Mu Namespace

(for libmu version *0.0.21*)

Types

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

Symbols:

`:byte`	*uint8_t*
`: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
`: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?

Characters

(**charp** *T*) *character* predicate

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'*
(**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*
(**terpri** *stream)* print newline to *stream*

Fixnums

(**fixnump** *T)* *fixnum* predicate
(**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'?*

(**floor** *fix fix'*) *fix* divided by *fix'* (*mod . rem)*
(**truncate** *fix fix'*) *fix* divided by *fix'* (*mod .rem*)

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

Floats

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

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-stream-string** *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-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

(**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-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*) make *namespace*, import *ns*
(**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*
*(***.letq** *symbol T*) modify lexical value
*(***env**) make *view* of *environment*
(**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** s*ymbol)*
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
*(***:t** *T T'*) return T
*(***:nil** *T T'*) return T'

Reader

(**read *stream***) read object from *stream*

; #|...|#	comment
(…)	list
'*T*	quote
"…."	*string*
#<*type fix (...)***>**	broket
#x	hexadecimal *fixnum*
#d	decimal *fixnum*
#o	octal *fixnum*
#*character*	*character*
#(*type* **…)**	*vector*
#'*function*	closure
#:*symbol*	uninterned *symbol*
#.*T*	read time eval
ns:*symbol*	namespaced external *symbol*
ns::*symbol*	namespaced internal *symbol*
`\`	`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