

mu/lib Reference

lib: namespace version 0.0.41

Type Keywords and aliases

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

Heap

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

Frame

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

Struct

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

Symbol

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

Special Forms

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

Core

apply fn list	<i>T</i>	apply <i>function</i> to <i>list</i>
eval form	<i>T</i>	evaluate <i>form</i>
eq T T'	<i>bool</i>	are <i>T</i> and <i>T'</i> identical?
type-of T	<i>keyword</i>	
*await async	<i>T</i>	return value of <i>async</i> future
*abort async	<i>T</i>	abort future
compile form	<i>T</i>	<i>mu</i> form compiler
view form	<i>vector</i>	vector of object
utime	<i>fixnum</i>	elapsed time usec
repr type T	<i>T</i>	tag representation
	<i>type</i>	- :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	<i>T</i>	fixpoint of <i>function</i> on <i>form</i>
gc bool	<i>bool</i>	garbage collection, verbose
version	<i>string</i>	type <i>symbol</i> , version string

Fixnum

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

Float

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

Conses/Lists

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

Vector

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

Reader/Printer

read stream bool T	<i>T</i>	read stream object
write T bool stream	<i>T</i>	write escaped object

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

Features

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

nix: uname
std: command, exit
sysinfo: sysinfo

mu/lib 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