

# libenv Reference

lib namespace, version 0.1.47

## Type Keywords and aliases

<b>supertype</b>	<i>T</i>
<b>bool</b>	() , :nil are false, otherwise true
<b>condition</b>	keyword, see <b>Exception</b>
<b>list</b>	cons or () , :nil
<b>frame</b>	cons, see <b>Frame</b>
<b>ns</b>	keyword or () , see <b>Namespace</b>
: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
:stream	stream, strm file or string type
:struct	struct typed vector
:symbol	symbol, sym LISP-1 symbol
:vector	vector, string, str
	:char :t :byte :fixnum :float

## Heap

<b>hp-info</b>	vector heap static information #(:t type pages pagesize)
<b>hp-stat</b>	vector heap allocations #(:t :type size total free ...)
<b>hp-size T</b>	fixnum heap occupancy in bytes

## Frame

frame binding: (fn . #(:t ...))

<b>frames</b>	list active frame binding list
<b>fr-pop fn</b>	fn, pop.function's top frame binding
<b>fr-push frame</b>	cons push frame binding
<b>fr-ref fix fix</b>	T frame id, offset

## Symbol

<b>boundp sym</b>	bool is symbol bound?
<b>keyword str</b>	key keyword from string
<b>symbol str</b>	symbol uninterned symbol
<b>sy-ns sym</b>	key symbol namespace
<b>sy-name sym</b>	string symbol name binding
<b>sy-val sym</b>	T symbol value binding

## Special Forms

<b>:lambda list . list'</b>	function anonymous function
<b>:quote form</b>	list quoted form
<b>:if form T T'</b>	T conditional

## Core

<b>apply fn list</b>	T apply function to list
<b>eval form</b>	T evaluate form
<b>eq T T'</b>	bool are T and T' identical?
<b>type-of T</b>	keyword

<b>compile form</b>	T mu form compiler
<b>view form</b>	vector vector of object
<b>utime</b>	fixnum elapsed time usec
<b>repr type T</b>	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.

<b>fix fn form</b>	T fixpoint of function on form
<b>gc bool</b>	bool garbage collection, verbose

<b>version</b>	string type symbol, version string
----------------	------------------------------------

## Future

<b>future fn list</b>	struct create future
<b>fwait struct</b>	T wait for complete
<b>fdone struct</b>	bool poll completion

## Fixnum

<b>fx-mul fix fix'</b>	fixnum product
<b>fx-add fix fix'</b>	fixnum sum
<b>fx-sub fix fix'</b>	fixnum difference
<b>fx-lt fix fix'</b>	bool fix < fix'?
<b>fx-div fix fix'</b>	fixnum quotient
<b>ash fix fix'</b>	fixnum arithmetic shift
<b>logand fix fix'</b>	fixnum bitwise and
<b>logor fix fix'</b>	fixnum bitwise or
<b>lognot fix</b>	fixnum bitwise complement

## Float

<b>fl-mul fl fl'</b>	float product
<b>fl-add fl fl'</b>	float sum
<b>fl-sub fl fl'</b>	float difference
<b>fl-lt fl fl'</b>	bool fl < fl'?
<b>fl-div fl fl'</b>	float quotient

## Conses/Lists

<b>append list T</b>	list append
<b>car list</b>	list head of list
<b>cdr list</b>	T tail of list
<b>cons T T'</b>	cons (form . form')
<b>length list</b>	fixnum length of list
<b>nth fix list</b>	T nth car of list
<b>nthcdr fix list</b>	T nth cdr of list

## Vector

<b>vector key list</b>	vector specialized vector from list
<b>sv-len vector</b>	fixnum length of vector
<b>sv-ref vector fix T</b>	nth element
<b>sv-type vector</b>	key type of vector

## Reader/Printer

<b>read strm bool T → T</b>	read stream object
<b>write T bool strm → T</b>	write escaped object

## Struct

<b>struct key list</b>	struct of type key from list
<b>st-type struct</b>	key struct type keyword
<b>st-vec struct</b>	vector of struct members

## Exception

**with-ex** *fn fn' T* catch exception  
*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  
 :ns :over :under :unbound

## Stream

**std-in** *symbol* standard input *stream*  
**std-out** *symbol* standard output *stream*  
**err-out** *symbol* standard error *stream*

**open** *type direction string*  
*stream* open *stream*  
*type* - :file :string  
*direction* - :input :output :bidir

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

**flush** *stream 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

**make-ns** *ns key* make namespace  
**ns-map** *list* list of mapped namespaces  
**unbound** *ns string*  
*symbol* intern unbound symbol  
**intern** *ns string value*  
*symbol* intern bound symbol  
**ns-find** *ns string*  
*symbol* map *string* to *symbol*  
**ns-syms** *type ns*  
*T* namespace's *symbols*  
*type* - :list :vector

## Features

[dependencies]  
 default = [ "nix", "std", "sysinfo" ]

**nix:** uname  
**std:** command, exit  
**sysinfo:** sysinfo

## libenv API

[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

;  
 # | ... | #  
 'form  
 `form  
 `(...)  
 ,form  
 ,@form  
 (...)  
 ()  
 (... . .)

comment to end of line  
 block comment  
 quoted form  
 backquoted form  
 backquoted list (proper lists only)  
 eval backquoted form  
 eval-splice backquoted form  
 constant *list*  
 empty *list*, prints as :nil  
 dotted *list*

"..."  
 |  
 #x  
 #\c  
 #(:type ...)  
 #s(:type ...)  
 #:symbol  
 "`;  
 #  
 !\$%&\*+-.  
 <=>?@[| |  
 :^\_{ }~/  
 A..Za..z  
 0..9

*string*, *char vector*  
 single escape in strings  
 hexadecimal *fixnum*  
*char*  
*vector*  
*struct*  
 uninterned *symbol*  
 terminating macro char  
 non-terminating macro char  
 symbol constituents

0x09 #\tab whitespace  
 0x0a #\linefeed  
 0x0c #\page  
 0x0d #\return  
 0x20 #\space

## Runtime

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