

Mu Namespace

mu version 0.0.14

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 Exceptions)
<i>type</i>	<i>type-of</i> returns <i>keyword</i>
<i>list</i>	cons or () , :nil
<i>frame</i>	see Frames
<i>ns</i>	see Namespaces
: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

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

frames

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

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

Reader/Printer

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

Structs

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

Symbols

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

Special Forms

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

Core

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

System

real-tm <i>T</i>	<i>fixnum</i> , system clock secs
run-us <i>T</i>	<i>fixnum</i> , process time μ s

Fixnums

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

Floats

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

Conses and Lists

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

Vectors

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

Exceptions

with-ex <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

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(: <i>ns 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
```