

libcore Reference

libcore version 0.0.39

Type Keywords and aliases

<i>supertype</i>	<i>T</i>	
<i>bool</i>	<code>()</code> , <code>:nil</code> are false, otherwise true	
<i>condition</i>	keyword, see Exception	
<i>list</i>	<code>cons</code> or <code>()</code> , <code>:nil</code>	
<i>frame</i>	<code>cons</code> , see Frame	
<code>:null</code>	<code>()</code> , <code>:nil</code>	
<code>:asyncid</code>	<i>async</i>	async future id
<code>:char</code>	<i>char</i>	
<code>:cons</code>	<i>cons</i>	
<code>:fixnum</code>	<i>fixnum</i> , <i>fix</i>	56 bit signed integer
<code>:float</code>	<i>float</i> , <i>fl</i>	32 bit IEEE float
<code>:func</code>	<i>function</i> , <i>fn</i>	function
<code>:keyword</code>	<i>keyword</i> , <i>key</i>	symbol
<code>:stream</code>	<i>stream</i>	file or string type
<code>:struct</code>	<i>struct</i>	typed vector
<code>:symbol</code>	<i>symbol</i> , <i>sym</i>	LISP-1 symbol
<code>:vector</code>	<i>vector</i> , <i>string</i> , <i>str</i>	
	<code>:char</code> <code>:t</code> <code>:byte</code> <code>:fixnum</code> <code>:float</code>	

Heap

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

Frame

		frame binding: <code>(fn . #(:t ...))</code>
frames	<i>list</i>	active frame binding list
fr-pop <i>fn</i>	<i>fn</i> ,	pop function's top frame binding
fr-push <i>frame</i>	<i>cons</i>	push frame binding
fr-ref <i>fix fix</i>	<i>T</i>	frame id, offset

Struct

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

Symbol

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

Special Forms

*:async <i>fn . list</i>	<i>async</i>	create <i>future</i> context
:lambda <i>list . list'</i>	<i>function</i>	anonymous function
:quote <i>form</i>	<i>list</i>	quoted form
:if <i>form T T'</i>	<i>T</i>	conditional

Core

apply <i>fn list</i>	<i>T</i>	apply <i>function</i> to <i>list</i>
eval <i>form</i>	<i>T</i>	evaluate <i>form</i>
eq <i>T T'</i>	<i>bool</i>	are <i>T</i> and <i>T'</i> identical?
type-of <i>T</i>	<i>keyword</i>	
*await <i>async</i>	<i>T</i>	return value of <i>async</i> future
*abort <i>async</i>	<i>T</i>	abort future
compile <i>form</i>	<i>T</i>	<i>mu</i> form compiler
view <i>form</i>	<i>vector</i>	vector of object
repr <i>type T</i>	<i>T</i>	tag representation
	<i>type</i>	- <code>:t</code> <code>:vector</code>
		if type is <code>:vector</code> , return 8 byte byte vector of argument tag bits, otherwise convert argument byte vector to tag.

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

Fixnum

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

Float

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

Conses/Lists

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

Vector

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

System

sys-tm	<i>fixnum</i>	system time in <i>us</i>
proc-tm	<i>fixnum</i>	process time in <i>us</i>
getpid	<i>fixnum</i>	process id
getcwd	<i>string</i>	<code>getcwd(2)</code>
uname		<code>struct</code> <code>uname(2)</code>
spawn <i>str list</i>	<i>fixnum</i>	spawn command
sysinfo		<code>struct</code> <code>sysinfo(2)</code>
exit	<i>fixnum</i>	exit shell with <i>fixnum</i>

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

Reader/Printer

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

libcore API

```
[dependencies]
mu = { git =
  "https://github.com/Software-Knife-and-Tool/mu.git",
  branch=main }
```

```
use libcore::{Condition, Config, Exception, Mu, Result, Tag}
```

```
config string format: "npages:N,gcmode:GCMODE"
GCMODE = { none, auto, demand }
```

```
impl Mu {
  const Mu::VERSION: &str

  fn config(config: String) -> Option<Config>;
  fn new(config: &Config)-> Mu;

  fn apply(&self, func: Tag, args: Tag)-> Result;
  fn compile(&self, form: Tag) -> Result;
  fn eq(&self, func: Tag, args: Tag) -> bool;
  fn exception_string(&self, ex: Exception) -> String;
  fn eval(&self, expr: Tag) -> Result;
  fn eval_str(&self, expr: &str) -> Result;
  fn load(&self, file_path: &str) -> Result;
  fn load_image(&self, file_path: &str) -> Result;
  fn read(&self, stream: Tag, eofp: bool, eof: Tag) -> Result;
  fn read_str(&self, str: &str) -> Result;
  fn err_out(&self) -> Tag
  fn save_and_exit(&self, file_path: &str) -> Result;
  fn std_in(&self) -> Tag
  fn std_out(&self) -> Tag
  fn write(&self, expr: Tag, esc: bool, stream: Tag) -> Result
  fn write_str(&self, str: &str, stream: Tag) -> Result;
  fn write_to_string(&self, stream: Tag) -> Result;
}
```

Reader Syntax

;
 #|...|# 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

Runtime

mu-sys: x.y.z: [-h?pvcelq] [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