

# libenv Reference

lib namespace, version 0.1.54

## 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>	:ns or ( ), see <b>Namespace</b>	
<b>:null</b>	( ), :nil	
<b>:char</b>	char	
<b>:cons</b>	cons	
<b>:fixnum</b>	fixnum, fix	56 bit signed integer
<b>:float</b>	float, fl	32 bit IEEE float
<b>:func</b>	function, fn	function
<b>:keyword</b>	keyword, key	symbol
<b>:ns</b>	namespace, ns	namespace
<b>:stream</b>	stream, strm	file or string type
<b>:struct</b>	struct	typed vector
<b>:symbol</b>	symbol, sym	LISP-1 symbol
<b>:vector</b>	vector, string, str	
	:char :t :byte :fixnum :float	

## Heap

<b>heap-info</b>	vector	heap static information
	#(:t type pages pagesize)	
<b>heap-stat</b>	vector	heap allocations
	#(:t :type size total free ...)	
<b>heap-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>make-symbol str</b>	symbol	uninterned symbol
<b>symbol-ns sym</b>	key	namespace
<b>symbol-name sym</b>	string	name binding
<b>symbol-value sym</b>	T	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>	key	type 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>%if T T' T''</b>	key	:if implementation
<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</b>	bool	garbage collection, verbose
<b>version</b>	string	type symbol, version string

## Future

<b>defer fn list</b>	struct	future application
<b>detach fn list</b>	struct	future application
<b>force struct</b>	T	force completion
<b>poll 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</b>	T	read stream object
<b>write T bool strm</b>	T	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 n

**unwind** *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 :future :ns
:over :under :unbound :return
```

## Streams n

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

**open** *type direction string* 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* read *byte* from *stream*, error on eof, *T*: eof value  
*byte*

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

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

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

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

## Namespace Excepi

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

## 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",
  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 only)  
*,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