librt Reference

lib **namespace, version *0.1.59***

type keywords and aliases

supertype	*T*	
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*	file or string type
`:struct`	*struct*	typed vector
`:symbol`	*symbol, sym*	LISP-1 symbol
`:vector`	*vector, string*	
	`:char :t :byte :fixnum :float`	

Heap

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

Frame

frames	*list*	active *frames*
frame-pop *fn*	*fn*	pop *function's* top frame binding
	frame binding: `(fn . #(:t ...))`	
frame-push *frame*	*cons*	push frame binding
frame-ref *fix fix*	*T*	frame id, offset

Symbol

boundp *symbol*	*bool*	is *symbol* bound?
make-symbol *string*	*symbol*	uninterned *symbol*
makunbound *string*	*symbol*	unbound *symbol*
symbol-ns *symbol*	*key*	*namespace*
symbol-name *symbol*	*string*	name binding
symbol-value *symbol*	*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*	*T* and *T'* identical?
type-of *T*	*key*	type keyword
compile *form*	*T*	*lib* 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*
gc	*bool*	garbage collection
version	*string*	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

product *fix fix'*	*fixnum*	product
sum *fix fix'*	*fixnum*	sum
difference *fix fix'*	*fixnum*	difference
less-than *fix fix'*	*bool*	*fix < fix'?*
quotient *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*	*n*th *car* of *list*
nthcdr *fix list*	*T*	*n*th *cdr* of *list*

Vector

make-vector *key list*	*vector*	specialized vector from list
vector-len *vector*	*fixnum*	length of *vector*
vector-ref *vector fix*	*T*	*n*th element
vector-type *vector*	*key*	type of *vector*

Reader/Printer

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

Struct

make-struct *key list*	*struct*	of type *key* from *list*
struct-type *struct*	*key*	*struct* type *keyword*
struct-vec *struct*	*vector*	of *struct* members

Exception

unwind-protect *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

standard-input	*symbol*	std input *stream*
standard-output	*symbol*	std output *stream*
error-output	*symbol*	std error *stream*
open *`type dir`* *string*	*stream*	open *stream*

```
type    :file   :string
dir     :input  :output  :bidir
```

close *stream*	*bool*	close *stream*
openp *stream*	*bool*	is *stream* open?
flush *stream*	*bool*	flush output *steam*
get-string *stream*	*string*	from *string stream*
read-byte *stream bool T*	*byte*	read *byte* from *stream,* error on eof, *T:* eof value
read-char *stream bool T*	*char*	read *char* from *stream,* error on `eof`, *T:* `eof` value
unread-char *char stream*	*char*	push *char* onto *stream*
write-byte *byte stream*	*byte*	write *byte* to *stream*
write-char *char stream*	*char*	write *char* to *stream*

Namespace

make-ns *string*	*ns*	make *namespace*
ns-map *ns*	*list*	list of mapped *namespaces*
ns-name *ns*	*string*	*namespace* name
unintern *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*	*list*	*namespace symbols*

Features

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

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

librt 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

`;`	comment to end of line
`#|...|#`	block comment
'*form*	quoted form
`` ` ``*form*	backquoted form
`` ` ``(…)	backquoted list (proper lists)
,*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*
`#.`	read-time eval
`#\.`	*char*
`#(:type …)`	*vector*
`#s(:type …)`	*struct*
`#:symbol`	uninterned *symbol*
" ` , ;	terminating macro char
`#`	non-terminating macro char
`!$%&*+-.` `<>=?@[]|` `:^_{}~/` `A..Za..z` `0..9`	symbol constituents
`0x09 #\tab` `0x0a #\linefeed` `0x0c #\page` `0x0d #\return` `0x20 #\space`	whitespace

mu-sys

```
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
```