

# Mu Library Reference

mu name space, version 0.1.80

## 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, str	
	: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

## Frames

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

## Symbols

<b>boundp</b> symbol	bool	is symbol bound?
<b>make-symbol</b> string	symbol	uninterned symbol
<b>symbol-namespace</b> symbol	key	namespace
<b>symbol-name</b> symbol	string	name binding
<b>symbol-value</b> symbol	T	value binding

## Special Forms

<b>:lambda</b> list . List'	function	anonymous function
<b>:quote</b> form	list	quoted form
<b>:if</b> form T T'	T	conditional

## Core

<b>apply</b> fn list	T	apply function to list
<b>eval</b> form	T	evaluate form
<b>eq</b> T T'	bool	T and T' identical?
<b>type-of</b> T	key	type keyword
<b>compile</b> form	T	mu form compiler
<b>view</b> form	vector	vector of object
<b>internal-run-time</b>	fixnum	elapsed time usec
<b>%if</b> T T' T''	key	:if implementation
<b>repr</b> 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.

<b>fix</b> fn T	T	fixpoint of function
<b>gc</b>	bool	garbage collection
<b>*version*</b>	string	version string

## Futures

<b>defer</b> fn list	struct	future application
<b>detach</b> fn list	struct	future application
<b>force</b> struct	T	force completion
<b>poll</b> struct	bool	poll completion

## Fixnum

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

## Float

<b>fmul</b> fl fl'	float	product
<b>fadd</b> fl fl'	float	sum
<b>fsub</b> float	float	difference
<b>fless-than</b> fl fl'	bool	fl < fl'?
<b>fdiv</b> fl fl'	float	quotient

## Conses/Lists

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

## Vectors

<b>make-vector</b> key list	vector	specialized vector from list
<b>vector-size</b> vector	fixnum	length of vector
<b>vector-type</b> vector	key	type of vector
<b>svref</b> vector fix	T	nth element

## Reader/Printer

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

## Structs

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

## Exception n

**with-exception** *fn fn' T* catch exception

```
fn - (:lambda (obj cond src) . body)
fn' - (:lambda () . body)
```

**raise** *T keyword* raise exception on *T* with condition:

```
:arity :div0 :eof :error :except
:future :ns :open :over :quasi
:range :read :return :sigint :stream
:syntax :syscall :type :unbound :under
:write
```

## 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 n

**make-namespace** *str ns* make *namespace*  
**namespace-map** *list* list of mapped *namespaces*

**namespace-name** *ns string* *namespace* name  
**intern** *ns str value* *symbol* intern bound symbol  
**find-namespace** *str ns* map *string* to *namespace*

**find** *ns string* *symbol* map *string* to *symbol*  
**namespace-symbols** *ns list* *namespace* symbols

## Features I

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

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

## mu library API I

[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 n

**mu-sys: 0.0.2:** [celq] [file...]

```
c: [name:value,...]
e: eval [form] and print result
l: load [path]
q: eval [form] quietly
```