

# Mu Library Reference

mu name space, version 0.1.83

## 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</b> <i>T</i>	fixnum	heap occupancy

## Frames

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

## Symbols

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

## Special Forms

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

## Core

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

## Futures

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

## Fixnum

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

## Float

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

## Conses/Lists

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

## Vectors

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

## Reader/Printer

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

## Structs

<b>make-struct</b> <i>key list</i>	struct	of type <i>key</i> from <i>list</i>
<b>struct-type</b> <i>struct</i>	key	struct type keyword
<b>struct-vec</b> <i>struct</i>	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* 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 1

```
[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 1

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

```
use mu::{
  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
```