

# Mu Reference

mu version 0.0.26

## 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>:map</code>	<i>map</i>	key/value hash
<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 :t :byte :fixnum :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 T</b>	<i>fixnum</i>	heap occupancy in bytes

## Frame

		frame binding: <code>(fn . #(:t ...))</code>
<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>make-st</b>	key list	
	<i>struct</i>	of type <i>key</i> from list
<b>st-type</b>	<i>struct</i>	key struct type keyword
<b>st-vec</b>	<i>struct</i>	vector 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>make-sy</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>	keyword	

<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 - :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 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
<b>exit</b>	<i>fix</i>		exit process with return code

## 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

## 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 . 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>make-sv</b>	keyword <i>list</i>	<i>vector</i>	typed vector from <i>list</i>
<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>

## Map

<b>make-mp</b>	<i>list</i>	<i>map</i>	<i>map</i> from assoc <i>list</i>
----------------	-------------	------------	-----------------------------------

<b>mp-ref</b>	<i>map T</i>	<i>T</i>	reference <i>map</i>
<b>mp-has</b>	<i>map T</i>	<i>bool</i>	is key resident?
<b>mp-size</b>	<i>map</i>	<i>fixnum</i>	size of <i>map</i>
<b>mp-list</b>	<i>map</i>	<i>list</i>	<i>map</i> contents

## Exception n

**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  
: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?  
**eof** *stream bool* is *stream* at end of file?  
**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*

## System m

**real-tm** *T fixnum* system clock secs  
**run-us** *T fixnum* process time  $\mu$ s

## 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 n

**read** *stream bool T*  
*T* read *stream* object

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

## Mu library API l

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

use mu::{Condition, Config, Exception,  
Mu, Result, System, Tag}

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

```
const Mu::VERSION: &str
Mu::new(config: &Config)-> Mu
Mu::config(config: String) -> Option<Config>
Mu::apply(&self, func: Tag, args: Tag)-> Result
Mu::eq(&self, func: Tag, args: Tag) -> Result
Mu::eval(&self, expr: Tag) -> Result
Mu::compile(&self, form: Tag) -> Result
Mu::read(&self, stream: Tag, eofp: bool, value: Tag) -> Result
Mu::write(&self, form: Tag, esc: bool, stream: Tag) -> Result
Mu::get_string(&self, stream: Tag) -> Result
Mu::write_string(&self, str: String, stream: Tag) -> Result
Mu::from_u64(&self, tag: u64) -> Tag
Mu::as_u64(&self, tag: Tag) -> u64
Mu::std_in(&self) -> Tag
Mu::std_out(&self) -> Tag
Mu::err_out(&self) -> Tag
```

```
System::new(config: &Config)-> System
System::config(config: String) -> Option<Config>
System::mu(&self) -> &Mu
System::eval(&self, expr: &String) -> Result
System::error(&self, ex: Exception) -> String
System::read(&self, string: String) -> Result
System::write(&self, expr: Tag, escape: bool) -> String
System::load(&self, file_path: &String) -> 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 uninerted *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

## Runtime

mu-shell: 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