

# Mu Reference

libmu version 0.0.38

## Type Keywords and aliases

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

## Heap

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

## Frame

frame binding: `(fn . #(:t ...))`

<b>frames</b>	<i>list</i>	active frame binding list
<b>fr-pop</b> <i>fn</i>	<i>fn</i> ,	pop function's top frame binding
<b>fr-push</b> <i>frame</i>	<i>cons</i>	push frame binding
<b>fr-ref</b> <i>fix fix</i>	<i>T</i>	frame id, offset

## Struct

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

## Symbol

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

## Special Form

<b>:async</b> <i>fn . list</i>	<i>async</i>	create <i>future</i> context
<b>:lambda</b> <i>list . 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 <i>function</i> 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>	are <i>T</i> and <i>T'</i> identical?
<b>type-of</b> <i>T</i>	<i>keyword</i>	
<b>*await</b> <i>async</i>	<i>T</i>	return value of <i>async</i> future
<b>*abort</b> <i>async</i>	<i>T</i>	abort future
<b>compile</b> <i>form</i>	<i>T</i>	mu form compiler
<b>view</b> <i>form</i>	<i>vector</i>	vector of object
<b>repr</b> type <i>T</i>	<i>T</i>	tag representation
	type	- <code>:t</code> <code>:vector</code>
	if type is <code>:vector</code> , return 8 byte byte vector of argument tag bits, otherwise convert argument byte vector to tag.	
<b>fix</b> <i>fn form</i>	<i>T</i>	fixpoint of <i>function</i> on <i>form</i>
<b>gc</b> <i>bool</i>	<i>bool</i>	garbage collection, verbose

## Fixnum

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

## Float

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

## Conses/Lists

<b>append</b> <i>list T</i>	<i>list</i>	append
<b>car</b> <i>list</i>	<i>list</i>	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>	<i>cons</i>	( <i>form</i> . <i>form'</i> )
<b>length</b> <i>list</i>	<i>fixnum</i>	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>

## Vector

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

## System

<b>sys-tm</b>	<i>fixnum</i>	system time in <i>us</i>
<b>proc-tm</b>	<i>fixnum</i>	process time in <i>us</i>
<b>getpid</b>	<i>fixnum</i>	process id
<b>getcwd</b>	<i>string</i>	<code>getcwd(2)</code>
<b>uname</b>		<code>struct</code> <code>uname(2)</code>
<b>spawn</b> <i>str list</i>	<i>fixnum</i>	spawn command
<b>sysinfo</b>		<code>struct</code> <code>sysinfo(2)</code>
<b>exit</b>	<i>fixnum</i>	exit shell with <i>fixnum</i>

## 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*

## Reader/Printer

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

## libmu API

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

use libmu::{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 . . Z a . . 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