

Mu Namespace

mu version 0.0.15

Type keywords and aliases

| | |
|------------------|--|
| <i>supertype</i> | <i>T</i> , form |
| <i>bool</i> | () , :nil is false, otherwise true |
| <i>condition</i> | keyword, see Exceptions |
| <i>type</i> | type-of returns keyword |
| <i>list</i> | cons or () , :nil |
| <i>frame</i> | see Frames |
| <i>ns</i> | see Namespaces |
| | |
| :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 |
| :stream | stream, file or string type |
| :struct | struct |
| :symbol | sym, symbol, keyword |
| :vector | simple vector, string (:char) :t :byte :fixnum :float |

Heap

| | |
|----------------|---|
| hp-info | vector, heap allocations #(:t type total alloc in-use) |
|----------------|---|

frames

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

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

Reader/Printer

| | |
|----------------------------|-------------------------|
| read stream bool T | T, read stream object |
| write T bool stream | T, write escaped object |

Structs

| | |
|-----------------------------|-----------------------------------|
| make-st keyword list | struct, of type keyword from list |
| st-type struct | keyword, struct type keyword |
| st-vec struct | vector, of struct members |

Symbols

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

Special Forms

| | |
|-----------------------------|---------------------|
| :lambda list . list' | function, anonymous |
| :quote form | list, quoted form |
| :if form fn' fn'' | T, conditional |

Core

| | |
|-----------------------|--|
| | async: #<async id=xxxxx> |
| eval form | T, evaluate form |
| eq form form' | bool, are form and form' identical? |
| type-of form | keyword |
| apply fn list | T, apply function to list |
| *async fn list | :async, create future context |
| *await : async | T, return value of async future |
| *abort : async | T, abort future |
| compile form | T, library form compiler |
| view form | vector, vector of object |
| repr bool T | T, tag representation conversion: if bool is (), return 8 byte vector of argument tag bits, otherwise convert argument byte vector to tag |
| fix fn form | T, fixpoint of function on form |
| *gc | bool, garbage collection |

System

| | |
|------------------|------------------------------|
| real-tm T | fixnum, system clock secs |
| run-us T | fixnum, process time μ s |

Fixnums

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

Floats

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

Conses and Lists

| | |
|------------------------|------------------------|
| car list | list, head of list |
| cdr list | list, tail of list |
| cons form form' | cons, (form . form') |
| length list | fixnum, length of list |
| nth fix list | T, nth car of list |
| nthcdr fix list | T, nth cdr of list |

Vectors

| | |
|-----------------------------|------------------------------|
| make-sv keyword list | vector, typed vector of list |
| sv-len vector | fixnum, length of vector |
| sv-ref vector fix | T, nth element |
| sv-type vector | keyword, type of vector |

Exceptions

| | |
|-----------------------|---|
| 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 :unbound
```

Streams

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

Namespaces

| | |
|-------------------------------|--|
| | <i>ns</i> : #s(:ns <i>name</i>) |
| make-ns string ns | <i>ns</i> , make <i>namespace</i> |
| map-ns string ns | <i>ns</i> , map <i>string</i> to <i>namespace</i> |
| untern ns string | <i>symbol</i> , intern unbound <i>symbol</i> |
| intern ns string value | <i>symbol</i> , intern bound <i>symbol</i> |
| ns-find ns string | <i>symbol</i> , map <i>string</i> to <i>symbol</i> |
| ns-name ns | <i>string</i> , namespace's name |
| ns-syms ns | <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
```