

Mu Runtime Reference

version 0.2.13

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>	<i>keyword</i> , see exceptions	
<i>list</i>	<code>:cons</code> or <code>()</code> , <code>:nil</code>	
<i>ns</i>	<code>#s(:ns #(:t fixnum symbol))</code>	
<i>ns-designator</i>	<code>ns</code> , <code>:nil</code> , <code>:unqual</code>	
<i>:null</i>	<code>()</code> , <code>:nil</code>	
<i>:char</i>	<i>char</i>	8 bit ASCII
<i>:cons</i>	<i>cons</i> , <i>list</i>	list, cons, dotted pair
<i>:fixnum</i>	<i>fixnum</i> , <i>fix</i>	56 bit signed integer
<i>:float</i>	<i>float</i> , <i>fl</i>	32 bit IEEE float
<i>:func</i>	<i>function</i> , <i>fn</i>	function
<i>:keyword</i>	<i>keyword</i> , <i>key</i>	symbol
<i>:stream</i>	<i>stream</i>	file or string type
<i>:struct</i>	<i>struct</i>	see structs
<i>:symbol</i>	<i>symbol</i> , <i>sym</i>	LISP-1 symbol
<i>:vector</i>	<i>vector</i> , <i>string</i> , <i>str</i>	typed vector
	<code>:bit</code> <code>:char</code> <code>:t</code>	
	<code>:byte</code> <code>:fixnum</code> <code>:float</code>	

core

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>for</i>	<i>vector</i>	vector of object
fix <i>fn T</i>	<i>T</i>	fixpoint of <i>fn</i>
gc	<i>bool</i>	garbage collection
repr <i>T</i>	<i>vector</i>	tag representation
unrepr <i>vector</i>	<i>T</i>	tag representation

special forms

<code>:lambda</code> <i>list . list'</i>	<i>function</i>	anonymous <i>fn</i>
<code>:alambda</code> <i>list . list'</i>	<i>function</i>	anonymous <i>fn</i>
<code>:quote</code> <i>T</i>	<i>list</i>	quoted form
<code>:if</code> <i>T T' T''</i>	<i>T</i>	conditional
vector is an 8 element :byte vector of little-endian argument tag bits.		

frames

frame binding: `(fn . #(:t ...))`

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

symbols

boundp <i>sym</i>	<i>bool</i>	is symbol bound?
make-symbol <i>string</i>	<i>sym</i>	uninterned symbol
symbol-namespace <i>sym</i>	<i>ns-designator</i>	namespace designator
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 list
cdr <i>list</i>	<i>T</i>	tail of list
cons <i>T T'</i>	<i>cons</i>	(<i>T</i> . <i>T'</i>)
length <i>list</i>	<i>fixnum</i>	length of list
nth <i>fix list</i>	<i>T</i>	<i>nth</i> car of list
nthcdr <i>fix list</i>	<i>T</i>	<i>nth</i> cdr of list

vectors

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

namespaces

runtime namespaces: *mu* (static), *keyword*

make-namespace <i>str</i>	<i>ns</i>	make namespace
namespace-name <i>ns</i>	<i>string</i>	namespace name
intern <i>ns str value</i>	<i>symbol</i>	intern symbol in non-static namespace
find-namespace <i>str</i>	<i>ns</i>	map string to namespace
find <i>ns string</i>	<i>symbol</i>	map string to symbol

structs

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

streams

standard-input	<i>stream</i>	std input stream
standard-output	<i>stream</i>	std out stream
error-output	<i>stream</i>	std error stream
open <i>type dir str bool</i>	<i>stream</i>	open stream, raise error if <i>bool</i>
	<i>type</i> <i>dir</i>	<code>:file</code> <code>:string</code> <code>:input</code> <code>:output</code> <code>:bidir</code>

close <i>stream</i>	<i>bool</i>	close stream
openp <i>stream</i>	<i>bool</i>	is stream open?
flush <i>stream</i>	<i>bool</i>	flush stream
get-string <i>stream</i>	<i>string</i>	from string stream
read-byte <i>stream bool T</i>	<i>byte</i>	read byte from stream, error on eof, <i>T</i> : eof-value
read-char <i>stream bool T</i>	<i>char</i>	read char from stream, error on eof, <i>T</i> : eof-value
unread-char <i>char stream char</i>		push char onto stream
write-byte <i>byte stream</i>	<i>byte</i>	write byte
write-char <i>char stream</i>	<i>char</i>	write char
read <i>stream bool T</i>	<i>T</i>	read stream
write <i>T bool stream</i>	<i>T</i>	write with escape

exceptions

with-exception *fn fn'* *T* catch exception

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

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

raise-from *T symbol keyword* raise exception on *T* with *keyword* condition

:arity	:div0	:eof	:error	:except
:future	:ns	:open	:over	:quasi
:range	:read	:exit	:signal	:stream
:syntax	:syscall	:type	:unbound	:under
:write	:storage	:user		

Features

```
[dependencies]
default = [ "core", "env", "system" ]
```

feature/core	core	list	core state
	delay	<i>fixnum</i>	microseconds
	process-mem-virt	<i>fixnum</i>	vmem
	process-mem-res	<i>fixnum</i>	reserve
	process-time	<i>fixnum</i>	microseconds
	time-units-per-sec	<i>fixnum</i>	
	ns-symbols <i>ns</i> <i>nil</i>		
		<i>list</i>	<i>symbol</i> list

feature/env	env	list	env state
	heap-info	()	heap info to stdout
	heap-room	<i>vector</i>	allocations
		<i>#(:t size total free ...)</i>	
	heap-size <i>keyword</i>	<i>fixnum</i>	type size
	cache-room	<i>vector</i>	allocations
		<i>#(:t size total ...)</i>	

feature/system	<i>uname</i>	:t	system info
	shell <i>string</i> <i>list</i>	<i>fixnum</i>	shell command
	exit <i>fixnum</i>		
	sysinfo	:t	not on macOS

feature/instrument

instrument-control <i>key</i>	:on :off :get
<i>key</i> <i>vec</i>	

environment

JSON config format:

```
{
  "pages": N,
  "gc-mode": "none" | "auto",
}
```

Mu library API

```
[dependencies]
mu = {
  git = "https://github.com/Software-Knife-and-Tool/mu.git",
  branch = "main"
}
```

```
use mu::{ Condition, Core, Env, Exception,
          Mu, Result, Tag };
```

```
impl Mu {
  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 lo8_ptad(_: &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 version() -> &str
  fn write_str(_: &Env, _: &str, _: Tag) -> Result<()>
  fn write_to_string(_: &Env, _: Tag, _: bool) -> String
  fn write(_: &Env, _: Tag, _: bool, _: Tag) -> Result<()>
}
```

```
;
#|...|#
```

```
'form
`form
`(...)
,form
,@form
```

```
(...)
()
(... . .)
"..."
|
```

ns:name

name

```
##
#x
#.
#\
#(:type ...)
#s(:type ...)
#::...
```

```
"`,;
#
```

```
!$%&*+- .
<=>?@[ ] |
: ^ _ { } ~ /
A..Za..z
0..9
```

```
0x09 #\tab
0x0a #\linefeed
0x0c #\page
0x0d #\return
0x20 #\space
```

comment to end of line
 block comment

quoted form
 backquoted form
 backquoted list (proper lists)
 eval backquoted form
 eval-splice backquoted form

constant *list*
 empty *list*, prints as *nil*
 dotted *list*
string, *char* vector
 single escape in strings

qualified *symbol*, where *ns* and *name* are *symbol constituents*
 lexical *symbol*

bit vector
 hexadecimal *fixnum*
 read-time eval
char
vector
struct
 uninterned *symbol*

terminating macro char
 non-terminating macro char

symbol constituent

character designators