

# Core Library Reference

mu name space, version 0.1.67

## type keywords and aliases

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

<b>heap-info</b>	vector	heap 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

## Frame

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

## Symbol

<b>boundp symbol</b>	bool	is symbol bound?
<b>make-symbol string</b>	symbol	uninterned symbol
<b>makunbound string</b>	symbol	unbound symbol
<b>symbol-ns symbol</b>	key	namespace
<b>symbol-name symbol</b>	string	name binding
<b>symbol-value symbol</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	T and T' identical?
<b>type-of T</b>	key	type keyword
<b>compile form</b>	T	lib 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
<b>gc</b>	bool	garbage collection
<b>*version*</b>	string	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>product fix fix'</b>	fixnum	product
<b>sum fix fix'</b>	fixnum	sum
<b>difference fix fix'</b>	fixnum	difference
<b>less-than fix fix'</b>	bool	fix < fix'?
<b>quotient 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>make-vector key list</b>	vector	specialized vector from list
<b>vector-len vector</b>	fixnum	length of vector
<b>vector-ref vector fix</b>	T	nth element
<b>vector-type vector</b>	key	type of vector

## Reader/Printer

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

## Struct

<b>make-struct key list</b>	struct	of type key from list
<b>struct-type struct</b>	key	struct type keyword
<b>struct-vec struct</b>	vector	of struct members

## Exception n

**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 n

**\*standard-input\*** *stream* std input *stream*  
**\*standard-output\*** *stream* std output *stream*  
**\*error-output\*** *stream* 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 *stream*  
**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 namespaces* list of mapped *namespaces*  
**ns-name** *ns string namespace name*  
**unintern** *ns string symbol* *unintern 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** *types ns list namespace symbols*

## Features

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

**nix** uname  
**std** command, exit  
**sysinfo** sysinfo (disabled on macOS)  
**ffi** Rust FFI

## core library API

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

```
use crux::{
  Condition, Config, Env, Exception, Result, Tag
};
```

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

```
impl Env {
  const VERSION: &str
  fn signal_exception() // enable ^C :sigint exception
  fn config(config: Option<String>) -> Option<Config>
  fn new(config: &Config, Option<Vec<u8>>) -> Env
  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 read(&self, st: Tag, eofp: bool, eof: Tag) -> Result<Tag>
  fn read_str(&self, str: &str) -> Result<Tag>
  fn image(&self) -> Result<Vec<u8>>
  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)
,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

#*... bit vector
#x... hexadecimal fixnum
#. read-time eval
#\ 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

**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
```