

# Mu Runtime Reference

mu namespace, version 0.2.6

## 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>	
<code>:null</code>	<code>()</code> , <code>:nil</code>	
<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>:ns</code>	<i>namespace</i> , <i>ns</i>	namespace
<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>:bit</code> <code>:char</code> <code>:t</code>	
	<code>:byte</code> <code>:fixnum</code> <code>:float</code>	

## Features

[dependencies]			
default = [ "env", "procinfo", "std", "nix", "sysinfo" ]			
<b>env</b>	<b>heap-room</b>	<i>vector</i>	allocations
	<code>#(:t :type size total free ...)</code>		
	<b>heap-info</b>	<i>list</i>	heap info
	<code>(type page-size npages)</code>		
	<b>heap-size</b>	<i>keyword</i> <i>fixnum</i>	type size
	<b>heap-free</b>	<i>fixnum</i>	bytes free
	<b>env</b>	<i>list</i>	env state
	<b>core</b>	<i>list</i>	core state
<b>nix</b>	<b>uname</b>		
<b>std</b>	<b>command</b> , <b>exit</b>		
<b>sysinfo</b>	<b>sysinfo</b> (disabled on macOS)		
<b>procinfo</b>	<b>process-mem-virt</b>	<i>fixnum</i>	virtual memory in bytes
	<b>process-mem-res</b>	<i>fixnum</i>	reserve in bytes
	<b>process-time</b>	<i>fixnum</i>	microseconds
	<b>time-units-per-sec</b>	<i>fixnum</i>	
<b>prof</b>	<b>prof-control</b>		enable semispace heap

## configuration API

config string format:

"npages:N, gc-mode:GCMODE, page-size:N, heap-type:HEAPTYPE"

*N*: unsigned integer  
*GCMODE*: none | auto | demand  
*HEAPTYPE*: semispace | bump // needs semispace feature

## 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</i> <i>T</i> <i>T'</i>	<i>T</i>	conditional

## Reader/Printer

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

## Core

<b>*null/*</b>	<i>ns</i>	null namespace
<b>apply</b> <i>fn</i> <i>list</i>	<i>T</i>	apply <i>fn</i> to <i>list</i>
<b>eval</b> <i>form</i>	<i>T</i>	evaluate <i>form</i>
<b>eq</b> <i>T</i> <i>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>	<i>vector</i>	vector of object

<b>%if</b> <i>fn</i> <i>fn'</i> <i>fn''</i>	<i>bool</i>	<b>:if</b> implementation
---	-------------	---------------------------

<b>repr</b> <i>T</i>	<i>vector</i>	tag representation
<b>unrepr</b> <i>vector</i>	<i>T</i>	tag representation

*vector* is an 8 element `:byte` vector of little-endian argument tag bits.

<b>fix</b> <i>fn</i> <i>T</i>	<i>T</i>	fixpoint of <i>fn</i>
<b>gc</b>	<i>bool</i>	garbage collection

## Frames

<b>%frame-stack</b>	<i>list</i>	active frames
<b>%frame-pop</b> <i>fn</i>	<i>fn</i>	pop <i>function's</i> top frame binding
frame binding: ( <i>fn</i> . #(:t ...))		
<b>%frame-push</b> <i>frame</i>	<i>cons</i>	push frame
<b>%frame-ref</b> <i>fn</i> <i>fix</i>	<i>T</i>	<i>function</i> , 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>ns</i>	namespace
<b>symbol-name</b> <i>symbol</i>	<i>string</i>	name binding
<b>symbol-value</b> <i>symbol</i>	<i>T</i>	value binding

## Fixnums

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

## Floats

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

## Conses/Lists

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

## Vectors

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

## Streams n

**\*standard-input\*** *stream* std input *stream*  
**\*standard-output\*** *stream* std output *stream*  
**\*error-output\*** *stream* std error *stream*

**open** *type dir string bool*  
                                   *stream* open *stream*  
                                   raise error if *bool*

*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 *byte* to *stream*

## Namespaces .

**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

## Exceptions 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 :exit :signal :stream  
 :syntax :syscall :type :unbound :under  
 :write :storage

## Structs f

**make-struct** *key list struct* of type *key* from *list*  
**struct-type** *struct key* *struct* type *keyword*  
**struct-vec** *struct vector* of *struct* members

## mu library API f

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

```
use mu_runtime::{ Condition, Config, Env, Exception, Result,
Tag };
```

```
impl Env {
  const VERSION: &str

  fn config(config: Option<String>) -> Option<Config>
  fn new(config: &Config, Option<(Vec<u8>, 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>, 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 .

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

c: name:value,... runtime configuration  
 e: form eval and print result  
 l: path load from path  
 q: form eval quietly