

Mu Reference

version 0.2.9

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", "mu", "std", "nix", "sysinfo"]

%mu	core	<i>list</i>	core state
	delay	<i>fixnum</i>	microseconds
	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>	
%env	heap-room	<i>vector</i>	allocations
	#(:t :type size total free ...)		
	heap-info	<i>list</i>	heap info
	(type page-size npages)		
	heap-size keyword	<i>fixnum</i>	type size
	heap-free	<i>fixnum</i>	bytes free
%nix	env	<i>list</i>	env state
	uname		
	command , exit		
	sysinfo (disabled on macOS)		
	prof-control		toggle enable
	%prof		

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>	function	anonymous function
:alambda <i>list</i> . <i>list'</i>	function	anonymous function
:quote <i>form</i>	<i>list</i>	quoted form
:if <i>T 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>
compile <i>form</i>	<i>T</i>	mu form compiler
eq <i>T T'</i>	<i>bool</i>	<i>T</i> and <i>T'</i> identical?
eval <i>form</i>	<i>T</i>	evaluate <i>form</i>
type-of <i>T</i>	<i>key</i>	type keyword
view <i>form</i>	<i>vector</i>	vector of object

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-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 symbol

symbol-name <i>symbol</i>	<i>string</i>	name binding
symbol-value <i>symbol</i>	<i>T</i>	value binding

Fixnums

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

Floats

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

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	<i>stream</i>	std input <i>stream</i>
standard-output	<i>stream</i>	std output <i>stream</i>
error-output	<i>stream</i>	std error <i>stream</i>
open <i>type dir string bool</i>		
	<i>stream</i>	open <i>stream</i> , raise error if <i>bool</i>
	<i>type</i> :file :string	
	<i>dir</i> :input :output :bidir	
close <i>stream</i>	<i>bool</i>	close <i>stream</i>
openp <i>stream</i>	<i>bool</i>	is <i>stream</i> open?
flush <i>stream</i>	<i>bool</i>	flush output <i>stream</i>
get-string <i>stream</i>	<i>string</i>	from <i>string stream</i>
read-byte <i>stream bool T</i>		
	<i>byte</i>	read <i>byte</i> from <i>stream</i> , error on eof, <i>T</i> : eof-value
read-char <i>stream bool T</i>		
	<i>char</i>	read <i>char</i> from <i>stream</i> , error on eof, <i>T</i> : eof-value
unread-char <i>char stream</i>		
	<i>char</i>	push <i>char</i> onto <i>stream</i>
write-byte <i>byte stream byte</i>		write <i>byte</i> to <i>stream</i>
write-char <i>char stream</i>		
	<i>char</i>	write <i>byte</i> to <i>stream</i>

Namespaces

make-namespace <i>str ns</i>		make <i>namespace</i>
namespace-map <i>list</i>		list of mapped <i>namespaces</i>
namespace-name <i>ns string</i>		<i>namespace</i> name
intern <i>ns str value</i>	<i>symbol</i>	intern bound symbol
find-namespace <i>str ns</i>		map <i>string</i> to <i>namespace</i>
find <i>ns string</i>	<i>symbol</i>	map <i>string</i> to <i>symbol</i>
namespace-symbols <i>ns list</i>		<i>namespace symbols</i>

Exceptions

with-exception <i>fn fn' T</i>	catch exception			
<i>fn</i> - (:lambda (<i>obj cond src</i>) . <i>body</i>)				
<i>fn'</i> - (:lambda () . <i>body</i>)				
raise <i>T keyword</i>	raise exception on <i>T</i> 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

make-struct <i>key list</i>	<i>struct</i>	type <i>key</i> from <i>list</i>
struct-type <i>struct</i>	<i>key</i>	<i>struct</i> type <i>key</i>
struct-vec <i>struct</i>	<i>vector</i>	of <i>struct</i> members

mu library API

```
[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 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 <i>list</i>
()	empty <i>list</i> , prints as :nil
(... . .)	dotted <i>list</i>
"..."	<i>string</i> , <i>char vector</i>
	single escape in strings
#*...	bit vector
#X...	hexadecimal <i>fixnum</i>
#.	read-time eval
#\.	<i>char</i>
#(:type ...)	<i>vector</i>
#s(:type ...)	<i>struct</i>
#:symbol	uninterned <i>symbol</i>
"` , ;	terminating macro <i>char</i>
#	non-terminating macro <i>char</i>
!\$%&*+-.	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