

# Mu Namespace

mu version 0.0.1

## Type keywords and aliases

<i>supertype</i>	<i>T</i> , form
<i>bool</i>	() , :nil is false, otherwise true
<i>condition</i>	condition <i>keyword</i> (see <b>Exceptions</b> )
<i>type</i>	type-of returns <i>keyword</i> of:
<i>list</i>	cons or ()   :nil
:null	() , :nil
:char	char
:cons	cons,
:fixnum	fix, fixnum, a 61 bit signed integer
:float	float, fl a 32 bit IEEE float
:func	fn, a function
:ns	ns, collection of symbol bindings
:stream	stream, file or string type
:struct	struct
:symbol	sym, symbol, keyword
:vector	simple vector, string (:char)
	:t :byte:fixnum :float

## Heap

<b>hp-info</b>	vector, heap allocations #(:t type total alloc in-use)
----------------	---

## frames

<b>fr-get</b> <i>fn</i>	struct, copy frame binding
<b>fr-pop</b> <i>fn</i>	function, pop frame binding
<b>fr-push</b> struct	struct, push frame binding
<b>:fr-ref</b> fix fix	<i>T</i> , ref frame variable

## Reader/Printer

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

## Structs

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

## Symbols

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

## Special Forms

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

## Core

<b>eval</b> form	<i>T</i> , evaluate form
<b>eq</b> form form'	bool, are form and form' identical?
<b>type-of</b> form	keyword

<b>apply</b> fn list	<i>T</i> , apply function to list
<b>compile</b> form	<i>T</i> , library form compiler

<b>view</b> form	vector, vector of object
<b>fix</b> fn form	<i>T</i> , fixpoint of function on form

<b>::if</b> <i>T</i> fn fn'	<i>T</i> , :if implementation
<b>::frames</b>	cons, active frame list
<b>::gc</b>	bool, garbage collection

## System

<b>real-tm</b> <i>T</i>	fixnum, system clock secs
<b>run-us</b> <i>T</i>	fixnum, process time $\mu$ s

## Fixnums

<b>fx-mul</b> fix fix"	fixnum, product
<b>fx-add</b> fix fix'	fixnum, sum
<b>fx-sub</b> fix fix'	fixnum, difference
<b>fx-lt</b> fix fix'	bool, is fix less than fix'?
<b>fx-div</b> fix fix'	fixnum, quotient

<b>logand</b> fix fix'	fixnum, bitwise and
<b>logor</b> fix fix'	fixnum, bitwise or

## Floats

<b>fl-mul</b> fl fl"	float, product
<b>fl-add</b> fl fl'	float, sum
<b>fl-sub</b> fl fl'	float, difference
<b>fl-lt</b> fl fl'	bool, is fl less than fl'?
<b>fl-div</b> fl fl'	float, quotient

## Conses and Lists

<b>car</b> list	list, head of list
<b>cdr</b> list	list, tail of list
<b>cons</b> form form'	cons, from <i>T</i> and <i>T'</i>
<b>length</b> list	fixnum, length of list
<b>nth</b> fix list	<i>T</i> , nth car of list
<b>nthcdr</b> fix list	<i>T</i> , nth cdr of list

## Vectors

<b>make-sv</b> keyword list	vector, typed vector of list
<b>sv-len</b> vector	fixnum, length of vector
<b>sv-ref</b> vector fix	<i>T</i> , nth element
<b>sv-type</b> vector	keyword, type of vector

## Exceptions

<b>with-ex</b> fn fn' <i>T</i>	catch exception
	fn - (:lambda (obj condition) . list)
	fn' - (:lambda () . list)

**raise** *T* keyword raise exception with condition:

:arity :eof :open :read  
:write :error :syntax  
:type :unbound :div0  
:stream :except :range

## Streams

<b>std-in</b>	<i>symbol</i> , standard input <i>stream</i>
<b>std-out</b>	<i>symbol</i> , standard output <i>stream</i>
<b>err-out</b>	<i>symbol</i> , standard error <i>stream</i>
<b>open</b> type direction <i>string</i>	
	<i>stream</i> , open <i>stream</i>
	type - :file :string
	direction - :input :output
<b>close stream</b>	<i>bool</i> , close <i>stream</i>
<b>openp stream</b>	<i>bool</i> , is <i>stream</i> open?
<b>eof stream</b>	<i>bool</i> , is <i>stream</i> at end of file?
<b>flush stream</b>	<i>bool</i> , flush output steam
<b>get-str stream</b>	<i>string</i> , from <i>string stream</i>
<b>rd-byte stream</b>	<i>bool form</i>
	<i>byte</i> , read <i>byte</i> from <i>stream</i> ,
	<i>bool</i> : error on eof, <i>form</i> : eof value
<b>rd-char stream</b>	<i>bool form</i>
	<i>char</i> , read <i>char</i> from <i>stream</i> ,
	<i>bool</i> : error on eof, <i>form</i> : eof value
<b>wr-byte</b> <i>byte stream</i>	
	<i>byte</i> , write <i>byte</i> to <i>stream</i>
<b>wr-char</b> <i>char stream</i>	
	<i>char</i> , write <i>char</i> to <i>stream</i>
<b>un-char</b> <i>char stream</i>	
	<i>char</i> , push <i>char</i> onto <i>stream</i>

## Namespaces

<b>make-ns</b> <i>string ns</i>	
	<i>ns</i> , make <i>namespace</i>
<b>map-ns</b> <i>string ns</i>	<i>ns</i> , map <i>string</i> to <i>namespace</i>
<b>untern</b> <i>ns scope string</i>	
	<i>symbol</i> , intern unbound symbol
	scope - :intern :extern
<b>intern</b> <i>ns scope string value</i>	
	<i>symbol</i> , intern bound symbol
	scope - :intern :extern
<b>ns-find</b> <i>ns scope string</i>	
	<i>symbol</i> , map <i>string</i> to <i>symbol</i>
	scope - :intern :extern
<b>ns-imp</b> <i>ns</i>	<i>ns</i> , namespace's import
<b>ns-name</b> <i>ns</i>	<i>string</i> , namespace's name
<b>ns-int</b> <i>ns</i>	<i>list</i> , namespace's interns
<b>ns-ext</b> <i>ns</i>	<i>list</i> , namespace's externs

## library API

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

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

const Mu::VERSION: &str

Mu::new(config: String)-> Mu
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: String)-> System
System::mu(&self)-> &Mu
System::version(&self) -> String
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
,form	eval backquoted form
,@form	eval-splice backquoted form (not yet)
(...)	constant <i>list</i>
()	empty <i>list</i> , prints as :nil
"..."	<i>string</i> , <i>char vector</i>
#x	hexadecimal <i>fixnum</i>
#\c	<i>char</i>
#(:type ...)	<i>vector</i>
#s(:type ...)	<i>struct</i>
#:symbol	uninterned <i>symbol</i>
\	single escape in strings
"` , ;	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

runtime: xx.xx.xx: [-h?pvcedlq] [file...]

? : usage message  
h : usage message  
c : [name:value,...]  
d : enable debugging  
e : eval [form] and print result  
l : load [path]  
p : pipe mode (no repl)  
q : eval [form] quietly  
v : print version and exit