Ctrl-R	return from `break` with a value
Ctrl-S	move to son environment's frame
Ctrl-U	move up to the caller's stack frame
Ctrl-V	evaluate one expression in the current environment
Ctrl-W	where (display the current stack frame)

Using the Mouse

All calls are handled through a "mouse" object:
`(mouse` *message* [*parameters*]`)` ⟹ *result*

Here *button-set* is {`'LEFT` `'RIGHT` `'CENTER`}
button ∈ *button-set*
buttons-down ⊂ *button-set*

`'RESET`
`'SHOW`
`'HIDE`
`'INQ` ..... ⇒ (*buttons-down x y*)
`'PRESS` *button* ..... ⇒ (*buttons-down press-count x y*)
`'RELEASE` *button* ..... ⇒ (*buttons-down release-count x y*)
`'MOVE` *x y*
`'LIMITS` `'HORIZONTAL` x_0 x_1
`'LIMITS` `'VERTICAL` y_0 y_1
`'LIMITS` `'BOTH` x_0 x_1 y_0 y_1
`'SHAPE` *shape*
symbol or (*x-hot-spot y-hot-spot and-mask xor-mask*)
`'CURSOR` *symbol* ..... ⇒ a cursor. Available cursors are:

`'CENTER`	`'LEFT`	`'RIGHT`	`'SMALL`	`'ARROW`
`'UL-POINT`	`'UR-POINT`	`'LL-POINT`	`'LR-POINT`	`'HAND`
`'KEYBOARD`	`'DISK`	`'STAR4`	`'STAR6`	`'TARGET`
`'X`	`'CROSSHAIR`	`'CROSS`	`'MOVE`	`'BLOCK`
`'CIRCLE`	`'EXCLAIM`	`'HOURGLAS`	`'CLOCK`	`'KITE`
`'MESH`	`'SQUARE`	`'CALIPERS`	`'CRAYON`	`'ERASER`
`'HAIRS`	`'HUMAN`	`'IBAR`	`'LASSO`	`'PAINTCAN`
`'PENCIL`	`'PENCIL2`	`'PLIERS`	`'SWORD`	`'THE-BOOT`
`'SCISSORS`	`'SCREWDRIVER`			

`'TEXT-TYPE` `'SOFTWARE` *and-item xor-item*
and-item and *xor-item* are (*character . attribute*)
`'TEXT-TYPE` `'HARDWARE` *start-row end-row*
`'SMOOTH` [`'CHECK` | `'ENABLE` | `'DISABLE`] Precise Point services
`'HANDLER` [*new-value*] ..... ⇒ previous *handler*
a *handler* is (*events . proc*)
events ∈ (`'MOVE` `'LEFT-DOWN` `'LEFT-UP` `'LEFT` `'RIGHT-DOWN` `'RIGHT-UP` `'RIGHT` `'CENTER-DOWN` `'CENTER-UP` `'CENTER` `'UP` `'DOWN` `'BUTTONS`)
proc is λ(*events buttons-down x y mickeys-x mickeys-y*)
`'ENABLE` `'DISABLE`
`'PEN-ON` `'PEN-OFF`
`'EXCLUDE` x_0 y_0 x_1 y_1
`'MICKEY-RATIO` *x y*
`'SPEED-THRESHOLD` *speed*
`'SENSITIVITY` [*new-value*] ... ⇒ (*x y speed*)
`'MICKEYS` ..... ⇒ (*x y*)
relative movement. A Mickey is ≈ 1/200in
`'POINTER-PAGE` [*new-value*] .. ⇒ *page*
`'INFORMATION` [*new-value*] ... ⇒ (*version type IRQ*)
type is `'BUS`, `'SERIAL`, `'INPORT`, `'PS/2` or `'HP`
`'LANGUAGE` [*new-value*] ..... ⇒ *language*
language is `'ENGLISH`, `'FRENCH`, `'DUTCH`, `'GERMAN`, `'SWEDISH`, `'FINNISH`, `'SPANISH`, `'PORTUGESE` or `'ITALIAN`
`'INTERRUPT-RATE` *rate* *rate* is `'NONE`, 30, 50, 100 200

PCS/Geneva Reference

Version 4.02—©1993 Larry Bartholdi, Marc Vuilleumier
E-Mail: schemege@cui.unige.ch — FTP: cui.unige.ch:pub/pcs

Semantics

Binding Forms

(**lambda** *formals-list exp ...*)
(**named-lambda** (*name formals*) *exp ...*)
(**rec** *label exp*)
(**let[*,rec]** [*label*] ((*var value*) ...) *exp ...*)
(**do** ((*var* [*init* [*step*]]) ...)
    (*terminate? result-exp ...*)
    statement ...)

Fluid Environment

(**fluid-bound?** *var*)
(**fluid** *var*) ⟹ *var*'s fluid binding
(**fluid-lambda** *formals-list exp ...*)
(**fluid-let** ((*var value*) ...) *exp ...*)
(**set-fluid!** *var obj*) changes a fluid binding

Literals

(**quote** *pattern*) ≡ '(*pattern*)
(**quasiquote** *pattern*) ≡ `(*pattern*)
(**unquote** *expression*) ≡ ,(*exp*). Valid within **quasiquote**
(**unquote-splicing** *exp*) ≡ ,@(*exp*). Valid within **quasiquote**

Sequencing & Control

(**if** *predicate consequent* [*alternative*])
(**when** *predicate exp ...*)
(**apply-if** *predicate* λ(*trigger*) *exp-false*)
(**case** *item* (*selector exp ...*) ... [(**else** *exp ...*)])
    selector is *item-value* or (*item-value ...*)
(**cond** *clause ...* [(**else** *exp ...*)])
    clause is (*predicate exp ...*) or (*predicate* **=>** λ(*trigger*))
(**and** *exp ...*) ⟹ value of last true *exp*, or **#F**
(**or** *exp ...*) ⟹ value of first true *exp*, or **#F**
(**not** *exp*)
(**begin** exp_1 ... exp_n) ⟹ exp_n
(**begin0** exp_1 ... exp_n) ⟹ exp_1

Syntax & Errors

(**alias** *new-name old-name*)
(**syntax** *pattern expansion*) *pattern* is a list structure
(**define-integrable** *name value*) *name* will be expanded inline
(**macro** *name expander*) *expander* is λ(*exp*)
    if *expander* is '(), *name* is unaliased

(**assert** *predicate message ...*)
(**bkpt** *message irritant-exp*)
(**error** *message irritant-exp ...*)

Operators

Booleans

#T **#F** '() is currently evaluated to **#F**
(**boolean?** *obj*)

Equivalence Predicates

(**eq?** obj_1 obj_2) tests physical equality
(**eqv?** obj_1 obj_2) tests numbers, strings & characters
(**equal?** obj_1 obj_2) tests visual appearances

Pairs & Lists

'() the empty list
(**pair?** *obj*)
(**null?** *obj*)
(**atom?** *obj*) ≡ (**not** (**pair?** *obj*))
(**list?** *obj*) ⟹ **#T** if *obj* is '() or (*obj . list*)
(**list->stream** *list*)
(**list->string** *list*)
(**list->vector** *list*)
(**implode** *list*) ⟹ a symbol (built of *list*'s elements)
(**append[!]** *list ...*) **append!** alters all lists but the last
(**apply** λ(*arg ...*) *argument-list*)
(**[assoc,assq,assv]** *obj pair-list*)
(**c[[a,d] ...]r** *pair*) up to 4 levels of "a" and "d"
(**cons** obj_1 obj_2)
(**copy** *list*)
(**[delete!,delq!]** *obj list*)
(**last-pair** *list*)
(**length** *list*)
(**list** *obj ...*)
(**list*** *obj ...*) creates a dotted list
(**list-ref** *list index*) *index* starts at 0
(**list-tail** *list index*)
(**[member,memq,memv]** *obj list*) ⟹ list-tail starting with *obj* or **#F**
(**reverse[!]** *list*) **reverse!** destroys its argument
(**set-c[a,d]r!** *pair obj*)

(**for-each** λ(*arg ...*) *list ...*) calls *proc* with an item of each list
(**map** λ(*arg ...*) *list ...*) ⟹ list of resulting values

Streams: Lists Evaluating on Demand

THE-EMPTY-STREAM
(**stream?** *obj*)
(**empty-stream?** *stream*)
(**stream->list** *stream*)
(**cons-stream** obj_1 obj_2)
(**head** *stream*)
(**tail** *stream*)
(**delayed-object?** *obj*)
(**delay** *exp*) freezes and memoises *exp*
(**force** *delayed-object*)
(**freeze** *exp*)
(**thaw** *freezed-object*)

Numbers

Current release uses IEEE 64-bit floating point and precise integer computations in range $\approx \pm 10^{40000}$. Exact floating point, rational and complex numbers are currently not implemented.

(**number?** *obj*)
(**complex?** *obj*) currently ≡ **number?**
(**real?** *obj*) currently ≡ **number?**
(**rational?** *obj*) currently ≡ **integer?**
(**integer?** *obj*)
    whether *obj* results from exact integer computations
(**float?** *obj*) ≡ (**and** (**number?** *obj*) (**not** (**integer?** *obj*)))
(**exact?** *number*) ⟹ currently ≡ **integer?**
(**inexact?** *number*) ⟹ currently ≡ **float?**
(**negative?** *number*)
(**zero?** *number*)
(**positive?** *number*)
(**number->string** *number* [*radix*]) *radix* is 2, 8, 10 or 16
(**float** *number*) ⟹ IEEE 64-bit representation
(**truncate** *number*) ⟹ *number* rounded to zero
(**round** *number*) ⟹ nearest integer, tie breaks even
(**floor** *number*) ⟹ integer ∈]*number* − 1; *number*]
(**ceiling** *number*) ⟹ integer ∈ [*number*; *number* + 1[
(**1+** *number*) ≡ **ADD1**
(**-1+** *number*) ≡ **SUB1**
(**[minus,-]** *number*) unary minus
(**/** *number*) inverse
(**abs** *number*)
(**+** [$number_1$ $number_2$] ...) (**-** [$number_1$ $number_2$] ...)
(***** [$number_1$ $number_2$] ...) (**/** [$number_1$ $number_2$] ...)
(**<** [*number*] ...) (**>=** [*number*] ...)
(**>** [*number*] ...) (**<=** [*number*] ...)
(**=** [*number*] ...) (**<>** [*number*] ...)
(**min** [*number*] ...) (**max** [*number*] ...)
(**sqrt** *number*)
(**exp** [*base*] *exponent*) ≡ **expt**
(**log** *number* [*base*])
(**sin** *number*) (**asin** *number*)
(**cos** *number*) (**acos** *number*)
(**tan** *number*) (**atan** *number* [*divisor*])

(**even?** *integer*)
(**odd?** *integer*)
(**integer->string** *integer radix*) *radix* is any arbitrary integer
(**integer->char** *integer*) ⟹ n^{th} character from ASCII code
(**ascii->symbol** *integer*) ⟹ a one-char symbol
(**quotient** *dividend divisor*) rounds to zero
(**remainder** *dividend divisor*) same sign as *dividend*
(**divide** *dividend divisor*) rounds to $-\infty$
(**modulo** *dividend divisor*) same sign as *divisor*
(**gcd** *integer ...*) greatest common divisor
(**lcm** *integer ...*) least common multiple
(**bitwise-and** *integer ...*)
(**bitwise-or** *integer ...*)
(**bitwise-xor** *integer ...*)
(**random** *range*) ⟹ a non-negative integer < *range*
(**randomize** [*integer*]) random seed; defaults to time-of-day

Characters

#\BACKSPACE **#\ESCAPE** **#\NEWLINE** **#\PAGE**
#\RETURN **#\RUBOUT** **#\SPACE** **#\TAB**

(**char?** *obj*)
(**char->integer** *char*) using ASCII character set
(**string** *char ...*) ⟹ a string built of *char*s
(**char-downcase** *char*)
(**char-upcase** *char*)
(**char-ci[<,<=,=,>,>=]?** $char_1$ $char_2$) case-insensitive
(**char[<,<=,=,>,>=]?** $char_1$ $char_2$)

Strings

Strings are enclosed in "..."; All indexes start at 0

(**string?** *string*)
(**string-null?** *string*)
(**string->list** *string*) ⟹ a list of chars
(**string->number** *string* [*radix*]) *radix* is 2, 8, 10 or 16
(**string->symbol** *string*)
(**string->uninterned-symbol** *string*) ⟹ a unique symbol
(**make-string** *length* [*init-char*])
(**string-append** [*string*] ...) returns "" when no arguments
(**string-copy** *string*)
(**string-fill!** *string fill-char*)
(**string-length** *string*) ⟹ an integer
(**string-ref** *string index*) ⟹ n^{th} char of *string*
(**string-set!** *string index char*)
(**string-ci[<,=]** $string_1$ $string_2$) case-insensitive
(**string[<,<=,=,>=,>]** $string_1$ $string_2$)

(**substring** *string start end*) ⟹ *string*[*start*...*end*-1]
(**substring-fill!** *string start end fill-char*)
(**substring-find-next-char-in-set** *string start end char-set*)
(**substring-find-previous-char-in-set** *string start end char-set*)
    ⟹ index or **#F**. *char-set* is a character or a string
(**substring-find-next-string[-ci]** *string start end match*)
(**substring-find-previous-string[-ci]** *string start end match*)
(**substring-move-[left,right]!** $string_1$ $start_1$ end_1 $string_2$ $start_2$)
    The direction is important only if the substrings overlap
(**substring-ci[<,=]** $string_1$ $start_1$ end_1 $string_2$ $start_2$ end_2)
(**substring[<,<=,=,>=,>]** $string_1$ $start_1$ end_1 $string_2$ $start_2$ end_2)
(**sprintf** *template* [*obj*] ...) ⟹ a string
(**sscanf** *input-string template*) ⟹ ([*obj*] ...)

Symbols

Symbols can be enclosed in |...|

(**symbol?** *obj*)
(**symbol->string** *symbol*)
(**symbol->ascii** *symbol*) ⟹ ASCII code of 1^{st} char of *symbol*
(**explode** *symbol*) ⟹ a list of one-char symbols
(**gensym** [*string, number*]) ⟹ a new uninterned symbol
    optionally sets the string prefix or initialize the counter suffix
(**property** *symbol* [*property-name* [*new-value*]]) ⟹ old value(s)
    if *new-value* is *#!UNDEFINED*, the property is removed

Ports & Windows

`'CONSOLE`	a port: the display and keyboard
`MAX-CONSOLE`	(*lines* . *cols*), the display's size
`PCS-STATUS-WINDOW`	the bottom line's window
`STANDARD-[INPUT,OUTPUT]`	the standard I/O ports
`[INPUT,OUTPUT]-PORT`	fluidly bound to current ports

`(port? `*obj*`)`
`(window? `*obj*`)`
`(input-string? `*obj*`)`
`(input-port? `*port*`)`
`(output-port? `*port*`)`
`(char-ready? `*port*`)`
`(eof-object? `*obj*`)`
`(open[-binary]-[input,output]-file `*filename*`)`
`(open-extend-file `*filename*`)`
`(open-input-string `*string*`)` } ⟹ a port
`(close-[input,output]-port `*port*`)`
`(current-[input,output]-port)` ⟹ a port
`(line-length [`*port*`])`
`(current-column [`*port*`])`
`(flush-input [`*port*`])`
`(read [`*port*`])` read a full Scheme expression
`(read-atom [`*port*`])`
`(read-char [`*port*`])`
`(read-line [`*port*`])`
`(display `*obj* `[`*port*`])` human-readable style; ≡ `princ`
`(write `*obj* `[`*port*`])` machine style; ≡ `prin1`, `print`
`(write-char `*char* `[`*port*`])`
`(writeln `obj_1 `...)`
`(pp `*obj* `[`*port* `[`*width*`]])` pretty prints Scheme objects
`(newline [`*port*`])`
`(fresh-line [`*port*`])`
`(print-length `*obj*`)` ⟹ an integer
`(set-line-length! `*length* `[`*port*`])`
`(get-file-position `*port*`)` ⟹ an integer
`(set-file-position! `*port num-bytes whence*`)`
whence is `'SET` (0), `'CUR` (1) or `'END` (2)
`(call-with-[input,output]-file `*filename* *λ*(*port*)`)`

`(make-window [`*label* `[`*border?*`]])`
`(window-clear `*window*`)`
`(window-scroll-[up,down] `*window* `[`*start-line* `[`*end-line*`]])`
`(window-delete `*window*`)`
`(window-popup[-delete] `*window*`)`
`(window-get-attribute `*window name*`)`
name is `'[BORDER,TEXT]-ATTRIBUTES` or `'WINDOW-FLAGS`
`(window-set-attribute! `*window name value*`)`
`(window-reverse-text! `*window*`)`
`(window-get-cursor `*window*`)` ⟹ a pair: (*line* . *column*)
`(window-set-cursor! `*window cursor-line cursor-column*`)`
`(window-get-position `*window*`)` ⟹ a pair. The top is at `(0 . 0)`
`(window-set-position! `*window ul-line ul-column*`)`
`(window-get-size `*window*`)` ⟹ a pair
`(window-set-size! `*window #lines #columns*`)`
`(window-save-contents `*window*`)` ⟹ contents: binary string
`(window-restore-contents `*window contents*`)`

Vectors

Vectors are enclosed in `#(...)`

`(vector? `*vector*`)`
`(vector->list `*vector*`)`
`(make-vector `*length* `[`*init-value*`])`
`(vector `*obj* `...)` ⟹ a new vector
`(vector-fill! `*vector obj*`)`
`(vector-length `*vector*`)` ⟹ an integer
`(vector-ref `*vector index*`)` *index* starts at 0
`(vector-set! `*vector index obj*`)`

Environments

`USER-GLOBAL-ENVIRONMENT`	parent of `USER-INITIAL-ENVIRONMENT`
`USER-INITIAL-ENVIRONMENT`	top-level environment

`(environment? `*obj*`)`
`(unbound? `$symbol_1$ ... *env*`)`
`(access `$symbol_1$ ... *env*`)` looks up $symbol_1$ in (... in *env*)
`(define `*ident* `[`*exp*`])` *ident* is a name or a formals-list
`(make-environment `*scheme_definition_or_exp* `...)` ⟹ an env
`(the-environment)` ⟹ the current lexical environment
`(procedure-environment `*procedure*`)` ⟹ *procedure*'s environment
`(environment-parent `*env*`)`
`(environment-son `*env*`)` ⟹ an (empty) child
`(environment-bindings `*env*`)` ⟹ a list of name-value pairs
`(set! `*ident exp*`)`
ident is *name*, `(fluid `*name*`)` or `(access `$name_1$ ... *env*`)`
`(unbind `*symbol env*`)`
`(eval `*exp* `[`*env*`])`

Procedures, Continuations & Engines

`(procedure? `*obj*`)` `#T` for procedures and continuations
`(continuation? `*obj*`)`
`(call/cc `*λ*`(`*continue*`))` ≡ `call-with-current-continuation`
`(make-engine `*thunk*`)` *thunk* is *λ*(*ticks success failure*)
`(engine-return `*value*`)` returns *value* to the engine's caller

Operating System

`(dos-call `*progname parameters* `[`*memory-to-free* `[`*restore-screen?*`]])`
`(dos-chdir `*directory-string*`)` ⟹ previous directory
`(dos-change-drive `*drive-string*`)`
`(dos-delete `*filename*`)` ≡ `delete-file`
`(dos-get-dir [`*drive-string*`])` ⟹ a string: the current directory
`(dos-dir `*mask-string*`)` ⟹ a list of filenames
`(dos-get-env `*env-variable-name*`)`
`(dos-put-env `*string*`)` doesn't change parent's environment
`(dos-search-file `*filespec*`)` searches upon `PATH`
`(dos-file-copy `*source-filename dest-filename*`)`
`(dos-rename `*current-filename new-filename*`)` can move files
`(dos-file-size `*filename*`)`
`(%system-file-name `*filename*`)` prepends PCS files' path
`(filename-split `*filename*`)` ⟹ (*drive path name ext*)
`(filename-merge (`*drive path name ext*`))` ⟹ *filename*

Command line

`SCHEME-TOP-LEVEL`	fluidly bound to the interpreter's loop
`(reset)`	resets the current interpreter loop
`(reset-scheme-top-level)`	binds the top-level to its default
`(scheme-reset)`	

`(%c `*n*`)` ⟹ query of n^{th} command-line entry
`(%d `*n*`)` ⟹ result of n^{th} command-line entry

`(get-history)` ⟹ a list of strings: previous input lines
`(push-history `*string[–list]*`)`
`(clear-history)`
`(pcs-learn-symbols `*symbol-list*`)`
`(pcs-recognize-symbol `*symbol* `[`*len*`])` ⟹ completion of *symbol*
`(pcs-macro-keys ((`*key* . *replacement*`) ...))`
key is an ASCII code or a scan-code·`#x100`
replacement is a [dotted-]list of strings

Miscellaneous

`PCS-DEBUG-MODE` whether debug information is generated
`PCS-INTEGRATE-PRIMITIVES`
whether primitives are inline-expanded
`PCS-INTEGRATE-INTEGRABLES`
whether `define-integrable`'s are inline-expanded
`*THE-NON-PRINTING-OBJECT*`
never printed, even by `write`
`PCS-GC-MESSAGE`
string displayed during garbage collecting
`PCS-GC-RESET`
string normally displayed in the status window. Contains up to 2 "`%lu`": free scheme memory and kernel memory (in bytes)

`CLOCK-TICK`	number of ticks per second
`(clock)`	⟹ number of ticks since start of PCS

there are 2^{16} ticks per hour, i.e. ≈ 18.2 per second
`(time [`'UNIX |'GM |'LOCAL`] [`*language*`] [`*source-time*`])`
⟹ *converted-time*
*language*s supported are `'ENGLISH` and `'FRENCH`
a *time* is an integer (UNIX-time) or a prefix of the list:
(*sec min hour month-day month year week-day year-day*)
source-time's fields default to current time values

`(text-mode `BIOS-*mode*`)`	−1 for previous mode, `#x40` for 50 lines
`(full-screen)`	makes the console port as big as possible
`(split-screen `*height*`)`	shrinks the console to the bottom lines
`(gc-screen)`	positions `PCS-STATUS-WINDOW`

`(load `*filename*`)` *filename* can be scheme, web or FSL code
`(fast-load `*FSL-filename*`)`
`(fast-save-file `*source-file[–list]* `[`*dest-file*`])`
`(autoload-from-file `*filename list-of-variables* `[`*environment*`])`
`(transcript-on `*filename*`)`
`(transcript-off)`
`(edit `*pair*`)` the yukky list editor
`(gc [`*compact?*`])` invokes the garbage collector
`(freesp)` ⟹ amount of free Scheme memory
`(exit [`*DOS-return-value*`])`

SCOOPS: the Object Package

`(load-scoops)` load Scoops package

`(define-class `*name*
`[(classvars `*item* `...)]`
`[(instvars [`*item*`,`*active-item*`] ...)]`
`[(mixins `*item* `...)]`
`[(options [`*option*`]`
`[(`*option variable* `...)] ...)]`)
item is (*var init-value*) or just *var* to leave it unbound
active-item is (*var* `(active` *init-value get-proc set-proc*))
option is `[gettable,settable,inittable]-variables`
option applies on all variables when none is specified
`(define-method (`*class method*`) ([`*parameter*`] ...) `*exp* `...)`
`(delete-method (`*class method*`))`
`(make-instance `*class var init-val* `...)`
`(class-of-object `*object*`)` ⟹ *name*
`(send[-if-handles] `*object msg* `[`*arg*`] ...)`
`send-if-handles` returns `#F` when *object* cannot handle *msg*

`(class-compiled? `*class*`)`
`(compile-class `*class*`)`
`(name->class `*name*`)` ⟹ *class*
`(rename-class (`*class name*`))`
`(describe [`*class*`,`*object*`])` displays a full description
`(classvars `*class*`)` ⟹ a list of class variable names
`(all-classvars `*class*`)` includes inherited class variables
`(instvars `*class*`)`
`(all-instvars `*class*`)`
`(methods `*class*`)`
`(all-methods `*class*`)`
`(mixins `*class*`)`
`(getcv `*class var*`)` ⟹ value of *var* in *class*
`(setcv `*class var exp*`)`

Directory Structure

bin	executables and run-time librairies
brief	interface to BORLAND's editor
doc	LaTeX, POSTSCRIPT and ASCII docs
samples	small Scheme-`WEB` demos
sources/asm	Kernel Turbo Assembler 3.1 source
/c	Kernel Borland C++ 3.1 source
/s	Kernel 4.02 Scheme source
/bgi	BGI tools, various sources
/hershey	Hershey fonts `WEB` source
/lisp	Lisp compatibility Scheme source
/web	Scheme-`WEB`, various sources