

# Mu Namespace

mu version 0.0.11

## Type keywords and aliases

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

## Heap

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

## frames

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 <i>function</i> '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

## Reader/Printer

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

## Structs

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

## Symbols

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

## Special Forms

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

## Core

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

<b>apply</b> <i>fn list</i>	<i>T</i> , apply <i>function</i> to <i>list</i>
<b>arity</b> <i>fn</i>	<i>fixnum</i> , function arity
<b>compile</b> <i>form</i>	<i>T</i> , library form compiler
<b>view</b> <i>form</i>	<i>vector</i> , vector of object
<b>repr</b> <i>bool T</i>	<i>T</i> , tag representation conversion: if <i>bool</i> is (), return 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> , garbage collection

## System

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

## Fixnums

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

## Floats

<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 and Lists

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

## Vectors

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

## Exceptions

<b>with-ex</b> <i>fn fn'</i>	<i>T</i> , catch exception <i>fn</i> - (:lambda ( <i>obj cond src</i> ) . <i>body</i> ) <i>fn'</i> - (:lambda () . <i>body</i> )
------------------------------	--

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

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

## 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> stream, 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>un-char char stream</b>	<i>char</i> , push <i>char</i> onto <i>stream</i>
<b>wr-byte byte stream</b>	<i>byte</i> , write <i>byte</i> to <i>stream</i>
<b>wr-char char stream</b>	<i>char</i> , write <i>char</i> to <i>stream</i>

## Namespaces

	<i>ns: #s(:ns name import)</i>
<b>make-ns string ns</b>	<i>ns</i> , make <i>namespace</i>
<b>map-ns string ns</b>	<i>ns</i> , map <i>string</i> to <i>namespace</i>
<b>untern ns string</b>	<i>symbol</i> , intern unbound <i>symbol</i>
<b>intern ns string value</b>	<i>symbol</i> , intern bound <i>symbol</i>
<b>ns-find ns string</b>	<i>symbol</i> , map <i>string</i> to <i>symbol</i>
<b>ns-imp ns</b>	<i>ns</i> , namespace's import
<b>ns-name ns</b>	<i>string</i> , namespace's name
<b>ns-syms ns</b>	<i>list</i> , namespace's symbols

## 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
`(...)	backquoted list (proper lists only)
,form	eval backquoted form
,@form	eval-splice backquoted form
(...)	constant <i>list</i>
()	empty <i>list</i> , prints as :nil
"..."	<i>string</i> , <i>char vector</i>
\	single escape in strings
#x	hexadecimal <i>fixnum</i>
#\c	<i>char</i>
#(:type ...)	<i>vector</i>
#s(:type ...)	<i>struct</i>
#:symbol	uninterned <i>symbol</i>
"`,";	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: x.y.z: [-h?pvcdlq] [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
```