

Core Library Reference

core name space, version 0.0.6

type identifiers

%lambda	closure lambda
%exception	exception
%vector	vector
%closure	lexical closure
bool	false if (), otherwise true
char	
cons	
fixnum	fix
float	
func	
keyword	
ns	
null	
stream	
string	
struct	
symbol	sym
vector	

Core

+version+	string	version string
%format	T string list	formatted output
load-file	string bool	load file through core reader
%make-keyword	string	make keyword
%quote	T cons	quote form
eval	T	eval form
apply	func list	apply func to list
compile	T	compile T in null environment
gensym	sym	create unique uninterned symbol

Special Form

%defmacro	sym list . body	symbol	define macro
%lambda	list . body	func	define closure
if	T 'T	T	conditional
if	T 'T 'T	T	conditional

Fixnum

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

%dropl	list fixnum	list	drop left
%dropr	list fixnum	list	drop right
%findl-if	func list	T	element if applied function returns an atom, () otherwise
%foldl	func T list	list	left fold
%foldr	func T list	list	right fold
%mapc	func list		apply func to list cars, return list
%mapcar	func list	list	new list from applying func to list cars
%mapl	func list	list	apply func to list cdrs, return list
%maplist	func list	list	new list from applying func to list cdrs
%positionl-if	func list	T	index of element if func returns an atom, otherwise ()
%append	list	list	append lists
reverse	list	list	reverse list

String

%string-position	char string	index of char in string, nil if not found
%substr	string fix 'fix string	substring of string from start to end

Vector

%make-vector	list vector	specialized vector from list
%map-vector	func vector vector	make vector of func applications on vector elements
make-vector	list vector	general vector from list
bit-vector-p	vector bool	bit vector?
vector-displaced-p	vector bool	a displaced vector?
vector-length	vector fix	length of vector
vector-ref	vector fix T	element of vector at index fix
vector-slice	vector fix 'fix vector	displaced vector from start to end
vector-type	vector symbol	vector type

Macro

define-symbol-macro	sym T symbol	define symbol macro
macro-function	sym list T	extract macro function with environment
macroexpand	T list T	expand macro expression in environment
macroexpand-1	T list T	expand macro expression once in environment

Predicate s			Exception n			Modules s		
minusp <i>fix</i>	<i>bool</i>	negative <i>fix</i>	%exceptionf <i>stream string bool struct</i>			provide <i>string list</i>	<i>T</i>	module definition
numberp <i>T</i>	<i>bool</i>	<i>float</i> or <i>fixnum</i>		<i>string</i>	format exception	require <i>string</i>	<i>bool</i>	module load
%uninternedp <i>sym</i>	<i>bool</i>	<i>symbol</i> interned	%make-exception <i>sym T string sym list</i>			require-lib <i>string</i>	<i>bool</i>	lib module load
charp <i>T</i>	<i>bool</i>	<i>char</i>		<i>struct</i>	create exception	Reader Syntax x		
consp <i>T</i>	<i>bool</i>	<i>cons</i>	error <i>T symbol list</i>	<i>string</i>	error format	<i>;</i>		comment to end of line
fixnump <i>T</i>	<i>bool</i>	<i>fixnum</i>	exceptionp <i>struct</i>	<i>bool</i>	predicate	<i># ... #</i>		block comment
floatp <i>T</i>	<i>bool</i>	<i>float</i>	raise <i>T symbol list</i>		raise exception	<i>'form</i>		quoted form
functionp <i>T</i>	<i>bool</i>	function	raise-env <i>T symbol list</i>		raise exception	<i>`form</i>		backquoted form
keywordp <i>T</i>	<i>bool</i>	keyword	warn <i>T string</i>	<i>T</i>	warning	<i>`(...)</i>		backquoted list (proper lists)
listp <i>T</i>	<i>bool</i>	<i>cons</i> or <i>()</i>	with-exception <i>func func</i>	<i>T</i>	catch exception	<i>,form</i>		eval backquoted form
namespacep <i>T</i>	<i>bool</i>	<i>namespace</i>	Macro Definitions s			<i>,@form</i>		eval-splice backquoted form
null <i>T</i>	<i>bool</i>	<i>:nil</i> or <i>()</i>				<i>(...)</i>		constant <i>list</i>
streamp <i>T</i>	<i>bool</i>	<i>stream</i>	and &rest ...	<i>T</i>	and of ...	<i>()</i>		empty <i>list</i> , prints as <i>:nil</i>
stringp <i>T</i>	<i>bool</i>	<i>char vector</i>	cond &rest ...	<i>T</i>	cond switch	<i>(... . .)</i>		dotted <i>list</i>
structp <i>T</i>	<i>bool</i>	<i>struct</i>	let <i>list</i> &rest ...	<i>T</i>	lexical bindings	<i>"..."</i>		<i>string</i> , <i>char vector</i>
symbolp <i>T</i>	<i>bool</i>	<i>symbol</i>	let* <i>list</i> &rest ...	<i>T</i>	dependent list	<i> </i>		single escape in strings
vectorp <i>T</i>	<i>bool</i>	<i>vector</i>						
Type System t			or &rest ...	<i>T</i>	or of ...	<i>#*...</i>		bit vector
%core-type-p <i>T</i>	<i>bool</i>	a core type?	progn &rest ...	<i>T</i>	evaluate rest list,	<i>#x...</i>		hexadecimal <i>fixnum</i>
def-type <i>symbol list</i>	<i>struct</i>	create core type			return last evaluation	<i>#.</i>		read-time eval
		of name <i>symbol</i>	unless <i>T</i> &rest ...	<i>T</i>	if <i>T</i> is <i>()</i> , (progn ...)	<i>#\.</i>		<i>char</i>
type-of <i>T</i>	<i>sym</i>	core type symbol			otherwise <i>()</i>	<i>#(:type ...)</i>		<i>vector</i>
typesp <i>T typespec</i>	<i>bool</i>	does <i>T</i> conform to	when <i>T</i> &rest ...	<i>T</i>	if <i>T</i> is an <i>atom</i> ,	<i>#s(:type ...)</i>		<i>struct</i>
		<i>typespec</i> ?			(progn ...) otherwise	<i>#:symbol</i>		uninterned <i>symbol</i>
					<i>()</i>			
Stream xu						<i>"` , ;</i>		terminating macro char
%peek-char <i>stream char</i>		read char from	Closures s			<i>#</i>		non-terminating macro char
%format <i>T string list T</i>		stream, unread				<i>!\$%&*+- .</i>		symbol constituents
		formatted output	append &rest ...	<i>list</i>	append lists	<i><=>?@[] </i>		
read <i>stream bool T</i>	<i>T</i>	to stream	format <i>T string &rest ...</i>	<i>T</i>	formatted output	<i>:^_{ }~ /</i>		
		with EOF				<i>:^_{ }~ /</i>		
		handling	funcall <i>func &rest ...</i>	<i>T</i>	apply <i>func</i> to ...	<i>A..Za..z</i>		
write <i>T bool stream</i>		write escaped	list &rest ...	<i>list</i>	<i>list</i> of ...	<i>0..9</i>		
		object to stream	list* &rest ...	<i>list</i>	append ...	<i>0x09 #\tab</i>		whitespace
			vector &rest	<i>vector</i>	<i>vector</i> of ...	<i>0x0a #\linefeed</i>		
						<i>0x0c #\page</i>		
						<i>0x0d #\return</i>		
						<i>0x20 #\space</i>		