

The Koioslisp Specification and Notes, version 0.0

Yash Tulsyan

February 18, 2013

Contents

1	Preface	9
2	Acknowledgements	11
3	Inspiration and Design Features	13
4	Introduction	15
4.1	Do We Really Need Another Dialect?	15
4.2	Conventions Used in This Book	15
5	Data Types	17
5.1	Class Hierarchy of Koioslisp	17
5.2	Numbers	17
5.2.1	Integers	17
5.2.2	Ratios	17
5.2.3	Floating Point Numbers	18
5.2.4	Complex Numbers	18
5.3	Characters	18
5.3.1	Standard Characters	18
5.3.2	Line Divisions	18
5.3.3	Non-standard Characters	18
5.3.4	Character Attributes	19
5.4	Symbols	19
5.5	Arrays	19
5.6	Vectors	19
5.7	Strings	19
5.8	Bit-Vectors	19
5.9	Hash Tables	19
5.10	Readtables	19
5.11	Packages	19
5.12	Pathnames	19
5.13	Sockets	19
5.14	Streams	19
5.15	Random-States	19
5.16	Structures	19
5.17	Functions	19
5.18	Objects	19
5.19	Continuations	19
5.20	Laziness and Lazy Types	19
5.21	Unreadable Data Objects	20
5.22	Overlap, Inclusion, and Disjointness of Types	20

6	Scope and Extent	21
7	Types	23
7.1	Type Specifiers	23
7.1.1	Algebraic Datatypes	23
7.2	Monads	23
7.2.1	Built-in Monads	24
8	Program Structure	25
9	Predicates	27
10	Control Structure	29
10.1	Constants and Variables	29
10.1.1	Reference	29
10.1.2	Assignment and Bindings	29
10.2	Sequencing	30
10.3	Iteration	30
10.3.1	Low-Level Iteration	31
10.3.2	LOOP	31
10.4	List Comprehension	33
10.5	Catch, Throw, and Unwind-Protect	34
11	Macros	35
11.1	Definition of Macros	35
11.2	Expansion of Macros	35
11.3	Compiler Macros	35
11.4	Environments	35
12	Declarations	37
12.1	Declaration Specifiers	37
12.2	Forms and Type Declaration	37
13	Symbols	39
14	Packages and Systems	41
14.1	Colon syntax	42
14.2	Built-in Modules	43
15	Mathematics	45
15.1	The Math library	45
16	Characters	47
17	Sequences	49
17.1	Basic Sequence Functions	49
17.2	Advanced Sequence Functions	50
17.3	Searching Sequences	51
17.4	Sorting and Merging	51

18 Lists	53
18.1 Conses	53
18.2 Lists	54
18.3 Association Lists	56
18.4 Mapping	56
19 Sets	57
20 Hash Tables	59
21 Arrays	63
21.1 Vectors	63
22 Conclists	65
23 Strings	67
23.1 Comparison of Strings	67
23.2 Regular Expressions	68
23.2.1 Regular Expression Syntax	68
23.2.2 Regular Expression Filters	69
24 The Evaluator	71
25 Lazy Evaluation	73
25.1 Series	73
26 Continuations	75
27 Logic Programming	77
27.1 Constraints	77
28 Sockets	79
29 Streams	81
30 Objects	83
30.1 Classes	84
30.1.1 Defining Classes	84
30.1.2 Instance Creation	84
30.1.3 Slots	84
30.1.4 Accessing Slots	84
30.1.5 Inheritance	84
30.1.6 Built-in Classes	84
30.1.7 Determining the Class Precedence List	84
30.2 Generic Functions and Methods	84
30.2.1 Introduction	84
30.2.2 Agreement on Parameter Specializers and Qualifiers	84
30.2.3 Keyword Arguments	84
30.2.4 Method Selection	84
30.2.5 Determinating the Effective Method	84
30.2.6 Standard Method Combination	84
30.2.7 Other Method Combinations	84
30.2.8 Defining Method Combinations	84
30.2.9 SET methods	84

30.3	Structures	84
30.3.1	Defstruct Options	84
30.3.2	BOA Constructors	84
30.4	The Meta-Object Protocol	84
31	Input/Output	87
31.1	Readtables	87
32	File System Interface	89
32.1	File Names	89
32.1.1	Pathnames	89
32.1.2	Case Conventions	90
32.1.3	Structured Directories	90
32.1.4	Extended Wildcards and Globbing	90
32.1.5	Logical Pathnames	90
32.1.6	Pathname Functions	90
32.2	File and Directory Operations	90
32.2.1	File I/O	90
32.2.2	Renaming, Deleting, and Other File Operations	90
32.2.3	Accessing Directories	90
33	Errors	91
34	Advice	93
35	Miscellaneous Features	95
35.1	The Compiler	95
35.1.1	Compiler Diagnostics	95
35.1.2	Compiled Functions	95
35.1.3	Compilation Environment	95
35.1.4	Similarity of Constants	95
35.2	Documentation	95
35.3	Debugging Tools	96
35.4	Environment	96
36	Pretty-Printing	99
37	Concurrency	101
37.1	Refs	101
37.2	The Dining Philosophers Problem	102
38	Backquote	103
39	Implementation Notes	105
40	KLIM	107
40.1	Text Styles	116
40.2	Color	117
40.3	Designs	119
40.4	Presentations	119
40.5	The Drawing Plane	122
40.6	Coordinates	122
40.7	Controlling the Look and Feel	122

40.8 Streams	122
40.9 Extended Input/Output	122
41 Batteries—Utility Libraries	123
41.1 Logging	123
41.2 Data Compression	123
41.3 Foreign Function Interfaces	123
41.4 OS Services and Adaptations	123
41.4.1 UNIX-specific	123
41.5 Cryptography	123
41.6 Internet Data Handling	123
41.7 Markup Language Processing Tools	123
41.8 Internet Protocols	123
41.9 Multimedia Services	123
42 Community Things	125
42.1 CKLAN	125
42.2 Lake	125
42.3 LispDoc	125
42.4 ASDF-like tools	125
43 References	127

Chapter 1

Preface

Chapter 2

Acknowledgements

If Koioslisp is a better language than others (a questionable proposition), it is, proverbially, because it stood on the shoulders of giants. First and foremost, I must acknowledge all the Lispers who came before me—the hackers of MIT, BBN, Xerox, the Lisp Machine world, and the Lisp community of today—for without their work, none of this would be possible. In particular, I must acknowledge the great implementors and theorists—giants of the LISP community: Professor John McCarthy, Steve Russell, Jonl White, Guy L. Steele, David Moon, Richard P. Gabriel, Richard M. Stallman, Gerald Jay Sussman, and more. I must also acknowledge the giants of recent times such as Paul Graham, Peter Seibel, Pascal Costanza, Edi Weitz, Zach Beane, François-René Rideau, James Kalenius, Nikodemus Siivola, Shriram Krishnamurthi, Stanislav Datskovskiy and Rich Hickey.

Chapter 3

Inspiration and Design Features

Primary inspiration for KOIOSLISP came from the COMMON LISP programming language (indeed, the document you are reading right now is structured based on the authoritative COMMON LISP reference, *Common Lisp, the Language* by Guy L. Steele)[4], with influences from SCHEME, CLOJURE, and PROLOG. A design influence comes from the language PYTHON and other modern scripting languages—in that “batteries” (i.e. utilities to connect to the rest of the world) should come included in the standard library, which is indeed part of the reason features which seem superfluous are included in KOIOSLISP. In the specification, batteries will be marked clearly, as such utilities are far more likely to change over time. KOIOSLISP attempts to unify the strengths of these Lisp dialects into one—COMMON LISP for its completeness, macros, object system, historical significance, large amounts of libraries (hopefully, the amount of compatibility given by KOIOSLISP will encourage people to port useful libraries to it, such as ASDF, or SWANK (which is used for SLIME, the free Lisp IDE on Emacs)), and continuity; SCHEME for its elegance, continuations, and assorted innovations; CLOJURE for its concurrency support, clean hash tables, and lazy evaluation. It also features PROLOG-style logic programming. In doing so, it tries to avoid the problems of the languages above mentioned. While COMMON LISP and SCHEME have aged relatively well, COMMON LISP has non-lispy hash tables and some marks of age, while SCHEME, though good for pedagogy, is crippled by its size and lack of powerful macros (though it does have “hygenic” macros), as well as some marks of age. Meanwhile, CLOJURE is dependent on the JVM (or, in the case of its new ports, the CLR and JavaScript—which create some incompatibility between implementations, as CLOJURE feeds off of those engines and absorbs their libraries and object systems) and departs from the “mud ball” metaphor of lisp, and PROLOG is entirely unusable for programming-in-the-large, being useful mainly as an extension to Lisp. Throughout this document, some compatibility notes will be recorded.

Chapter 4

Introduction

This is a specification for a KOIOSLISP, a LISP dialect, which will evolve as time goes on and the fickle attitudes of the BDFL, Yash Tulsyan, change. If you are a programmer from C-like languages who expects a C-like syntax and a C++ (or Java/C#) like object-oriented system imposed upon the user, then please either accept that this will not cater to that palate, or leave. If you are a programmer from Haskell or ML-like languages who is similarly put-off by s-expressions, want a functional style to be imposed upon the user, and caution against side-effects, we advise a similar course of action. As of this moment, this specification does not come with any standard implementation, due to the fact that the BDFL is currently a simple student who has not yet learned how to create a fully-featured compiler or an interpreter.

4.1 Do We Really Need Another Dialect?

This question appears whenever a new language appears, and in Lisp it is especially important, due to both its history and COMMON LISP macros. My answer to this: Maybe. If so, this is going to be it. If not, we (or at least, I) will probably use Common Lisp.

4.2 Conventions Used in This Book

Chapter 5

Data Types

5.1 Class Hierarchy of Koioslisp

5.2 Numbers

As in most programming languages, there are several different types of numbers in KOIOSLISP, and, as in Common Lisp, they are divided into four main categories: *integers*, *ratios*, *floating-point numbers*, and *complex numbers*. Integers and ratios are contained by the type *rational*. Rational numbers and floating-point numbers are of the type *real*, and finally real numbers and complex numbers are of type *number*.

5.2.1 Integers

Integers function as in Common Lisp—as representations of mathematical integers. There are no standard restrictions on the magnitude of an integer, as storage should be automatically allocated for representing large integers. However, there might still be a low-level distinction between very efficient integers of smaller magnitude (*fixnums*) and larger integers, called *bignums*. Though this distinction may exist in implementations of the language, Koioslisp is designed, like Common Lisp, to hide the distinction whenever possible. Because the numbers which may be considered *fixnums* and those which may be considered *bignums* will vary from implementation to implementation, the constants `+most-positive-fixnum+` and `+most-negative-fixnum+` are provided. As in most programming languages, integers are canonically written as they are canonically in arithmetic—with decimal digits, possibly preceded by a sign and followed by a decimal point. Integers can be written in bases other than the default 10. The syntax `#nnrdddd`, case insensitively, will refer to the integer represented in *nn*-radix by *dddd*, where *nn* is between 2 and 36, inclusive, and *dddd* is an integer. When representing numbers in radices above 10, use of letters to represent numbers is case-insensitive—that is, `#16rFACADE` is the same as `#16RfacaDE`. Binary, octal, and hexadecimal are relatively common, and so they are granted the abbreviations `#b`, `#o` and `#x`.

5.2.2 Ratios

A *ratio* is, as in COMMON LISP, a representation of a mathematical ratio, and the other part (along with integers) that constitute the type *rational*. The canonical representation of a ratio is either an integer if it is equal to an integer, or the ratio of a *numerator* and *denominator*, which are both integers, and whose greatest common denominator is one. The character `\` separates the numerator from the denominator. One can input a non-canonical value, but a ratio will always be printed by `print` in its canonical form. As all other reals, ratios may be signed, (with the same syntax as other real numbers). Internally, ratios in non-canonical form will be converted to canonical form. The denominator of a ratio must not be zero; if it is zero, an error is signaled. As with integers, ratios can be used in different radices, with the same syntax.

5.2.3 Floating Point Numbers

5.2.4 Complex Numbers

5.3 Characters

5.3.1 Standard Characters

Koioslisp uses a standard character set in order to enhance portability—Koioslisp programs that use only the standard set of characters can be (assuming no other implementation-specific (mis)features are used) read by any Koioslisp implementation. The standard character set of KOIOSLISP is nearly identical to the Unicode character set. As the Unicode character set has several thousands of characters, it will not be displayed here. It also includes ways to denote non-printing characters, such as

```
#\newline #\space #\backspace #\tab #\linefeed #\page #\return
```

. Of the non-blank printing characters, most are not used in KOIOSLISP syntax. For brevity, we shall note the subset of the subset of the Unix code subset “Basic Latin” which denotes punctuation which is not commonly used in KOIOSLISP syntax: [] { } ! ^ _ ~ \$ %. Almost all of these, however, are used as format directives. Other than that, the following characters are explicitly reserved for the user’s syntactic extension purposes: [,] , { , } , \$, % , ! , . This preserves the syntactic purity and clarity of Lisp, unlike some recent dialects which insist on the introduction of other forms of brackets, allegedly for purposes of clarity. See chapter “Input/Output” for greater detail on transforming constituent characters into macro characters

5.3.2 Line Divisions

Unlike the nightmare of implementing a standard method of line division in previous decades, the structure of line divisions has simplified from a variety of schemas involving linefeeds, carriage returns, and other such characters into two schemes—either a bare linefeed (LF) character, used by UNIX and its descendants; or a linefeed following a carriage return (CR), used by MS-DOS and Microsoft Windows. Like Common Lisp and C, KOIOSLISP uses the abstract requirement of requiring a single character,

```
#\newline
```

to act as a delimiter between lines. This character may be externally translated to whatever sort of line-division schema the underlying operating system uses, however, it must internally use one character, and one character only. Another issue is raised. Suppose one types the following:

```
#\Return#\Newline
```

. Implementations should not suppress the call to

```
#\Return
```

, for there are certain standards (such as the Internet Relay Chat (IRC) protocol) which require CRLF sequences.

5.3.3 Non-standard Characters

KOIOSLISP implementations may add extra characters, however, this may render programs using such characters unportable.

5.3.4 Character Attributes

5.4 Symbols

5.5 Arrays

5.6 Vectors

5.7 Strings

5.8 Bit-Vectors

5.9 Hash Tables

5.10 Readtables

5.11 Packages

5.12 Pathnames

5.13 Sockets

5.14 Streams

5.15 Random-States

5.16 Structures

5.17 Functions

5.18 Objects

5.19 Continuations

KOIOSLISP, like Scheme, provides *continuations*, convenient and abstract mechanisms for controlling the control state of a program. Unlike Scheme, however, these are *delimited* continuations. Though they can be discussed here as structures, due to their nature as GOTO-like mechanisms (though theoretically much more fundamental), they are discussed in greater detail in the Continuations chapter of this book.

5.20 Laziness and Lazy Types

Though *lazy evaluation* refers to an evaluation strategy, it is useful to speak of *lazy* forms of data structures, of which the most central is the lazy list.

5.21 Unreadable Data Objects

5.22 Overlap, Inclusion, and Disjointness of Types

Chapter 6

Scope and Extent

Scope and *extent* are important when discussing features of a language. *Scope* is the spatial region on which a variable is defined—*extent* is the temporal region. Koioslisp, like Common Lisp, utilizes two systems of scope and two systems of extent—*lexical scope*, *indefinite scope*, *dynamic extent*, and *indefinite extent*.

Chapter 7

Types

7.1 Type Specifiers

7.1.1 Algebraic Datatypes

[*Macro*]

`datatype name lambda-list &body body`

`datatype` defines *name* to be a type specified in *body*, with the use of *lambda-list* to allow for parametric data types. The new type specifier can be referred to as *name* if *lambda-list* is empty, but otherwise, it can be referred to as *(name arg1 arg2 ...)*. If symbols used in the body of `datatype` are not quoted, they are taken to refer to other types (and thus cease to be symbols); if no such type has been defined, it is a newly created singleton type. The syntax of the body of `deftype` is that of a type specifier, as that is what is being created.

An example of the use of `datatype` is given below.

```
(datatype Rank () (or ace 2 3 4 5 6 7 8 9 10 page knight queen king))
(datatype Suit () (or swords wands coins cups))
(datatype Minor-Arcana () (list Rank Suit))
(datatype Major-Arcana ()
  (or Magician High-Priestess Empress Emperor Hierophant Lovers Chariot
    Strength Hermit Wheel-of-Fortune Justice Hanged-Man Death Temperance
    Devil Tower Star Moon Sun Judgement World Fool))
(datatype Tarot () (or Major-Arcana Minor-Arcana))
(type-of Magician) ; → Magician
(type? Magician Major-Arcana) ; → T, T
(type? Magician 'symbol) ; → NIL, T
(type-of '(8 cups)) ; → List
(type? '(8 cups) Tarot) ; → T, T
```

7.2 Monads

KOIOSLISP has a monad facility, inspired by Haskell and Clojure. Because KOIOSLISP is not a pure language, certain (normally monadic) constructs do not need to be wrapped in monads. [*Macro*]

`defmonad name (result-ll result-body) (bind-ll bind-body) &optional docstring`

The first argument to `defmonad` is the name of the resulting monad. The second is the `result` function, in the form `(lambda-list body)`. It is followed by the fourth argument, the `bind` function, also in the `(lambda-list body)` format. Optionally, the body may be followed by a `docstring`, which is simply a string. The `lambda-list`

is technically unnecessary, but allows relative ease in remembering variable names.

[*Macro*]

`domonad` *domonad* takes the name of a monad

[*Macro*]

`with-monad` *name* &body *forms*

`with-monad` evaluates *forms* as if they were evaluated in the monad *name*

7.2.1 Built-in Monads

Identity

Maybe

A function in the `Maybe` returns either `Just value` or `Nothing`.

List

In the `List` monad, `result` returns a singleton list containing the argument passed to it. The `bind` function acts as if `mappend` were called with the *return-type* being `list`.

Vector

In the `Vector` monad, `result` returns a singleton vector containing the argument passed to it. The `bind` function acts as if `mappend` were called with the *return-type* being `vector`.

Conclist

In the `Conclist` monad, `result` returns a singleton conclist containing the argument passed to it. The `bind` function acts as if `mappend` were called with the *return-type* being `conclist`.

Series

In the `Series` monad, `result` returns a singleton series containing the argument passed to it. The `bind` function acts as if `mappend` were called with the *return-type* being `series`.

IO

In the `IO` monad, `result` returns `IO value`. The `bind` function passes the value returned by its first argument to its second argument.

STM

In the `STM` monad, `result` returns `STM value`. The `bind` function passes the value returned by its first argument to its second argument.

Chapter 8

Program Structure

Chapter 9

Predicates

[Function]
eq?
[Function]
eqv?
[Generic Function]
=?
[Function]
not *x*
[Function]
type? *object type*

Chapter 10

Control Structure

[*Macro*]
and &rest *forms*
[*Macro*]
or &rest *forms*
[*Function*]
xor &rest *forms*
[*Macro*]
zap!
[*Macro*]
alias
[*Macro*]
partial *function args-or-placeholders*
A *placeholder* is represented the atom _ and the placeholder arguments form the arguments of the returned function
[*Function*]
memoize
[*Function*]
compose *f* &rest *funcs*
[*Function*]
conjoin &rest *predicates*
[*Function*]
disjoin &rest *predicates*

10.1 Constants and Variables

10.1.1 Reference

10.1.2 Assignment and Bindings

Low-Level Assignment

Like TAGBODY, PROG and GO, many of these functions are not used often in casual code, but they are necessary in the implementation of some of the higher-level functions and indispensable in the implementation of new languages upon Koios. [*Function*]

%set

[*Function*]

%setq

Generalized References

[*Macro*]
set!
[*Macro*]
pset!
[*Macro*]
rotate!
[*Macro*]
shift!
[*Macro*]
defzap
[*Macro*]
defsetter
[*Macro*]
defsetter*

Establishing New Bindings

[*Special form*]
let ({*var* | (*var value*))* {*declaration*}* {*form*}*
A **let** form introduces new variable bindings to be used in the forms it contains.

Unlike previous lisps (except Clojure), in Koioslisp, **let** supports *destructuring*, of the sort found in the **destructuring-bind** construct in COMMON LISP. It also supports the functionality previously in the COMMON LISP construct **multiple-value-bind**. See also the Koioslisp constraint-programming facilities.

10.2 Sequencing

[*Special form*]
begin {*form*}*
[*Special form*]
%progx *integer* {*form*}*

10.3 Iteration

[*Macro*]
do ({*var* | (*var* [*init* [*step*]])}* (*end-test* {*result*}*) {*declaration*}* {*tag* | *statement*}*
do is a very powerful, useful, and arcane (at least, to newcomers) macro meant to facilitate complex iterations.

[*Macro*]
do*({*var* | (*var* [*init* [*step*]])}* (*end-test* {*result*}*) {*declaration*}* {*tag* | *statement*}*
As the name suggests, **do*** is a variant of **do**

[*Macro*]
doseq (*var listform* [*resultform*]) {*declaration*}* {*tag* | *expression*}*

```
doseq
[Macro]
dotimes (var listform [countform]) {declaration}* {tag | expression}*
dotimes
```

10.3.1 Low-Level Iteration

Like %SET and %SETQ, many of these functions are not used often in casual code, but they are necessary in the implementation of some of the higher-level functions and indispensable in the implementation of new languages upon Koios.

```
[Special form]
%tagbody {tag | expression}*
[Special form]
%go tag

[Macro]
%prog ( {var | (var [init])}* ) {declaration}* {tag | expression}*
[Macro]
%prog* ( {var | (var [init])}* ) {declaration}* {tag | expression}*
```

10.3.2 LOOP

The syntax and semantics of the LOOP macro in Koioslisp is very different than that of Common Lisp, which has been criticized for being inelegant, inextensible, and generally un-Lispy; though regarded as an excellent demonstration of Lisp's ability to create domain-specific languages. Instead, it is based upon the related ITERATE library for Common Lisp.

```
[Macro]
loop
[Loop clause]
repeat n
[Loop clause]
for var &sequence
&sequence is one of the following symbols: :from, :upfrom, :downfrom, :to, :downto, :above, :below, and :by.
[Loop clause]
for var :in sequence &sequence
[Loop clause]
for key val :in-table table
[Loop clause]
for var :in-package package &optional :external-only external-only?
[Loop clause]
for var :in-packages package &optional :having-access symbol-types
[Loop clause]
for var :in-file filename &optional :using reader
[Loop clause]
for var :in-stream stream &optional :using reader
[Loop clause]
for var :next expr
[Loop clause]
for var :do-next form
[Loop clause]
```

```

for pvar :previous var &optional :initially init :back n
[Loop clause]
with var &optional := value
[Loop clause]
for var := expr
[Loop clause]
for var :initially init-expr :then then-expr
[Loop clause]
for var :first first-expr :then then-expr
[Loop clause]
sum expr &optional :into var
[Loop clause]
multiply expr &optional :into var
[Loop clause]
counting expr &optional :into var
[Loop clause]
maximize expr &optional :into var
[Loop clause]
reducing expr :by func &optional :initial-value init-val :into var
[Loop clause]
collecting expr &optional :into var :at place :result-type type
place is one of :start, :beginning (a synonym for :start) or :end.
[Loop clause]
adjoining expr &optional :into var :at place :result-type type
place is one of :start, :beginning (a synonym for :start) or :end.
[Loop clause]
appending expr &optional :into var :at place :result-type type
place is one of :start, :beginning (a synonym for :start) or :end.
[Loop clause]
appending! expr &optional :into var :at place :result-type type
place is one of :start, :beginning (a synonym for :start) or :end.
[Loop clause]
unioning expr &optional :into var :at place :result-type type
place is one of :start, :beginning (a synonym for :start) or :end.
[Loop clause]
unioning! expr &optional :into var :at place :result-type type
place is one of :start, :beginning (a synonym for :start) or :end.
[Loop clause]
accumulate expr :by func &optional :initial-value init-val :into var
[Loop clause]
finding expr :such-that test &optional :into var :on-failure failure-value
[Loop clause]
finding expr :maximizing m-expr &optional :into var
[Loop clause]
finding expr :minimizing m-expr &optional :into var
[Loop clause]
first-iteration?
[Loop clause]
first-time?
[Loop clause]
always expr
[Loop clause]

```



```

never expr
[Loop clause]
there-is expr
[Loop clause]
finish
[Loop clause]
leave &optional value
[Loop clause]
next-iteration
[Loop clause]
while expr
[Loop clause]
until expr
[Loop clause]
if-first-time then &optional else
[Loop clause]
initially &rest forms
[Loop clause]
after-each &rest forms
[Loop clause]
else &rest forms
[Loop clause]
finally &rest forms
[Loop clause]
finally-protected &rest forms
[Loop clause]
in name &rest forms
[Function]
display-loop-clauses &optional clause-spec
[Macro]
defloop arglist &body body
[Macro]
defloop-driver arglist &body body
[Macro]
defloop-alias syn word
[Macro]
defloop-sequence element-name index-name &key access-fn size-fn sequence-type element-type element-doc-string index-doc-string

```

10.4 List Comprehension

List comprehension refers to a convenient method of generating collections of items that fit a specific criterion. [*Function*]

```
iota start &optional end &key :step
```

This function has the alias ι , that is, the literal small iota character (U+03B9 and optionally U+2373). If an *end* is not specified, it returns a series; otherwise, it returns a list.

```
[Macro]
collect
```

```
[Collect option]
```

```
while
```

[*Collect option*]
in
[*Collect option*]
when

10.5 Catch, Throw, and Unwind-Protect

[*Special form*]
catch *tag* {*form*}*
[*Special form*]
unwind-protect *protected-form* {*cleanup-form*}*
[*Special form*]
throw *tag* *result*

Chapter 11

Macros

11.1 Definition of Macros

[*Function*]
macro-function *symbol*

[*Macro*]
defmacro *name* *lambda-list* [*{declaration}** | *doc-string*] *{form}**

[*Macro*]
defmacro* *name* *lambda-list* [*{declaration}** | *doc-string*] *{form}** defmacro* is like defmacro, except that it ensures that the macros are *hygienic*, such as to prevent variable capture.

[*Macro*]
define-syntax

[2] [*Macro*]
syntax-rules

11.2 Expansion of Macros

[*Function*]
macroexpand *form* &optional *env*

[*Function*]
macroexpand-1 *form* &optional *env*

[*Variable*]
macroexpand-hook

11.3 Compiler Macros

11.4 Environments

Chapter 12

Declarations

[Special form]
`declare` {*decl-spec*}*
[Special form]
`locally` {*declaration*}* {*form*}*
[Function]
`proclaim` *decl-spec*
[Macro]
`declaim` {*decl-spec*}*
[Macro]
`define-declaration` *decl-name lambda-list* {*form*}*

12.1 Declaration Specifiers

`special`
`type`
type
`pure`
`multithreaded`
`return-type`
`ftype`
`inline`
`notinline`
`ignore`
`optimize`
`declaration`
`dynamic-extent`

12.2 Forms and Type Declaration

[Special form]
`the` *value-type form*

Chapter 13

Symbols

Chapter 14

Packages and Systems

The Common Lisp package system is *not* inherited, as it turned out to be quite a mess. [*Function*]

`load`

[*Function*]

`provide`

[*Function*]

`require`

[*Function*]

`use-system`

[*Function*]

`shadow`

[*Function*]

`intern`

[*Function*]

`unuse-system`

[*Function*]

`unintern`

[*Function*]

`import` *module symbols*

[*Function*]

`export`

[*Function*]

`defsystem`

[*System option*]

`:name`

[*System option*]

`:version`

[*System option*]

`:maintainer`

[*System option*]

`:author`

[*System option*]

`:licence`

Because some projects may be multi-licensed, or certain parts of projects can be licensed differently than others, the item following the `:licence` option can be a list, and modules (see the `:module` form) can include this option. A few pre-defined licences are provided:

`:gpl` The GNU General Public Licence

`:l1gpl` The Lisp-GNU Lesser General Public Licence

```

:agpl The Affero General Public Licence
:x11 The X11 Licence (often referred to as the MIT Licence)
:bsd3 The 3 Clause BSD Licence (BSD Licence sans advertising clause, also referred to as the “Modified
BSD Licence”)
:bsd2 The 2 Clause BSD Licence (FreeBSD licence)
:public Public domain

```

```

    [System option]
:description
[System option]
:long-description
[System option]
:serial
[System option]
:depends-on
[System option]
:components
[System-components form]
:file
[System-components form]
:depends-on
[System option]
:module
[System-module form]
:pathname
[System-module form]
:components
[System-module form]
:aliases
[System-module form]
:licence
[System option]
:aliases

```

14.1 Colon syntax

```
foo:bar
```

```
foo::bar
```

```
foo::(bar quux frob)
```

When read, evaluates `(bar quux frob)` as if it were read in the `F00` module. This feature is currently provided in Steel Bank Common Lisp and Allegro Common Lisp. `foo::bar::quux` Modules can be nested, as in this example.

```
:bar
```

This interns the symbol `BAR` as an external symbol in the `keyword` module.

```
#:bar
```

14.2 Built-in Modules

koios-lisp

koios-lisp-user

keyword

system

math

Chapter 15

Mathematics

KOIOSLISP's standard mathematical features have been changed somewhat from COMMON LISP's in light of the IEEE standard for floating-point arithmetic. However, they are on the whole fairly similar, and because of the presence of complex numbers in Koioslisp, some NaN (not a number results) are removed.

15.1 The Math library

[*Function*]
rref
[*Function*]
mean *sample*
[*Function*]
median *sample*
[*Function*]
variance *sample* &key *biased*
[*Function*]
standard-deviation *sample* &key *biased*
[*Function*]
map-iota *function* *n* &key *start* *step*
[*Function*]
clamp *number* *min* *max*
[*Function*]
binomial-coefficient *n* *k*
[*Function*]
count-permutations *n* &optional *k*
[*Function*]
lerp *v* *a* *b*
[*Function*]
factorial *n*
[*Function*]
subfactorial *n*
[*Function*]
gaussian-random &optional *min* *max*
[*Function*]
invert-matrix
[*Function*]
transpose

[*Function*]
conjugate-transpose
[*Function*]
eigenvalues
[*Function*]
eigenvectors

Chapter 16

Characters

Chapter 17

Sequences

17.1 Basic Sequence Functions

[*Generic Function*]
nth *sequence index*
[*Generic Function*]
with *object sequence*
[*Generic Function*]
with! *object sequence*
[*Generic Function*]
subseq *sequence start* &optional *end*
[*Generic Function*]
copy-seq *sequence*
[*Generic Function*]
length *sequence*
[*Generic Function*]
reverse *sequence*
[*Generic Function*]
reverse! *sequence*
[*Generic Function*]
make-sequence *type size* &key *initial-element*
[*Generic Function*]
union
[*Generic Function*]
union!
[*Generic Function*]
intersection
[*Generic Function*]
intersection!
[*Generic Function*]
set-difference
[*Generic Function*]
set-difference!
[*Function*]
complement *function*
[*Generic Function*]
flatten *sequence*

17.2 Advanced Sequence Functions

All of the pure advanced sequence generic functions (save `map`) should have an implementation where the *sequence* arguments are of type `sequence`. They may optionally provide implementations for sub-classes of `sequence` in the interests of efficiency.

[*Generic Function*]

`map` *result-type function sequence* &rest *more-sequences*

[*Generic Function*]

`map!` *result-sequence function* &rest *sequences*

[*Generic Function*]

`append` *result-type* &rest *sequences*

[*Function*]

`mappend` *result-type function sequence* &rest *more-sequences*

[*Function*]

`some` *predicate sequence* &rest *more-sequences*

[*Function*]

`every` *predicate sequence* &rest *more-sequences*

[*Function*]

`notany` *predicate sequence* &rest *more-sequences*

[*Function*]

`notevery` *predicate sequence* &rest *more-sequences*

[*Function*]

`reduce` *function sequence* &key *:from-end :start :end :initial-value :key*

[*Generic Function*]

`map-reduce`

[*Generic Function*]

`fill` *sequence item* &key *:start :end*

[*Generic Function*]

`fill!` *sequence item* &key *:start :end*

[*Generic Function*]

`replace` *sequence1 sequence2* &key *:start1 :end1 :start2 :end2*

[*Generic Function*]

`replace!` *sequence1 sequence2* &key *:start1 :end1 :start2 :end2*

[*Generic Function*]

`remove` *item sequence* &key *:from-end :test :start :end :count :key*

[*Generic Function*]

`remove-if` *predicate sequence* &key *:from-end :start :end :count :key*

[*Generic Function*]

`remove-duplicates` *sequence* &key *:from-end :test :start :end :key*

[*Generic Function*]

`remove!` *item sequence* &key *:from-end :test :start :end :count :key*

[*Generic Function*]

`remove-if!` *predicate sequence* &key *:from-end :start :end :count :key*

[*Generic Function*]

`filter` *item sequence* &key *:from-end :test :start :end :count :key*

[*Generic Function*]

`filter-if` *predicate sequence* &key *:from-end :start :end :count :key*

[*Generic Function*]

`filter!` *item sequence* &key *:from-end :test :start :end :count :key*

[*Generic Function*]

`filter-if!` *predicate sequence* &key *:from-end :start :end :count :key*

[*Generic Function*]

```

partition item sequence &key :from-end :test :start :end :count :key
[Generic Function]
partition-if predicate sequence &key :from-end :start :end :count :key
[Generic Function]
remove-duplicates! sequence &key :from-end :test :start :end :key
[Generic Function]
substitute newitem olditem sequence &key :from-end :test :start :end :count :key
[Generic Function]
substitute-if new-item test sequence &key :from-end :start :end :count :key
[Generic Function]
substitute! newitem olditem sequence &key :from-end :test :start :end :count :key
[Generic Function]
substitute-if! new-item test sequence &key :from-end :start :end :count :key

```

17.3 Searching Sequences

```

[Generic Function]
member
[Generic Function]
member-if
[Generic Function]
find item sequence &key :from-end :test :test-not
[Generic Function]
find-if predicate sequence &key :from-end :start :end :key
[Generic Function]
position item sequence &key :from-end :test :test-not :start :end :key
[Generic Function]
position-if predicate sequence &key :from-end :start :end :key
[Generic Function]
count item sequence &key :from-end :test :test-not :start :end :key
[Generic Function]
count-if predicate sequence &key :from-end :start :end :key
[Generic Function]
mismatch sequence1 sequence2 &key :from-end :test :test-not :key :start1 :start2 :end1 :end2
[Generic Function]
search sequence1 sequence2 &key :from-end :test :test-not :key :start1 :start2 :end1 :end2

```

17.4 Sorting and Merging

```

[Generic Function]
sort sequence predicate &key :key
[Generic Function]
stable-sort sequence predicate &key :key
[Generic Function]
sort! sequence predicate &key :key
[Generic Function]
stable-sort! sequence predicate &key :key
[Generic Function]
merge result-type sequence1 sequence2 predicate &key :key

```

[*Generic Function*]
merge! *result-type sequence1 sequence2 predicate &key :key*

Chapter 18

Lists

Lists are the primary—and the first—data structure of Lisp. A list is a chain of *conses* (which are data structures with two fields, the *car* and the *cdr*)—linked to each other via *cdrs*.

18.1 Conses

[*Function*]

car *list*

[*Function*]

cdr *list*

[*Function*]

caar *list*

[*Function*]

cadr *list*

[*Function*]

cdar *list*

[*Function*]

cddr *list*

[*Function*]

caaar *list*

[*Function*]

caadr *list*

[*Function*]

cadar *list*

[*Function*]

caddr *list*

[*Function*]

cdaar *list*

[*Function*]

cdadr *list*

[*Function*]

cdadar *list*

[*Function*]

cdddr *list*

[*Function*]

caaaar *list*

[*Function*]

`caaddr list`
[Function]
`caadar list`
[Function]
`caaddr list`
[Function]
`cadaar list`
[Function]
`cadadr list`
[Function]
`caddar list`
[Function]
`caddr list`
[Function]
`cdaaar list`
[Function]
`cdaadr list`
[Function]
`cdadar list`
[Function]
`cdaddr list`
[Function]
`cddaar list`
[Function]
`cddadr list`
[Function]
`cdddar list`
[Function]
`cdddr list`
[Function]
`cons x y`
[Function]
`tree-equal x y`

18.2 Lists

[Function]
`end?`
[Method]
`length`
[Method]
`nth`
[Function]
`first`
[Function]
`second`
[Function]
`third`
[Function]
`fourth`

[*Function*]
fifth
[*Function*]
sixth
[*Function*]
seventh
[*Function*]
eighth
[*Function*]
ninth
[*Function*]
tenth
[*Function*]
rest
[*Function*]
nthcdr
[*Method*]
last
[*Function*]
list
[*Function*]
list*
[*Function*]
list? *object*
[*Function*]
make-list
[*Method*]
append
[*Function*]
copy-list
[*Function*]
copy-tree
[*Method*]
revappend
[*Method*]
append!
[*Method*]
revappend!
[*Macro*]
push!
[*Macro*]
pushnew!
[*Macro*]
pop!
[*Function*]
butlast
[*Function*]
nbutlast
[*Function*]
ldiff
[*Method*]
subst

[*Method*]
subst-if
[*Method*]
subst!
[*Method*]
subst-if!
[*Function*]
sublis
[*Method*]
member
[*Method*]
member-if
[*Function*]
tail?
[*Function*]
circular-list?

18.3 Association Lists

[*Function*]
acons
[*Function*]
pairlis
[*Method*]
assoc
[*Method*]
assoc-if
[*Method*]
rassoc
[*Method*]
rassoc-if
[*Function*]
copy-alist

18.4 Mapping

[*Function*]
maplist *function list &rest more-lists*

Chapter 19

Sets

[*Function*]

make-set

[*Function*]

adjoin

[*Function*]

less

[*Function*]

empty-set

[*Function*]

subset?

[*Function*]

disjoint?

Chapter 20

Hash Tables

Hash tables in Koioslisp, like in Haskell and Clojure, can be operated similar to linked lists, specifically, association lists; as opposed to the support in COMMON LISP, where most of the functions are destructive. Unlike in COMMON LISP, hash tables in KOIOSLISP, hash tables have standard, **readable**, printed representations, borrowed from RACKET and ARC. For example, a plausible hash-table is provided

```
#hash((vladimir . estragon) (rosencrantz . guildenstern))
```

In this hash-table, the *symbol* *vladimir* is mapped to the symbol *estragon*, and the symbol *rosencrantz* is mapped to the symbol *guildenstern*. If the hash table had an equality test of *eq?*, then it would be notated as

```
#hash(:eq? (vladimir . estragon)(rosencrantz . guildenstern)) [Function]
```

```
make-hash-table &key initial-contents
```

```
[Function]
```

```
empty-hash
```

```
[Function]
```

```
hash-table? object
```

```
[Method]
```

```
assoc item table &key key test
```

```
[Method]
```

```
assoc-if predicate table &key key
```

```
[Method]
```

```
rassoc item table &key key test test-not
```

```
[Method]
```

```
rassoc-if predicate table &key key
```

```
[Method]
```

```
pair keys values &optional table
```

```
[Method]
```

```
remove key table
```

```
[Method]
```

```
remove! key table
```

```
[Method]
```

```
reduce
```

```
[Function]
```

```
h-cons key value table
```

h-cons takes a key, a value, and a hash table (respectively) as its arguments, and returns a new hash-table with the contents of *table* and a new association between *key* and *value*. If *key* is already contained (as in, the keys are =) as a key in *table*, the table returned will use the value *value* instead of the value mapped to *key* in *table*.

(*h-cons* 'foo 'bar baz) is equivalent to (pair 'foo 'bar baz)

[Function]

h-append *table1 table2 &key test*

h-append is like **h-cons**, except that it takes two hash-tables as arguments instead of a key, a value, and a hash-table. It returns a new hash-table with the contents of both the first table and the second table. If the second hash-table contains an entry with the same (=) key as the first hash-table, then the entry from the second hash-table is used. If the optional parameter *new-table-test* (which may be the values EQ or =) is supplied, the hash-table returned has membership specified by the given parameter as a test.

[Function]

h-push! *key value table*

h-push is the destructive counterpart to **h-cons**, modifying *table* as a result. (**h-push** 'foo 'bar baz) is equivalent to (zap #'(λ (x) (h-cons 'foo 'bar x)) baz).

[Function]

h-append! *table1 table 2*

h-append! is the destructive counterpart to **h-append**, modifying *table1* as a result. (**h-append!** foo baz) is equivalent to (zap #'(λ (x) (h-append x bar)) foo), unless *table1* uses a different test than the default (such as =, which will be used for this example), in which case it would be (zap #'(λ (x) (h-append x bar :test '=)) foo)

[Method]

map-keys *result-class function hash-table &rest more-sequences*

This is the method of map specialized for *hash-tables*. It behaves like the rest of the methods of *map*, except that the function is called on the set of keys in the hash-table rather than both the keys and the values.

[Function]

maphash *function hash-table*

The *maphash* function maps over every entry of the hash-table *hash-table*; calling the function *function* with the arguments being the key and the value of the entry; then it returns a list of the results of the evaluation of the function over the entries. In this it is incompatible with COMMON LISP's *maphash* function, which unconditionally returns nil. If the entries in *hash-table* are destructively modified by *function*, the results are unpredictable.

[Macro]

with-hash-table-iterator

[Method]

member

[Method]

member-if

[Method]

sort *table*

[Method]

sort! *table*

[Method]

clear

[Function]

sxhash *object*

[Method]

get-in *table &rest keys*

[Function]

keys *table*

[Function]

hash-values *table*

[Method]

length

[Function]

merge-with

Chapter 21

Arrays

[*Function*]
make-array
[*Constant*]
+array-rank-limit+
[*Constant*]
+array-dimension-limit+
[*Constant*]
+array-total-size-limit+
[*Function*]
aref
[*Function*]
array-element-type
[*Function*]
array-rank
[*Function*]
array-dimension
[*Function*]
array-dimensions
[*Function*]
array-total-size
[*Function*]
array-in-bounds-p
[*Function*]
array-row-major-index
[*Function*]
row-major-aref
[*Function*]
adjustable-array-p

21.1 Vectors

What Common Lisp called `vector-push-extend` has been rechristened `vector-push`; Common Lisp's `vector-push` (a less-useful case of `vector-push-extend`) has been removed from the language.

Chapter 22

Conclists

In KOIOSLISP, conclists are designed to potentially produce particularly efficient parallel code. They are modelled, in part, after their use in the Fortress programming language.

```
[Function]  
concllist  
[Function]  
concllist?  
[Function]  
conc  
[Method]  
empty?  
[Function]  
singleton?  
[Function]  
item  
[Function]  
left  
[Function]  
right  
[Function]  
rebalance  
[Function]  
add-left  
[Function]  
add-right  
[Function]  
split  
[Function]  
divide-and-conc
```


Chapter 23

Strings

A string is a vector of characters. Used as a type specifier, `string` is equivalent to `(vector character)` and `(array character (*))`.

[Function]

`str &rest args`

`str` is in the spirit of the Clojure function of the same name: it takes an arbitrary number of arguments, and concatenates the printed forms of them into a single string, which it then returns.

23.1 Comparison of Strings

[Method]

`equal (a string) (b string) &rest keys &key :recursive (:case-sensitive t) :start1 :end1 :start2 :end2 &allow-other-keys`

`equal` behaves like `string=` if the key `:case-sensitive` is not `nil`, which is the default. Otherwise, it behaves like `string-equal`. It accepts

[Function]

`string= string1 string2 &key :start1 :end1 :start2 :end2`

`string=`

[Function]

`string-equal`

[Function]

`string<`

[Function]

`string>`

[Function]

`string<=`

[Function]

`string>=`

[Function]

`string/ =`

[Function]

`string-less?`

[Function]

`string-greater?`

[Function]

`string-not-less?`

[*Function*]
string-not-greater?
[*Function*]
string-not-equal?

23.2 Regular Expressions

Koioslisp supports the use of `string-not-greater?` for quick-and-dirty parsing, the API being inspired by Edi Weitz’s excellent CL-PPCRE[5] [*Function*]

`splitre`
[*Variable*]
`*property-resolver*`

23.2.1 Regular Expression Syntax

What are here called “regular expressions” can in reality parse more than what traditional regular expressions can; they are simply called regular expressions for historical reasons. A regular expression literal begins with the characters “#” and ends with the character “”. There are a few variations:

To treat the string matched as multiple lines—that is, to change the behavior of “^” and “\$” to match the beginning and ends of lines within strings as well as the beginning and ending of the string—one begins the regular expression with “#m” and ends with “”.

To allow “.” to match newlines, the regular expression begins with “#s” and ends with “”.

A *case-insensitive* regular expression begins with “#i” and ends with “”.

These variations, known as modifiers, can be combined: e.g. a regular expression beginning with “#msi” and ending with “” would use all of them.

Within the regular expression, the syntax follows that of Perl-compatible regular expressions.

Regular Expression parse trees

Regular expressions can also be specified using *regular expression parse trees* (*REPT*) a la CL-PPCRE[5] (and indeed much of the syntax is directly borrowed from it). In a REPT, none of the regular expression metacharacters are treated specially—they are all interpreted as part of the tree. The syntax for the trees is:

1. The keyword `:null-string` is equivalent to the null string.
2. The keyword `:match-all` is equivalent to the dot in regex syntax.
3. The keyword `:word-boundary` is equivalent to “\b” and the keyword `:not-word-boundary` is equivalent to “\B” in regex syntax.
4. Corresponding to Perl’s *character classes* are the keywords `:digit-char` (“\d”), `:non-digit-char` (“\D”), `:word-char` (“\w”), `:non-word-char` (“\W”), `:whitespace-char` (“\s”), and `:non-whitespace-char` (“\S”).
5. The following keywords correspond to anchors, that is: `:start-anchor` is equivalent to “^”, `:end-anchor` is equivalent to “\$”, `:modeless-start-anchor` to “\A”, `:modeless-end-anchor` to “\Z”, and `:modeless-end-anchor-no-newline` to “\Z”.
6. The following keywords correspond to embedded modifiers (as such, they are kept local to the most local segment of the parse tree they are introduced in): `:case-insensitive` (“(?i)”), `:case-sensitive` (“(?-i)”), `:multi-line-mode` (“(?m)”), `:not-multi-line-mode` (“(?-m)”), `:single-line-mode` (“(?s)”), and `:not-single-line-mode` (“(?-s)”).

7. `(:sequence parse-trees*)` matches its arguments in sequence. For example, the rept `(:sequence #\q #\u #\u #\x)` is equivalent to `#R"quux"`
8. `(:alternation parse-trees*)` matches any one of its arguments. Thus, `(:alternation "foo" "baz" "bar" "quux")` is equivalent to `#R"foo|baz|bar|quux"`.
9. `(:group parse-trees*)` act like Perl's `"(?:pattern)"` construct
10. `(:branch test rept1 rept2)` is treated as the `(?(condition)yes-pattern|no-pattern)` is in Perl
11. The constructs `(:positive-lookahead rept)`, `(:negative-lookahead rept)`, `(:positive-lookbehind rept)`, and `(:negative-lookbehind rept)` do exactly what they say.
12. `(:register rept)` denotes a register group; they are counted from left to right, indexed at 1 (unfortunately, this is necessary in order for compatibility with CL-PPCRE, and thus Perl)
13. `(:named-register name rept)` denotes a register, but also assigns it a name *name* (a string), which is case-sensitive, not necessarily unique, and can be referred to using `:back-reference`
14. `(:back-reference reference)` is a reference to a register, which may be either a number, or, if it is a named register, a string
15. `(:greedy-repetition min max rept)`, where *min* is a non-negative integer and *max* is either nil or an integer larger than or equal to *min* will try to match *rept* at least *min* times and, if *max* is an integer, at most *max* times; if *max* is nil, then it will attempt to match *rept* as much as possible. `(:non-greedy-repetition min max rept)` works similarly, but non-greedily.
16. `(:property property)` is a named regular expression property (or in the case of `:inverted-property`, its inverse) with *property* being a function designator—see `*property-resolver*`
17. `(:standalone rept)` is an independent expression, e.g. `(:standalone "foo")` is the same as `#R"(?>bar)"`.
18. `(:filter function &optional length)` specifies a filter (see “Regular Expression Filters”), where *function* is a function of one argument and *length* is either a non-negative integer or nil
19. `(:regex regex)`, where *regex* is a regular expression in the regular syntax.
20. `(:char-class item*)` is a character class (or, in the case of `:inverted-char-class`, its inversion) where *item* is either a character, a named property (see above), or a character range, which is `(:range char1 char2)`, where `(char<= char1 char2)` is true.

23.2.2 Regular Expression Filters

Filters are as in CL-PPCRE[5]:user-defined functions usable in regular-expression parse trees. The function accepts one argument, which is an integer *pos* that represents the current position within the string being matched.

[*Variable*]

matching-string The target string of the current matching

[*Variable*]

start-pos The start index (an integer) of the current matching, corresponding to the `start` argument in `scan`. In repeated scans, `*start-pos*` will move forwards; there `*initial-start-pos*` may be more useful.

[*Variable*]

end-pos The end index (an integer) of the current matching, corresponding to the `end` argument in `scan`

[*Variable*]

initial-start-pos This is like `*start-pos*`, but in repeated scans, it will remain at the same position

[*Variable*]

reg-starts A vector whose contents are the beginning indices of the corresponding registers in the current

regular expression. The “default” value of an index is nil.

[*Variable*]

reg-endsA vector whose contents are the ending indices of the corresponding registers in the current regular expression. The “default” value of an index is nil.

Chapter 24

The Evaluator

Chapter 25

Lazy Evaluation

Lazy evaluation is a method for efficiently manipulating large (even theoretically infinite) data structures. Though usually evaluation is eager, Koioslisp provides support for lazy evaluation with explicit annotation. The Koioslisp lazy evaluation facility is partially based on the CLAZY library for Common Lisp, and the Clojure lazy evaluation facility. See also the section on List Comprehension.

[*Macro*]
deflazy
[*Function*]
lazycall
[*Macro*]
lazy
[*Macro*]
lazylambda
[*Macro*]
lazylet
[*Function*]
force

25.1 Series

Series are sequences, and thus all sequence generic functions should “work” on them, in manners defined here. Generally they are to be defined lazily; that is, to generate a result, one must **take** (or **take-all**) elements from them. [*Method*]

take

Unlike **take-all**, **take** is useful for large sequences in general—thus it has been promoted to a sequence generic function.

[*Function*]

take-all

[*Function*]

drop

[*Function*]

head

[*Function*]

tail

Chapter 26

Continuations

Like Scheme and Qi, KOIOSLISP provides continuations, which are mechanisms for abstract control over the flow of execution of a program. Unlike Scheme or Qi, these are *delimited continuations*

[*Function*]

`call/cc function`

The argument *function* is a function which takes one parameter. `call/cc` can only be called within a `with-call/cc` form or an implicit `with-call/cc` form. `call/cc` has the alias `call-with-current-continuation`

[*Macro*]

`with-call/cc`

[*Macro*]

`let/cc`

[*Macro*]

`lambda/cc`

[*Macro*]

`defun/cc`

[*Macro*]

`without-call/cc`

[*Macro*]

`defmethod/cc`

[*Macro*]

`defgeneric/cc`

Chapter 27

Logic Programming

The KOIOSLISP logic programming system is derived from Dorai Sitaram's excellent Schelog and Racklog.

[*Function*]

logic-var?

[*Macro*]

which

[*Macro*]

find-all

[*Macro*]

rel

[*Macro*]

assert-rel!

[*Macro*]

assert-rel!*

[*Macro*]

cut-delimiter

[*Cut syntax*]

! ! is used to denote the cut within cut-delimiter or rel

[*Function*]

unifiable?

27.1 Constraints

Chapter 28

Sockets

Chapter 29

Streams

Chapter 30

Objects

The Koioslisp Object System is based on CLOS (Common Lisp Object System) and TELOS (The EuLisp Object System), and provides reasonably better integration with the rest of Koioslisp than CLOS does with Common Lisp.

30.1 Classes

30.1.1 Defining Classes

30.1.2 Instance Creation

30.1.3 Slots

30.1.4 Accessing Slots

30.1.5 Inheritance

Inheritance of Methods

Inheritance of Slots

Examples

30.1.6 Built-in Classes

30.1.7 Determining the Class Precedence List

Sorting the Class Precedence Lists

Examples

30.2 Generic Functions and Methods

30.2.1 Introduction

30.2.2 Agreement on Parameter Specializers and Qualifiers

30.2.3 Keyword Arguments

30.2.4 Method Selection

30.2.5 Determinating the Effective Method

30.2.6 Standard Method Combination

30.2.7 Other Method Combinations

30.2.8 Defining Method Combinations

30.2.9 SET methods

30.3 Structures

30.3.1 Defstruct Options

30.3.2 BOA Constructors

30.4 The Meta-Object Protocol

This is based on the Common Lisp Meta-Object Protocol, described in Kiczales 1991[3] [*Generic Function*]
add-dependent

[*Generic Function*]

add-direct-method

[*Generic Function*]

add-direct-subclass
[*Generic Function*]
add-method

Chapter 31

Input/Output

```
(defvar *country* ``Eurasia'')
(format t ``Oceania has always been at war with ~A' ' *country*)
;→ ``Oceania has always been at war with Eurasia''
```

31.1 Readtables

KOIOSLISP adopts the mechanism of *named readtables*. There are two preregistered readtables: `:koioslisp` and `:current`: they refer to standard KOIOSLISP syntax and the current readtable respectively. This readtable API was inspired by the paper “EDITOR-HINTS.NAMED-READTABLES - 0.9” by Tobias C. Rittweiler. [*Macro*]

defreadtable *name* &body *options*

defreadtable, as indicated by its name, defines a new readtable with its name the symbol *name*, unless *name* is already used for another readtable, in which case it is redefined. **defreadtable** takes several options:

(`:merge readtables+`)

(`:merge-overwriting readtables+`)

(`:dispatch-macro-symbol d-macro-symbol function`)

(`:macro-char macro-char function &optional non-terminating-p`)

(`:syntax-from from-readtable from-char to-char`)

[*Function*]

copy-readtable

[*Function*]

ensure-readtable

[*Function*]

find-readtable *name*

Looks for a readtable of name *name* (which is of type (`or readtable symbol`)), and returns it if it is found, or nil otherwise.

[*Macro*]

in-readtable *name*

The code following a call to **in-readtable** are to be treated as if they follow the syntax defined by readtable *name*.

[*Function*]

merge-readtables-into *result* &rest *readtables*

The contents of *readtables* are copied into *result*. If a conflict is discovered, an error of type `reader-macro-conflict` is signaled.

Chapter 32

File System Interface

Often, programs are packaged in *files*. Because of their use, KOIOSLISP provides a standard method for interacting with a filesystem. The system is designed with its basis on the Common Lisp filesystem interface because of its generality and ability to handle most, if not all, forms of filesystems. These are useful features because of the historical and current variation in file systems and their conventions in representing files. As an example of such variation, a table of different methods to represent a certain file on different filesystems important (both currently and historically) in the Lisp world is given:

System	File name
TOPS-20	<code><LISP10>FORMAT.FASL.13</code>
TOPS-10	<code>FORMAT.FAS[1,4]</code>
ITS	<code>LISP10;FORMAT FASL</code>
MULTICS	<code>>udd>Lisp10>format.fasl</code>
TENEX	<code><LISP10>FORMAT.FASL;13</code>
VAX/VMS	<code>[LISP10]FORMAT.FAS;13</code>
Unix, GNU, *BSD, Plan 9, Mac OS X, etc.	<code>/usr/Lisp10/format.fasl</code>
MS-DOS, Windows, ReactOS, etc.	<code>C:\Program Files\Lisp10\format.fasl</code>

This large proliferation of differing filesystems makes optimizing to one specific type in the core language rather arbitrary and shortsighted. Thus, Koioslisp uses a very general method for dealing with filesystems; namely, adapting the Common Lisp system—providing *namestrings* (which are strings in the customary file system-representing form) and *pathnames* (which are abstract methods of dealing with files) that work in a very similar manner to their predecessors.

32.1 File Names

32.1.1 Pathnames

Filesystems in Koioslisp are seen through the framework of the *pathname* data object, with its components defined below. This is used in order to enhance portability over file systems.

host

A *host* is the name of the file system which contains the file

device

The “file structure”, or “device” component of file systems

directory

The “directory” component of most file systems; a structure containing a group of possibly related files

name

The label associated with a file

type The “extension” or “filetype” of a file system—quite literally the type of file, which determines which programs are to handle it.

version

An increasingly rare component; a number which represents which revision the file is in, and how many revisions it has undergone

32.1.2 Case Conventions

32.1.3 Structured Directories

32.1.4 Extended Wildcards and Globbing

32.1.5 Logical Pathnames

Syntax of Logical Pathnames

Parsing of Logical Pathname Namestrings

Using Logical Pathnames

Logical Pathname Examples

32.1.6 Pathname Functions

32.2 File and Directory Operations

32.2.1 File I/O

[*Function*]

read-file-into-string

[*Function*]

rewind

Loading files

32.2.2 Renaming, Deleting, and Other File Operations

32.2.3 Accessing Directories

Chapter 33

Errors

Chapter 34

Advice

[*Macro*]

`defadvice` (*function advice-name time &key where documentation*) (*lambda-list*) &body *body*

Chapter 35

Miscellaneous Features

35.1 The Compiler

[*Function*]
compile
[*Function*]
compile-file
[*Variable*]
compile-verbose
[*Variable*]
compile-print
[*Variable*]
compile-file-pathname
[*Variable*]
compile-file-truename
[*Special form*]
load-time-value
[*Function*]
disassemble
[*Function*]
function-lambda-expression
[*Macro*]
with-compilation-unit

35.1.1 Compiler Diagnostics

35.1.2 Compiled Functions

35.1.3 Compilation Environment

35.1.4 Similarity of Constants

35.2 Documentation

[*Generic Function*]
documentation
[*Generic Function*]

apropos

35.3 Debugging Tools

[*Macro*]
trace
[*Macro*]
untrace
[*Macro*]
step
[*Macro*]
time
[*Function*]
describe
[*Generic Function*]
describe-object
[*Method*]
describe-object
[*Function*]
inspect
[*Function*]
room
[*Function*]
ed
[*Function*]
dribble

35.4 Environment

Here there is significant influence from Erik Naggum's paper "A Long, Painful History of Time "

[*Class*]
<local-time>
Superclasses: <standard-object>
Slots:
[*Function*]
make-local-time &key *universal internal unix* (msec 0) (zone 0)
[*Function*]
local-time-day
[*Function*]
local-time-sec
[*Function*]
local-time-msec
[*Function*]
local-time-zone
[*Function*]
local-time<
[*Function*]
local-time<=
[*Function*]


```

local-time>
[Function]
local-time>=
[Function]
local-time=
[Function]
local-time/=
[Function]
local-time-adjust source timezone &optional destination
local-time-adjust returns the values of the adjusted slots unless destination is provided as a local-time
instance, in which case it will first adjust the slots and then return the destination local-time instance
[Function]
get-local-time
[Function]
get-french-time
This function returns the current time in the French Republican Calendar. The values it returns are: the
year, the month, the day of the month, the hour, minute, and second. The current year is the number
of years since the establishment of the calendar (in the Gregorian Calendar, 22 September 1792 C.E.; the
New Year is thus set at 22 September in the KOIOSLISP implementation, though in actual usage it varied
depending on the Autumn Equinox). The month is one of the following values: “Vendémiaire”, “Brumaire”,
“Frimaire”, “Nivôse”, “Pluviôse”, “Ventôse”, “Germinal”, “Floréal”, “Prairial”, “Messidor”, “Thermidor”,
“Fructidor”, or “Fêtes”; each month is 30 days long, except for the Fêtes, which are 5 days long on normal
years, 6 on leap years. A leap year occurs with the same rule as in the Gregorian calendar, but the origin
date is the year 20.
[Function]
get-formatted-french-time &optional timestamp
This function returns a formatted string representing the current French Republican Calendar time, or, if
supplied a timestamp argument, formats the time represented by the timestamp. The format is specified as:
time ::= [[date-of-month month | fête ]]An year de la Révolution, hour:minute:second
[Function]
timestamp->french
[Function]
sleep seconds
[Function]
syscall
[Function]
lisp-implementation-type
Examples: “Seattle Koioslisp”, “Alonzo”.
[Function]
lisp-implementation-version
[Function]
os-type
Returns a list of symbols which identify the operating system upon which Koioslisp runs—a list is chosen,
of course, because some OSes implement a larger standard, or are part of a family that can be treated as a
single unit. Some are provided here as standards for common systems:
:gnu The GNU system, often erroneously called Linux when paired with said kernel. Matches GNU/Linux
systems, GNU/kFreeBSD systems, GNU/HURD systems, and any other GNU systems.
:bsd The BSD family of operating systems.
:posix The POSIX standard for operating systems.
:loper The Loper Operating System
:plan-9 The Plan 9 Operating System
:os-x The Mac OS X operating system

```

:windows The Microsoft Windows operating system, and the React Operating System

[*Function*]

os-version

[*Function*]

software-type

[*Variable*]

features

[*Function*]

identity *object*

Chapter 36

Pretty-Printing

Chapter 37

Concurrency

[Function]
spawn
[Macro]
recieve
[Recieve option]
after
[Function]
send

[Function]
exit-proc
[Function]
spawn-remotely
[Function]
atomic
[Atomic anaphor]
retry
[Macro]
or-else &rest *args*

37.1 Refs

A *ref* is like a Concurrent Haskell *TVar*; it is used for mutable shared state in concurrent programs[1].

[Function]
ref
[Macro]
defref
(defref *x y*) is syntactic sugar for (defvar *x* (ref *y*))
[Macro]
getref

37.2 The Dining Philosophers Problem

```
(defclass <chopstick> ()
  ((owner :accessor chopstick-owner :initform (ref nil))))
(defvar *chopsticks*
  '((make-instance '<chopstick>)
    (make-instance '<chopstick>)
    (make-instance '<chopstick>)
    (make-instance '<chopstick>)
    (make-instance '<chopstick>)
    (make-instance '<chopstick>)
    (make-instance '<chopstick>)
    (make-instance '<chopstick>)))
(defclass <philosopher> (<thread>)
  ((name :accessor name-of :initarg name)
   (meals-eaten :accessor phil-meals :initform (ref 0))
   (left :accessor phil-left :initarg left)
   (right :accessor phil-right :initarg right)))
(defvar *philosophers*
  '((make-instance '<philosopher> :name 'Chomsky :left (nth *chopsticks* 0) :right (nth *chopsticks* 1))
    (make-instance '<philosopher> :name 'Kongfuzi :left (nth *chopsticks* 1) :right (nth *chopsticks* 2))
    (make-instance '<philosopher> :name 'Nietzsche :left (nth *chopsticks* 2) :right (nth *chopsticks* 3))
    (make-instance '<philosopher> :name 'Marx :left (nth *chopsticks* 3) :right (nth *chopsticks* 4))
    (make-instance '<philosopher> :name 'Spinoza :left (nth *chopsticks* 4) :right (nth *chopsticks* 5))
    (make-instance '<philosopher> :name 'Wittgenstein :left (nth *chopsticks* 5) :right (nth *chopsticks* 6))
    (make-instance '<philosopher> :name 'de Beauvoir :left (nth *chopsticks* 6) :right (nth *chopsticks* 7))
    (make-instance '<philosopher> :name 'Dostoyevsky :left (nth *chopsticks* 7) :right (nth *chopsticks* 8))
    (make-instance '<philosopher> :name 'de Saussure :left (nth *chopsticks* 8) :right (nth *chopsticks* 0))))
(defmethod run ((thread <philosopher>))
  (until (=? (phil-meals philosopher) 30)
    (atomic
      (if (not
          (and (null? (chopstick-owner (phil-left philosopher)))
              (null? (chopstick-owner (phil-right philosopher)))))
        (retry)
        (begin
          (set! (get-ref (chopstick-owner (phil-left philosopher))) (name-of philosopher))
          (set! (get-ref (chopstick-owner (phil-right philosopher))) (name-of philosopher))
          (format t "~A has acquired two chopsticks" (name-of philosopher))))))
    (atomic
      (inc (get-ref (phil-meals philosopher)))
      (format t "~A has eaten a tasty bite of rice" (name-of philosopher))
      (set! (get-ref (chopstick-owner (phil-left philosopher))) nil)
      (set! (get-ref (chopstick-owner (phil-right philosopher))) nil)
      (format t "~A has returned to thinking" (name-of philosopher))))
    (format t "~A has left the table" (name-of philosopher)))
  (mapc #'spawn *philosophers*))
```

Chapter 38

Backquote

Chapter 39

Implementation Notes

Some notes as to how to implement this are located here, however, the notes speak from a high-level, design perspective. KOIOSLISP's syntax and semantics have been designed in such a way that if one has a modern, working implementation of COMMON LISP (such as Steel Bank Common Lisp (SBCL)), one can adapt it with little effort to implement much of the KOIOSLISP standard library. An interesting perspective on implementation, proposed by Richard P. Gabriel in his paper "Lisp: Good News, Bad News, How to Win Big", and followed by the experimental Lisp dialect Shen, is to create a kernel language small enough to implement easily, yet powerful enough to express the language—and implement the rest of the language as a standard library for the kernel (Shen's kernel language is called Kernel Lisp, or 'KL', which, like Koioslisp, has used '.kl' as an extension. Koioslisp is not compatible with Kernel Lisp, and has no relation to it whatsoever). Another solution would be to implement it as an embedded language within Common Lisp. Whatever the solution turns out to be, we hope that these suggestions will prompt quicker implementation of KOIOSLISP.

Chapter 40

KLIM

The standard mechanism for windowing is the KOIOSLISP INTERFACE MANAGER, or KLIM. It is roughly based around the Common Lisp Interface Manager, which itself is descended from the GUI provided in the Genera operating system in Symbolics Lisp Machines. Like the rest of Koioslisp, it should be portable, which in this context means it should be supportable across window environments.

[*Function*]
defapplication-frame
[*Function*]
defapplication-pane
[*Function*]
draw-point
[*Function*]
draw-point*
[*Function*]
draw-points
[*Function*]
draw-points*
[*Function*]
draw-arrow
[*Function*]
draw-arrow*
[*Function*]
draw-arrows*
[*Function*]
region-intersection
[*Function*]
region-difference
[*Function*]
draw-text
[*Function*]
draw-text*
[*Function*]
draw-line
[*Function*]
draw-line*
[*Function*]
draw-lines
[*Function*]

draw-lines*
[Function]
draw-polygon
[Function]
draw-polygon*
[Function]
draw-rectangle
[Function]
draw-rectangle*
[Function]
draw-rectangles
[Function]
draw-rectangles*
[Function]
draw-ellipse
[Function]
draw-ellipse*
[Function]
draw-circle
[Function]
draw-circle*
[Function]
draw-oval
[Function]
draw-oval*
[Function]
draw-bezier-curve
[Function]
draw-bezier-curve*
[Generic Function]
medium-draw-point*
[Generic Function]
medium-draw-points*
[Generic Function]
medium-draw-line*
[Generic Function]
medium-draw-lines*
[Generic Function]
medium-draw-rectangle*
[Generic Function]
medium-draw-rectangles*
[Generic Function]
medium-copy-area
[Generic Function]
medium-draw-polygon*
[Generic Function]
medium-draw-ellipse*
[Generic Function]
medium-draw-text*
[Function]
copy-to-pixmap
[Function]

copy-from-pixmap
 [Function]
 draw-pixmap*
 [Function]
 draw-pixmap
 [Generic Function]
 copy-area
 [Function]
 allocate-pixmap
 [Function]
 deallocate-pixmap
 [Generic Function]
 pixmap-width
 [Generic Function]
 pixmap-height
 [Generic Function]
 pixmap-depth
 [Macro]
 with-output-to-pixmap
 [Class]
 <region>
 Superclasses: <standard-object>
 Slots:
 [Class]
 <point>
 Superclasses: <standard-object>
 Slots:
 [Class]
 <path>
 Superclasses: <region>
 Slots:
 [Class]
 <area>
 Superclasses: <region>
 Slots:
 [Constant]
 +everywhere+
 [Constant]
 +nowhere+
 [Generic Function]
 region-equal
 [Generic Function]
 region-contains-region-p
 [Generic Function]
 region-contains-position-p
 [Generic Function]
 region-intersects-region-p
 [Method]
 union
 [Method]
 intersection
 [Method]

```

difference
[Class]
<region-set>
Superclasses: <region>
Slots:
[Generic Function]
region-set-function
[Generic Function]
region-set-regions
[Generic Function]
map-regions
[Class]
<point>
Superclasses: <standard-object>
Slots:
[Class]
<standard-point>
Superclasses: <standard-object>
Slots:
[Function]
make-point
[Generic Function]
point-position
[Generic Function]
point-x
[Generic Function]
point-y
[Class]
<polyline>
Superclasses: <path>
Slots:
[Class]
<polygon>
Superclasses: <area>
Slots:
[Class]
<standard-polyline>
Superclasses: <polyline>
Slots:
[Class]
<standard-polygon>
Superclasses: <polygon>
Slots:
[Function]
make-polygon
[Function]
make-polygon*
[Function]
make-polyline
[Function]
make-polyline*
[Generic Function]

```

```

polyline-closed-p
[Generic Function]
map-polygon-coordinates
[Generic Function]
map-polygon-segments
[Class]
<line>
Superclasses: <polyline>
Slots:
[Class]
<standard-line>
Superclasses: <line>
Slots:
[Function]
make-line
[Function]
make-line*
[Generic Function]
line-start-point
[Generic Function]
line-start-point*
[Generic Function]
line-end-point
[Generic Function]
line-end-point*
[Class]
<rectangle>
Superclasses: <polygon>
Slots:
[Class]
<standard-rectangle>
Superclasses: <rectangle>
Slots:
[Generic Function]
rectangle-min-point
[Generic Function]
rectangle-max-point
[Function]
rectangle-min-x
[Function]
rectangle-min-y
[Function]
rectangle-width
[Function]
rectangle-height
[Function]
rectangle-size
[Class]
<bounding-rectangle>
Superclasses: <standard-object>
Slots:
[Class]

```

```

<standard-bounding-rectangle>
Superclasses: <bounding-rectangle>, <rectangle>
Slots:
[Function]
make-bounding-rectangle
[Generic Function]
bounding-rectangle*
[Generic Function]
bounding-rectangle
[Function]
bounding-rectangle-min-x
[Function]
bounding-rectangle-min-y
[Function]
bounding-rectangle-top
[Function]
bounding-rectangle-max-x
[Function]
bounding-rectangle-max-y
[Function]
bounding-rectangle-bottom
[Function]
bounding-rectangle-size
[Function]
bounding-rectangle-width
[Function]
bounding-rectangle-height
[Class]
<ellipse>
Superclasses: <area>
Slots:
[Class]
<elliptical-arc>
Superclasses: <path>
Slots:
[Class]
<standard-ellipse>
Superclasses: <ellipse>
Slots:
[Class]
<standard-elliptical-arc>
Superclasses: <elliptical-arc>
Slots:
[Function]
make-ellipse
[Function]
make-ellipse*
[Function]
make-elliptical-arc
[Function]
make-elliptical-arc*
[Function]

```



```

ellipse-center-point
[Function]
ellipse-center-point*
[Function]
ellipse-radii
[Function]
ellipse-start-angle
[Function]
ellipse-end-angle
[Generic Function]
medium-foreground medium
[Generic Function]
medium-background medium
[Generic Function]
medium-ink medium
[Generic Function]
medium-transformation medium
[Generic Function]
medium-clipping-region medium
[Generic Function]
medium-line-style medium
[Generic Function]
medium-text-style medium
[Generic Function]
medium-default-text-style medium
[Macro]
with-drawing-options(medium &key ink clipping-region transformation line-style line-unit line-thickness
line-dashes line-joint-shape line-cap-shape text-style text-family text-face text-size) &body body
[Generic Function]
invoke-with-drawing-options stream function &key ink clipping-region transformation line-style line-unit
line-thickness line-dashes line-joint-shape line-cap-shape text-style text-family text-face text-size
[Drawing option]
:clipping-region
[Drawing option]
:ink
[Drawing option]
:transformation
[Drawing option]
:text-style
[Drawing option]
:line-style
[Class]
<line-style>
Superclasses: <standard-object>
Slots:
[Function]
make-line-style &key (unit :normal) (thickness 1) dashes (joint-shape :miter) (cap-shape :butt)
[Generic Function]
line-style-thickness line-style
[Generic Function]
line-style-joint-shape line-style
[Generic Function]

```

line-style-cap-shape*line-style*
 [Generic Function]
line-style-dashes*line-style*
 [Generic Function]
line-style-unit*line-style*
 [Drawing option]
:line-unit
 [Suboption]
:normal
 [Suboption]
:point
 [Drawing option]
:line-thickness
 [Drawing option]
:line-dashes
 [Drawing option]
:line-joint-shape
 [Drawing option]
:line-cap-shape
 [Function]
make-contrasting-dash-patterns *n* &optional *k*
 [Function]
contrasting-dash-patterns-limit
 [Function]
make-translation-transformation δ *x* δ *y*
 [Function]
make-rotation-transformation θ &optional *origin*
 [Function]
make-rotation-transformation**angle origin-x origin-y*
 [Function]
make-scaling-transformation*mx my* &optional *origin*
 [Function]
make-reflection-transformation *point-1 point-2*
 [Function]
make-reflection-transformation* *x1 y1 x2 y2*
 [Function]
make-transformation *mxx mxy myx myy tx ty*
 [Function]
make-3-point-transformation *point-1 point-2 point-3 point-1-image point-2-image point-3-image*
 [Function]
make-3-point-transformation* *x1 y1 x2 y2 x3 y3 x1-image y1-image x2-image y2-image x3-image y3-image*
 [Class]
 <transformation>
 Superclasses: <standard-object>
 Slots:
 [Constant]
+identity-transformation+
 [Method]
 = *transform1 transform2*
 [Generic Function]
identity-transformation-p *transform*
 [Generic Function]

translation-transformation-p *transform*
[Generic Function]
invertible-transformation-p *transform*
[Generic Function]
reflection-transformation-p *transform*
[Generic Function]
rigid-transformation-p *transform*
[Generic Function]
even-scaling-transformation-p *transform*
[Generic Function]
scaling-transformation-p *transform*
[Generic Function]
rectilinear-transformation-p *transform*
[Generic Function]
compose-transformations &rest *transformations*
[Generic Function]
compose-translation-with-transformation *transform dx dy*
[Generic Function]
compose-scaling-with-transformation *transform mx my* &optional *origin*
[Generic Function]
compose-rotation-with-transformation *transform angle* &optional *origin*
[Generic Function]
compose-transformation-with-translation *transform dx dy*
[Generic Function]
compose-transformation-with-scaling *transform mx my* &optional *origin*
[Generic Function]
compose-transformation-with-rotation *transform angle* &optional *origin*
[Generic Function]
invert-transformation *transform*
[Macro]
with-rotation (*medium angle* &optional *origin*) &body *body*
[Macro]
with-translation (*medium dx dy*) &body *body*
[Macro]
with-scaling (*medium sx* &optional *sy*) &body *body*
[Macro]
with-room-for-graphics(&optional *stream* &key *height* (first-quadrant t) (move-cursor t) *record-type*) &body
body
[Macro]
with-local-coordinates(&optional *stream x y*) &body *body*
[Macro]
with-first-quadrant-coordinates (&optional *stream x y*) &body *body*
[Generic Function]
transform-region *transformation region*
[Generic Function]
untransform-region *transformation region*
[Generic Function]
transform-position *transform x y*
[Generic Function]
untransform-position *transform x y*
[Generic Function]
transform-distance *transform dx dy*

[*Generic Function*]
 untransform-distance *transform dx dy*
 [*Generic Function*]
 transform-rectangle* *transform x1 y1 x2 y2*
 [*Generic Function*]
 untransform-rectangle* *transform x1 y1 x2 y2*

40.1 Text Styles

family

KLIM allows one to include font families installed on a system (or perhaps packaged with the software) in typographic decisions—they should be specified by a string which corresponds to the name of the file containing the font; what file types that are to be considered fonts are of course implementation dependent (though two font types, TrueType and OpenType, are recommended, if only for the fact that as of this writing they are in common use), e.g. “Helvetica”. In order to maintain a well-defined text style, one is allowed to pass in a *precedence list* of font families to use—if the first font-family cannot be displayed for whatever reason, the next one is tried, and if it can’t be displayed, the next; and so on . KLIM also supports 4 default family values, which are implementation-dependent: `:mono`, `:serif`, `:sans-serif`, and `nil`. An example of a family precedence list is given: (`'Helvetica'` `'TeXGyreHeros'` `'Helvetica Neue'` `'Nimbus Sans L'` `'Akzidenz-Grotesk'` `'Univers'` `'Arial'` `:sans-serif`)

face

A modification of the family. KLIM supports the faces `:roman`, `:bold`, `:italic`, `(:bold :italic)`, or `nil`.

size

The size of the characters. Either it is one of the sizes provided(`:tiny`, `:very-small`, `:small`, `:normal`, `:large`, `:very-large`, `:huge`, `:smaller`, `:larger`), or the point size, or `nil`.

[*Variable*]

default-text-style

[*Text style option*]

:text-family

[*Function*]

text-style-family

[*Text style option*]

:text-face

[*Function*]

text-style-face

[*Text style option*]

:text-size

[*Function*]

text-style-size

[*Generic Function*]

parse-text-style *text-style*

[*Generic Function*]

merge-text-style *style1 style2*

[*Generic Function*]

text-style-components *text-style medium*

[*Generic Function*]

text-style-ascent *text-style medium*

[*Generic Function*]

text-style-descent *text-style medium*

[*Generic Function*]

`text-style-height` *text-style medium*
 [Generic Function]
`text-style-width` *text-style medium*
 [Generic Function]
`text-style-fixed-width-p` *text-style medium*
 [Generic Function]
`text-style-mapping` *port style &optional character-set*
 [Method]
`(setf text-style-mapping)` *mapping port text-style &optional character-set*
 [Function]
`make-text-style` *family face size*
 [Macro]
`with-text-style` (*medium style*) &body *body*
 [Macro]
`with-text-family` (*medium family*) &body *body*
 [Macro]
`with-text-face` (*medium face*) &body *body*
 [Macro]
`with-text-size` (*medium size*) &body *body*

40.2 Color

[Class]
 <palette>
Superclasses: <standard-object>
Slots:
 [Function]
`palette?` *object*
 [Generic Function]
`palette-color?` *palette*
 [Generic Function]
`palette-mutable-p` *palette*
 [Generic Function]
`make-palette` *port*
 [Generic Function]
`frame-palette` *frame*
 [Generic Function]
`frame-manager-palette` *frame-manager*
 [Condition]
`palette-full`
 [Generic Function]
`palette-full-palette` *palette-full-condition*
 [Generic Function]
`palette-full-color` *palette-full-condition*
 [Variable]
`*use-closest-color*`
 [Generic Function]
`find-closest-matching-color` *palette desired-color*
 [Named restart]
`use-other-color`

```

[Generic Function]
cadd-colors-to-palette palette &rest colors
[Generic Function]
remove-colors-from-palette palette &rest colors
[Generic Function]
port-default-palette port
[Function]
make-ihc-color intensity hue saturation
[Function]
make-rgb-color red green blue
[Function]
make-gray-color luminance
[Generic Function]
color-ihc color
[Generic Function]
color-rgb color
[Function]
make-contrasting-inks n &optional k
[Function]
contrasting-inks-limit port
[Class]
<color>
Superclasses: <standard-object>
Slots:
[Class]
<device-color>
Superclasses: <color>
Slots:
[Generic Function]
device-color-pixel device-color
[Generic Function]
device-color-palette device-color
[Generic Function]
device-color-color device-color
[Generic Function]
make-device-color palette pixel
[Function]
convert-rgb-to-ihc red green blue
[Function]
convert-ihc-to-rgb intensity hue saturation
[Generic Function]
make-dynamic-color &key (:color +black+)
[Generic Function]
dynamic-color-color dynamic-color
[Method]
(set dynamic-color-color) color dynamic-color
[Generic Function]
recolor-dynamic-color dynamic-color color
[Macro]
with-delayed-recoloring &body body
[Function]
make-layered-color-set &rest layers

```

```

[Generic Function]
layered-color set &rest layers
[Constant]
+black+
[Constant]
+white+
[Constant]
+red+
[Constant]
+green+
[Constant]
+blue+
[Constant]
+magenta+
[Constant]
+cyan+
[Constant]
+yellow+
[Function]
find-named-color name palette &key :errorp

```

40.3 Designs

```

[Constant]
+foreground-color+
[Constant]
+background-color+

```

40.4 Presentations

```

[Macro]
with-output-as-presentation (stream object type &key :single-box :allow-sensitive-inferiors :modifier
:parent :record-type) &body body
[Function]
present object &optional presentation-type &key (:stream *standard-output*) :view :modifier :acceptably
:for-context-type :single-box :allow-sensitive-inferiors :sensitive :query-identifier :prompt :record-type
[Function]
present-to-string object &optional presentation-type &key :view :acceptably :for-context-type :string
:index
[Class]
<presentation>
Superclasses: <standard-object>
Slots:
[Class]
<standard-presentation>
Superclasses: <presentation>
Slots:
[Function]
presentationp object

```

[Generic Function]
presentation-object *presentation*
[Generic Function]
presentation-type *presentation*
[Variable]
input-context
[Function]
input-context-type *context-entry*
[Macro]
with-input-context (*type* &key :**override**) (&optional *object-var type-var event-var options-var*) *form* &body *clauses*
[Function]
accept *type* &rest *accept-args* &key :**stream** :**view** :**default** :**default-type** :**history** :**provide-default** :**prompt** :**prompt-mode** :**display-default** :**query-identifier** :**activation-gestures** :**additional-activation-gestures** :**delimiter-gestures** :**additional-delimiter-gestures** :**insert-default** (:**replace-input** *t*) :**present-p** (:**active-p** *t*)
[Function]
accept-from-string *type string* &key :**view** :**default** :**default-type** :**activation-gestures** :**additional-activation-gestures** :**delimiter-gestures** :**additional-delimiter-gestures** (:**start** 0) :**end**
[Presentation type]
t
[Presentation type]
null
[Presentation type]
boolean
[Presentation type]
symbol
[Presentation type]
keyword
[Presentation type]
number
[Presentation type]
complex &optional *type*
[Presentation type]
real &optional *low high*
[Presentation type]
rational
[Presentation type]
ratio &optional *low high*
[Presentation type]
integer &optional *low high*
[Presentation type]
float &optional *low high*
[Presentation type]
character
[Presentation type]
string &optional *length*
[Presentation type]
pathname
[Presentation type]
completion *sequence* &key :**test** :**value-key**
[Presentation type abbreviation]


```

member-sequence sequence &key :test
[Presentation type abbreviation]
member-alist alist &key :test
[Presentation type]
subset-completion sequence &key :test :value-key
[Presentation type abbreviation]
subset &rest elements
[Presentation type abbreviation]
subset-sequence sequence &key :test
[Presentation type abbreviation]
subset-alist alist &key :test
[Presentation type]
sequence element-type
[Presentation type]
sequence-enumerated &rest element-type
[Presentation type]
or &rest types
[Presentation type]
and &rest types
[Presentation type abbreviation]
token-or-type tokens type
[Presentation type abbreviation]
nil-or-type type
[Presentation type abbreviation]
type-or-string type
[Presentation type]
expression
[Presentation type]
form
[Macro]
defpresentation-type name parameters &key :options :inherit-from :description :history :parameters-are-types
[Macro]
defpresentation-method name {qualifiers}* lambda-list &body body
[Macro]
defpresentation-generic generic-function-name presentation-function-name lambda-list &rest options
[Macro]
def-default-presentation-method presentation-function-name qualifiers* specialized-lambda-list &body body

```

```

[Function]
defpresentation->command

```

- 40.5 The Drawing Plane**
- 40.6 Coordinates**
- 40.7 Controlling the Look and Feel**
- 40.8 Streams**
- 40.9 Extended Input/Output**

Chapter 41

Batteries--Utility Libraries

Many of these “batteries” are connected to the outside world in such a way that if something were to drastically change, they would be obsoleted and will be updated. They are suggested, but not required, to be included in implementations of KOIOSLISP as standard utility libraries—this is simply a description of a standard “way” to do the things described here. Some more complex utilities, such as a BitTorrent library, are not specified here as standard libraries but are encouraged to be developed by KOIOSLISP users.

41.1 Logging

Standard logging levels include, from least to most severe, `+dribble+`, `+debug+`, `+info+`, `+warn+`, `+error+`, and `+fatal+`

41.2 Data Compression

41.3 Foreign Function Interfaces

41.4 OS Services and Adaptations

41.4.1 UNIX-specific

41.5 Cryptography

41.6 Internet Data Handling

41.7 Markup Language Processing Tools

41.8 Internet Protocols

41.9 Multimedia Services

Chapter 42

Community Things

These are (probably) not to be implemented as part of the language. Rather, they are services designed for the Koioslisp community. In fact, with some of them (such as Lake or LispDoc), there is encouraged to

42.1 CKLAN

CKLAN is a proposed common repository for the hosting of useful software libre (FLOSS) KOIOSLISP libraries, programs, and scripts. It is inspired by the CPAN (Comprehensive Perl Archive Network), which does the same for the Perl programming language.

42.2 Lake

Lake is the Koioslisp build language, similar to Unix make, Apache Ant, and Ruby Rake. It is also the interpreter for such a program

42.3 LispDoc

LispDoc generates documentation (in the form of \LaTeX or HTML) based on docstrings in a given selection of Koioslisp code. It is roughly similar to JavaDoc and RDoc in this respect.

42.4 ASDF-like tools

Chapter 43

References

Bibliography

- [1] Tim Harris, Simon Marlow, Simon Peyton-Jones, and Maurice Herlihy. Composable memory transactions. In *Proceedings of the tenth ACM SIGPLAN symposium on Principles and practice of parallel programming*, PPOPP '05, pages 46–60, New York, NY, USA, 2005. ACM.
- [2] Richard Kelsey, William Clinger, Jonathan Rees, Hal Abelson, N. I. Adams IV, R. Kent Dybvig, Christopher T. Haynes, Guillermo J. Rozas, Daniel P. Friedman, D. H. Bartley, R. Halstead, D. Oxley, Eugene Kohlbecker, G. J. Sussman, G. Brooks, Chris Hanson, Guy L. Steele Jr., Kent M. Pitman, and Mitchell Wand. Revised5 report on the algorithmic language. *ACM SIGPLAN Notices*, 33:26–76, 1998.
- [3] Gregor Kiczales and Jim des Rivieres. *The Art of the Metaobject Protocol*. MIT Press, Cambridge, MA, USA, 1991.
- [4] Guy L. Steele. *Common LISP: the Language*. Digital Press, Newton, MA, USA, 2 edition, 1990.
- [5] Edi Weitz. CL-PPCRE - Portable Perl-compatible regular expressions for Common Lisp.

Index

(set dynamic-color-color), 118
(setf text-style-mapping), 117
compile-file-pathname, 95
compile-file-truename, 95
compile-print, 95
compile-verbose, 95
default-text-style, 116
end-pos, 69
features, 98
initial-start-pos, 69
input-context, 120
macroexpand-hook, 35
matching-string, 69
property-resolver, 68
reg-ends, 70
reg-starts, 69
start-pos, 69
use-closest-color, 117
+array-dimension-limit+, 63
+array-rank-limit+, 63
+array-total-size-limit+, 63
+background-color+, 119
+black+, 119
+blue+, 119
+cyan+, 119
+everywhere+, 109
+foreground-color+, 119
+green+, 119
+identity-transformation+, 114
+magenta+, 119
+nowhere+, 109
+red+, 119
+white+, 119
+yellow+, 119
:aliases, 42
:author, 41
:clipping-region, 113
:components, 42
:depends-on, 42
:description, 42
:file, 42
:ink, 113
:licence, 41, 42
:line-cap-shape, 114
:line-dashes, 114
:line-joint-shape, 114
:line-style, 113
:line-thickness, 114
:line-unit, 114
:long-description, 42
:maintainer, 41
:module, 42
:name, 41
:normal, 114
:pathname, 42
:point, 114
:serial, 42
:text-face, 116
:text-family, 116
:text-size, 116
:text-style, 113
:transformation, 113
:version, 41
=, 114
=?, 27
%go, 31
%prog, 31
%prog*, 31
%progx, 30
%set, 29
%setq, 30
%tagbody, 31
accept, 120
accept-from-string, 120
accumulate, 32
acons, 56
add-dependent, 84
add-direct-method, 84
add-direct-subclass, 85
add-left, 65
add-method, 85
add-right, 65
adjoin, 57
adjoining, 32
adjustable-array-p, 63

- after, 101
- after-each, 33
- alias, 29
- allocate-pixmap, 109
- always, 32
- and, 29, 121
- append, 50, 55
- appending, 32
- apropos, 96
- aref, 63
- array-dimension, 63
- array-dimensions, 63
- array-element-type, 63
- array-in-bounds-p, 63
- array-rank, 63
- array-row-major-index, 63
- array-total-size, 63
- assert-rel, 77
 - *, 77
- assoc, 56, 59
- assoc-if, 56, 59
- atomic, 101

- begin, 30
- binomial-coefficient, 45
- boolean, 120
- bounding-rectangle, 112
- bounding-rectangle*, 112
- bounding-rectangle-bottom, 112
- bounding-rectangle-height, 112
- bounding-rectangle-max-x, 112
- bounding-rectangle-max-y, 112
- bounding-rectangle-min-x, 112
- bounding-rectangle-min-y, 112
- bounding-rectangle-size, 112
- bounding-rectangle-top, 112
- bounding-rectangle-width, 112
- butlast, 55

- caaaar, 53
- caaadr, 54
- caaar, 53
- caadar, 54
- caaddr, 54
- caadr, 53
- caar, 53
- cadaar, 54
- cadadr, 54
- cadar, 53
- cadd-colors-to-palette, 118
- caddar, 54
- cadddr, 54
- caddr, 53
- cadr, 53
- call/cc, 75
- car, 53
- catch, 34
- cdaaar, 54
- cdaadr, 54
- cdaar, 53
- cdadar, 54
- cdaddr, 54
- cdadr, 53
- cdar, 53
- cddaar, 54
- cddadr, 54
- cddar, 53
- cdddar, 54
- cdddr, 53
- cddr, 53
- cdr, 53
- character, 120
- circular-list?, 56
- clamp, 45
- clear, 60
- collect, 33
- collecting, 32
- color-ihs, 118
- color-rgb, 118
- compile, 95
- compile-file, 95
- complement, 49
- completion, 120
- complex, 120
- compose, 29
- compose-rotation-with-transformation, 115
- compose-scaling-with-transformation, 115
- compose-transformation-with-rotation, 115
- compose-transformation-with-scaling, 115
- compose-transformation-with-translation, 115
- compose-transformations, 115
- compose-translation-with-transformation, 115
- conc, 65
- concllist, 65
- concllist?, 65
- conjoin, 29
- conjugate-transpose, 46
- cons, 54
- contrasting-dash-patterns-limit, 114
- contrasting-inks-limit, 118
- convert-ihs-to-rgb, 118
- convert-rgb-to-ihs, 118
- copy-alist, 56

- copy-area, 109
- copy-from-pixmap, 109
- copy-list, 55
- copy-readtable, 87
- copy-seq, 49
- copy-to-pixmap, 108
- copy-tree, 55
- count, 51
- count-if, 51
- count-permutations, 45
- counting, 32
- cut-delimiter, 77

- datatype, 23
- deallocate-pixmap, 109
- declaim, 37
- declare, 37
- def-default-presentation-method, 121
- defadvice, 93
- defapplication-frame, 107
- defapplication-pane, 107
- defgeneric/cc, 75
- define-declaration, 27
- define-syntax, 35
- deflazy, 73
- defloop, 33
- defloop-alias, 33
- defloop-driver, 33
- defloop-sequence, 33
- defmacro, 35
- defmacro*, 35
- defmethod/cc, 75
- defmonad, 23
- defpresentation->command, 121
- defpresentation-generic, 121
- defpresentation-method, 121
- defpresentation-type, 121
- defreadtable, 87
- defref, 101
- defsetter, 30
- defsetter*, 30
- defsystem, 41
- defun/cc, 75
- defzap, 30
- describe, 96
- describe-object, 96
- device-color-color, 118
- device-color-palette, 118
- device-color-pixel, 118
- difference, 110
- disassemble, 95
- disjoin, 29

- disjoint?, 57
- display-loop-clauses, 33
- divide-and-conc, 65
- do, 30
- do*, 30
- documentation, 95
- domonad, 24
- doseq, 30
- dotimes, 31
- draw-arrow, 107
- draw-arrow*, 107
- draw-arrows*, 107
- draw-bezier-curve, 108
- draw-bezier-curve*, 108
- draw-circle, 108
- draw-circle*, 108
- draw-ellipse, 108
- draw-ellipse*, 108
- draw-line, 107
- draw-line*, 107
- draw-lines, 107
- draw-lines*, 108
- draw-oval, 108
- draw-oval*, 108
- draw-pixmap, 109
- draw-pixmap*, 109
- draw-point, 107
- draw-point*, 107
- draw-points, 107
- draw-points*, 107
- draw-polygon, 108
- draw-polygon*, 108
- draw-rectangle, 108
- draw-rectangle*, 108
- draw-rectangles, 108
- draw-rectangles*, 108
- draw-text, 107
- draw-text*, 107
- dribble, 96
- drop, 73
- dynamic-color-color, 118

- ed, 96
- eigenvalues, 46
- eigenvectors, 46
- eighth, 55
- ellipse-center-point, 113
- ellipse-center-point*, 113
- ellipse-end-angle, 113
- ellipse-radii, 113
- ellipse-start-angle, 113
- else, 33

- empty-hash, 59
- empty-set, 57
- empty?, 65
- end?, 54
- ensure-readtable, 87
- eq?, 27
- equal, 67
- eqv?, 27
- even-scaling-transformation-p, 115
- every, 50
- exit-proc, 101
- export, 41
- expression, 121

- factorial, 45
- fifth, 55
- fill, 50
- filter, 50
- filter-if, 50
- finally, 33
- finally-protected, 33
- find, 51
- find-all, 77
- find-closest-matching-color, 117
- find-if, 51
- find-named-color, 119
- find-readtable, 87
- finding, 32
- finish, 33
- first, 54
- first-iteration?, 32
- first-time?, 32
- flatten, 49
- float, 120
- for, 31, 32
- force, 73
- form, 121
- fourth, 54
- frame-manager-palette, 117
- frame-palette, 117
- function-lambda-expression, 95

- gaussian-random, 45
- get-formatted-french-time, 97
- get-french-time, 97
- get-in, 60
- get-local-time, 97
- getref, 101

- h-append, 60
- h-cons, 59
- h-push, 60
- hash-table?, 59

- hash-values, 60
- head, 73

- identity, 98
- identity-transformation-p, 114
- if-first-time, 33
- import, 41
- in, 33, 34
- in-readtable, 87
- initially, 33
- input-context-type, 120
- inspect, 96
- integer, 120
- intern, 41
- intersection, 49, 109
- invert-matrix, 45
- invert-transformation, 115
- invertible-transformation-p, 115
- invoke-with-drawing-options, 113
- iota, 33
- item, 65

- keys, 60
- keyword, 120

- lambda/cc, 75
- last, 55
- layered-color, 119
- lazy, 73
- lazycall, 73
- lazylambda, 73
- lazylet, 73
- ldiff, 55
- leave, 33
- left, 65
- length, 49, 54, 60
- lerp, 45
- less, 57
- let, 30
- let/cc, 75
- line-end-point, 111
- line-end-point*, 111
- line-start-point, 111
- line-start-point*, 111
- line-style-cap-shape, 114
- line-style-dashes, 114
- line-style-joint-shape, 113
- line-style-thickness, 113
- line-style-unit, 114
- lisp-implementation-type, 97
- lisp-implementation-version, 97
- list, 55
- list*, 55

- list?, 55
- load, 41
- load-time-value, 95
- local-time<, 96
- local-time<=, 96
- local-time>, 97
- local-time>=, 97
- local-time-adjust, 97
- local-time-day, 96
- local-time-msec, 96
- local-time-sec, 96
- local-time-zone, 96
- local-time/=, 97
- local-time=, 97
- locally, 37
- logic-var?, 77
- loop, 31

- macro-function, 35
- macroexpand, 35
- macroexpand-1, 35
- make-3-point-transformation, 114
- make-3-point-transformation*, 114
- make-array, 63
- make-bounding-rectangle, 112
- make-contrasting-dash-patterns, 114
- make-contrasting-inks, 118
- make-device-color, 118
- make-dynamic-color, 118
- make-ellipse, 112
- make-ellipse*, 112
- make-elliptical-arc, 112
- make-elliptical-arc*, 112
- make-gray-color, 118
- make-hash-table, 59
- make-ihs-color, 118
- make-layered-color-set, 118
- make-line, 111
- make-line*, 111
- make-line-style, 113
- make-list, 55
- make-local-time, 96
- make-palette, 117
- make-point, 110
- make-polygon, 110
- make-polygon*, 110
- make-polyline, 110
- make-polyline*, 110
- make-reflection-transformation, 114
- make-reflection-transformation*, 114
- make-rgb-color, 118
- make-rotation-transformation, 114
- make-rotation-transformation*, 114
- make-scaling-transformation, 114
- make-sequence, 49
- make-set, 57
- make-text-style, 117
- make-transformation, 114
- make-translation-transformation, 114
- map, 50
- map-iota, 45
- map-keys, 60
- map-polygon-coordinates, 111
- map-polygon-segments, 111
- map-reduce, 50
- map-regions, 110
- maphash, 60
- maplist, 56
- mappend, 50
- maximize, 32
- mean, 45
- median, 45
- medium-background, 113
- medium-clipping-region, 113
- medium-copy-area, 108
- medium-default-text-style, 113
- medium-draw-ellipse*, 108
- medium-draw-line*, 108
- medium-draw-lines*, 108
- medium-draw-point*, 108
- medium-draw-points*, 108
- medium-draw-polygon*, 108
- medium-draw-rectangle*, 108
- medium-draw-rectangles*, 108
- medium-draw-text*, 108
- medium-foreground, 113
- medium-ink, 113
- medium-line-style, 113
- medium-text-style, 113
- medium-transformation, 113
- member, 51, 56, 60
- member-alist, 120
- member-if, 51, 56, 60
- member-sequence, 120
- memoize, 29
- merge, 51, 52
- merge-readtables-into, 87
- merge-text-style, 116
- merge-with, 60
- mismatch, 51
- multiply, 32

- nbutlast, 55
- never, 33

- next-iteration, 33
- nil-or-type, 121
- ninth, 55
- not, 27
- notany, 50
- notevery, 50
- nth, 49, 54
- nthcdr, 55
- null, 120
- number, 120
- or, 29, 121
- or-else, 101
- os-type, 97
- os-version, 98
- pair, 59
- pairlis, 56
- palette-color?, 117
- palette-full, 117
- palette-full-color, 117
- palette-full-palette, 117
- palette-mutable-p, 117
- palette?, 117
- parse-text-style, 116
- partial, 29
- partition, 51
- partition-if, 51
- pathname, 120
- pixmap-depth, 109
- pixmap-height, 109
- pixmap-width, 109
- point-position, 110
- point-x, 110
- point-y, 110
- polyline-closed-p, 111
- pop, 55
- port-default-palette, 118
- position, 51
- position-if, 51
- present, 119
- present-to-string, 119
- presentation-object, 119
- presentation-type, 120
- presentationp, 119
- proclaim, 37
- provide, 41
- pset, 30
- push, 55
- pushnew, 55
- rassoc, 56, 59
- rassoc-if, 56, 59
- ratio, 120
- rational, 120
- read-file-into-string, 90
- real, 120
- rebalance, 65
- recieve, 101
- recolor-dynamic-color, 118
- rectangle-height, 111
- rectangle-max-point, 111
- rectangle-min-point, 111
- rectangle-min-x, 111
- rectangle-min-y, 111
- rectangle-size, 111
- rectangle-width, 111
- rectilinear-transformation-p, 115
- reduce, 50, 59
- reducing, 32
- ref, 101
- reflection-transformation-p, 115
- region-contains-position-p, 109
- region-contains-region-p, 109
- region-difference, 107
- region-equal, 109
- region-intersection, 107
- region-intersects-region-p, 109
- region-set-function, 110
- region-set-regions, 110
- regular expressions, 68
- rel, 77
- remove, 50, 59
- remove-colors-from-palette, 118
- remove-duplicates, 50, 51
- remove-if, 50
- repeat, 31
- replace, 50
- require, 41
- rest, 55
- retry, 101
- revappend, 55
- reverse, 49
- rewind, 90
- right, 65
- rigid-transformation-p, 115
- room, 96
- rotate, 30
- row-major-aref, 63
- rref, 45
- scaling-transformation-p, 115
- search, 51
- second, 54
- send, 101

- sequence, 121
- sequence-enumerated, 121
- set, 30
- set-difference, 49
- seventh, 55
- shadow, 41
- shift, 30
- singleton?, 65
- sixth, 55
- sleep, 97
- software-type, 98
- some, 50
- sort, 51, 60
- spawn, 101
- spawn-remotely, 101
- split, 65
- splitre, 68
- stable-sort, 51
- standard-deviation, 45
- step, 96
- str, 67
- string, 120
- string/ =, 67
- string<, 67
- string<=, 67
- string=, 67
- string>, 67
- string>=, 67
- string-equal, 67
- string-greater?, 67
- string-less?, 67
- string-not-equal?, 68
- string-not-greater?, 68
- string-not-less?, 67
- subfactorial, 45
- sublis, 56
- subseq, 49
- subset, 121
- subset-alist, 121
- subset-completion, 121
- subset-sequence, 121
- subset?, 57
- subst, 55, 56
- subst-if, 56
- substitute, 51
- substitute-if, 51
- sum, 32
- sxhash, 60
- symbol, 120
- syntax-rules, 35
- syscall, 97
- t, 120
- tail, 73
- tail?, 56
- take, 73
- take-all, 73
- tenth, 55
- text-style-ascent, 116
- text-style-components, 116
- text-style-descent, 116
- text-style-face, 116
- text-style-family, 116
- text-style-fixed-width-p, 117
- text-style-height, 116
- text-style-mapping, 117
- text-style-size, 116
- text-style-width, 117
- the, 37
- there-is, 33
- third, 54
- throw, 34
- time, 96
- timestamp->french, 97
- token-or-type, 121
- trace, 96
- transform-distance, 115
- transform-position, 115
- transform-rectangle*, 116
- transform-region, 115
- translation-transformation-p, 115
- transpose, 45
- tree-equal, 54
- type-or-string, 121
- type?, 27
- unifiable?, 77
- unintern, 41
- union, 49, 109
- unioning, 32
- until, 33
- untrace, 96
- untransform-distance, 116
- untransform-position, 115
- untransform-rectangle*, 116
- untransform-region, 115
- unuse-system, 41
- unwind-protect, 34
- use-other-color, 117
- use-system, 41
- variance, 45
- when, 34
- which, 77

- while, 33
- with, 32, 49
- with-call/cc, 75
- with-compilation-unit, 95
- with-delayed-recoloring, 118
- with-drawing-options, 113
- with-first-quadrant-coordinates, 115
- with-hash-table-iterator, 60
- with-input-context, 120
- with-local-coordinates, 115
- with-monad, 24
- with-output-as-presentation, 119
- with-output-to-pixmap, 109
- with-room-for-graphics, 115
- with-rotation, 115
- with-scaling, 115
- with-text-face, 117
- with-text-family, 117
- with-text-size, 117
- with-text-style, 117
- with-translation, 115
- without-call/cc, 75

- xor, 29

- zap, 29