

# Mu Namespace

mu version 0.0.21

## Type keywords and aliases

<i>supertype</i>	<i>T</i> , object
<i>bool</i>	() , :nil is false, otherwise true
<i>condition</i>	keyword, see <b>Exceptions</b>
<i>type</i>	type-of returns keyword
<i>list</i>	cons or () , :nil
<i>frame</i>	see <b>Frames</b>
<i>string</i>	char vector
:null	() , :nil
:async id	async future id
:char	char
:cons	cons
:fixnum	fix, fixnum, 56 bit signed integer
:float	float, fl, 32 bit IEEE float
:func	fn, a function
:keyword	keyword symbol
:map	map object
:stream	stream, file or string type
:struct	struct
:symbol	sym, symbol
:vector	simple vector, string (:char) :t :byte :fixnum :float

## Heap

<b>hp-info</b>	vector, heap allocations #(:t type total alloc in-use)
<b>size-of T</b>	fixnum, size in bytes of object

## Frame

frame binding: (fn . #(:t ...))

<b>frames</b>	list, active frame binding list
<b>fr-pop fn</b>	fn, pop function's top frame binding
<b>fr-push frame</b>	cons, push frame binding
<b>fr-ref fix fix</b>	T, frame id, offset

## Struct

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

## Symbol

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

## Special Forms

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

## Core

<b>apply fn list</b>	T, apply function to list
<b>eval form</b>	T, evaluate form
<b>eq T T'</b>	bool, are T and T' identical?
<b>type-of T</b>	keyword

<b>*await: async</b>	T, return value of async future
<b>*abort: async</b>	T, abort future

<b>compile form</b>	T, library form compiler
<b>view form</b>	vector, vector of object

<b>repr bool T</b>	T, tag representation conversion: if bool is (), return 8 byte vector of argument tag bits, otherwise convert argument byte vector to tag
--------------------	--

<b>fix fn form</b>	T, fixpoint of function on form
<b>*gc</b>	bool, garbage collection
<b>exit fix</b>	exit process with return code

## Fixnum

<b>fx-mul fix fix'</b>	fixnum, product
<b>fx-add fix fix'</b>	fixnum, sum
<b>fx-sub fix fix'</b>	fixnum, difference
<b>fx-lt fix fix'</b>	bool, fix < fix'
<b>fx-div fix fix'</b>	fixnum, quotient
<b>logand fix fix'</b>	fixnum, bitwise and
<b>logor fix fix'</b>	fixnum, bitwise or

## Float

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

## Conses and Lists

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

## Vector

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

## Map

<b>make-mp</b>	map, make a new map
<b>mp-add map T T'</b>	map, add pair to map
<b>mp-get map T</b>	T, reference map
<b>mp-has map T</b>	bool, is key resident?
<b>mp-size map</b>	fixnum, size of map
<b>mp-list map</b>	cons, map contents

## 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  
: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

**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*,  
*bool*: error on eof, *T*: eof value

**rd-char** *stream* *bool T*  
*char*, read *char* from *stream*,  
*bool*: 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

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

## Namespace

**make-ns** *keyword*  
*keyword*, make namespace

**untern** *keyword string*  
*symbol*, intern unbound symbol

**intern** *keyword string value*  
*symbol*, intern bound symbol

**ns-find** *keyword string*  
*symbol*, map *string* to *symbol*

**ns-syms** *keyword*  
*list*, namespace's symbols

## Reader/Printer

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

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

## 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 *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

runtime: x.y.z: [-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