

libenv Reference

lib namespace, version 0.1.54

Type Keywords and aliases

supertype	<i>T</i>	
bool	(), :nil are false, otherwise true	
condition	keyword, see Exception	
list	:cons or (), :nil	
frame	cons, see Frame	
ns	:ns or (), 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, 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

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

Frame

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

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

Symbol

boundp sym	bool	is symbol bound?
make-symbol str	symbol	uninterned symbol
symbol-ns sym	key	namespace
symbol-name sym	string	name binding
symbol-value sym	T	value binding

Special Forms

:lambda list . list'	function	anonymous function
:quote form	list	quoted form
:if form T T'	T	conditional

Core

apply fn list	T	apply function to list
eval form	T	evaluate form
eq T T'	bool	are T and T' identical?
type-of T	key	type keyword
compile form	T	mu 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 on form
gc	bool	garbage collection, verbose
version	string	type symbol, version string

Future

defer fn list	struct	future application
detach fn list	struct	future application
force struct	T	force completion
poll struct	bool	poll completion

Fixnum

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

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

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	nth car of list
nthcdr fix list	T	nth cdr of list

Vector

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

Reader/Printer

read strm bool T	T	read stream object
write T bool strm	T	write escaped object

Struct

struct key list	struct	of type key from list
st-type struct	key	struct type keyword
st-vec struct	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*
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 Excepi

make-ns *string ns* make *namespace*
ns-map *list* list of mapped namespaces
untern *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*
T *namespace's symbols*
type - :list :vector

Features

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

nix: uname
std: command, exit
sysinfo: sysinfo (disabled on macOS)

libenv API

[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 . . Z a . . 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