

# mu/lib Reference

lib: namespace version 0.0.42

## Type Keywords and aliases

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

## Heap

<b>hp-info</b>	<i>vector</i>	heap static information #(:t <i>type pages pagesize</i> )
<b>hp-stat</b>	<i>vector</i>	heap allocations #(:t :type size total free ...)
<b>hp-size T</b>	<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 fn</b>	<i>fn,</i>	pop function's top frame binding
<b>fr-push frame</b>	<i>cons</i>	push frame binding
<b>fr-ref fix fix</b>	<i>T</i>	frame id, offset

## Struct

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

## Symbol

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

## Special Forms

<b>*:async fn . list</b>	<i>async</i>	create <i>future</i> context
<b>:lambda list . list'</b>	<i>function</i>	anonymous function
<b>:quote form</b>	<i>list</i>	quoted form
<b>:if form T T'</b>	<i>T</i>	conditional

## Core

<b>apply fn list</b>	<i>T</i>	apply <i>function</i> to <i>list</i>
<b>eval form</b>	<i>T</i>	evaluate <i>form</i>
<b>eq T T'</b>	<i>bool</i>	are <i>T</i> and <i>T'</i> identical?
<b>type-of T</b>	<i>keyword</i>	
<b>*await async</b>	<i>T</i>	return value of <i>async</i> future
<b>*abort async</b>	<i>T</i>	abort future
<b>compile form</b>	<i>T</i>	<i>mu</i> form compiler
<b>view form</b>	<i>vector</i>	vector of object
<b>utime</b>	<i>fixnum</i>	elapsed time usec
<b>repr type T</b>	<i>T</i>	tag representation
	<i>type</i>	- :t :vector
	if <i>type</i> is :vector, return 8 byte byte vector of argument tag bits, otherwise convert argument byte vector to tag.	

<b>fix fn form</b>	<i>T</i>	fixpoint of <i>function</i> on <i>form</i>
<b>gc bool</b>	<i>bool</i>	garbage collection, verbose
<b>version</b>	<i>string</i>	type <i>symbol</i> , version string

## Fixnum

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

## Float

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

## Conses/Lists

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

## Vector

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

## Reader/Printer

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

## 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

## Features

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

**nix:** uname  
**std:** command, exit  
**sysinfo:** sysinfo

## mu/lib API

[dependencies]  
 mu = {  
 git = "https://github.com/Software-Knife-and-Tool/mu.git",  
 branch=main  
 }  
 use mu::{Condition, Config, Exception, Mu, Result, Tag}  
 config string format: "npages:N,gcmode:GCMODE"  
 GCMODE - { none, auto, demand }  
 impl Mu {  
 const VERSION: &str  
 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 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

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

## Runtime