*Mu* *Reference*

**mu version *0.0.25***

***Type keywords and aliases***

*supertype T*

*bool* (),:nil are false, otherwise true

*condition* *keyword,* see ***Exception***

*list cons* or (),:nil

*frame* *cons*, see ***Frame***

:null (),:nil

:asyncid *async* async future id

:char *char*

:cons *cons*

:fixnum *fixnum, fix* 56 bit signed integer

:float *float*, *fl* 32 bit IEEE float

:func *function*, *fn* function

:keyword *keyword, key symbol*

:map *map key/value* hash

:stream *stream* file or string type

:struct*struct*typed vector

:symbol *symbol, sym* LISP-1symbol

:vector *vector*, *string, str*

:char :t :byte :fixnum :float

***Heap***

**hp-info** *vector* heap static information

#(:t type *pages* *pagesize*)

**hp-stat** *vector* heap allocations

#(:t :*type* *size* *total* *free ...*)

**hp-size** *T**fixnum* heap occupancy in bytes

***Frame***

*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

***Struct***

**make-st** *key list*

*struct of* type *key* from list

**st-type** *struct* *key* structtype keyword

**st-vec** *struct* *vector* of struct members

***Symbol***

**boundp** s*ym bool* is*symbol* bound?

**keyword** *str key* *keyword* from *string*

**make-sy** *str symbol* uninterned *symbol*

**sy-ns** *sym* *key symbol* namespace

**sy-name** *sym string symbol* name binding

**sy-val** *sym T symbol* value binding

***Special*** ***Forms***

**:async** *fn* *. list* *async*create *future* context

**:lambda** *list* . *list’*

*function* anonymous function

***:*quote** *form* *list* quoted form

***:*if** *form* *T T’**T* conditional

***Core***

**apply** *fn* *list* *T*apply *function* to *list*

**eval** *form* *T* evaluate *form*

**eq** *T*  *T’ bool**are T* and *T’* identical?

**type-of** *T* *keyword*

**\*await**:async *T*return value of async future

**\*abort**:async *T*abort future

**compile** *form* *T**mu* form compiler

**view** *form* *vector* vector of object

**repr** *bool T* *T* tag representation

conversion: if *bool* is (),

return 8 byte *fixnum* vector

of argument tag bits,

otherwise convert

argument byte vector to tag

**fix** *fn form* *T* fixpoint of *function* on *form*

**gc** *bool**bool* garbage collection, verbose

**exit** *fix* exit process with return code

***Fixnum***

**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

**ash** *fix* *fix’* *fixnum*arithmetic shift

**logand** *fix* *fix’* *fixnum*bitwise and

**logor** *fix* *fix*’ *fixnum*bitwise or

***Float***

**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***

**%append** *list T* *list* append

**car** *list* *list* head of *list*

**cdr** *list* *T* tail of *list*

**cons** *T* *T’* *cons* (*form* . *form*’)

**length** *list* *fixnum* length of *list*

**nth** *fix* *list* *T n*th *car* of *list*

**nthcdr** *fix* *list* *T n*th *cdr* of *list*

***Vector***

**make-sv** *keyword* *list*

*vector* typed vector from list

**sv-len** *vector* *fixnum* length of *vector*

**sv-ref** *vector* *fix* *T* *n*th element

**sv-type** *vector* *key* type of *vector*

***Map***

**make-mp** *map* make a new map

**mp-add** *map T T’*

*map* add pair to map

**mp-get** *map T T* reference map

**mp-has** *map* *T bool* is key resident?

**mp-size** *map* *fixnum* sizeof map

**mp-list** *map* *list* map contents

***Exception***

**with-ex** *fn fn’* *T*catchexception

*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** s*tream* *bool* is *stream* at end of file?

**flush** s*tream* *bool* flush output steam

**get-str** *stream string* from *string* *stream*

**rd-byte** *stream bool T*

*byte* read *byte* from *stream,*

error on eof, *T:* eof value

**rd-char** *stream bool T*

*char* read *char* from *stream,*

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 *μs*

***namespaces***

**make-ns** *keyword*

*key* 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

*mu library API*

*[dependencies]*

*mu = { git = “*[*https://github.com/Software-Knife-and-Tool/thorn.git*](https://github.com/Software-Knife-and-Tool/dyad.git)*”, branch=main }*

use mu::{Condition, Config, Exception,

Mu, Result, System, Tag}

*config string format: “npages:N,gcmode:GCMODE”*

*GCMODE – { none, auto, demand }*

const Mu::VERSION: &str

Mu::new(config: &Config)-> Mu

Mu::config(config: String) -> Option<Config>

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: &Config)-> System

System::config(config: String) -> Option<Config>

System::mu(&self)-> &Mu

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 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*

mu-local: x.y.z: [-h?pvcelq] [file…]

?: usage message

h: usage message

c: [name:value,…]

e: eval [form] and print result

l: load [path]

p: pipe mode (no repl)

q: eval [form] quietly

v: print version and exit