

Mu Library Reference

mu name space, version 0.1.84

type keywords and aliases

<i>supertype</i>	<i>T</i>	
<i>bool</i>	() , :nil are false, otherwise true	
<i>condition</i>	keyword, see Exception	
<i>list</i>	:cons or () , :nil	
:null	() , :nil	
:char	char	
:cons	cons	
:fixnum	fixnum, fix	56 bit signed integer
: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-1 symbol
:vector	vector, string, str	
	:char :t :byte :fixnum :float	

Heap

heap-info	vector	heap information
	#(:t type pages pagesize)	
heap-stat	vector	heap allocations
	#(:t :type size total free ...)	
heap-size <i>T</i>	fixnum	heap occupancy

Frames

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

Symbols

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

Special Forms

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

Core

apply <i>fn list</i>	<i>T</i>	apply function to list
eval <i>form</i>	<i>T</i>	evaluate form
eq <i>T T'</i>	bool	<i>T</i> and <i>T'</i> identical?
type-of <i>T</i>	key	type keyword
compile <i>form</i>	<i>T</i>	mu form compiler
view <i>form</i>	vector	vector of object
%if <i>T T' T''</i>	key	:if implementation
repr <i>type T</i>	<i>T</i>	tag representation
	type :t :vector	

if type is :vector, return 8 byte
byte vector of argument tag bits,
otherwise convert argument byte
vector to tag.

fix <i>fn T</i>	<i>T</i>	fixpoint of function
gc	bool	garbage collection
+version+	string	version string

Futures

defer <i>fn list</i>	struct	future application
detach <i>fn list</i>	struct	future application
force <i>struct</i>	<i>T</i>	force completion
poll <i>struct</i>	bool	poll completion

Fixnum

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

Float

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

Conses/Lists

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

Vectors

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

Reader/Printer

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

Structs

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

Exception 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 :return :sigint :stream
:syntax :syscall :type :unbound :under
:write
```

Streams n

standard-input *stream* std input *stream*
standard-output *stream* std output *stream*
error-output *stream* std error *stream*

open *type dir string stream* open *stream*

```
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 *char* to *stream*

Namespace n

make-namespace *str ns* make *namespace*
namespace-map *list 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

Features 1

[dependencies]
default = ["cpu-time", "std", "nix", "ffi", "sysinfo"]

cpu-time process-time, time-units-per-sec
nix uname
std command, exit
sysinfo sysinfo (disabled on macOS)
ffi Rust FFI
prof prof-control

mu library API 1

[dependencies]
mu = {
git = "https://github.com/Software-Knife-and-Tool/mu.git",
branch=main
}
use mu::{
Condition, Config, Env, Exception, Result, Tag
};
config string format: "npages:N,gcmode:GCMODE"
GCMODE - { none, auto, demand }
impl Env {
const VERSION: &str
fn signal_exception() // enable ^C :sigint exception
fn config(config: Option<String>) -> Option<Config>
fn new(config: &Config, Option<Vec<u8>>) -> Env
fn apply(&self, func: Tag, args: Tag) -> Result<Tag>
fn compile(&self, form: Tag) -> Result<Tag>
fn eq(&self, func: Tag, args: Tag) -> bool;
fn exception_string(&self, ex: Exception) -> String
fn eval(&self, exp: Tag) -> Result<Tag>
fn eval_str(&self, exp: &str) -> Result<Tag>
fn load(&self, file_path: &str) -> Result<bool>
fn read(&self, st: Tag, eofp: bool, eof: Tag) -> Result<Tag>
fn read_str(&self, str: &str) -> Result<Tag>
fn image(&self) -> Result<Vec<u8>>
fn err_out(&self) -> Tag
fn std_in(&self) -> Tag
fn std_out(&self) -> Tag
fn write(&self, exp: Tag, esc: bool, st: Tag) -> Result<()>
fn write_str(&self, str: &str, st: Tag) -> Result<()>
fn write_to_string(&self, exp: Tag, esc: 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 n

mu-sys: 0.0.2: [celq] [file...]

```
c: [name:value,...]
e: eval [form] and print result
l: load [path]
q: eval [form] quietly
```