Mu Namespace

mu version *0.0.11*

Type keywords and aliases

supertype	*T*, *form*
bool	`(),:nil` is false, otherwise true
condition	*keyword* (see ***Exceptions***)
type	*type-of* returns *keyword*
list	*cons* or `(),:nil`
frame	see ***Frames***
ns	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

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*
arity *fn* *fixnum*, function arity
compile *form* *T*, library form compiler
view *form* *vector*, vector of object
repr *bool T* *T*, tag representation conversion: if *bool* is (), return 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 *µs*

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*, *n*th 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 *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 form*
byte, read *byte* from *stream,*
bool: error on eof, *form:* eof value

rd-char *stream bool form*
char, read *char* from *stream,*
bool: error on `eof`, *form:* `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*

Namespaces

ns: `#s(:ns name import)`

make-ns *string ns*
ns, make *namespace*

map-ns *string* *ns,* map *string* to namespace

untern *ns string*
symbol, intern unbound symbol

intern *ns string value*
symbol, intern bound symbol

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

ns-imp *ns* *ns,* namespace's import
ns-name *ns* *string,* namespace's name
ns-syms *ns* *list,* 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 *list*
`()`	empty *list*, prints as `:nil`
`"..."`	*string, char vector*
`\`	single escape in strings
`#x`	hexadecimal *fixnum*
`#\c`	*char*
`#(:type ...)`	*vector*
`#s(:type ...)`	*struct*
`#:symbol`	uninterned *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
```