***Mu*** ***Runtime*** ***Reference e***

**version *0.2.9***

***type keywords and aliases s***

*supertype T*

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

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

*list* :consor (),:nil

:null (),:nil

:char *char*

:cons *cons*

:fixnum *fixnum, fix* 56 bit signed int

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

:func *function*, *fn* function

:keyword *keyword, key* symbol

:ns *namespace, ns* namespace

:stream *stream* file or string type

:struct*struct*typed vector

:symbol *symbol, sym* LISP-1symbol

:vector *vector*, *string, str*

:bit :char :t

:byte :fixnum :float

***Features***  *I*

**[dependencies]**

**default = [ “env”, “mu”, “std”, “prof”, “nix”, "sysinfo" ]**

**mu/core** **core** *list* corestate

**delay** *fixnum* microseonds

**process-mem-virt** *fixnum* vmem

**process-mem-res** *fixnum* reserve

**process-time** *fixnum* microseconds

**time-units-per-sec** *fixnum*

**mu/env heap-room** *vector* allocations

#(:t :*type* *size* *total* *free …*)

**heap-info** *list* heap info

(*type* *page-size* *npages*)

**heap-size** *keyword**fixnum* type size

**heap-free** *fixnum* bytes free

**env** *list* envstate

**mu/nix uname**

**mu/std command**, **exit**

**mu/sysinfo sysinfo** (disabled on macOS)

**mu*/*prof prof-control** toggleenable

***configuration API z***

*config string format:*

*“npages:N, gc-mode:GCMODE, page-size:N, heap-type:HEAPTYPE”*

*N: unsigned integer*

*GCMODE: none | auto | demand*

*HEAPTYPE: semispace | bump // needs semispace feature*

***Special*** ***Forms s***

**:lambda** *list* . l*ist’* *function* anonymous *fn*

**:alambda** list . list’ *function* anonymous *fn*

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

**:if***T T’ T’’* *T* conditional

***Core s***

**\*null/\*** *ns* null namespace

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

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

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

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

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

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

**repr**  *T* *vector* tag representation

**unrepr**  *vector*  *T* tag representation

*vector* is an 8 element :byte vector

of little-endian argument tag bits.

**fix** *fn T*  *T* fixpoint of *fn*

**gc** *bool* garbage collection

***Frames e***

**%frame-stack** *list*active *frame*s

**%frame-pop** *fn fn* pop *function’s* top

frame binding

*frame* binding:(*fn* . #(:t *…*))

**%frame-push** *frame cons* push frame

**%frame-ref** *fn fix T function*, offset

***Symbols l***

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

**make-symbol** *string sym* uninterned *symbol*

**symbol-namespace** *sym*

*ns namespace*

**symbol-name** *symbol string* name binding

**symbol-value** *symbol T* value binding

***Fixnums m***

**add** *fix* *fix’* *fixnum*sum

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

**div** *fix* *fix*’ *fixnum*quotient

**less-than** *fix* *fix’* *bool fix* < *fix’?*

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

**lognot** *fix* *fixnum*bitwise complement

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

**mul** *fix* *fix’* *fixnum*product

**sub** *fix* *fix’* *fixnum*difference

***Floats t***

**fadd** *fl* *fl’* *float* sum

**fdiv** *fl* *fl’* *float* quotient

**fless-than** *fl* *fl’* *bool fl* < *fl’?*

**fmul** *fl* *fl’* *float* product

**fsub** *fl fl’* *float* difference

***Conses/Lists s***

**append** *list* *list* append lists

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

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

**cons** *T* *T’* *cons* (*T* . *T*’)

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

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

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

***Vectors s***

**make-vector** *key* *list* *vector* specialized vector

from list

**vector-length** *vector* *fixnum* length of *vector*

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

**svref** *vector* *fix* *T* *n*th element

***Streams n***

**\*s****tandard*-*input\*** *stream* std input *stream*

**\*standard*-*output\*** *stream* std out *stream*

**\*error*-*output\*** *stream* std error *stream*

**open** *type* *dir string bool*

*stream* open *stream,*

raise error if *bool*

*type* :file :string

*dir* :input :output :bidir

**close** *stream* *bool* close *stream*

**openp** *stream* *bool* is *stream* open?

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

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

**read-byte** *stream bool T*

*byte* read *byte* from

*stream,* error on

eof, *T:* eof-value

**read-char** *stream bool T*

*char* read *char* from *stream,* error on

eof, *T:* eof-value

**unread-char** *char* *stream*

*char* push *char* onto

*stream*

**write-byte** *byte* *stream* *byte* write *byte*

**write-char** *char stream*

*char* write *char*

**read** *stream* *bool* *T* *T* read *stream*

**write** T *bool* *stream* *T* write with escape

***Namespaces .***

**make-namespace** *str* *ns* make *namespace*

**namespace-name** *ns* *string namespace* name

**intern** *ns str value symbol* intern symbol

**find-namespace** *str* *ns* map *string* to *namespace*

**find** *ns* *string* *symbol* map *string* to *symbol*

**namespace-symbols** *ns list symbol* list

***Exceptions n***

**with-exception** *fn fn’* *T* catchexception

*fn*  - (:lambda (*obj**cond src*) ***.*** *body*)

*fn’* - (:lambda () **.** *body*)

**raise** *T* *keyword* raise exception

on *T* with

condition:

:arity :div0 :eof :error :except

:future :ns :open :over :quasi

:range :read :exit :signal :stream

:syntax :syscall :type :unbound :under

:write :storage

***Structs t***

**make-struct** *key list struct* type *key* from *list*

**struct-type** *struct* *key struct* type *key*

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

***mu******library******API I***

*[dependencies]*

*mu = {*

*git = “*[*https://github.com/Software-Knife-and-Tool/mu.git*](https://github.com/Software-Knife-and-Tool/mu.git)*”,*

*branch = “main”*

*}*

use mu::{ Condition, Config, Env, Exception,

Core, Mu, Result, Tag };

impl Mu {

const VERSION: &str

fn apply(\_: &Env, \_: Tag, \_: Tag) → Result<Tag>

fn compile(\_: &Env, \_: Tag) → Result<Tag>

fn config(\_: Option<String>) → Option<Config>

fn core() → &Core

fn eq(\_: Tag, \_: Tag) → bool;

fn err\_out() → Tag

fn eval\_str(\_: &Env, \_: &str) → Result<Tag>

fn eval(\_: &Env, \_: Tag) → Result<Tag>

fn exception\_string(\_: &Env, \_: Exception) → String

fn load(\_: &Env, \_: &str) → Result<bool>

fn make\_env(\_: &Config) → Env

fn read\_str(\_: &Env, \_: &str) → Result<Tag>

fn read(\_: &Env, \_: Tag, \_: bool, \_: Tag) → Result<Tag>

fn std\_in() → Tag

fn std\_out() → Tag

fn write\_str(\_: &Env, \_: &str, \_: Tag) → Result<()>

fn write\_to\_string(\_: &Env, \_: Tag, \_: bool) → String

fn write(\_: &Env, \_: Tag, \_: bool, \_: Tag) → Result<()>

***Reader Syntax x***

; comment to end of line

#|...|# block comment

‘*form* quoted form

`*form* backquoted form

`(*...)* backquoted list (proper lists)

,*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

#\* bit vector

#x hexadecimal *fixnum*

#. read-time eval

#\ *char*

#(:type …) *vector*

#s(:type …) *struct*

#: uninterned *symbol*

“`,; terminating macro char

# non-terminating macro char

!$%&\*+-. symbol constituent

<>=?@[]|

:^\_{}~/

A..Za..z

0..9

character designators

0x09 #\tab

0x0a #\linefeed

0x0c #\page

0x0d #\return

0x20 #\space

***mu-sys .***

**mu-sys: 0.0.2: [celq] [file…]**

c: name:value,… runtime configuration

e: form eval and print result

l: path load from path

q: form eval quietly