*Mu* *Namespace*

mu version *0.0.14*

***Type keywords***

***T***  type superclass

**:t** **:nil** *bool*

**:char** *char*

**:cons** *cons, list (cons* or **:nil)**

**:fixnum** *fix,* 61 bit signed integer

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

**:func** *fn,* function

**:ns** *ns,* *symbol* bindings

**:stream** *stream,*file, string

**:struct** *struct,* general *vector*

**:symbol** LISP-1 binding: *symbol, keyword*

**:vector** *vector*, *string* (**:char**)

**:t** **:byte:fixnum** **:float**

***Heap***

**hp-info** *vector,* of type allocations

**:type :total :alloc :in-use**

***Frame***

**fr-get** *fn* *struct,*copyframe 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***

**struct** *type* *list struct,* from list

**st-type** *vector* *keyword*, structtype

**st-vec** *vector* *fix* *vector,* of members

***Symbols***

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

**keyp** s*ymbol* *bool,* *keyword* predicate

**keyword** *string* *symbol*, *keyword* from *string*

**symbol** *string symbol,* uninterned *symbol*

**sy-ns** *symbol* *ns, symbol namespace*

**sy-name** *symbol string, symbol* name binding

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

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

**:lambda** *list* . *body* *function,* anonymous

***:*quote** *T* *list*, quote form

***:*if** *T T’ T’’*  *T,* conditional

***Core***

**coerce** *T keyword T****,*** *coerce* totype keyword

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

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

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

**apply** *fn* *list* *T****,***apply *function* to arg *list*

**compile** *T* *T****,***library form compiler

**tag-of** *T* *fixnum****,***of object tag

**view** *T* *struct,* vector of object

**fix** *fn T* *T****,***fixpoint of *function*

**:if** *T fn fn’* *T****,*****:if** implementation

**:frames**  *cons****,*** active frame list

**\*:gc**  *bool,* garbage collection

***Fixnums***

**fx-mul** *fix* *fix’* *fixnum****,*** product

**fx-add** *fix* *fix’* *fixnum****,*** sum

**fx-sub** *fix* *fix’* *fixnum****,*** difference

**fx-lt** *fix* *fix’* *bool****,*** is *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** *float* *float’* *float,* product

**fl-add** *float* *float’* *float,* sum

**fl-sub** *float* *float’* *float,* difference

**fl-lt** *float* *float’* *bool,* is *float* less than *float’?*

**fl-div** *float* *float’* *float,* quotient

***Lists***

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

**cdr** *list* *list,* tail of *list*

**cons** *T* *T’* *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***

**vector** *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*

***Namespaces***

**make-ns** *string* *ns*

*ns,* make *namespace*

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

**intern** *ns scope string value*

*symbol,* intern bound symbol

*scope* **:intern :extern**

**ns-find** *ns* *scope* *string*

*symbol,* map *string* to *symbol*

*scope* **:intern :extern**

**ns-imp** *ns* *ns, namespace’s* import

**ns-name** *ns* *string, namespace’s* name

**ns-int** *ns* *list, namespace’s* interns

**ns-ext** *ns* *list, namespace’s* externs

***Streams***

**std*-*in** *symbol,* standard input *stream*

**std*-*out** *symbol,* standard output *stream*

**err*-*out** *symbol,* standard error *stream*

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

*stream,* open *stream*

*type* **:file** | **:string**

*dir* **: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,*

*g*et *string* from *string* *stream*

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

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

**wr-byte** *byte* *stream*

*byte,* write *byte* to *stream*

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

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

**wr*-*char** *char* *stream*

*char,* write *char* to *stream*

**un-char** *char* *stream*

*char,* push *char* onto *stream*

***Exceptions***

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

*fn* **(:lambda (***T* *keyword***) *.*** *body****)***

*fn’* **(:lambda ()** . *body****)***

**raise** ***T*** *keyword* raise exception with condition

***Condition Keywords***

:**arity :eof :open :read**

**:write :error :syntax :type**

**:unbound :div0 :range :stream**

**:except**

***Rust*** ***API***

**use crate::mu::core::mu::{**

**Exception,**

**Extern,**

**Mu,**

**MuCondition,**

**Tag**

**},**

**<Mu as Extern>::new(config: String) -> Mu**

**config:** comma-separated list of

**name:value** pairs

**heap:*npages***

**gc:*on|off***

**&’static str <Mu as Extern>::VERSION**

**pub trait Export for Mu {**

**fn nil() -> Tag**

**fn eq(tag: Tag, tag1: Tag) -> bool**

**fn apply(&self, func: Tag, args) →**

**Exception::Result<Tag>**

**fn compile(&self, expr: Tag) ->**

**Exception::Result<Tag>**

**fn** **eof(&self, stream: Tag) ->**

**Exception::Result<Tag>**

**fn eval(&self,** **expr: Tag) ->**

**Exception::Result<Tag>**

**fn read\_stream(&self,** **stream: Tag,**

**eof: Tag,**

**eof\_value: Tag) ->**

**Exception::Result<Tag>**

**fn read\_string(&self, expr: String) ->**

**Exception::Result<Tag>**

**fn write(&self, expr: Tag,**

**escape: bool, stream: Tag) ->**

**Exception::Result<()>**

**fn write\_string(&self, string: String,**

**stream: Tag) ->**

**Exception::Result<()>**

**}**

***Reader  Syntax***

**;** comment to end of line

**#|...|#** block comment

**(…)**  constant list

**()** empty list, prints as **:nil**

**‘** quoted form

**“…”**  string/char vector

**#x**  hexadecimal *fixnum*

**#\**  character

**#(:*vector-type*** **…)** *vector*

**#s(:struct*-type*** **…)** *struct*

**#:symbol** uninterned *symbol*

**\** single escape in strings

**“`,;** terminating macro char

**#** non-terminating macro char

**!$%&\*+-.** symbol constituent:

**<>=?@[]|**

**:^\_{}~/**

**A..Za..z**

**0..9**

**backspace**

**rubout**

**0x09 tab** whitespace:

**0x0a linefeed**

**0x0c page**

**0x0d return**

**0x20 space**

*Runtime*

**runtime: 0.0.10: [-h?psvcedlq] [file...]**

**?: usage message**

**h: usage message**

**c: [name:value,…]**

**d: enable debugging**

**e: eval [form] and print result**

**l: load [path]**

**p: pipe mode**

**q: eval [form] quietly**

**s: script mode**

**v: print version and exit**