

Mu Runtime Reference

version 0.2.8

type keywords and aliases

<i>supertype</i>	<i>T</i>	
<i>bool</i>	<code>()</code> , <code>:nil</code> are false, otherwise true	
<i>condition</i>	keyword, see Exception	
<i>list</i>	<code>:cons</code> or <code>()</code> , <code>:nil</code>	
<code>:null</code>	<code>()</code> , <code>:nil</code>	
<code>:char</code>	<i>char</i>	
<code>:cons</code>	<i>cons</i>	
<code>:fixnum</code>	<i>fixnum</i> , <i>fix</i>	56 bit signed integer
<code>:float</code>	<i>float</i> , <i>fl</i>	32 bit IEEE float
<code>:func</code>	<i>function</i> , <i>fn</i>	function
<code>:keyword</code>	<i>keyword</i> , <i>key</i>	symbol
<code>:ns</code>	<i>namespace</i> , <i>ns</i>	namespace
<code>:stream</code>	<i>stream</i>	file or string type
<code>:struct</code>	<i>struct</i>	typed vector
<code>:symbol</code>	<i>symbol</i> , <i>sym</i>	LISP-1 symbol
<code>:vector</code>	<i>vector</i> , <i>string</i> , <i>str</i>	
	<code>:bit</code> <code>:char</code> <code>:t</code>	
	<code>:byte</code> <code>:fixnum</code> <code>:float</code>	

Features

[dependencies] default = ["env", "procinfo", "std", "timer", "nix", "sysinfo"]			
env	heap-room	<i>vector</i>	allocations
	<code>#(:t :type size total free ...)</code>		
	heap-info	<i>list</i>	heap info
	<code>(type page-size npages)</code>		
	heap-size	<i>keyword</i> <i>fixnum</i>	type size
	heap-free	<i>fixnum</i>	bytes free
	env	<i>list</i>	env state
	core	<i>list</i>	core state
nix	uname		
std	command , exit		
sysinfo	sysinfo (disabled on macOS)		
procinfo	process-mem-virt	<i>fixnum</i>	virtual memory
	process-mem-res	<i>fixnum</i>	reserve
	process-time	<i>fixnum</i>	microseconds
	time-units-per-sec	<i>fixnum</i>	
prof	prof-control		enable
timer	delay	<i>fixnum</i>	microseconds

configuration API

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

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

Reader/Printer

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

Core

null/	<i>ns</i>	null namespace
apply <i>fn list</i>	<i>T</i>	apply <i>fn</i> to <i>list</i>
eval <i>form</i>	<i>T</i>	evaluate <i>form</i>
eq <i>T T'</i>	<i>bool</i>	<i>T</i> and <i>T'</i> identical?
type-of <i>T</i>	<i>key</i>	type keyword
compile <i>form</i>	<i>T</i>	<i>mu</i> form compiler
view <i>form</i>	<i>vector</i>	vector of object

%if *fn fn' fn* *bool* **:if** implementation

repr <i>T</i>	<i>vector</i>	tag representation
unrepr <i>vector</i>	<i>T</i>	tag representation

vector is an 8 element `:byte` vector of little-endian argument tag bits.

fix <i>fn T</i>	<i>T</i>	fixpoint of <i>fn</i>
gc	<i>bool</i>	garbage collection

Frames

%frame-stack	<i>list</i>	active frames
%frame-pop <i>fn</i>	<i>fn</i>	pop <i>function's</i> top frame binding
frame binding: (<i>fn</i> . #(:t ...))		
%frame-push <i>frame</i>	<i>cons</i>	push frame
%frame-ref <i>fn fix</i>	<i>T</i>	<i>function</i> , offset

Symbols

boundp <i>symbol</i>	<i>bool</i>	is <i>symbol</i> bound?
make-symbol <i>string</i>	<i>symbol</i>	uninterned <i>symbol</i>
symbol-namespace <i>symbol</i>	<i>ns</i>	namespace
symbol-name <i>symbol</i>	<i>string</i>	name binding
symbol-value <i>symbol</i>	<i>T</i>	value binding

Fixnums

mul <i>fix fix'</i>	<i>fixnum</i>	product
add <i>fix fix'</i>	<i>fixnum</i>	sum
sub <i>fix fix'</i>	<i>fixnum</i>	difference
less-than <i>fix fix'</i>	<i>bool</i>	<i>fix</i> < <i>fix'</i> ?
div <i>fix fix'</i>	<i>fixnum</i>	quotient
ash <i>fix fix'</i>	<i>fixnum</i>	arithmetic shift
logand <i>fix fix'</i>	<i>fixnum</i>	bitwise and
logor <i>fix fix'</i>	<i>fixnum</i>	bitwise or
lognot <i>fix</i>	<i>fixnum</i>	bitwise complement

Floats

fmul <i>fl fl'</i>	<i>float</i>	product
fadd <i>fl fl'</i>	<i>float</i>	sum
fsub <i>fl fl'</i>	<i>float</i>	difference
fless-than <i>fl fl'</i>	<i>bool</i>	<i>fl</i> < <i>fl'</i> ?
fdiv <i>fl fl'</i>	<i>float</i>	quotient

Conses/Lists

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

Vectors

make-vector <i>key list</i>	<i>vector</i>	specialized vector from <i>list</i>
vector-length <i>vector</i>	<i>fixnum</i>	length of <i>vector</i>
vector-type <i>vector</i>	<i>key</i>	type of <i>vector</i>
svref <i>vector fix</i>	<i>T</i>	<i>nth</i> element

Streams n

standard-input *stream* std input *stream*
standard-output *stream* std output *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 *stream bool* flush output *stream*
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* to *stream*
write-char *char stream char* write *byte* to *stream*

Namespaces .

make-namespace *str ns* make *namespace*
namespace-map *list* list of mapped
namespaces
namespace-name *ns string* *namespace* name
intern *ns str value symbol* intern bound symbol
find-namespace *str ns* map *string* to
namespace
find *ns string symbol* map *string* to
symbol
namespace-symbols *ns list* *namespace* symbols

Exceptions n

with-exception *fn fn' T* catch exception

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 f

make-struct *key list struct* of type *key* from *list*
struct-type *struct key* *struct* type *keyword*
struct-vec *struct vector* of *struct* members

mu library API I

[dependencies]
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 config(_: Option<String>) -> Option<Config>
  fn make_env(_: &Config) -> Env
  fn apply(_: &Env, _: Tag, _: Tag) -> Result<Tag>
  fn compile(_: &Env, _: Tag) -> Result<Tag>
  fn eq(_: Tag, _: Tag) -> bool;
  fn exception_string(_: &Env, _: Exception) -> String
  fn eval(_: &Env, _: Tag) -> Result<Tag>
  fn eval_str(_: &Env, _: &str) -> Result<Tag>
  fn load(_: &Env, _: &str) -> Result<bool>
  fn read(_: &Env, _: Tag, _: bool, _: Tag) -> Result<Tag>
  fn read_str(_: &Env, _: &str) -> Result<Tag>
  fn core() -> &Core
  fn err_out() -> Tag
  fn std_in() -> Tag
  fn std_out() -> Tag
  fn write(_: &Env, _: Tag, _: bool, _: Tag) -> Result<()>
  fn write_str(_: &Env, _: &str, _: Tag) -> Result<()>
  fn write_to_string(_: &Env, _: Tag, _: bool) -> String
}
```

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*
#: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

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