*libenv* Reference e

***lib* namespace, version *0.1.55***

***type keywords and aliases s***

*supertype T*

*bool* (),:nil are false, otherwise true

*condition* *keyword,* see ***Exception***

*list* :consor (),:nil

*frame* *cons*, see ***Frame***

*ns* :nsor(), see ***Namespace***

:null (),:nil

:char *char*

:cons *cons*

:fixnum *fixnum, fix* 56 bit signed integer

:float *float*, *fl* 32 bit IEEE float

:func *function*, *fn* function

:keyword *keyword, key* symbol

:ns *namespace, ns* namespace

:stream *stream* file or string type

:struct*struct*typed vector

:symbol *symbol, sym* LISP-1symbol

:vector *vector*, *string*

:char :t :byte :fixnum :float

***Heap p***

**heap-info** *vector* heap information

#(:t *type* *pages* *pagesize*)

**heap-stat** *vector* heap allocations

#(:t :*type* *size* *total* *free ...*)

**heap-size** *T**fixnum* heap occupancy

***Frame e***

**frames** *list*active *frame*s

**frame-pop** *fn fn* pop *function’s* top

frame binding

*frame* binding:(*fn* . #(:t *…*))

**frame-push** *frame cons* push frame binding

**frame-ref** *fix fix T* frame id, offset

***Symbol l***

**boundp** s*ymbol bool* is*symbol* bound?

**make-symbol** *string symbol* uninterned *symbol*

**symbol-ns** *symbol* *key namespace*

**symbol-name** *symbol string* name binding

**symbol-value** *symbol T* value binding

***Special*** ***Forms s***

**:lambda** *list* . *List’* *function* anonymous function

***:*quote** *form* *list* quoted form

***:*if** *form* *T T’**T* conditional

***Core s***

**apply** *fn* *list* *T*apply *function* to *list*

**eval** *form* *T* evaluate *form*

**eq** *T*  *T’ bool**T* and *T’* identical?

**type-of** *T* *key* type keyword

**compile** *form* *T**lib* form compiler

**view** *form* *vector* vector of object

**utime** *fixnum* elapsed time usec

**%if** *T T’ T’’* *key* **:if** implementation

**repr**  *type* *T* *T* tag representation

*type* :t :vector

if type is :vector, return 8 byte

byte vector of argument tag bits,

otherwise convert argument byte vector to tag.

**fix** *fn form* *T* fixpoint of *function*

**gc** *bool* garbage collection

**version** *string*versionstring

***Future s***

**defer** *fnlist* *struct* future application

**detach** *fnlist* *struct* future application

**force** *struct* *T* force completion

**poll** *struct* *bool* poll completion

***Fixnum m***

**fx-mul** *fix* *fix’* *fixnum*product

**fx-add** *fix* *fix’* *fixnum*sum

**fx-sub** *fix* *fix’* *fixnum*difference

**fx-lt** *fix* *fix’* *bool fix* < *fix’?*

**fx-div** *fix* *fix*’ *fixnum*quotient

**ash** *fix* *fix’* *fixnum*arithmetic shift

**logand** *fix* *fix’* *fixnum*bitwise and

**logor** *fix* *fix*’ *fixnum*bitwise or

**lognot** *fix* *fixnum*bitwise complement

***Float t***

**fl-mul** *fl* *fl’* *float* product

**fl-add** *fl* *fl’* *float* sum

**fl-sub** *fl* *fl’* *float* difference

**fl-lt** *fl* *fl’* *bool fl* < *fl’?*

**fl-div** *fl* *fl’* *float* quotient

***Conses/Lists s***

**append** *list T* *list* append

**car** *list* *list* head of *list*

**cdr** *list* *T* tail of *list*

**cons** *T* *T’* *cons* (*form* . *form*’)

**length** *list* *fixnum* length of *list*

**nth** *fix* *list* *T n*th *car* of *list*

**nthcdr** *fix* *list* *T n*th *cdr* of *list*

***Vector s***

**make-vector** *key* *list* *vector* specialized vector

from list

**vector-len** *vector* *fixnum* length of *vector*

**vector-ref** *vector* *fix* *T* *n*th element

**vector-type** *vector* *key* type of *vector*

***Reader/Printer s***

**read** *stream* *bool* *T* *T* read stream object

**write** *T* *bool stream T* write escaped object

***Struct t***

**make-struct** *key list struct* of type *key* from *list*

**struct-type** *struct* *key struct* type *keyword*

**struct-vec** *struct* *vector* of *struct* members

***Exception n***

**unwind-protect** *fn fn’* *T* catchexception

*fn*  - (:lambda (*obj**cond src*) ***.*** *body*)

*fn’* - (:lambda () **.** *body*)

**raise** *T* *keyword* raise exception

with condition

:arity :eof :open :read

:syscall :write :error :syntax

:type :sigint :div0 :stream

:range :except :future :ns

:over :under :unbound :return

***Streams n***

**standard*-*input** *symbol* std input *stream*

**standard*-*output** *symbol* std output *stream*

**error*-*output** *symbol* std error *stream*

**open** *type* *dir string stream* open *stream*

*type* :file :string

*dir* :input :output :bidir

**close** *stream* *bool* close *stream*

**openp** *stream* *bool* is *stream* open?

**flush** s*tream* *bool* flush output steam

**get-str** *stream string* from *string* *stream*

**rd-byte** *stream bool T byte* read *byte* from

*stream,* error on

eof, *T:* eof value

**rd-char** *stream bool T char* read *char* from *stream,* error on

eof, *T:* eof value

**un-char** *char* *stream* *char* push *char* onto

*stream*

**wr-byte** *byte* *stream* *byte* write *byte* to *stream*

**wr*-*char** *char* *stream char* write *char* to *stream*

***Namespace*  *Excepti***

**make-ns** *string* *ns* make *namespace*

**ns-map** *ns* *list* list of mapped *namespaces*

**ns-name** *ns* *string namespace* name

**unintern** *ns string symbol* intern unbound symbol

**intern** *ns string value symbol* intern bound symbol

**find-ns** *string* *ns* map *string* to *namespace*

**find** *ns* *string* *symbol* map *string* to *symbol*

**symbols** *type* *ns list namespace symbols*

***Features***  *I*

**[dependencies]**

**default = [ “nix”, "std", "sysinfo" ]**

**nix** uname

**std** command, exit

**sysinfo** sysinfo (disabled on macOS)

***libenv******API*** *I*

*[****dependencies]***

***mu = {***

***git = “***[***https://github.com/Software-Knife-and-Tool/mu.git***](https://github.com/Software-Knife-and-Tool/mu.git)***”,***

***branch=main***

***}***

**use libenv::{Condition, Config, Env, Exception, Result, Tag}**

***config string format: “npages:N,gcmode:GCMODE”***

***GCMODE – { none, auto, demand }***

***If the signal\_exception() interface is called, ^C will***

***generate a :sigint exception.***

**impl Env {**

**const VERSION: &str**

**fn signal\_exception()**

**fn config(config: Option<String>) → Option<Config>**

**fn new(config: &Config) → Mu**

**fn apply(&self, func: Tag, args: Tag) → Result<Tag>**

**fn compile(&self, form: Tag) → Result<Tag>**

**fn eq(&self, func: Tag, args: Tag) → bool;**

**fn exception\_string(&self, ex: Exception) → String**

**fn eval(&self, exp: Tag) → Result<Tag>**

**fn eval\_str(&self, exp: &str) → Result<Tag>**

**fn load(&self, file\_path: &str) → Result<bool>**

**fn load\_image(&self, path: &str) → Result<bool>;**

**fn read(&self, st: Tag, eofp: bool, eof: Tag) → Result<Tag>**

**fn read\_str(&self, str: &str) → Result<Tag>**

**fn save\_and\_exit(&self, path: &str) → Result<bool>**

**fn err\_out(&self) → Tag**

**fn std\_in(&self) → Tag**

**fn std\_out(&self) → Tag**

**fn write(&self, exp: Tag, esc: bool, st: Tag) → Result<()>**

**fn write\_str(&self, str: &str, st: Tag) → Result<()>**

**fn write\_to\_string(&self, exp: Tag, esc: bool) → String**

***Reader Syntax x***

**; comment to end of line**

**#|...|# block comment**

**‘*form* quoted form**

**`*form* backquoted form**

**`(*...)* backquoted list (proper lists)**

**,*form* eval backquoted form**

**,@*form* eval-splice backquoted form**

**(…) constant *list***

**() empty *list*, prints as :nil**

**(… . .) dotted *list***

**“…” *string, char vector***

***\* single escape in strings**

**#x hexadecimal *fixnum***

**#\c *char***

**#(:type …) *vector***

**#s(:type …) *struct***

**#:symbol uninterned *symbol***

**“`,; terminating macro char**

**# non-terminating macro char**

**!$%&\*+-. symbol constituents**

**<>=?@[]|**

**:^\_{}~/**

**A..Za..z**

**0..9**

**0x09 #\tab whitespace**

**0x0a #\linefeed**

**0x0c #\page**

**0x0d #\return**

**0x20 #\space**

*mu-sys* ***x***

**mu-sys: x.y.z: [-h?pvcelq0] [file…]**

?: usage message

h: usage message

c: [name:value,…]

e: eval [form] and print result

l: load [path]

p: pipe mode (no repl)

q: eval [form] quietly

v: print version and exit

0: null terminate