***Core Library*** ***Referencee***

***core* name space, version *0.0.9***

***type identifiers s***

%lambda closure lambda

%exception exception

%vector vector

%closure lexical closure

*bool* false if (), otherwise true

*char*

*cons*

*fixnum fix*

*float*

*function fn*

*keyword*

*ns*

*null*

*stream*

*string*

*struct*

*symbol sym*

*vector vec*

***Core s***

**%format** *T* *string list* *string* formatted output

**load-file** *string bool* load file through

core reader

**%make-keyword** string make keyword

**%quote** *T* *cons* quote form

**eval** *T*  *T*eval form

**apply** *fn* *list* *T*apply *fn* to *list*

**compile** *T T* compile T in null

environment

**gensym** *sym* create unique

uninterned symbol

**eql** *T T* *bool* eql predicate

***Special*** ***Forms s***

**%defmacro** *sym list . body*

*symbol* define macro

**%lambda** *list* . body *fn* define closure

**%if** *T 'T T* conditional

**%if** *T 'T ''T T* conditional

***Fixnum m***

**1+** *fix* *fix* increment *fix*

**1-** *fix* *fix* decrement *fix*

**logand** *fix* '*fix fix* bitwise and

**lognot** *fix* *fix* bitwise negate

**logor** *fix* '*fix* *fix* bitwise or

**logxor** *fix* '*fix* *fix* bitwise xor

***List s***

**%dropl** *list* *fixnum* *list* drop left

**%dropr** *list* *fixnum list* drop right

**%findl-if** *fn* *list T* element if applied

function returns

an atom, ()

otherwise

**%foldl** *fn T list list* left fold

**%foldr** *fn T list* *list* right fold

**%mapc** *fn list* *list* apply *fn* to *list*

cars, return *list*

**%mapcar** *fn list list* new list from

applying *fn* to

*list* cars

**%mapl** *fn list* *list* apply *fn* to *list*

cdrs, return *list*

**%maplist** *fn* *list list* new list from

applying *fn* to

*list* cdrs

**%positionl-if** *fn* *list* index of element

*T* if *fn* returns an

atom, otherwise

()

**%append** *list*  *list* append lists

**reverse** *list list* reverse *list*

***String*** ***s***

**%string-position** *char string* index of char in

*fix*  *string*, nil if not

found

**%substr** *string fix 'fix string* substring of

*string* from start

to end

**%string=** *string string’* string predicate

*bool*

***Vector s***

**%make-vector** *list* *vector*  specialized

vector from list

**%map-vector** *fn vector* mapc for vectors

*vector*

**make-vector** *list* *vector* general vector

from list

**bit-vector-p** *vector bool* bit vector?

**vector-displaced-p** *vector* a displaced

*bool* vector?

**vector-length** *vector fix* length of *vector*

**vector-ref** *vector* *fix T* element of *vector*

at index *fix*

**vector-slice** *vector* *fix* '*fix* displaced vector

*vector* from start for length

**vector-type** *vector symbol* vector type

***Macro***   ***s***

**define-symbol-macro** *sym T* define symbol

*symbol* macro

**macro-function** *sym list* extract macro

*T* function with

environment

**macroexpand** *T list T* expand macro

expression in

environment

**macroexpand-1***T list* expand macro

*T* expression once

in environment

***Predicate s***

**minusp** *fix bool* negative *fix*

**numberp** *T* *bool* *float* or *fixnum*

**%uninternedp** *sym bool* *symbol* interned

**charp** *T bool* *char*

**consp** *T* *bool cons*

**fixnump** *T* *bool* *fixnum*

**floatp** *T* *bool* *float*

**functionp** *T* *bool* fntion

**keywordp** *T* *bool* keyword

**listp** *T* *bool* *cons* or ()

**namespacep** *T* *bool*  *namespace*

**null** *T* *bool* :nil or ()

**streamp** *T* *bool* *stream*

**stringp** *T* *bool* *char vector*

**structp** *T* *bool* *struct*

**symbolp** *T* *bool* *symbol*

**vectorp** *T* *bool* vector

***Type System t***

**%core-type-p** *T bool* a core type?

**def-type** *symbol list struct* createcoretype

ofname *symbol*

**type-of** *T sym* core type symbol

**typep** *T typespec bool* does *T* conformto

typespec*?*

***Stream xu***

**%peek-char** *stream* *char* read char from

stream, unread

**%format** *T string list T* formatted output

to stream

**read** *stream bool T* *T* read from stream

with EOF handling

**write** *T bool stream* *T* write escaped

object to stream

***Exception n***

**%exceptionf** *stream string bool struct*

*string* format exception

**%make-exception** *sym T string sym list*

*struct* create exception

**error** *T symbol list string* error format

**exceptionp** *struct* *bool* predicate

**raise** *T symbol list* raise exception

**raise-env** *T symbol list*  raise exception

**warn** *T string* *T* warning

**with-exception** *fn fn T* catch exception

***Macro Definitions***  *s*

**and** &rest ... *T* and of ...

**cond** &rest ... *T* cond switch

**let** *list* &rest ... *T* lexical bindings

**let\*** *list* &rest ... *T* dependent list

of bindings

**or** &rest ... *T* or of ...

**progn** &rest ... *T* evaluate rest list,

return last evaluation

**unless** *T* &rest ... *T* if T is (), (**progn** …)

otherwise ()

**when** *T* &rest ... *T* if T is an *atom*,

(**progn** ...)

otherwise ()

***Closures*** *s*

**append** &rest *…* *list* append lists

**format** *T* *string* &rest ... formatted output

*T*

**funcall** *fn* &rest … *T* apply *fn to ...*

**list** &rest … *list list of ...*

**list\*** &rest ... *list* append ...

**mapc** *fn* &rest ... *list* mapcof ...

**mapcar** *fn* &rest ... *list* mapcarof ...

**mapl** *fn* &rest ... *list* maplof ...

**maplist** *fn* &rest ... *list* maplistof ...

***Modules s***

**modules** *list* module definitions

**module-version** *string*

*string* module version

**module-namespace** *string* module

*ns* namespace

**provide** *string list* *T* define module

**require** *string* *bool* load module

***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