

Mu Namespace

mu version 0.0.4

Type keywords and aliases

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

Heap

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

frames

fr-get fn	struct, copy frame binding
fr-pop fn	function, pop frame binding
fr-push struct	struct, push frame binding
::fr-ref fix fix	T, ref frame variable

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
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 form form'	T, conditional

Core

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
compile form	T, library form compiler

view form	vector, vector of object
fix fn form	T, fixpoint of function on form

::frames	cons, active frame list
*::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 less than 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 less than 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, from T and T'
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 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

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

Namespaces

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