Mu Namespace

(for libox version *0.0.1*)

Types

`T`	type superclass
`byte`	*uint8_t*
`boolean`	*:t, :nil*
`fix`	*fixnum* synonym
`fn`	*function* synonym
`list`	*cons*, (), **`:nil`**
`ns`	*namespace*
`string`	*character* vector
`type`	type keyword

`:char`	*character*
`:code`	*code*
`:cons`	*cons*
`:double`	64 bit IEEE *float*
`:except`	*exception*
`:fixnum`	62 bit signed *integer*
`:float`	32 bit IEEE *float*
`:func`	*lambda*, native
`:keyword`	7 byte *keyword*
`:macro`	*macro* forms
`:ns`	*symbol* bindings
`:stream`	file, string, socket, function
`:struct`	*defstruct*
`:symbol`	Lisp-1 binding
`:string`	`:char` vector
`:vector`	*T, byte*, *`:char`*, *`:fixnum`*, *`:float`*

(**type-of** *T*)	type *keyword symbol*
(**eq** *T T'*)	are *T* and *T'* identical?
(**null** *T*)	is *T* () or **`:nil`**?

Characters

(**charp** *T*)	*character* predicate
(**char** *T)*	coerce *T* to *character*

Symbols

(**symbolp** *T)*	*symbol* predicate
(**boundp** *symbol*)	is *symbol* bound?
(**keywordp** *symbol)*	*keyword* predicate
(**keyword** *string)*	make *keyword* of *string*
(**symbol-name** *symbol)*	*symbol* name binding
(**symbol-value** *symbol)*	*symbol* value binding
(**symbol-ns** *symbol)*	*symbol ns* binding
(**make-symbol** *string)*	make uninterned *symbol*

Conses/Lists

(**consp** *T*)	*cons* predicate
(**car** *list*)	head of *list*
(**cdr** *list*)	tail of *list*
(**cons** *T T'*)	make a *cons* from *T* and *T'*
(**list-length** *list)*	length of *list*
(**.mapc** *fn list)*	map *function* over *list cars*
(**.mapcar** *fn list)*	make *list* from *list cars*
(**.mapl** *fn list)*	map *function* over *list cdrs*
(**.maplist** *fn list)*	make *list* from *list cdrs*
(**nth** *fix list)*	nth *car* of *list*
(**nthcdr** *fix list)*	nth *cdr* of *list*

Exceptions

*(***exceptionp** *T)*	*exception* predicate
(**exception** *keyword string T)*	make *exception*
*(***raise** *string T*)	raise type *exception*
(**raise-exception** *exception)*	throw *exception*
(**with-exception** *function function*)	catch *exception*

Printer

(**.print** *T stream boolean)*	print with escapes to *stream*
(**print-unreadable** *T stream)*	print unreadable to *stream*
(**terpri** *stream)*	print newline to *stream*

Fixnums

*(***fixnump** *T*)	*fixnum* predicate
*(***fixnum** *T)*	coerce *T* to *fixnum*
*(***fixnum*** *fix fix'*)	product of *fix* and *fix'*
*(***fixnum+** *fix fix'*)	sum of *fix* and *fix'*
*(***fixnum-** *fix fix'*)	difference of *fix* and *fix'*
*(***fixnum<** *fix fix'*)	is *fix* less than *fix'?*
*(***fixnum/** *fix fix'*)	*fix* divided by *fix'* (floor)

(**logand** *fix fix'*)	bitwise and of *fix* and *fix'*
(**logor** *fix fix'*)	bitwise or *fix* and *fix'*
(**mod** *fix fix'*)	modulus of *fix* and *fix'*

Floats

*(***floatp** *T*)	*float* predicate
*(***float** *T)*	coerce *T* to *float*
*(***float*** *float float'*)	product of *float* and *float'*
*(***float+** *float float'*)	sum of *float* and *float'*
*(***float-** *float F'*)	difference of *float* and *float'*
*(***float<** *float float'*)	is *float* less than *float'?*
*(***float/** *float float'*)	*float* divided by *float'*

(**asin** *float*)	arcsine of *float* degrees
(**acos** *float*)	arccosine of *float* degrees
(**atan** *float*)	arctangent of *float* degrees
(**sin** *float*)	sine of *float* degrees
(**cos** *float*)	cosine of *float* degrees
(**tan** *float*)	tangent of *float* degrees
(**exp** *float float'*)	natural exponential
(**pow** *float float'*)	power function
(**log** *float*)	natural logarithm
(**log10** *float*)	base 10 logarithm
(**sqrt** *float*)	square root

Vectors

(**vectorp** *T*)	*vector* predicate
(**.vec-length** *vector)*	*fixnum* length of *vector*
(**.vec-map** *fn vector)*	make vector from *vector*
(**.vec-mapc** *fn list)*	map function over *vector*
(**.vec-ref** *vector fix)*	*nth* element
(**.vec-type** *vector)*	type of *vector* elements

Streams

standard-input	standard input stream
standard-output	standard output stream
error-output	standard error stream
(**streamp** *T)*	*stream* predicate
(**close** *stream)*	close *stream*
(**eofp** *stream)*	is *stream* at end of file?
(**get-output-string-stream** *stream)*	get *string* from *stream*
(**load** *string)*	load file
(**open-input-file** *string)*	returns file *stream*
(**open-input-string** *string*)	returns *string stream*
(**open-output-file** *string)*	returns file *stream*
(**open-output-string** *string*)	returns *string stream*
(**open-function-stream** *fn*)	returns *function stream*
(**open-socket-server** *fixnum)*	returns socket *stream*
(**open-socket-stream** *fixnum fixnum'*)	returns socket *stream*
(**accept-socket-stream** *stream)*	accept socket *stream*
(**connect-socket-stream** *stream)*	connect socket *stream*
(**read-byte** *stream*)	read *byte* from *stream*
(**read-char** *stream*)	read *char* from *stream*
(**unread-char** *stream)*	push *char* onto *stream*
(**write-char** *char stream)*	write *char* to *stream*
(**write-byte byte** *stream)*	write *byte* to *stream*

Functions

(**codep** *T*)	*code* predicate
(**functionp** *T*)	*function* predicate
(**.apply** *F* ***list)***	apply *function* to arg *list*
(**eval** *T)*	evaluate form
(**closure** *fn*)	reify lexical environment
(**frame-ref** *fix fix'*)	lexical variable of frame
(**.trampoline** *fn*)	trampoline

Namespaces

(**namespacep** *T)*	*namespace* predicate
(**intern** *ns* **:***keyword string T***)**	intern in *namespace*
(**find-ns** *string*)	map *string* to *namespace*
(**find-in-ns** *ns :keyword string*)	map *string* to *symbol*
(**find-symbol** *ns string*)	resolve *symbol* in *namespace*
(**in-ns** *ns*)	set the current *namespace*
(**ns** *string ns*)	xsmake *namespace*, import *ns*
(**ns-current**)	current *namespace*
(**ns-name** *ns)*	*namespaces's* name
(**ns-symbols** *ns*)	list of *namespace's* symbols
(**ns-import** *ns)*	*namespace's* import

Miscellaneous

(**.block** *symbol fn*)	establish named *block*
(**.return** *symbol T*)	return value from *block*
(**.if** *f*n *fn' fn''*)	support **if** *macro*
(**.letq** *symbol T*)	modify lexical value
(**.env-stack** *fix fix'*)	stack as a list of frames
(**.env-stack-depth**)	stack depth as a *fixnum*
(**gc** *boolean*)	garbage collection
(**heap-info** *T*)	heap occupancy for type
(**heap-log** *boolean*)	enable heap logging
(**view** *T*)	make *view* of *T*

Structs

(**structp** *T*)	*struct* predicate
(**struct** *keyword list...*)	make *struct*
(**struct-type** *struct*)	get *struct* type
(**struct-slots** ***struct***)	get *struct* slot values

Special Forms

(**special-operatorp** *symbol)*	special operator predicate
(**:defcon** *symbol form*)	define constant *symbol*
(**:lambda** *list . body*)	define anonymous *function*
(**:letq** *symbol T*)	modify lexical value
(**:macro** *list . body*)	define *macro* expander
(**:quote** *T*)	quote form

Reader

(read *stream*)	read object from *stream*
; #|...|#	comment
(...)	list
'*T*	quote
"...."	*string*
#<...>	broket
#x	hexadecimal *fixnum*
#d	decimal *fixnum*
#o	octal *fixnum*
#\*character*	*character*
#(*type* **...)**	*vector*
#'*function*	closure
#:*symbol*	uninterned *symbol*
#.*T*	read time eval
`\`	`single escape (in strings)`
`" ' ( )`	`terminating macro char`
`` ` , ; ``	`terminating macro char`
`#`	`non-terminating macro char`
`! $ % & *`	`constituent`
`+ - . / :`	`constituent`
`< = > ? @`	`constituent`
`[ | ] ^ _`	`constituent`
`{ } ~`	`constituent`
`A--Z a--z`	`constituent`
`0--9`	`constituent`
`Backspace`	`constituent`
`Rubout`	`constituent`
`Linefeed`	`whitespace`
`Newline`	`whitespace`
`Page`	`whitespace`
`Return`	`whitespace`
`Space`	`whitespace`
`Tab`	`whitespace`

Macros (see **:macro** special operator)

(**macro-function** *macro*)	extract *macro function*
(**macroexpand** *T)*	expand *macro* call
(**set-macro-character** *char fn*)	reader interface

Libmu API

(for libmu version *0.0.20*)

```
char** Environment()
int System(const std::string)
std::string Invoke(uint64_t, std::string)
void* libmu_t();
void* libmu_nil();
const char* libmu_version();
void* libmu_eval(void*, void*);
void* libmu_read_stream(void*, void*);
void* libmu_read_string(void*, std::string);
void* libmu_read_cstr(void*, const char*);
void  libmu_print(void*, void*, void*, bool)
const char* libmu_print_cstr(void*, void*,
bool);
void libmu_terpri(void*, void*);
void libmu_withException(void*,
std::function<void(void*)>);
void* libmu_env_default(Platform*);
void* libmu_env(Platform*, Platform::StreamId,
Platform::StreamId, Platform::StreamId);
```

Mu Defined Forms

(for mu version *0.0.17*)

in mu namespace from core/mu.l

.version	*symbol* constant string
(**defun** *symbol list . body*)	
	define recursive *function*
*(***defmacro** *symbol list . body)*	
	define *macro* expander
*(***defconstant** *symbol T)*	define constant *symbol*
(.recur *symbol list . body)*	
	recursive *function* binding
(**append** . lists)	append lists, last may be atom
(**block** *symbol . body*)	named *block macro*
*(***bool** *T*)	coerce *T* to *boolean*
(**return** *T*)	return from nil *block macro*
(**return-from** *symbol T*)	return from *block macro*
(**and** . *body*)	**and** *macro*
(**check-type** *T T' string*)	error if *T* isn't *T' macro*
(**cond** . *clauses*)	**cond** *macro*
(**.foldl** *fn init list*)	reduce *list* left iterative
(**.foldr** *fn init list*)	reduce *list* right recursive
(**gensym**)	generate unique *symbol*
(**identity** *T*)	identity *function*
(**if** *fn form form'*)	**if** *macro*
(**let** *list . clauses*)	parallel lexical bind *macro*
(**let*** *list . clauses*)	sequential lexical bind *macro*
(**letf** *list . clauses*)	parallel lexical defun *macro*
(**letf*** *list . clauses*)	sequential lexical defun *macro*
(**listp** *T)*	is *T* a *cons* or `:nil`*?*
(**or** . *body*)	**or** *macro*
(**progn** . *body*)	**progn** *macro*
(**load-once** *symbol string*)	
	load file discipline
(**unless** *T . body*)	**if** syntactic sugar *macro*
(**when** *T . body*)	**if** syntactic sugar *macro*
(**list** . *body)*	make *list* of *body*
(**list*** . *body)*	make dotted *list* of *body*