This manual is for SLIB (version 3b4, May 2013), the portable Scheme library.


Permission is granted to copy, distribute and/or modify this document under the terms of the GNU Free Documentation License, Version 1.3 or any later version published by the Free Software Foundation; with no Invariant Sections, no Front-Cover Texts, and no Back-Cover Texts. A copy of the license is included in the section entitled “GNU Free Documentation License.”
# Table of Contents

1 The Library System ........................................ 1
   1.1 Feature ................................................ 1
   1.2 Require ............................................... 2
   1.3 Library Catalogs ......................................... 3
   1.4 Catalog Creation ....................................... 3
   1.5 Catalog Vicinities ..................................... 4
   1.6 Compiling Scheme ..................................... 6
      1.6.1 Module Conventions ............................... 6
      1.6.2 Module Manifests ................................. 6
      1.6.3 Module Semantics ............................... 9
      1.6.4 Top-level Variable References ................ 9
      1.6.5 Module Analysis ................................. 10

2 Universal SLIB Procedures ................................. 11
     2.1 Vicinity ............................................. 11
     2.2 Configuration ...................................... 12
     2.3 Input/Output ....................................... 13
     2.4 System .............................................. 15
     2.5 Miscellany ......................................... 16
        2.5.1 Mutual Exclusion ............................ 16
        2.5.2 Legacy ......................................... 17

3 Scheme Syntax Extension Packages ................... 18
       3.1 Defmacro .......................................... 18
          3.1.1 Defmacroexpand .............................. 18
       3.2 R4RS Macros ....................................... 19
       3.3 Macro by Example .................................. 19
          3.3.1 Caveat ......................................... 19
       3.4 Macros That Work ................................ 20
          3.4.1 Definitions ................................... 22
          3.4.2 Restrictions .................................. 22
       3.5 Syntactic Closures ................................ 23
          3.5.1 Syntactic Closure Macro Facility .......... 24
             3.5.1.1 Terminology .............................. 24
             3.5.1.2 Transformer Definition ................... 25
             3.5.1.3 Identifiers ............................... 28
             3.5.1.4 Acknowledgements ....................... 30
       3.6 Syntax-Case Macros ............................... 30
          3.6.1 Notes .......................................... 32
       3.7 Define-Structure .................................. 32
       3.8 Define-Record-Type ................................ 33
       3.9 Fluid-Let .......................................... 33
4 Textual Conversion Packages

4.1 Precedence Parsing
   4.1.1 Precedence Parsing Overview
   4.1.2 Rule Types
   4.1.3 Ruleset Definition and Use
   4.1.4 Token definition
   4.1.5 Nud and Led Definition
   4.1.6 Grammar Rule Definition

4.2 Format (version 3.1)
   4.2.1 Format Interface
   4.2.2 Format Specification (Format version 3.1)
      4.2.2.1 Implemented CL Format Control Directives
      4.2.2.2 Not Implemented CL Format Control Directives
      4.2.2.3 Extended, Replaced and Additional Control Directives
      4.2.2.4 Configuration Variables
   4.2.2.5 Compatibility With Other Format Implementations

4.3 Standard Formatted I/O
   4.3.1 stdio
   4.3.2 Standard Formatted Output
      4.3.2.1 Exact Conversions
      4.3.2.2 Inexact Conversions
      4.3.2.3 Other Conversions
   4.3.3 Standard Formatted Input

4.4 Program and Arguments
   4.4.1 Getopt
   4.4.2 Getopt—
   4.4.3 Command Line
   4.4.4 Parameter lists
   4.4.5 Getopt Parameter lists
   4.4.6 Filenames
   4.4.7 Batch

4.5 HTML

4.6 HTML Forms

4.7 HTML Tables
   4.7.1 HTML editing tables
   4.7.2 HTML databases

4.8 HTTP and CGI
5 Mathematical Packages .................................. 103
5.1 Bit-Twiddling ......................................... 103
  5.1.1 Bitwise Operations ................................. 103
  5.1.2 Integer Properties ................................ 104
  5.1.3 Bit Within Word .................................. 105
  5.1.4 Field of Bits ...................................... 106
  5.1.5 Bits as Booleans .................................. 107
5.2 Modular Arithmetic ................................... 107
5.3 Irrational Integer Functions ............................ 108
5.4 Irrational Real Functions ............................... 109
5.5 Prime Numbers ....................................... 111
5.6 Random Numbers ...................................... 111
  5.6.1 Exact Random Numbers ......................... 112
  5.6.2 Inexact Random Numbers ...................... 112
5.7 Discrete Fourier Transform ............................ 113
5.8 Cyclic Checksum ...................................... 114
5.9 Graphing ............................................. 117
  5.9.1 Character Plotting ................................. 117
  5.9.2 PostScript Graphing ............................... 120
    5.9.2.1 Column Ranges ............................... 121
    5.9.2.2 Drawing the Graph ......................... 121
    5.9.2.3 Graphics Context ......................... 122
    5.9.2.4 Rectangles ................................. 123
    5.9.2.5 Legending .................................. 124
    5.9.2.6 Legacy Plotting .............................. 125
    5.9.2.7 Example Graph .............................. 125
6 Database Packages

6.1 Relational Database
   6.1.1 Using Databases ............................................. 162
   6.1.2 Table Operations ............................................ 164
      6.1.2.1 Single Row Operations ................................. 164
      6.1.2.2 Match-Keys ............................................. 166
      6.1.2.3 Multi-Row Operations ................................. 167
      6.1.2.4 Indexed Sequential Access Methods ................. 168
      6.1.2.5 Sequential Index Operations ......................... 168
      6.1.2.6 Table Administration ................................. 169
   6.1.3 Database Interpolation ..................................... 169
   6.1.4 Embedded Commands ......................................... 169
      6.1.4.1 Database Extension ..................................... 170
      6.1.4.2 Command Intrinsics .................................... 171
      6.1.4.3 Define-tables Example ............................... 171
      6.1.4.4 The *commands* Table ............................... 172
      6.1.4.5 Command Service ....................................... 173
      6.1.4.6 Command Example ...................................... 174
   6.1.5 Database Macros ............................................ 176
      6.1.5.1 Within-database ....................................... 176
      6.1.5.2 Within-database Example ............................. 178
   6.1.6 Database Browser ........................................... 179

6.2 Relational Infrastructure
   6.2.1 Base Table ................................................... 179
      6.2.1.1 The Base ................................................. 180
      6.2.1.2 Base Tables ............................................ 181
      6.2.1.3 Base Field Types ..................................... 182
      6.2.1.4 Composite Keys ....................................... 182
      6.2.1.5 Base Record Operations ............................. 182
7 Other Packages

7.1 Data Structures
7.1.1 Arrays
7.1.2 Subarrays
7.1.3 Array Mapping
7.1.4 Array Interpolation
7.1.5 Association Lists
7.1.6 Byte
7.1.7 Byte/Number Conversions
7.1.8 MAT-File Format
7.1.9 Portable Image Files
7.1.10 Collections
7.1.11 Dynamic Data Type
7.1.12 Hash Tables
7.1.13 Macroless Object System
7.1.14 Concepts
7.1.15 Procedures
7.1.16 Examples
7.1.16.1 Inverter Documentation
7.1.16.2 Number Documentation
7.1.16.3 Inverter code
7.1.17 Priority Queues
7.1.18 Queues
7.1.19 Records
7.2 Sorting and Searching
7.2.1 Common List Functions
7.2.1.1 List construction
7.2.1.2 Lists as sets
7.2.1.3 Lists as sequences
7.2.1.4 Destructive list operations
7.2.1.5 Non-List functions
7.2.2 Tree operations
7.2.3 Chapter Ordering
7.2.4 Sorting
7.2.5 Topological Sort
7.2.6 Hashing
<table>
<thead>
<tr>
<th>Section</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>7.2.7</td>
<td>Space-Filling Curves</td>
<td>235</td>
</tr>
<tr>
<td>7.2.7.1</td>
<td>Hilbert Space-Filling Curve</td>
<td>235</td>
</tr>
<tr>
<td>7.2.7.2</td>
<td>Gray code</td>
<td>236</td>
</tr>
<tr>
<td>7.2.7.3</td>
<td>Bitwise Lamination</td>
<td>237</td>
</tr>
<tr>
<td>7.2.7.4</td>
<td>Peano Space-Filling Curve</td>
<td>237</td>
</tr>
<tr>
<td>7.2.7.5</td>
<td>Sierpinski Curve</td>
<td>237</td>
</tr>
<tr>
<td>7.2.8</td>
<td>Soundex</td>
<td>238</td>
</tr>
<tr>
<td>7.2.9</td>
<td>String Search</td>
<td>239</td>
</tr>
<tr>
<td>7.2.10</td>
<td>Sequence Comparison</td>
<td>240</td>
</tr>
<tr>
<td>7.3</td>
<td>Procedures</td>
<td>241</td>
</tr>
<tr>
<td>7.3.1</td>
<td>Type Coercion</td>
<td>241</td>
</tr>
<tr>
<td>7.3.2</td>
<td>String-Case</td>
<td>241</td>
</tr>
<tr>
<td>7.3.3</td>
<td>String Ports</td>
<td>242</td>
</tr>
<tr>
<td>7.3.4</td>
<td>Line I/O</td>
<td>242</td>
</tr>
<tr>
<td>7.3.5</td>
<td>Multi-Processing</td>
<td>243</td>
</tr>
<tr>
<td>7.3.6</td>
<td>Metric Units</td>
<td>244</td>
</tr>
<tr>
<td>7.3.6.1</td>
<td>SI Prefixes</td>
<td>245</td>
</tr>
<tr>
<td>7.3.6.2</td>
<td>Binary Prefixes</td>
<td>245</td>
</tr>
<tr>
<td>7.3.6.3</td>
<td>Unit Symbols</td>
<td>245</td>
</tr>
<tr>
<td>7.4</td>
<td>Standards Support</td>
<td>247</td>
</tr>
<tr>
<td>7.4.1</td>
<td>RnRS</td>
<td>247</td>
</tr>
<tr>
<td>7.4.2</td>
<td>With-File</td>
<td>247</td>
</tr>
<tr>
<td>7.4.3</td>
<td>Transcripts</td>
<td>248</td>
</tr>
<tr>
<td>7.4.4</td>
<td>Rev2 Procedures</td>
<td>248</td>
</tr>
<tr>
<td>7.4.5</td>
<td>Rev4 Optional Procedures</td>
<td>249</td>
</tr>
<tr>
<td>7.4.6</td>
<td>Multi-argument / and -</td>
<td>249</td>
</tr>
<tr>
<td>7.4.7</td>
<td>Multi-argument Apply</td>
<td>249</td>
</tr>
<tr>
<td>7.4.8</td>
<td>Rationalize</td>
<td>249</td>
</tr>
<tr>
<td>7.4.9</td>
<td>Promises</td>
<td>250</td>
</tr>
<tr>
<td>7.4.10</td>
<td>Dynamic-Wind</td>
<td>250</td>
</tr>
<tr>
<td>7.4.11</td>
<td>Eval</td>
<td>250</td>
</tr>
<tr>
<td>7.4.12</td>
<td>Values</td>
<td>252</td>
</tr>
<tr>
<td>7.4.13</td>
<td>SRFI</td>
<td>252</td>
</tr>
<tr>
<td>7.4.13.1</td>
<td>SRFI-1</td>
<td>253</td>
</tr>
<tr>
<td>7.5</td>
<td>Session Support</td>
<td>256</td>
</tr>
<tr>
<td>7.5.1</td>
<td>Repl</td>
<td>257</td>
</tr>
<tr>
<td>7.5.2</td>
<td>Quick Print</td>
<td>257</td>
</tr>
<tr>
<td>7.5.3</td>
<td>Debug</td>
<td>258</td>
</tr>
<tr>
<td>7.5.4</td>
<td>Breakpoints</td>
<td>258</td>
</tr>
<tr>
<td>7.5.5</td>
<td>Tracing</td>
<td>259</td>
</tr>
<tr>
<td>7.6</td>
<td>System Interface</td>
<td>261</td>
</tr>
<tr>
<td>7.6.1</td>
<td>Directories</td>
<td>261</td>
</tr>
<tr>
<td>7.6.2</td>
<td>Transactions</td>
<td>262</td>
</tr>
<tr>
<td>7.6.3</td>
<td>CVS</td>
<td>264</td>
</tr>
<tr>
<td>7.7</td>
<td>Extra-SLIB Packages</td>
<td>265</td>
</tr>
</tbody>
</table>
8 About SLIB .............................................. 266
  8.1 Installation ........................................... 266
    8.1.1 Unpacking the SLIB Distribution .................. 266
    8.1.2 Install documentation and slib script .............. 266
    8.1.3 Configure Scheme Implementation to Locate SLIB ... 266
    8.1.4 Configure Scheme Implementation to Locate and
      Implementation Directory ......................... 267
    8.1.5 Loading SLIB Initialization File .................... 267
    8.1.6 Build New SLIB Catalog for the Implementation ... 267
    8.1.7 Implementation-specific Instructions ............... 268
  8.2 The SLIB script ........................................ 270
  8.3 Porting .................................................. 270
  8.4 Compiled and Implementation-Specific Features ........... 270
  8.5 Coding Guidelines ....................................... 271
    8.5.1 Modifications .................................. 271
  8.6 Copyrights ............................................... 272
    8.6.1 Putting code into the Public Domain ............... 272
    8.6.2 Explicit copying terms .......................... 272
    8.6.3 Example: Company Copyright Disclaimer ........... 273
  8.7 About this manual ....................................... 273
    8.7.1 GNU Free Documentation License ................... 273

Procedure and Macro Index ......................... 282

Variable Index ............................................ 293

Concept and Feature Index ............................ 294
Chapter 1: The Library System

SLIB is a portable library for the programming language Scheme. It provides a platform independent framework for using packages of Scheme procedures and syntax. As distributed, SLIB contains useful packages for all Scheme implementations. Its catalog can be transparently extended to accommodate packages specific to a site, implementation, user, or directory.

1.1 Feature

SLIB denotes features by symbols. SLIB maintains a list of features supported by a Scheme session. The set of features provided by a session may change during that session. Some features are properties of the Scheme implementation being used. The following intrinsic features detail what sort of numbers are available from an implementation:

- 'inexact
- 'rational
- 'real
- 'complex
- 'bignum

SLIB initialization (in ‘require.scm’) tests and provides any of these numeric features which are appropriate.

Other features correspond to the presence of packages of Scheme procedures or syntax (macros).

provided? feature

Returns #t if feature is present in the current Scheme session; otherwise #f. More specifically, provided? returns #t if the symbol feature is the software-type, the scheme-implementation-type, or if feature has been provided by a module already loaded; and #f otherwise.

In some implementations provided? tests whether a module has been required by any module or in any thread; other implementations will have provided? reflect only the modules required by that particular session or thread.

To work portably in both scenarios, use provided? only to test whether intrinsic properties (like those above) are present.

The feature argument can also be an expression calling and, or, and not of features. The boolean result of the logical question asked by feature is returned.

The generalization of provided? for arbitrary features and catalog is feature-eval:

feature-eval expression provided?

Evaluates and, or, and not forms in expression, using the values returned by calling provided? on the leaf symbols. feature-eval returns the boolean result of the logical combinations.

---

1 scheme-implementation-type is the name symbol of the running Scheme implementation (RScheme, 1STK1, Bigloo, chez, Elk, gambit, gauche, guile, JScheme, kawa, MacScheme, MITScheme, Pocket-Scheme, S7, Scheme48, Scheme>C, Scheme48, Scsh, SISC, T, umb-scheme, or Vscm). Dependence on scheme-implementation-type is almost always the wrong way to do things.
provide feature
  [Procedure]
  Informs SLIB that feature is supported in this session.
  
  \[
  \begin{align*}
  \text{(provided? 'foo)} & \Rightarrow \text{#f} \\
  \text{(provide 'foo)} & \\
  \text{(provided? 'foo)} & \Rightarrow \text{#t}
  \end{align*}
  \]

1.2 Require

SLIB creates and maintains a catalog mapping features to locations of files introducing procedures and syntax denoted by those features.

*catalog*  [Variable]
  Is an association list of features (symbols) and pathnames which will supply those features. The pathname can be either a string or a pair. If pathname is a pair then the first element should be a macro feature symbol, source, compiled, or one of the other cases described in Section 1.3 [Library Catalogs], page 3. The cdr of the pathname should be either a string or a list.

At the beginning of each section of this manual, there is a line like \text{(require 'feature)}. The Scheme files comprising SLIB are cataloged so that these feature names map to the corresponding files.

SLIB provides a form, require, which loads the files providing the requested feature.

require feature  [Procedure]
  \begin{itemize}
  \item If \text{(provided? feature)} is true, then require just returns.
  \item Otherwise, if feature is found in the catalog, then the corresponding files will be loaded and \text{(provided? feature)} will henceforth return \text{#t}. That feature is thereafter provided.
  \item Otherwise (feature not found in the catalog), an error is signaled.
  \end{itemize}

There is a related form require-if, used primarily for enabling compilers to statically include modules which would be dynamically loaded by interpreters.

require-if condition feature  [Procedure]
  Requires feature if condition is true.

The random module uses require-if to flag \text{object->string} as a (dynamic) required module.

  \begin{verbatim}
  (require 'byte)
  (require 'logical)
  (require-if 'compiling 'object->string)
  \end{verbatim}

The batch module uses require-if to flag \text{posix-time} as a module to load if the implementation supports large precision exact integers.

  \begin{verbatim}
  (require-if '(and bignum compiling) 'posix-time)
  \end{verbatim}

The catalog can also be queried using \text{slib:in-catalog?}.
Chapter 1: The Library System

**Function**

slib:in-catalog?  
*feature*  
Returns a CDR of the catalog entry if one was found for the symbol *feature* in the alist *catalog* (and transitively through any symbol aliases encountered). Otherwise, returns #f. The format of catalog entries is explained in Section 1.3 [Library Catalogs], page 3.

### 1.3 Library Catalogs

Catalog files consist of one or more association lists. In the circumstance where a feature symbol appears in more than one list, the latter list’s association is retrieved. Here are the supported formats for elements of catalog lists:

- \((feature . <symbol>)\)
  Redirects to the feature named <symbol>.

- \((feature . "<path>")\)
  Loads file <path>.

- \((feature source "<path>"))\)
  \(\text{slib:load}\)s the Scheme source file <path>.

- \((feature compiled "<path>" ...))\)
  \(\text{slib:load-compileds}\) the files <path> . . .

- \((feature aggregate <symbol> ...))\)
  \(\text{requires}\) the features <symbol> . . .

The various macro styles first **require** the named macro package, then just load <path> or load-and-macro-expand <path> as appropriate for the implementation.

- \((feature defmacro "<path>"))\)
  \(\text{defmacro:load}\)s the Scheme source file <path>.

- \((feature macro-by-example "<path>"))\)
  \(\text{defmacro:load}\)s the Scheme source file <path>.

- \((feature macro "<path>"))\)
  \(\text{macro:load}\)s the Scheme source file <path>.

- \((feature macros-that-work "<path>"))\)
  \(\text{macro:load}\)s the Scheme source file <path>.

- \((feature syntax-case "<path>"))\)
  \(\text{macro:load}\)s the Scheme source file <path>.

- \((feature syntactic-closures "<path>"))\)
  \(\text{macro:load}\)s the Scheme source file <path>.

### 1.4 Catalog Creation

At the start of an interactive session no catalog is present, but is created with the first catalog inquiry (such as \((\text{require 'random})\)). Several sources of catalog information are combined to produce the catalog:

- standard SLIB packages.
additional packages of interest to this site.

- packages specifically for the variety of Scheme which this session is running. This catalog, if it exists, is the file ‘implcat’ in implementation-invicinity, which is created by loading ‘mkimpcat.scm’ in implementation-invicinity if it exists.

- packages this user wants to always have available. This catalog is the file ‘homecat’ in the user’s HOME directory.

- packages germane to working in this (current working) directory. This catalog is the file ‘usercat’ in the directory to which it applies. One would typically cd to this directory before starting the Scheme session.

- packages which are part of an application program.

SLIB combines the catalog information which doesn’t vary per user into the file ‘slibcat’ in the implementation-vicinity. Therefore ‘slibcat’ needs change only when new software is installed or compiled. Because the actual pathnames of files can differ from installation to installation, SLIB builds a separate catalog for each implementation it is used with.

The definition of *slib-version* in SLIB file ‘require.scm’ is checked against the catalog association of *slib-version* to ascertain when versions have changed. It is a reasonable practice to change the definition of *slib-version* whenever the library is changed. If multiple implementations of Scheme use SLIB, remember that recompiling one ‘slibcat’ will update only that implementation’s catalog.

The compilation scripts of Scheme implementations which work with SLIB can automatically trigger catalog compilation by deleting ‘slibcat’ or by invoking require of a special feature:

```
require 'new-catalog'  # Procedure
This will load 'mklibcat', which compiles and writes a new 'slibcat'.
```

Another special feature of require erases SLIB’s catalog, forcing it to be reloaded the next time the catalog is queried.

```
require #f  # Procedure
Removes SLIB’s catalog information. This should be done before saving an executable image so that, when restored, its catalog will be loaded afresh.
```

### 1.5 Catalog Vicinities

Each file in the table below is described in terms of its file-system independent vicinity (see Section 2.1 [Vicinity], page 11). The entries of a catalog in the table override those of catalogs above it in the table.

**implementation-vicinity ‘slibcat’**

This file contains the associations for the packages comprising SLIB, the ‘implcat’ and the ‘sitecat’s. The associations in the other catalogs override those of the standard catalog.

**library-vicinity ‘mklibcat.scm’**

creates ‘slibcat’.
library-vicinity 'sitecat'
   This file contains the associations specific to an SLIB installation.

implementation-vicinity 'implcat'
   This file contains the associations specific to an implementation of Scheme.
   Different implementations of Scheme should have different implementation-vicinity.

implementation-vicinity 'mkimpcat.scm'
   if present, creates 'implcat'.

implementation-vicinity 'sitecat'
   This file contains the associations specific to a Scheme implementation installation.

home-vicinity 'homecat'
   This file contains the associations specific to an SLIB user.

user-vicinity 'usercat'
   This file contains associations affecting only those sessions whose working directory is user-vicinity.

Here is an example of a 'usercat' catalog. A program in this directory can invoke the 'run' feature with (require 'run).

;;; "usercat": SLIB catalog additions for SIMSYNCH. -*-scheme-*-

(  
  (simsynch  . "../synch/simsynch.scm")
  (run       . "../synch/run.scm")
  (schlep    . "schlep.scm")
)

Copying 'usercat' to many directories is inconvenient. Application programs which aren’t always run in specially prepared directories can nonetheless register their features during initialization.

catalog:read  vicinity catalog  [Procedure]
   Reads file named by string catalog in vicinity, resolving all paths relative to vicinity, and adds those feature associations to *catalog*.

catalog:read would typically be used by an application program having dynamically loadable modules. For instance, to register factoring and other modules in *catalog*, JACAL does:

   (catalog:read (program-vicinity) "jacalcat")

For an application program there are three appropriate venues for registering its catalog associations:
   • in a 'usercat' file in the directory where the program runs; or
   • in an 'implcat' file in the implementation-vicinity; or
   • in an application program directory; loaded by calling catalog:read.
1.6 Compiling Scheme

To use Scheme compilers effectively with SLIB the compiler needs to know which SLIB modules are to be compiled and which symbols are exported from those modules.

The procedures in this section automate the extraction of this information from SLIB modules. They are guaranteed to work on SLIB modules; to use them on other sources, those sources should follow SLIB conventions.

1.6.1 Module Conventions

- All the top-level `require` commands have one quoted argument and are positioned before other Scheme definitions and expressions in the file.
- Any conditionally required SLIB modules\(^2\) also appear at the beginning of their files conditioned on the feature `compiling` using `require-if` (see Section 1.2 [Require], page 2).

```
(require 'logical)
(require 'multiarg/and-)
(require-if 'compiling 'sort)
(require-if 'compiling 'ciexyz)
```

- Schmooz-style comments preceding a definition, identify that definition as an exported identifier (see Section 4.15 [Schmooz], page 100). For non-schmooz files, putting `';@'` at the beginning of the line immediately preceding the definition (define, define-syntx, or defmacro) suffices.

```
;@
(define (identity <obj>) <obj>)
```

- Syntax (macro) definitions are grouped at the end of a module file.
- Modules defining macros do not invoke those macros. SLIB macro implementations are exempt from this rule.

An example of how to expand macro invocations is:

```
(require 'macros-that-work)
(require 'yasos)
(require 'pprint-file)
(pprint-filter-file "collect.scm" macwork:expand)
```

1.6.2 Module Manifests

(require 'manifest)

In some of these examples, `slib:catalog` is the SLIB part of the catalog; it is free of compiled and implementation-specific entries. It would be defined by:

```
(define slib:catalog (cdr (member (assq 'null *catalog*) *catalog*)))
```

(file->requires file provided? catalog) [Function]

Returns a list of the features required by file assuming the predicate provided? and association-list catalog.

---

\(^2\) There are some functions with internal `require` calls to delay loading modules until they are needed. While this reduces startup latency for interpreters, it can produce headaches for compilers.
(define (provided+? . features)
  (lambda (feature)
    (or (memq feature features) (provided? feature))))

(file->requires "obj2str.scm" (provided+? 'compiling) '())
⇒ (string-port generic-write)

(file->requires "obj2str.scm" provided? '())
⇒ (string-port)

feature->requires feature provided? catalog
[Function]
Returns a list of the features required by feature assuming the predicate provided?
and association-list catalog.

(feature->requires 'batch (provided+? 'compiling) *catalog*)
⇒ (tree line-i/o databases parameters string-port
     pretty-print common-list-functions posix-time)

(feature->requires 'batch provided? *catalog*)
⇒ (tree line-i/o databases parameters string-port
     pretty-print common-list-functions)

(feature->requires 'batch provided? '((batch . "batch")))
⇒ (tree line-i/o databases parameters string-port
     pretty-print common-list-functions)

feature->requires* feature provided? catalog
[Function]
Returns a list of the features transitively required by feature assuming the predicate provided?
and association-list catalog.

file->requires* file provided? catalog
[Function]
Returns a list of the features transitively required by file assuming the predicate provided?
and association-list catalog.

file->loads file
[Function]
Returns a list of strings naming existing files loaded (load slib:load slib:load-source
macro:load defmacro:load syncase:load synclo:load macwork:load) by file or any of
the files it loads.

(file->loads (in-vicinity (library-vicinity) "scainit.scm"))
⇒ ("/usr/local/lib/slib/scaexpp.scm"
    "/usr/local/lib/slib/scaglob.scm"
    "/usr/local/lib/slib/scaoutp.scm")

load->path exp
[Function]
Given a (load '<expr>), where <expr> is a string or vicinity stuff), (load->path
<expr>) figures a path to the file. load->path returns that path if it names an
existing file; otherwise #f.
(load->path '(in-vicinity (library-vicinity) "mklibcat"))
⇒ "/usr/local/lib/slib/mklibcat.scm"

file->definitions file definer ...
[Function]
Returns a list of the identifier symbols defined by SLIB (or SLIB-style) file file. The optional arguments definers should be symbols signifying a defining form. If none are supplied, then the symbols define-operation, define, define-syntax, and defmacro are captured.

(file->definitions "random.scm")
⇒ (*random-state* make-random-state seed->random-state copy-random-state random random:chunk)

file->exports file definer ...
[Function]
Returns a list of the identifier symbols exported (advertised) by SLIB (or SLIB-style) file file. The optional arguments definers should be symbols signifying a defining form. If none are supplied, then the symbols define-operation, define, define-syntax, and defmacro are captured.

(file->exports "random.scm")
⇒ (make-random-state seed->random-state copy-random-state random)

(file->exports "randinex.scm")

feature->export-alist feature catalog
[Function]
Returns a list of lists; each sublist holding the name of the file implementing feature, and the identifier symbols exported (advertised) by SLIB (or SLIB-style) feature feature, in catalog.

feature->exports feature catalog
[Function]
Returns a list of all exports of feature.

In the case of aggregate features, more than one file may have export lists to report:

(feature->export-alist 'r5rs slib:catalog))
⇒ (("/usr/local/lib/slib/values.scm"
call-with-values values)
("/usr/local/lib/slib/mbe.scm"
define-syntax macro:expand macro:load macro:eval)
("/usr/local/lib/slib/eval.scm"
eval scheme-report-environment null-environment interaction-environment))

(feature->export-alist 'stdio *catalog*)
1.6.3 Module Semantics

For the purpose of compiling Scheme code, each top-level require makes the identifiers exported by its feature’s module defined (or defmacroed or defined-syntaxed) within the file (being compiled) headed with those requires.

Top-level occurrences of require-if make defined the exports from the module named by the second argument if the feature-expression first argument is true in the target environment. The target feature compiling should be provided during this phase of compilation.

Non-top-level SLIB occurrences of require and require-if of quoted features can be ignored by compilers. The SLIB modules will all have top-level constructs for those features.

Note that aggregate catalog entries import more than one module. Implementations of require may or may not be transitive; code which uses module exports without requiring the providing module is in error.

In the SLIB modules modular, batch, hash, common-lisp-time, commutative-ring, charplot, logical, common-list-functions, coerce and break there is code conditional on features being provided?. Most are testing for the presence of features which are intrinsic to implementations (inexacts, bignums, ...).

In all cases these provided? tests can be evaluated at compile-time using feature-eval (see Section 1.1 [Feature], page 1). The simplest way to compile these constructs may be to treat provided? as a macro.

1.6.4 Top-level Variable References

(require 'top-refs)

These procedures complement those in Section 1.6.2 [Module Manifests], page 6 by finding the top-level variable references in Scheme source code. They work by traversing expressions and definitions, keeping track of bindings encountered. It is certainly possible to foil these functions, but they return useful information about SLIB source code.

\textbf{top-refs} \textit{obj} \hfill [Function]

Returns a list of the top-level variables referenced by the Scheme expression \textit{obj}.

\textbf{top-refs<-file} \textit{filename} \hfill [Function]

\textit{filename} should be a string naming an existing file containing Scheme source code. \textbf{top-refs<-file} returns a list of the top-level variable references made by expressions in the file named by \textit{filename}. 
Code in modules which \textit{filename} \textbf{requires} is not traversed. Code in files loaded from top-level is traversed if the expression argument to \texttt{load}, \texttt{slib:load}, \texttt{slib:load-source}, \texttt{macro:load}, \texttt{defmacro:load}, \texttt{synclo:load}, \texttt{syncase:load}, or \texttt{macwork:load} is a literal string constant or composed of combinations of vicinity functions and string literal constants; and the resulting file exists (possibly with ".scm" appended).

The following function parses an \textit{Info} Index. \footnote{Although it will work on large info files, feeding it an excerpt is much faster; and has less chance of being confused by unusual text in the info file. This command excerpts the SLIB index into \texttt{'slib-index.info'}: \begin{verbatim} info -f slib2d6.info -n "Index" -o slib-index.info \end{verbatim}}

\begin{verbatim}
exports<-info-index file n ...
\end{verbatim}

\textit{n} . . . must be an increasing series of positive integers. \texttt{exports<-info-index} returns a list of all the identifiers appearing in the \textit{n}th . . . (info) indexes of \textit{file}. The identifiers have the case that the implementation's \texttt{read} uses for symbols. Identifiers containing spaces (eg. \texttt{close-base on base-table}) are not included. \texttt{#f} is returned if the index is not found.

Each info index is headed by a \texttt{'* Menu:'} line. To list the symbols in the first and third info indexes do:

\begin{verbatim}
(exports<-info-index "slib.info" 1 3)
\end{verbatim}

\subsection*{1.6.5 Module Analysis}

\begin{verbatim}
(require 'vet)
\end{verbatim}

\begin{verbatim}
vet-slib file1 ... [Function]
Using the procedures in the \texttt{top-refs} and \texttt{manifest} modules, \texttt{vet-slib} analyzes each SLIB module and \textit{file1}, . . ., reporting about any procedure or macro defined whether it is:

orphaned defined, not called, not exported;
missing called, not defined, and not exported by its \texttt{required} modules;
undocumented-export
Exported by module, but no index entry in \texttt{'slib.info'};

And for the library as a whole:

documented-unexport
Index entry in \texttt{'slib.info'}, but no module exports it.

This straightforward analysis caught three full days worth of never-executed branches, transitive require assumptions, spelling errors, undocumented procedures, missing procedures, and cyclic dependencies in SLIB.

The optional arguments \textit{file1}, . . . provide a simple way to vet prospective SLIB modules.

Chapter 2: Universal SLIB Procedures

2 Universal SLIB Procedures

The procedures described in these sections are supported by all implementations as part of the '*.init' files or by 'require.scm'.

2.1 Vicinity

A vicinity is a descriptor for a place in the file system. Vicinities hide from the programmer the concepts of host, volume, directory, and version. Vicinities express only the concept of a file environment where a file name can be resolved to a file in a system independent manner. Vicinities can even be used on flat file systems (which have no directory structure) by having the vicinity express constraints on the file name.

All of these procedures are file-system dependent. Use of these vicinity procedures can make programs file-system independent.

These procedures are provided by all implementations. On most systems a vicinity is a string.

- **make-vicinity dirpath**  
  Returns `dirpath` as a vicinity for use as first argument to `in-vicinity`.

- **pathname->vicinity path**  
  Returns the vicinity containing `path`.

  ```scheme
  (pathname->vicinity "/usr/local/lib/scm/Link.scm")
  ⇒ "/usr/local/lib/scm/
  ```

- **program-vicinity**  
  Returns the vicinity of the currently loading Scheme code. For an interpreter this would be the directory containing source code. For a compiled system (with multiple files) this would be the directory where the object or executable files are. If no file is currently loading, then the result is undefined. **Warning:** `program-vicinity` can return incorrect values if your program escapes back into a load continuation.

- **library-vicinity**  
  Returns the vicinity of the shared Scheme library.

- **implementation-vicinity**  
  Returns the vicinity of the underlying Scheme implementation. This vicinity will likely contain startup code and messages and a compiler.

- **user-vicinity**  
  Returns the vicinity of the current directory of the user. On most systems this is `""` (the empty string).

- **home-vicinity**  
  Returns the vicinity of the user’s `HOME` directory, the directory which typically contains files which customize a computer environment for a user. If scheme is running without a user (eg. a daemon) or if this concept is meaningless for the platform, then `home-vicinity` returns `#f`. 

vicinity:suffix? chr
Returns the ‘#t’ if chr is a vicinity suffix character; and #f otherwise. Typical vicinity suffixes are ‘/’, ‘:’, and ‘\’.

in-vicinity vicinity filename
Returns a filename suitable for use by slib:load, slib:load-source, slib:load-compiled, open-input-file, open-output-file, etc. The returned filename is filename in vicinity. in-vicinity should allow filename to override vicinity when filename is an absolute pathname and vicinity is equal to the value of (user-vicinity). The behavior of in-vicinity when filename is absolute and vicinity is not equal to the value of (user-vicinity) is unspecified. For most systems in-vicinity can be string-append.

sub-vicinity vicinity name
Returns the vicinity of vicinity restricted to name. This is used for large systems where names of files in subsystems could conflict. On systems with directory structure sub-vicinity will return a pathname of the subdirectory name of vicinity.

with-load-pathname path thunk
path should be a string naming a file being read or loaded. with-load-pathname evaluates thunk in a dynamic scope where an internal variable is bound to path; the internal variable is used for messages and program-vicinity. with-load-pathname returns the value returned by thunk.

2.2 Configuration
These constants and procedures describe characteristics of the Scheme and underlying operating system. They are provided by all implementations.

char-code-limit
An integer 1 larger that the largest value which can be returned by char->integer.

most-positive-fixnum
In implementations which support integers of practically unlimited size, most-positive-fixnum is a large exact integer within the range of exact integers that may result from computing the length of a list, vector, or string.
In implementations which do not support integers of practically unlimited size, most-positive-fixnum is the largest exact integer that may result from computing the length of a list, vector, or string.

slib:tab
The tab character.

slib:form-feed
The form-feed character.

software-type
Returns a symbol denoting the generic operating system type. For instance, unix, vms, macos, amiga, or ms-dos.
### 2.3 Input/Output

These procedures are provided by all implementations.

#### file-exists? filename

Returns `#t` if the specified file exists. Otherwise, returns `#f`. If the underlying implementation does not support this feature then `#f` is always returned.
delete-file filename

[Function]
Deletes the file specified by filename. If filename cannot be deleted, #f is returned. Otherwise, #t is returned.

open-file filename modes

[Function]
filename should be a string naming a file. open-file returns a port depending on the symbol modes:

- r: an input port capable of delivering characters from the file.
- rb: a binary input port capable of delivering characters from the file.
- w: an output port capable of writing characters to a new file by that name.
- wb: a binary output port capable of writing characters to a new file by that name.

If an implementation does not distinguish between binary and non-binary files, then it must treat rb as r and wb as w.

If the file cannot be opened, either #f is returned or an error is signalled. For output, if a file with the given name already exists, the effect is unspecified.

port? obj

[Function]
Returns #t if obj is an input or output port, otherwise returns #f.

close-port port

[Procedure]
Closes the file associated with port, rendering the port incapable of delivering or accepting characters.

close-file has no effect if the file has already been closed. The value returned is unspecified.

call-with-open-ports proc ports ...

[Function]
call-with-open-ports ports . . . proc

[Function]
Proc should be a procedure that accepts as many arguments as there are ports passed to call-with-open-ports. call-with-open-ports calls proc with ports . . . . If proc returns, then the ports are closed automatically and the value yielded by the proc is returned. If proc does not return, then the ports will not be closed automatically unless it is possible to prove that the ports will never again be used for a read or write operation.

tmpnam

[Function]
Returns a pathname for a file which will likely not be used by any other process. Successive calls to (tmpnam) will return different pathnames.

current-error-port

[Function]
Returns the current port to which diagnostic and error output is directed.

force-output

[Procedure]
force-output port

[Procedure]
Forces any pending output on port to be delivered to the output device and returns an unspecified value. The port argument may be omitted, in which case it defaults to the value returned by (current-output-port).
file-position port

Function
port must be open to a file. file-position returns the current position of the character in port which will next be read or written. If the implementation does not support file-position, then #f is returned.

file-position port #f

Function
port must be open to a file. file-position sets the current position in port which will next be read or written. If successful, #t is returned; otherwise file-position returns #f.

2.4 System

These procedures are provided by all implementations.

slib:load-source name

Procedure
Loads a file of Scheme source code from name with the default filename extension used in SLIB. For instance if the filename extension used in SLIB is ’.scm’ then (slib:load-source "foo") will load from file ‘foo.scm’.

slib:load-compiled name

Procedure
On implementations which support separately loadable compiled modules, loads a file of compiled code from name with the implementation’s filename extension for compiled code appended.

slib:load name

Procedure
Loads a file of Scheme source or compiled code from name with the appropriate suffixes appended. If both source and compiled code are present with the appropriate names then the implementation will load just one. It is up to the implementation to choose which one will be loaded.

If an implementation does not support compiled code then slab:load will be identical to slab:load-source.

slib:eval obj

Procedure
eval returns the value of obj evaluated in the current top level environment. Section 7.4.11 [Eval], page 250 provides a more general evaluation facility.

slib:eval-load filename eval

Procedure
filename should be a string. If filename names an existing file, the Scheme source code expressions and definitions are read from the file and eval called with them
sequentially. The `slib:eval-load` procedure does not affect the values returned by `current-input-port`, `current-error-port`, and `current-output-port`.

`slib:warn arg1 arg2 ...`  
Outputs a warning message containing the arguments.

`slib:error arg1 arg2 ...`  
Outputs an error message containing the arguments, aborts evaluation of the current form and responds in a system dependent way to the error. Typical responses are to abort the program or to enter a read-eval-print loop.

`slib:exit n`  
`slib:exit`  
Exits from the Scheme session returning status `n` to the system. If `n` is omitted or `#t`, a success status is returned to the system (if possible). If `n` is `#f` a failure is returned to the system (if possible). If `n` is an integer, then `n` is returned to the system (if possible). If the Scheme session cannot exit, then an unspecified value is returned from `slib:exit`.

`browse-url url`  
Web browsers have become so ubiquitous that programming languages should support a uniform interface to them.

If a browser is running, `browse-url` causes the browser to display the page specified by string `url` and returns `#t`.

If the browser is not running, `browse-url` starts a browser displaying the argument `url`. If the browser starts as a background job, `browse-url` returns `#t` immediately; if the browser starts as a foreground job, then `browse-url` returns `#t` when the browser exits; otherwise (if no browser) it returns `#f`.

2.5 Miscellany

These procedures are provided by all implementations.

`identity x`  
`identity` returns its argument.

Example:

```
(identity 3)  
⇒ 3
(identity '(foo bar))  
⇒ (foo bar)
(map identity lst)  
≡ (copy-list lst)
```

2.5.1 Mutual Exclusion

An `exchanger` is a procedure of one argument regulating mutually exclusive access to a resource. When an exchanger is called, its current content is returned, while being replaced by its argument in an atomic operation.
make-exchanger obj

Returns a new exchanger with the argument obj as its initial content.

(define queue (make-exchanger (list a)))

A queue implemented as an exchanger holding a list can be protected from reentrant execution thus:

(define (pop queue)
  (let ((lst #f))
    (dynamic-wind
      (lambda () (set! lst (queue #f)))
      (lambda () (and lst (not (null? lst))
        (let ((ret (car lst)))
          (set! lst (cdr lst))
          ret)))))

(pop queue)  ⇒  a

(pop queue)  ⇒  #f

2.5.2 Legacy

The following procedures were present in Scheme until R4RS (see Section “Language changes” in Revised(4) Scheme). They are provided by all SLIB implementations.

t

[Constant]

Defined as #t.

nil

[Constant]

Defined as #f.

last-pair l

[Function]

Returns the last pair in the list l. Example:

(last-pair (cons 1 2))
  ⇒  (1 . 2)

(last-pair '(1 2))
  ⇒  (2)
  ≡  (cons 2 '())
3 Scheme Syntax Extension Packages

3.1 Defmacro

Defmacros are supported by all implementations.

`gentemp`  [Function]

Returns a new (interned) symbol each time it is called. The symbol names are implementation-dependent.

```
(gentemp) ⇒ scm:G0
(gentemp) ⇒ scm:G1
```

`defmacro:eval e`  [Function]

Returns the `slib:eval` of expanding all defmacros in scheme expression `e`.

`defmacro:load filename`  [Function]

`filename` should be a string. If `filename` names an existing file, the `defmacro:load` procedure reads Scheme source code expressions and definitions from the file and evaluates them sequentially. These source code expressions and definitions may contain defmacro definitions. The `defmacro:load` procedure does not affect the values returned by `current-input-port`, `current-error-port`, and `current-output-port`.

`defmacro? sym`  [Function]

Returns `#t` if `sym` has been defined by `defmacro`, `#f` otherwise.

`macroexpand-1 form`  [Function]

`macroexpand form`  [Function]

If `form` is a macro call, `macroexpand-1` will expand the macro call once and return it. A `form` is considered to be a macro call only if it is a cons whose `car` is a symbol for which a `defmacro` has been defined.

`macroexpand` is similar to `macroexpand-1`, but repeatedly expands `form` until it is no longer a macro call.

`defmacro name lambda-list form . . .`  [Macro]

When encountered by `defmacro:eval`, `defmacro:macroexpand*`, or `defmacro:load` defines a new macro which will henceforth be expanded when encountered by `defmacro:eval`, `defmacro:macroexpand*`, or `defmacro:load`.

3.1.1 Defmacroexpand

`(require 'defmacroexpand)`

`defmacro:expand* e`  [Function]

Returns the result of expanding all defmacros in scheme expression `e`. 
3.2 R4RS Macros

(require 'macro) is the appropriate call if you want R4RS high-level macros but don’t care about the low level implementation. If an SLIB R4RS macro implementation is already loaded it will be used. Otherwise, one of the R4RS macros implementations is loaded.

The SLIB R4RS macro implementations support the following uniform interface:

**Function**

macro:expand sexpression

Takes an R4RS expression, macro-expands it, and returns the result of the macro expansion.

**Function**

macro:eval sexpression

Takes an R4RS expression, macro-expands it, evals the result of the macro expansion, and returns the result of the evaluation.

**Procedure**

macro:load filename

filename should be a string. If filename names an existing file, the macro:load procedure reads Scheme source code expressions and definitions from the file and evaluates them sequentially. These source code expressions and definitions may contain macro definitions. The macro:load procedure does not affect the values returned by current-input-port, current-error-port, and current-output-port.

3.3 Macro by Example

(require 'macro-by-example)

A vanilla implementation of Macro by Example (Eugene Kohlbecker, R4RS) by Dorai Sitaram, (dorai @ cs.rice.edu) using defmacro.

- generating hygienic global define-syntax Macro-by-Example macros cheaply.
- can define macros which use ....
- needn’t worry about a lexical variable in a macro definition clashing with a variable from the macro use context
- don’t suffer the overhead of redefining the repl if defmacro natively supported (most implementations)

3.3.1 Caveat

These macros are not referentially transparent (see Section “Macros” in Revised(4) Scheme). Lexically scoped macros (i.e., let-syntax and letrec-syntax) are not supported. In any case, the problem of referential transparency gains poignancy only when let-syntax and letrec-syntax are used. So you will not be courting large-scale disaster unless you’re using system-function names as local variables with unintuitive bindings that the macro can’t use. However, if you must have the full r4rs macro functionality, look to the more featureful (but also more expensive) versions of syntax-rules available in slib Section 3.4 [Macros That Work], page 20, Section 3.5 [Syntactic Closures], page 23, and Section 3.6 [Syntax-Case Macros], page 30.

define-syntax keyword transformer-spec

The keyword is an identifier, and the transformer-spec should be an instance of syntax-rules.
Chapter 3: Scheme Syntax Extension Packages

The top-level syntactic environment is extended by binding the keyword to the specified transformer.

```
(define-syntax let*
  (syntax-rules ()
    ((let* () body1 body2 ...) (let () body1 body2 ...))
    ((let* ((name1 val1) (name2 val2) ...) body1 body2 ...)
     (let ((name1 val1))
      (let* ((name2 val2) ...) body1 body2 ...)))
)
```

**syntax-rules** literals syntax-rule ...  

*literals* is a list of identifiers, and each *syntax-rule* should be of the form

```
(pattern template)
```

where the *pattern* and *template* are as in the grammar above.

An instance of *syntax-rules* produces a new macro transformer by specifying a sequence of hygienic rewrite rules. A use of a macro whose keyword is associated with a transformer specified by *syntax-rules* is matched against the patterns contained in the *syntax-rules*, beginning with the leftmost *syntax-rule*. When a match is found, the macro use is transcribed hygienically according to the template.

Each pattern begins with the keyword for the macro. This keyword is not involved in the matching and is not considered a pattern variable or literal identifier.

### 3.4 Macros That Work

(\texttt{require \textquoteleft macros-that-work\textquoteright})

*Macros That Work* differs from the other R4RS macro implementations in that it does not expand derived expression types to primitive expression types.

**macwork:expand** \texttt{expression}  

Takes an R4RS expression, macro-expands it, and returns the result of the macro expansion.

**macro:eval** \texttt{expression}  

*macro:eval* returns the value of \texttt{expression} in the current top level environment. \texttt{expression} can contain macro definitions. Side effects of \texttt{expression} will affect the top level environment.

**macwork:load** \texttt{filename}  

\texttt{filename} should be a string. If \texttt{filename} names an existing file, the \texttt{macwork:load} procedure reads Scheme source code expressions and definitions from the file and evaluates them sequentially. These source code expressions and definitions may contain
macro definitions. The \texttt{macro:load} procedure does not affect the values returned by \texttt{current-input-port}, \texttt{current-error-port}, and \texttt{current-output-port}.

References:

The \textit{Revised}^4 \textit{Report on the Algorithmic Language Scheme} Clinger and Rees [editors]. To appear in LISP Pointers. Also available as a technical report from the University of Oregon, MIT AI Lab, and Cornell.


The supported syntax differs from the R4RS in that vectors are allowed as patterns and as templates and are not allowed as pattern or template data.

\[
\text{transformer spec} \mapsto (\text{syntax-rules literals rules})
\]

\[
\text{rules} \mapsto ()
\|
\text{rule} . \text{rules}
\]

\[
\text{rule} \mapsto (\text{pattern template})
\]

\[
\text{pattern} \mapsto \text{pattern}\_\text{var} \quad ; \text{a symbol not in literals}
\|
\text{symbol} \quad ; \text{a symbol in literals}
\|
()\|
(\text{pattern} . \text{pattern})\|
(\text{ellipsis}\_\text{pattern})\|
#(\text{pattern}*) \quad ; \text{extends R4RS}
\|
#(\text{pattern}^\star \text{ellipsis}\_\text{pattern}) \quad ; \text{extends R4RS}
\|
\text{pattern}\_\text{datum}
\]

\[
\text{template} \mapsto \text{pattern}\_\text{var}
\|
\text{symbol}
\|
()
\|
(\text{template}^\star \text{template})\|
#(\text{template}*) \quad ; \text{extends R4RS}
\|
\text{pattern}\_\text{datum}
\]

\[
\text{template}^\star \mapsto \text{template}
\|
\text{ellipsis}\_\text{template}
\]

\[
\text{pattern}\_\text{datum} \mapsto \text{string} \quad ; \text{no vector}
\|
\text{character}
\|
\text{boolean}
\|
\text{number}
\]

\[
\text{ellipsis}\_\text{pattern} \mapsto \text{pattern} \ldots
\]

\[
\text{ellipsis}\_\text{template} \mapsto \text{template} \ldots
\]

\[
\text{pattern}\_\text{var} \mapsto \text{symbol} \quad ; \text{not in literals}
\]
3.4.1 Definitions

Scope of an ellipsis
Within a pattern or template, the scope of an ellipsis (...) is the pattern or template that appears to its left.

Rank of a pattern variable
The rank of a pattern variable is the number of ellipses within whose scope it appears in the pattern.

Rank of a subtemplate
The rank of a subtemplate is the number of ellipses within whose scope it appears in the template.

Template rank of an occurrence of a pattern variable
The template rank of an occurrence of a pattern variable within a template is the rank of that occurrence, viewed as a subtemplate.

Variables bound by a pattern
The variables bound by a pattern are the pattern variables that appear within it.

Referenced variables of a subtemplate
The referenced variables of a subtemplate are the pattern variables that appear within it.

Variables opened by an ellipsis template
The variables opened by an ellipsis template are the referenced pattern variables whose rank is greater than the rank of the ellipsis template.

3.4.2 Restrictions

No pattern variable appears more than once within a pattern.

For every occurrence of a pattern variable within a template, the template rank of the occurrence must be greater than or equal to the pattern variable’s rank.

Every ellipsis template must open at least one variable.

For every ellipsis template, the variables opened by an ellipsis template must all be bound to sequences of the same length.

The compiled form of a rule is

rule \(\mapsto\) (pattern template inserted)

pattern \(\mapsto\) pattern_var
| symbol
| ()
| (pattern . pattern)
| ellipsis_pattern
| #(pattern)  
| pattern_datum

```template  \mapsto \rightarrow \begin{align*} & \text{pattern_var} \\
& | \text{symbol} \\
& | () \\
& | (\text{template2} . \text{template2}) \\
& | #(pattern) \\
& | pattern_datum \end{align*}```

```template2  \mapsto \rightarrow \begin{align*} & \text{template} \\
& | \text{ellipsis_template} \end{align*}```

```pattern_datum  \mapsto \rightarrow \begin{align*} & \text{string} \\
& | \text{character} \\
& | \text{boolean} \\
& | \text{number} \end{align*}```

```pattern_var  \mapsto \rightarrow \begin{align*} & #(V \text{ symbol rank}) \end{align*}```

```ellipsis_pattern  \mapsto \rightarrow \begin{align*} & #(E \text{ pattern pattern_vars}) \end{align*}```

```ellipsis_template  \mapsto \rightarrow \begin{align*} & #(E \text{ template pattern_vars}) \end{align*}```

```inserted  \mapsto \rightarrow \begin{align*} & () \\
& | (\text{symbol} . \text{inserted}) \end{align*}```

```pattern_vars  \mapsto \rightarrow \begin{align*} & () \\
& | (\text{pattern_var} . \text{pattern_vars}) \end{align*}```

```rank  \mapsto \rightarrow \begin{align*} & \text{exact non-negative integer} \end{align*}```

where V and E are unforgeable values.

The pattern variables associated with an ellipsis pattern are the variables bound by the pattern, and the pattern variables associated with an ellipsis template are the variables opened by the ellipsis template.

If the template contains a big chunk that contains no pattern variables or inserted identifiers, then the big chunk will be copied unnecessarily. That shouldn’t matter very often.

### 3.5 Syntactic Closures

```
(\text{require} \ '\text{syntactic-closures}')
```

**macro:expand** expression  [Function]

**synclo:expand** expression  [Function]

Returns scheme code with the macros and derived expression types of expression expanded to primitive expression types.
**macrol:eval** expression

**synclo:eval** expression

`macrol:eval` returns the value of `expression` in the current top level environment. `expression` can contain macro definitions. Side effects of `expression` will affect the top level environment.

**macrol:load** filename

**synclo:load** filename

`filename` should be a string. If `filename` names an existing file, the `macrol:load` procedure reads Scheme source code expressions and definitions from the file and evaluates them sequentially. These source code expressions and definitions may contain macro definitions. The `macrol:load` procedure does not affect the values returned by `current-input-port`, `current-error-port`, and `current-output-port`.

### 3.5.1 Syntactic Closure Macro Facility

A Syntactic Closures Macro Facility

by Chris Hanson

9 November 1991

This document describes *syntactic closures*, a low-level macro facility for the Scheme programming language. The facility is an alternative to the low-level macro facility described in the *Revised^4 Report on Scheme*. This document is an addendum to that report.

The syntactic closures facility extends the BNF rule for `transformer spec` to allow a new keyword that introduces a low-level macro transformer:

```
transformer spec := (transformer expression)
```

Additionally, the following procedures are added:

- `make-syntactic-closure`
- `capture-syntactic-environment`
- `identifier?`
- `identifier=?`

The description of the facility is divided into three parts. The first part defines basic terminology. The second part describes how macro transformers are defined. The third part describes the use of `identifiers`, which extend the syntactic closure mechanism to be compatible with `syntax-rules`.

### 3.5.1.1 Terminology

This section defines the concepts and data types used by the syntactic closures facility.

- **Forms** are the syntactic entities out of which programs are recursively constructed. A form is any expression, any definition, any syntactic keyword, or any syntactic closure. The variable name that appears in a `set!` special form is also a form. Examples of forms:

  ```scheme
  17
  #t
  car
  (+ x 4)
  (lambda (x) x)
  ```
(define pi 3.14159)

if
define

- An alias is an alternate name for a given symbol. It can appear anywhere in a form that the symbol could be used, and when quoted it is replaced by the symbol; however, it does not satisfy the predicate symbol?. Macro transformers rarely distinguish symbols from aliases, referring to both as identifiers.

- A syntactic environment maps identifiers to their meanings. More precisely, it determines whether an identifier is a syntactic keyword or a variable. If it is a keyword, the meaning is an interpretation for the form in which that keyword appears. If it is a variable, the meaning identifies which binding of that variable is referenced. In short, syntactic environments contain all of the contextual information necessary for interpreting the meaning of a particular form.

- A syntactic closure consists of a form, a syntactic environment, and a list of identifiers. All identifiers in the form take their meaning from the syntactic environment, except those in the given list. The identifiers in the list are to have their meanings determined later. A syntactic closure may be used in any context in which its form could have been used. Since a syntactic closure is also a form, it may not be used in contexts where a form would be illegal. For example, a form may not appear as a clause in the cond special form. A syntactic closure appearing in a quoted structure is replaced by its form.

3.5.1.2 Transformer Definition

This section describes the transformer special form and the procedures make-syntactic-closure and capture-syntactic-environment.

transformer expression

Syntax: It is an error if this syntax occurs except as a transformer spec.

Semantics: The expression is evaluated in the standard transformer environment to yield a macro transformer as described below. This macro transformer is bound to a macro keyword by the special form in which the transformer expression appears (for example, let-syntax).

A macro transformer is a procedure that takes two arguments, a form and a syntactic environment, and returns a new form. The first argument, the input form, is the form in which the macro keyword occurred. The second argument, the usage environment, is the syntactic environment in which the input form occurred. The result of the transformer, the output form, is automatically closed in the transformer environment, which is the syntactic environment in which the transformer expression occurred.

For example, here is a definition of a push macro using syntax-rules:

(define-syntax push
  (syntax-rules ()
    ((push item list)
      (set! list (cons item list)))))

Here is an equivalent definition using transformer:
(define-syntax push
  (transformer
   (lambda (exp env)
     (let ((item
            (make-syntactic-closure env () (cadr exp)))
           (list
            (make-syntactic-closure env () (caddr exp))))
      '(set! ,list (cons ,item ,list))))))

In this example, the identifiers set! and cons are closed in the transformer environment, and thus will not be affected by the meanings of those identifiers in the usage environment env.

Some macros may be non-hygienic by design. For example, the following defines a loop macro that implicitly binds exit to an escape procedure. The binding of exit is intended to capture free references to exit in the body of the loop, so exit must be left free when the body is closed:

(define-syntax loop
  (transformer
   (lambda (exp env)
     (let ((body (cdr exp)))
       '(call-with-current-continuation
         (lambda (exit)
           (let f ()
             ,@(map (lambda (exp)
               (make-syntactic-closure env '(exit)
                exp))
             body)
           (f))))))))

To assign meanings to the identifiers in a form, use make-syntactic-closure to close the form in a syntactic environment.

make-syntactic-closure environment free-names form

[Function]

environment must be a syntactic environment, free-names must be a list of identifiers, and form must be a form. make-syntactic-closure constructs and returns a syntactic closure of form in environment, which can be used anywhere that form could have been used. All the identifiers used in form, except those explicitly excepted by free-names, obtain their meanings from environment.

Here is an example where free-names is something other than the empty list. It is instructive to compare the use of free-names in this example with its use in the loop example above: the examples are similar except for the source of the identifier being left free.

(define-syntax let1
  (transformer
   (lambda (exp env)
     (let ((id (cadr exp))
           (init (caddr exp))
           (f))))))
let1 is a simplified version of let that only binds a single identifier, and whose body consists of a single expression. When the body expression is syntactically closed in its original syntactic environment, the identifier that is to be bound by let1 must be left free, so that it can be properly captured by the lambda in the output form.

To obtain a syntactic environment other than the usage environment, use capture-syntactic-environment.

capture-syntactic-environment procedure

[Function]
capture-syntactic-environment returns a form that will, when transformed, call procedure on the current syntactic environment. procedure should compute and return a new form to be transformed, in that same syntactic environment, in place of the form.

An example will make this clear. Suppose we wanted to define a simple loop-until keyword equivalent to

\[
\text{(define-syntax loop-until}
\begin{aligned}
&\text{(syntax-rules ()}
&\text{((loop-until id init test return step)}
&\text{(letrec ((loop}
&\text{\quad (lambda (id)}
&\text{\quad \quad (if test return \text{(loop step)}))))}
&\text{\quad (loop init))}))
\end{aligned}
\]

The following attempt at defining loop-until has a subtle bug:

\[
\text{(define-syntax loop-until}
\begin{aligned}
&\text{(transformer}
&\text{\quad (lambda (exp env)}
&\text{\quad \quad (let ((id (cadr exp)))}
&\text{\quad \quad \quad (init (cadddr exp)))}
&\text{\quad \quad (test (cadddr exp)))}
&\text{\quad \quad (return (cadddr (cdr exp)))}
&\text{\quad \quad (step (cadddr (cddr exp)))}
&\text{\quad \quad (close}
&\text{\quad \quad \quad (lambda (exp free)}
&\text{\quad \quad \quad \quad (make-syntactic-closure env free exp))))}
&\text{\quad \quad \quad ‘(letrec ((loop}
&\text{\quad \quad \quad \quad \quad (lambda (,id)}
&\text{\quad \quad \quad \quad \quad \quad \quad (if ,\text{(close test (list id))}
&\text{\quad \quad \quad \quad \quad \quad \quad \quad ,\text{(close return (list id))}
&\text{\quad \quad \quad \quad \quad \quad \quad \quad \quad (loop ,\text{(close step (list id))}))))}
&\text{\quad \quad \quad \quad \quad \quad \quad (loop ,\text{(close init ‘()})))}))
\end{aligned}
\]

This definition appears to take all of the proper precautions to prevent unintended captures. It carefully closes the subexpressions in their original syntactic environment.
and it leaves the \texttt{id} identifier free in the \texttt{test}, \texttt{return}, and \texttt{step} expressions, so that it will be captured by the binding introduced by the \texttt{lambda} expression. Unfortunately it uses the identifiers \texttt{if} and \texttt{loop} within that \texttt{lambda} expression, so if the user of \texttt{loop-until} just happens to use, say, \texttt{if} for the identifier, it will be inadvertently captured.

The syntactic environment that \texttt{if} and \texttt{loop} want to be exposed to is the one just outside the \texttt{lambda} expression: before the user’s identifier is added to the syntactic environment, but after the identifier \texttt{loop} has been added. \texttt{capture-syntactic-environment} captures exactly that environment as follows:

\begin{verbatim}
(define-syntax loop-until
  (transformer
   (lambda (exp env)
     (let ((id (cadr exp))
           (init (caddr exp))
           (test (cadddr exp))
           (return (cadddr (cdr exp)))
           (step (cadddr (cddr exp)))
           (close
            (lambda (exp free)
              (make-syntactic-closure env free exp))))
     ,(letrec ((loop ,((capture-syntactic-environment
                      (lambda (env)
                        ,(lambda (,id)
                         ,(make-syntactic-closure env '() 'if)
                         ,(close test (list id))
                         ,(close return (list id))
                         ,(make-syntactic-closure env '()
                         ,(make-syntactic-closure env '()
                         ,(make-syntactic-closure env '()
                         ,(make-syntactic-closure env '()
                         ,(close step (list id)))))))))
         (loop ,((close init '()))))))

In this case, having captured the desired syntactic environment, it is convenient to construct syntactic closures of the identifiers \texttt{if} and the \texttt{loop} and use them in the body of the \texttt{lambda}.

A common use of \texttt{capture-syntactic-environment} is to get the transformer environment of a macro transformer:

\begin{verbatim}
(transformer
 (lambda (exp env)
   (capture-syntactic-environment
    (lambda (transformer-env)
      (...))))
\end{verbatim}

3.5.1.3 Identifiers

This section describes the procedures that create and manipulate identifiers. Previous syntactic closure proposals did not have an identifier data type – they just used symbols.
The identifier data type extends the syntactic closures facility to be compatible with the high-level \texttt{syntax-rules} facility.

As discussed earlier, an identifier is either a symbol or an \texttt{alias}. An alias is implemented as a syntactic closure whose \texttt{form} is an identifier:

\[
\text{make-syntactic-closure \ env \ '() \ 'a) \\
\Rightarrow \text{an alias}
\]

Aliases are implemented as syntactic closures because they behave just like syntactic closures most of the time. The difference is that an alias may be bound to a new value (for example by \texttt{lambda} or \texttt{let-syntax}); other syntactic closures may not be used this way. If an alias is bound, then within the scope of that binding it is looked up in the syntactic environment just like any other identifier.

Aliases are used in the implementation of the high-level facility \texttt{syntax-rules}. A macro transformer created by \texttt{syntax-rules} uses a template to generate its output form, substituting subforms of the input form into the template. In a syntactic closures implementation, all of the symbols in the template are replaced by aliases closed in the transformer environment, while the output form itself is closed in the usage environment. This guarantees that the macro transformation is hygienic, without requiring the transformer to know the syntactic roles of the substituted input subforms.

\textbf{Identifier?} \texttt{object} \\
Returns \#t if \texttt{object} is an identifier, otherwise returns \#f. Examples:

\[
\begin{align*}
(\text{identifier?} \ 'a) & \Rightarrow \#t \\
(\text{identifier?} \ (\text{make-syntactic-closure} \ \text{env} \ '() \ 'a)) & \Rightarrow \#t \\
(\text{identifier?} \ "a") & \Rightarrow \#f \\
(\text{identifier?} \ #\a) & \Rightarrow \#f \\
(\text{identifier?} \ 97) & \Rightarrow \#f \\
(\text{identifier?} \ #f) & \Rightarrow \#f \\
(\text{identifier?} \ '(a)) & \Rightarrow \#f \\
(\text{identifier?} \ '#(a)) & \Rightarrow \#f
\end{align*}
\]

The predicate \texttt{eq?} is used to determine if two identifiers are “the same”. Thus \texttt{eq?} can be used to compare identifiers exactly as it would be used to compare symbols. Often, though, it is useful to know whether two identifiers “mean the same thing”. For example, the \texttt{cond} macro uses the symbol \texttt{else} to identify the final clause in the conditional. A macro transformer for \texttt{cond} cannot just look for the symbol \texttt{else}, because the \texttt{cond} form might be the output of another macro transformer that replaced the symbol \texttt{else} with an alias. Instead the transformer must look for an identifier
that “means the same thing” in the usage environment as the symbol `else` means in the transformer environment.

```
(identifier=? environment1 identifier1 environment2 identifier2)   [Function]
```

`environment1` and `environment2` must be syntactic environments, and `identifier1` and `identifier2` must be identifiers. `identifier=?` returns `#t` if the meaning of `identifier1` in `environment1` is the same as that of `identifier2` in `environment2`, otherwise it returns `#f`. Examples:

```
(let-syntax
  ((foo
    (transformer
     (lambda (form env)
      (capture-syntactic-environment
       (lambda (transformer-env)
        (identifier=? transformer-env 'x env 'x))))))
   (list (foo)
   (let ((x 3))
     (foo))))
⇒ (#t #f)

(let-syntax ((bar foo))
  (let-syntax
    ((foo
      (transformer
       (lambda (form env)
        (capture-syntactic-environment
         (lambda (transformer-env)
          (identifier=? transformer-env 'foo env (cadr form))))))))
    (list (foo foo)
      (foobar)))
⇒ (#f #t)
```

### 3.5.1.4 Acknowledgements

The syntactic closures facility was invented by Alan Bawden and Jonathan Rees. The use of aliases to implement `syntax-rules` was invented by Alan Bawden (who prefers to call them `synthetic names`). Much of this proposal is derived from an earlier proposal by Alan Bawden.

### 3.6 Syntax-Case Macros

```
(require 'syntax-case)
```

```
macro:expand expression        [Function]
synccase:expand expression     [Function]
```

Returns scheme code with the macros and derived expression types of `expression` expanded to primitive expression types.
**Chapter 3: Scheme Syntax Extension Packages**

**Function**

`macro:eval  expression`  

`syncase:eval  expression`

`macro:eval` returns the value of `expression` in the current top level environment. `expression` can contain macro definitions. Side effects of `expression` will affect the top level environment.

**Procedure**

`macro:load  filename`

`syncase:load  filename`

`filename` should be a string. If `filename` names an existing file, the `macro:load` procedure reads Scheme source code expressions and definitions from the file and evaluates them sequentially. These source code expressions and definitions may contain macro definitions. The `macro:load` procedure does not affect the values returned by `current-input-port`, `current-error-port`, and `current-output-port`.

This is version 2.1 of `syntax-case`, the low-level macro facility proposed and implemented by Robert Hieb and R. Kent Dybvig.

This version is further adapted by Harald Hanche-Olsen <hanche@imf.unit.no> to make it compatible with, and easily usable with, SLIB. Mainly, these adaptations consisted of:

- Removing white space from `expand.pp` to save space in the distribution. This file is not meant for human readers anyway...
- Removed a couple of Chez scheme dependencies.
- Renamed global variables used to minimize the possibility of name conflicts.
- Adding an SLIB-specific initialization file.
- Removing a couple extra files, most notably the documentation (but see below).

If you wish, you can see exactly what changes were done by reading the shell script in the file `syncase.sh`.

The two PostScript files were omitted in order to not burden the SLIB distribution with them. If you do intend to use `syntax-case`, however, you should get these files and print them out on a PostScript printer. They are available with the original `syntax-case` distribution by anonymous FTP in `cs.indiana.edu:/pub/scheme/syntax-case`.

In order to use `syntax-case` from an interactive top level, execute:

```
(require 'syntax-case)
(require 'repl)
(repl:top-level macro:eval)
```

See the section Repl (see Section 7.5.1 [Repl], page 257) for more information.

To check operation of `syntax-case` get `cs.indiana.edu:/pub/scheme/syntax-case`, and type

```
(require 'syntax-case)
(syncase:sanity-check)
```

Beware that `syntax-case` takes a long time to load – about 20s on a SPARCstation SLC (with SCM) and about 90s on a Macintosh SE/30 (with Gambit).
3.6.1 Notes

All R4RS syntactic forms are defined, including delay. Along with delay are simple definitions for make-promise (into which delay expressions expand) and force.

syntax-rules and with-syntax (described in TR356) are defined.

syntax-case is actually defined as a macro that expands into calls to the procedure syntax-dispatch and the core form syntax-lambda; do not redefine these names.

Several other top-level bindings not documented in TR356 are created:

- the “hooks” in ‘hooks.ss’
- the build- procedures in ‘output.ss’
- expand-syntax (the expander)

The syntax of define has been extended to allow (define id), which assigns id to some unspecified value.

We have attempted to maintain R4RS compatibility where possible. The incompatibilities should be confined to ‘hooks.ss’. Please let us know if there is some incompatibility that is not flagged as such.

Send bug reports, comments, suggestions, and questions to Kent Dybvig (dyb@iu-vax.cs.indiana.edu).

3.7 Define-Structure

(require 'structure)

Included with the syntax-case files was ‘structure.scm’ which defines a macro define-structure. Here is its documentation from Gambit-4.0:

define-structure (name field...) [special form]

Record data types similar to Pascal records and C struct types can be defined using the define-structure special form. The identifier name specifies the name of the new data type. The structure name is followed by k identifiers naming each field of the record. The define-structure expands into a set of definitions of the following procedures:

- ‘make-name’ – A k argument procedure which constructs a new record from the value of its k fields.
- ‘name?’ – A procedure which tests if its single argument is of the given record type.
- ‘name-field’ – For each field, a procedure taking as its single argument a value of the given record type and returning the content of the corresponding field of the record.
- ‘name-field-set!’ – For each field, a two argument procedure taking as its first argument a value of the given record type. The second argument gets assigned to the corresponding field of the record and the void object is returned.

Gambit record data types have a printed representation that includes the name of the type and the name and value of each field.

For example:
3.8 Define-Record-Type

(require 'define-record-type) or (require 'srfi-9)


define-record-type <type-name> (<constructor-name> <field-tag> ... <predicate-name> <field-spec> ...)

Where

<field-spec> ≡ (<field-tag> <accessor-name>)
≡ (<field-tag> <accessor-name> <modifier-name>)

define-record-type is a syntax wrapper for the SLIB record module.

3.9 Fluid-Let

(require 'fluid-let)

Note: fluid-let is not thread-safe. It is better to use Section 3.10 [Parameter Objects], page 34 (srfi-39) or Section 7.1.11 [Dynamic Data Type], page 213, both of which will be made thread-safe in the future.

fluid-let (bindings ...) forms...
(fluid-let ((variable init) ...)
  expression expression ...)

The inits are evaluated in the current environment (in some unspecified order), the current values of the variables are saved, the results are assigned to the variables, the expressions are evaluated sequentially in the current environment, the variables are restored to their original values, and the value of the last expression is returned.

The syntax of this special form is similar to that of let, but fluid-let temporarily rebinds existing variables. Unlike let, fluid-let creates no new bindings; instead it assigns the values of each init to the binding (determined by the rules of lexical scoping) of its corresponding variable.
3.10 Parameter Objects

(require 'srfi-39)

3.11 Binding to multiple values

(require 'receive) or (require 'srfi-8)

receive formals expression body . . . [Special Form]

(require 'let-values) or (require 'srfi-11)

let-values ((formals expression) ...) body . . . [Special Form]
let-values* ((formals expression) ...) body . . . [Special Form]

3.12 Guarded LET* special form

(require 'and-let*) or (require 'srfi-2)

and-let* claws body . . . [Macro]

3.13 Guarded COND Clause

(require 'guarded-cond-clause) or (require 'srfi-61)


cond <clause1> <clause2> . . . [library syntax]

Syntax: Each <clause> should be of the form
(<test> <expression1> . . .)

where <test> is any expression. Alternatively, a <clause> may be of the form
(<test> => <expression>)

The <clause> production in the formal syntax of Scheme as written by R5RS in section
7.1.3 is extended with a new option:
<clause> => (<generator> <guard> => <receiver>)

where <generator>, <guard>, & <receiver> are all <expression>s.

Clauses of this form have the following semantics: <generator> is evaluated. It may return arbitrarily many values. <Guard> is applied to an
argument list containing the values in order that <generator> returned. If <guard> returns a true value for that argument list, <receiver> is applied with an equivalent argument list. If <guard> returns a false value, however, the clause is abandoned and the next one is tried.

The last <clause> may be an “else clause,” which has the form

(else <expression1> <expression2> ...).

This port->char-list procedure accepts an input port and returns a list of all the characters it produces until the end.

(define (port->char-list port)
  (cond ((read-char port) char?
         => (lambda (c) (cons c (port->char-list port))))
         (else '())))

(call-with-input-string "foo" port->char-list) ==> (#\f #\o #\o)

3.14 Yasos

(requires 'oop) or (require 'yasos)


Another reference is:

Ken Dickey.
Scheming with Objects
AI Expert Volume 7, Number 10 (October 1992), pp. 24-33.

3.14.1 Terms

Object Any Scheme data object.

Instance An instance of the OO system; an object.

Operation A method.

Notes: The object system supports multiple inheritance. An instance can inherit from 0 or more ancestors. In the case of multiple inherited operations with the same identity, the operation used is that from the first ancestor which contains it (in the ancestor let). An operation may be applied to any Scheme data object—not just instances. As code which creates instances is just code, there are no classes and no meta-anything. Method dispatch is by a procedure call a la CLOS rather than by send syntax a la Smalltalk.
Disclaimer:

There are a number of optimizations which can be made. This implementation is expository (although performance should be quite reasonable). See the L&FP paper for some suggestions.

### 3.14.2 Interface

**define-operation** (opname self arg ...) **default-body**

[Syntax]

Defines a default behavior for data objects which don’t handle the operation opname. The default behavior (for an empty `default-body`) is to generate an error.

**define-predicate** opname?

[Syntax]

Defines a predicate `opname?`, usually used for determining the type of an object, such that `(opname? object)` returns `#t` if `object` has an operation `opname?` and `#f` otherwise.

**object** ((name self arg ...) body) ...

[Syntax]

Returns an object (an instance of the object system) with operations. Invoking `(name object arg ...)` executes the `body` of the `object` with `self` bound to `object` and with argument(s) arg....

**object-with-ancestors** ((ancestor1 init1) ...) operation ...

[Syntax]

A `let`-like form of `object` for multiple inheritance. It returns an object inheriting the behaviour of `ancestor1` etc. An operation will be invoked in an ancestor if the object itself does not provide such a method. In the case of multiple inherited operations with the same identity, the operation used is the one found in the first ancestor in the ancestor list.

**operate-as** component operation self arg ...

[Syntax]

Used in an operation definition (of `self`) to invoke the `operation` in an ancestor `component` but maintain the object’s identity. Also known as “send-to-super”.

**print** obj port

[Procedure]

A default `print` operation is provided which is just `(format port obj)` (see Section 4.2 [Format], page 46) for non-instances and prints `obj` preceded by ‘#<INSTANCE>’ for instances.

**size** obj

[Function]

The default method returns the number of elements in `obj` if it is a vector, string or list, 2 for a pair, 1 for a character and by default id an error otherwise. Objects such as collections (see Section 7.1.10 [Collections], page 210) may override the default in an obvious way.

### 3.14.3 Setters

Setters implement generalized locations for objects associated with some sort of mutable state. A getter operation retrieves a value from a generalized location and the corresponding setter operation stores a value into the location. Only the getter is named – the setter is specified by a procedure call as below. (Dylan uses special syntax.) Typically, but not necessarily, getters are access operations to extract values from Yasos objects (see
Section 3.14 [Yasos], page 35]. Several setters are predefined, corresponding to getters \texttt{car}, \texttt{cdr}, \texttt{string-ref} and \texttt{vector-ref} e.g., \texttt{(setter car)} is equivalent to \texttt{set-car!}.

This implementation of setters is similar to that in Dylan(TM) (\textit{Dylan: An object-oriented dynamic language}, Apple Computer Eastern Research and Technology). Common LISP provides similar facilities through \texttt{setf}.

\begin{verbatim}
(setter getter)
\end{verbatim}

\begin{verbatim}
(set place new-value)
\end{verbatim}

\begin{verbatim}
(add-setter getter setter)
\end{verbatim}

\begin{verbatim}
(remove-setter-for getter)
\end{verbatim}

\begin{verbatim}
(define-access-operation getter-name)
\end{verbatim}

3.14.4 Examples

\begin{verbatim}
;; These definitions for PRINT and SIZE are
;; already supplied by
(require 'yasos)

(define-operation (print obj port)
  (format port
    (if (instance? obj) "#<instance>" "-s")
    obj))
\end{verbatim}
(define-operation (size obj)
  (cond
   ((vector? obj) (vector-length obj))
   ((list? obj) (length obj))
   ((pair? obj) 2)
   ((string? obj) (string-length obj))
   ((char? obj) 1)
   (else
    (slib:error "Operation not supported: size" obj)))))

(define-predicate cell?)
(define-operation (fetch obj))
(define-operation (store! obj newValue))

(define (make-cell value)
  (object
   ((cell? self) #t)
   ((fetch self) value)
   ((store! self newValue)
    (set! value newValue)
    newValue)
   ((size self) 1)
   ((print self port)
    (format port "#<Cell: ~s~s>" (fetch self))))))

(define-operation (discard obj value)
  (format #t "Discarding ~s~s" value))

(define (make-filtered-cell value filter)
  (object-with-ancestors
   ((cell (make-cell value)))
   ((store! self newValue)
    (if (filter newValue)
      (store! cell newValue)
      (discard self newValue)))))

(define-predicate array?)
(define-operation (array-ref array index))
(define-operation (array-set! array index value))

(define (make-array num-slots)
  (let ((anArray (make-vector num-slots)))
    (object
     ((array? self) #t)
     ((size self) num-slots)
     ((array-ref self index)
(vector-ref anArray index)
((array-set! self index newValue)
 (vector-set! anArray index newValue))
((print self port)
 (format port "#<Array ~s>" (size self))))

(define-operation (position obj))
(define-operation (discarded-value obj))

(define (make-cell-with-history value filter size)
  (let ((pos 0) (most-recent-discard #f))
    (object-with-ancestors
     ((cell (make-filtered-call value filter))
      (sequence (make-array size)))
     ((array? self) #f)
     ((position self) pos)
     ((store! self newValue)
      (operate-as cell store! self newValue)
      (array-set! self pos newValue)
      (set! pos (+ pos 1)))
     ((discard self value)
      (set! most-recent-discard value))
     ((discarded-value self) most-recent-discard)
     ((print self port)
      (format port "#<Cell-with-history ~s>" (fetch self)))))

(define-access-operation fetch)
(add-setter fetch store!)
(define foo (make-cell 1))
(print foo #f)
⇒ "#<Cell: 1>"
(set (fetch foo) 2)
⇒
(print foo #f)
⇒ "#<Cell: 2>"
(fetch foo)
⇒ 2
4 Textual Conversion Packages

4.1 Precedence Parsing

(receive 'precedence-parse) or (receive 'parse)

This package implements:

- a Pratt style precedence parser;
- a tokenizer which congeals tokens according to assigned classes of constituent characters;
- procedures giving direct control of parser rulesets;
- procedures for higher level specification of rulesets.

4.1.1 Precedence Parsing Overview

This package offers improvements over previous parsers.

- Common computer language constructs are concisely specified.
- Grammars can be changed dynamically. Operators can be assigned different meanings within a lexical context.
- Rule sets don’t need compilation. Grammars can be changed incrementally.
- Operator precedence is specified by integers.
- All possibilities of bad input are handled\(^1\) and return as much structure as was parsed when the error occurred; The symbol ? is substituted for missing input.

The notion of binding power may be unfamiliar to those accustomed to BNF grammars. When two consecutive objects are parsed, the first might be the prefix to the second, or the second might be a suffix of the first. Comparing the left and right binding powers of the two objects decides which way to interpret them.

Objects at each level of syntactic grouping have binding powers.

A syntax tree is not built unless the rules explicitly do so. The call graph of grammar rules effectively instantiate the syntax tree.

The JACAL symbolic math system (http://people.csail.mit.edu/jaffer/JACAL) uses precedence-parse. Its grammar definitions in the file ‘jacal/English.scm’ can serve as examples of use.

4.1.2 Rule Types

Here are the higher-level syntax types and an example of each. Precedence considerations are omitted for clarity. See Section 4.1.6 [Grammar Rule Definition], page 44 for full details.

```
nofix  bye  exit
    bye
```

[Grammar]

calls the function exit with no arguments.

\(^1\) How do I know this? I parsed 250kbyte of random input (an e-mail file) with a non-trivial grammar utilizing all constructs.
**prefix** - *negate*  
- 42  
Calls the function *negate* with the argument 42.

**infix** - *difference*  
\(x - y\)  
Calls the function *difference* with arguments \(x\) and \(y\).

**nary** + *sum*  
\(x + y + z\)  
Calls the function *sum* with arguments \(x\), \(y\), and \(y\).

**postfix** ! *factorial*  
5 !  
Calls the function *factorial* with the argument 5.

**prestfix** set set!  
set foo bar  
Calls the function set! with the arguments foo and bar.

**commentfix** /* */  
/* almost any text here */  
Ignores the comment delimited by /* and */.

**matchfix** { *list* }  
\{0, 1, 2\}  
Calls the function list with the arguments 0, 1, and 2.

**inmatchfix** ( *funcall* )  
\(f(x, y)\)  
Calls the function funcall with the arguments \(f\), \(x\), and \(y\).

**delim** ;  
set foo bar;  
delimits the extent of the restfix operator set.

### 4.1.3 Ruleset Definition and Use

* **syn-defs**  
A grammar is built by one or more calls to *prec:define-grammar*. The rules are appended to *syn-defs*. The value of *syn-defs* is the grammar suitable for passing as an argument to *prec:parse*.

* **syn-ignore-whitespace**  
Is a nearly empty grammar with whitespace characters set to group 0, which means they will not be made into tokens. Most rulesets will want to start with *syn-ignore-whitespace*
In order to start defining a grammar, either

(set! *syn-defs* '())

or

(set! *syn-defs* *syn-ignore-whitespace*)

**prec:define-grammar** rule1 ...

[Function]

Appends rule1 ... to *syn-defs*. **prec:define-grammar** is used to define both the character classes and rules for tokens.

Once your grammar is defined, save the value of *syn-defs* in a variable (for use when calling **prec:parse**).

(define my-ruleset *syn-defs*)

**prec:parse** ruleset delim column

[Function]

**prec:parse** ruleset delim column port

[Function]

The ruleset argument must be a list of rules as constructed by **prec:define-grammar** and extracted from *syn-defs*.

The token delim may be a character, symbol, or string. A character delim argument will match only a character token; i.e. a character for which no token-group is assigned. A symbol or string will match only a token string; i.e. a token resulting from a token group.

**prec:parse** reads a ruleset grammar expression delimited by delim from the given input port. **prec:parse** returns the next object parsable from the given input port, updating port to point to the first character past the end of the external representation of the object.

For the purpose of reporting problems in error messages, this package keeps track of the current column. Its initial value is passed as the third argument to **prec:parse**.

If an end of file is encountered in the input before any characters are found that can begin an object, then an end of file object is returned. If a delimiter (such as delim) is found before any characters are found that can begin an object, then #f is returned.

The port argument may be omitted, in which case it defaults to the value returned by **current-input-port**. It is an error to parse from a closed port.

### 4.1.4 Token definition

**tok:char-group** group chars chars-proc

[Function]

The argument chars may be a single character, a list of characters, or a string. Each character in chars is treated as though **tok:char-group** was called with that character alone.

The argument chars-proc must be a procedure of one argument, a list of characters. After **tokenize** has finished accumulating the characters for a token, it calls chars-proc with the list of characters. The value returned is the token which **tokenize** returns.
The argument group may be an exact integer or a procedure of one character argument. The following discussion concerns the treatment which the tokenizing routine, tokenize, will accord to characters on the basis of their groups.

When group is a non-zero integer, characters whose group number is equal to or exactly one less than group will continue to accumulate. Any other character causes the accumulation to stop (until a new token is to be read).

The group of zero is special. These characters are ignored when parsed pending a token, and stop the accumulation of token characters when the accumulation has already begun. Whitespace characters are usually put in group 0.

If group is a procedure, then, when triggered by the occurrence of an initial (no accumulation) chars character, this procedure will be repeatedly called with each successive character from the input stream until the group procedure returns a non-false value.

The following convenient constants are provided for use with tok:char-group.

**tok:decimal-digits**
Is the string "0123456789".

**tok:upper-case**
Is the string consisting of all upper-case letters ("ABCDEFGHIJKLMNOPQRSTUVWXYZ").

**tok:lower-case**
Is the string consisting of all lower-case letters ("abcdefghijklmnopqrstuvwxyz").

**tok:whitespaces**
Is the string consisting of all characters between 0 and 255 for which char-whitespace? returns true.

### 4.1.5 Nud and Led Definition

This section describes advanced features. You can skip this section on first reading.

The *Null Denotation* (or nud) of a token is the procedure and arguments applying for that token when Left, an unclaimed parsed expression is not extant.

The *Left Denotation* (or led) of a token is the procedure, arguments, and lbp applying for that token when there is a Left, an unclaimed parsed expression.

In his paper,


the left binding power (or lbp) was an independent property of tokens. I think this was done in order to allow tokens with NUDs but not LEDs to also be used as delimiters, which was a problem for statically defined syntaxes. It turns out that dynamically binding NUDs and LEDs allows them independence.

For the rule-defining procedures that follow, the variable tk may be a character, string, or symbol, or a list composed of characters, strings, and symbols. Each element of tk is treated as though the procedure were called for each element.
Character *tk* arguments will match only character tokens; i.e. characters for which no token-group is assigned. Symbols and strings will both match token strings; i.e. tokens resulting from token groups.

**prec:make-nud** *tk sop arg1* ...

Returns a rule specifying that *sop* be called when *tk* is parsed. If *sop* is a procedure, it is called with *tk* and *arg1* ... as its arguments; the resulting value is incorporated into the expression being built. Otherwise, (list *sop arg1* ...) is incorporated.

If no NUD has been defined for a token; then if that token is a string, it is converted to a symbol and returned; if not a string, the token is returned.

**prec:make-led** *tk sop arg1* ...

Returns a rule specifying that *sop* be called when *tk* is parsed and *left* has an unclaimed parsed expression. If *sop* is a procedure, it is called with *left*, *tk*, and *arg1* ... as its arguments; the resulting value is incorporated into the expression being built. Otherwise, *left* is incorporated.

If no LED has been defined for a token, and *left* is set, the parser issues a warning.

### 4.1.6 Grammar Rule Definition

Here are procedures for defining rules for the syntax types introduced in Section 4.1.1 [Precedence Parsing Overview], page 40.

For the rule-defining procedures that follow, the variable *tk* may be a character, string, or symbol, or a list composed of characters, strings, and symbols. Each element of *tk* is treated as though the procedure were called for each element.

For procedures **prec:delim**, . . ., **prec:prestfix**, if the *sop* argument is #f, then the token which triggered this rule is converted to a symbol and returned. A false *sop* argument to the procedures **prec:commentfix**, **prec:matchfix**, or **prec:inmatchfix** has a different meaning.

Character *tk* arguments will match only character tokens; i.e. characters for which no token-group is assigned. Symbols and strings will both match token strings; i.e. tokens resulting from token groups.

**prec:delim** *tk*

Returns a rule specifying that *tk* should not be returned from parsing; i.e. *tk*’s function is purely syntactic. The end-of-file is always treated as a delimiter.

**prec:nofix** *tk sop*

Returns a rule specifying the following actions take place when *tk* is parsed:

- If *sop* is a procedure, it is called with no arguments; the resulting value is incorporated into the expression being built. Otherwise, the list of *sop* is incorporated.

**prec:prefix** *tk sop bp rule1* ...

Returns a rule specifying the following actions take place when *tk* is parsed:

- The rules *rule1* ... augment and, in case of conflict, override rules currently in effect.
- **prec:parse1** is called with binding-power *bp*. 
• If sop is a procedure, it is called with the expression returned from \texttt{prec:parse1}; the resulting value is incorporated into the expression being built. Otherwise, the list of sop and the expression returned from \texttt{prec:parse1} is incorporated.
• The ruleset in effect before \texttt{tk} was parsed is restored; \texttt{rule1} \ldots are forgotten.

\textbf{Function} \texttt{prec:infix \ tk \ sop \ lbp \ bp \ rule1 \ldots} \\
Returns a rule declaring the left-binding-precedence of the token \texttt{tk} is \texttt{lbp} and specifying the following actions take place when \texttt{tk} is parsed:

• The rules \texttt{rule1} \ldots augment and, in case of conflict, override rules currently in effect.
• One expression is parsed with binding-power \texttt{lbp}. If instead a delimiter is encountered, a warning is issued.
• If \texttt{sop} is a procedure, it is applied to the list of \texttt{left} and the parsed expression; the resulting value is incorporated into the expression being built. Otherwise, the list of \texttt{sop}, the \texttt{left} expression, and the parsed expression is incorporated.
• The ruleset in effect before \texttt{tk} was parsed is restored; \texttt{rule1} \ldots are forgotten.

\textbf{Function} \texttt{prec:nary \ tk \ sop \ bp} \\
Returns a rule declaring the left-binding-precedence of the token \texttt{tk} is \texttt{bp} and specifying the following actions take place when \texttt{tk} is parsed:

• Expressions are parsed with binding-power \texttt{bp} as far as they are interleaved with the token \texttt{tk}.
• If \texttt{sop} is a procedure, it is applied to the list of \texttt{left} and the parsed expressions; the resulting value is incorporated into the expression being built. Otherwise, the list of \texttt{sop}, the \texttt{left} expression, and the parsed expressions is incorporated.

\textbf{Function} \texttt{prec:postfix \ tk \ sop \ lbp} \\
Returns a rule declaring the left-binding-precedence of the token \texttt{tk} is \texttt{lbp} and specifying the following actions take place when \texttt{tk} is parsed:

• If \texttt{sop} is a procedure, it is called with the \texttt{left} expression; the resulting value is incorporated into the expression being built. Otherwise, the list of \texttt{sop} and the \texttt{left} expression is incorporated.

\textbf{Function} \texttt{prec:prestfix \ tk \ sop \ bp \ rule1 \ldots} \\
Returns a rule specifying the following actions take place when \texttt{tk} is parsed:

• The rules \texttt{rule1} \ldots augment and, in case of conflict, override rules currently in effect.
• Expressions are parsed with binding-power \texttt{bp} until a delimiter is reached.
• If \texttt{sop} is a procedure, it is applied to the list of parsed expressions; the resulting value is incorporated into the expression being built. Otherwise, the list of \texttt{sop} and the parsed expressions is incorporated.
• The ruleset in effect before \texttt{tk} was parsed is restored; \texttt{rule1} \ldots are forgotten.

\textbf{Function} \texttt{prec:commentfix \ tk \ stp \ match \ rule1 \ldots} \\
Returns rules specifying the following actions take place when \texttt{tk} is parsed:
The rules \textit{rule1} \ldots augment and, in case of conflict, override rules currently in effect.

Characters are read until and end-of-file or a sequence of characters is read which matches the \textit{string match}.

If \textit{stp} is a procedure, it is called with the string of all that was read between the \textit{tk} and \textit{match} (exclusive).

The ruleset in effect before \textit{tk} was parsed is restored; \textit{rule1} \ldots are forgotten.

Parsing of commentfix syntax differs from the others in several ways. It reads directly from input without tokenizing; It calls \textit{stp} but does not return its value; nay any value. I added the \textit{stp} argument so that comment text could be echoed.

\textbf{[Function]}

\texttt{prec:matchfix \ tk \ sop \ sep \ match \ rule1 \ldots}

Returns a rule specifying the following actions take place when \textit{tk} is parsed:

- The rules \textit{rule1} \ldots augment and, in case of conflict, override rules currently in effect.
- A rule declaring the token \textit{match} a delimiter takes effect.
- Expressions are parsed with binding-power 0 until the token \textit{match} is reached. If the token \textit{sep} does not appear between each pair of expressions parsed, a warning is issued.
- If \textit{sop} is a procedure, it is applied to the list of parsed expressions; the resulting value is incorporated into the expression being built. Otherwise, the list of \textit{sop} and the parsed expressions is incorporated.
- The ruleset in effect before \textit{tk} was parsed is restored; \textit{rule1} \ldots are forgotten.

\textbf{[Function]}

\texttt{prec:inmatchfix \ tk \ sop \ sep \ match \ lbp \ rule1 \ldots}

Returns a rule declaring the left-binding-precedence of the token \textit{tk} is \textit{lbp} and specifying the following actions take place when \textit{tk} is parsed:

- The rules \textit{rule1} \ldots augment and, in case of conflict, override rules currently in effect.
- A rule declaring the token \textit{match} a delimiter takes effect.
- Expressions are parsed with binding-power 0 until the token \textit{match} is reached. If the token \textit{sep} does not appear between each pair of expressions parsed, a warning is issued.
- If \textit{sop} is a procedure, it is applied to the list of \textit{left} and the parsed expressions; the resulting value is incorporated into the expression being built. Otherwise, the list of \textit{sop}, the \textit{left} expression, and the parsed expressions is incorporated.
- The ruleset in effect before \textit{tk} was parsed is restored; \textit{rule1} \ldots are forgotten.

4.2 Format (version 3.1)

\texttt{(require 'format) or (require 'srffi-28)}
4.2.1 Format Interface

format destination format-string . arguments [Function]
An almost complete implementation of Common LISP format description according
compatible to most of the available Scheme format implementations.
Returns #t, #f or a string; has side effect of printing according to format-string. If
destination is #t, the output is to the current output port and #t is returned. If
destination is #f, a formatted string is returned as the result of the call. NEW: If
destination is a string, destination is regarded as the format string; format-string is
then the first argument and the output is returned as a string. If destination is a
number, the output is to the current error port if available by the implementation.
Otherwise destination must be an output port and #t is returned.

format-string must be a string. In case of a formatting error format returns #f and
prints a message on the current output or error port. Characters are output as if
the string were output by the display function with the exception of those prefixed
by a tilde (~). For a detailed description of the format-string syntax please consult
a Common LISP format reference manual. For a test suite to verify this format
implementation load ‘formatst.scm’.

4.2.2 Format Specification (Format version 3.1)
Please consult a Common LISP format reference manual for a detailed description of the for-
mat string syntax. For a demonstration of the implemented directives see ‘formatst.scm’.

This implementation supports directive parameters and modifiers (: and @ characters).
Multiple parameters must be separated by a comma (,). Parameters can be numerical pa-
rameters (positive or negative), character parameters (prefixed by a quote character (’),
variable parameters (v), number of rest arguments parameter (#), empty and default pa-
rameters. Directive characters are case independent. The general form of a directive is:

directive ::= ~{directive-parameter,}{::}@directive-character

directive-parameter ::= [ [-]+{0-9}+ | ’character | v | # ]

4.2.2.1 Implemented CL Format Control Directives
Documentation syntax: Uppercase characters represent the corresponding control directive
characters. Lowercase characters represent control directive parameter descriptions.

~A Any (print as display does).
~@A left pad.
~mincol, colinc, minpad, padcharA
full padding.

~S S-expression (print as write does).
~@S left pad.
~mincol, colinc, minpad, padcharS
full padding.
\^D  Decimal.  
\^D@  print number sign always.  
\^:D  print comma separated.  
\^\text{mincol, padchar, commachar}D  
padding.  

\^X  Hexadecimal.  
\^X@  print number sign always.  
\^:X  print comma separated.  
\^\text{mincol, padchar, commachar}X  
padding.  

\^0  Octal.  
\^0@  print number sign always.  
\^:0  print comma separated.  
\^\text{mincol, padchar, commachar}0  
padding.  

\^B  Binary.  
\^B@  print number sign always.  
\^:B  print comma separated.  
\^\text{mincol, padchar, commachar}B  
padding.  

\^nR  Radix n.  
\^n,\text{mincol, padchar, commachar}R  
padding.  

\^\text{@R}  print a number as a Roman numeral.  
\^:\text@R  print a number as an “old fashioned” Roman numeral.  
\^:\text@R  print a number as an ordinal English number.  
\^\text{R}  print a number as a cardinal English number.  

\^P  Plural.  
\^P@  prints y and ies.  
\^:\text{P}  as \^P but jumps 1 argument backward.  
\^:\text@P  as \^\text{P} but jumps 1 argument backward.  

\^C  Character.  
\^C@  prints a character as the reader can understand it (i.e. \# prefixing).  
\^:\text{C}  prints a character as emacs does (eg. \^C for ASCII 03).  

"F  Fixed-format floating-point (prints a flonum like mmm.nnn).
   ~width,digits, scale, overflowchar, padcharF
   ~@F If the number is positive a plus sign is printed.
"E  Exponential floating-point (prints a flonum like mmm.nnnEee).
   ~width, digits, exponentdigits, scale, overflowchar, padchar, exponentcharE
   ~@E If the number is positive a plus sign is printed.
"G  General floating-point (prints a flonum either fixed or exponential).
   ~width, digits, exponentdigits, scale, overflowchar, padchar, exponentcharG
   ~@G If the number is positive a plus sign is printed.
"$  Dollars floating-point (prints a flonum in fixed with signs separated).
   ~digits, scale, width, padchar$
   ~@$ If the number is positive a plus sign is printed.
   ~:@$ A sign is always printed and appears before the padding.
   ~:@$ The sign appears before the padding.
"%  Newline.
   ~n% print n newlines.
"&  print newline if not at the beginning of the output line.
   ~n& prints ~& and then n-1 newlines.
|  Page Separator.
   ~n | print n page separators.
--  Tilde.
   ~n~ print n tildes.
<newline>  Continuation Line.
   ~:<newline> newline is ignored, white space left.
   ~@<newline> newline is left, white space ignored.
T  Tabulation.
   ~@T relative tabulation.
   ~colnum, colincT full tabulation.
?  Indirection (expects indirect arguments as a list).
   ~@? extracts indirect arguments from format arguments.
<(str)> Case conversion (converts by string-downcase).
":(str~) converts by string-capitalize.
"@:(str~) converts by string-capitalize-first.
":@:(str~)
   converts by string-upcase.
"* Argument Jumping (jumps 1 argument forward).
"n* jumps n arguments forward.
":* jumps 1 argument backward.
"n:* jumps n arguments backward.
"@* jumps to the 0th argument.
"n@* jumps to the n-th argument (beginning from 0)
"[*str0~;str1~;...;strn~]
   Conditional Expression (numerical clause conditional).
"n[ take argument from n.
"@[ true test conditional.
":[ if-else-then conditional.
"; clause separator.
";; default clause follows.
"{str~} Iteration (args come from the next argument (a list)). Iteration bounding is
   controlled by configuration variables format:iteration-bounded and format:max-
   iterations. With both variables default, a maximum of 100 iterations will be
   performed.
"n{ at most n iterations.
":{ args from next arg (a list of lists).
":@[ args from the rest of arguments.
":@[ args from the rest args (lists).
"^^ Up and out.
"n^ aborts if n = 0
"n,m^ aborts if n = m
"n,m,k^ aborts if n <= m <= k

4.2.2.2 Not Implemented CL Format Control Directives
"~:A print #f as an empty list (see below).
"~:S print #f as an empty list (see below).
"<~> Justification.
"~:~ (sorry I don’t understand its semantics completely)
4.2.2.3 Extended, Replaced and Additional Control Directives

\[-\text{mincol}, \text{padchar}, \text{commachar}, \text{commawidth}\]
\[-\text{mincol}, \text{padchar}, \text{commachar}, \text{commawidth}\times\]
\[-\text{mincol}, \text{padchar}, \text{commachar}, \text{commawidth}\circ\]
\[-n, \text{mincol}, \text{padchar}, \text{commachar}, \text{commawidth}\text{R}\]

\text{commawidth} is the number of characters between two comma characters.

\[-\text{I}\]
print a R4RS complex number as \[-F\text{OFi}\] with passed parameters for \[-F\].

\[-\text{Y}\]
Pretty print formatting of an argument for scheme code lists.

\[-\text{K}\]
Same as \[-?\].

\[-\text{!}\]
Flushes the output if format \text{destination} is a port.

\[-\text{-}\]
Print a \#\text{space} character
\[-\text{n-}\]
print \text{n} \#\text{space} characters.

\[-/\]
Print a \#\text{tab} character
\[-\text{n/}\]
print \text{n} \#\text{tab} characters.

\[-\text{nC}\]
Takes \text{n} as an integer representation for a character. No arguments are consumed. \text{n} is converted to a character by integer->char. \text{n} must be a positive decimal number.

\[-:\text{S}\]
Print out readproof. Prints out internal objects represented as \#<...> as strings "#<...>", so that the format output can always be processed by \text{read}.

\[-:\text{A}\]
Print out readproof. Prints out internal objects represented as \#<...> as strings "#<...>", so that the format output can always be processed by \text{read}.

\[-\text{Q}\]
Print information and a copyright notice on the format implementation.

\[-:\text{Q}\]
prints format version.

\[-\text{F}, \text{E}, \text{G}, \text{$}\]
may also print number strings, i.e. passing a number as a string and format it accordingly.

4.2.2.4 Configuration Variables

Format has some configuration variables at the beginning of ‘format.scm’ to suit the systems and users needs. There should be no modification necessary for the configuration that comes with SLIB. If modification is desired the variable should be set after the format code is loaded. Format detects automatically if the running scheme system implements floating point numbers and complex numbers.

\text{format:symbol-case-conv}
Symbols are converted by symbol->string so the case type of the printed symbols is implementation dependent. \text{format:symbol-case-conv} is a one arg closure which is either \#f (no conversion), string-upcase, string-downcase or string-capitalize. (default \#f)
format:iobj-case-conv
As format:symbol-case-conv but applies for the representation of implementation internal objects. (default #f)

format:expch
The character prefixing the exponent value in "E printing. (default #\E)

format:iteration-bounded
When #t, a "{...}" control will iterate no more than the number of times specified by format:max-iterations regardless of the number of iterations implied by modifiers and arguments. When #f, a "{...}" control will iterate the number of times implied by modifiers and arguments, unless termination is forced by language or system limitations. (default #t)

format:max-iterations
The maximum number of iterations performed by a "{...}" control. Has effect only when format:iteration-bounded is #t. (default 100)

4.2.2.5 Compatibility With Other Format Implementations

SLIB format 2.x:
See 'format.doc'.

SLIB format 1.4:
Downward compatible except for padding support and ~A, ~S, ~P, ~X uppercase printing. SLIB format 1.4 uses C-style printf padding support which is completely replaced by the CL format padding style.

MIT C-Scheme 7.1:
Downward compatible except for ~, which is not documented (ignores all characters inside the format string up to a newline character). (7.1 implements ~a, ~s, ~newline, ~%, numerical and variable parameters and :/@ modifiers in the CL sense).

Elk 1.5/2.0:
Downward compatible except for ~A and ~S which print in uppercase. (Elk implements ~a, ~s, ~, and ~% (no directive parameters or modifiers)).

Scheme->C 01nov91:
Downward compatible except for an optional destination parameter: S2C accepts a format call without a destination which returns a formatted string. This is equivalent to a #f destination in S2C. (S2C implements ~a, ~s, ~c, ~%, and ~ (no directive parameters or modifiers)).

This implementation of format is solely useful in the SLIB context because it requires other components provided by SLIB.

4.3 Standard Formatted I/O

4.3.1 stdio
(requires 'stdio)

requires printf and scanf and additionally defines the symbols:
stdin
Defined to be (current-input-port).

stdout
Defined to be (current-output-port).

stderr
Defined to be (current-error-port).

4.3.2 Standard Formatted Output

(require 'printf)

printf format arg1 ... [Procedure]
fprintf port format arg1 ... [Procedure]
sprintf str format arg1 ... [Procedure]
sprintf #f format arg1 ... [Procedure]
sprintf k format arg1 ... [Procedure]

Each function converts, formats, and outputs its arg1 ... arguments according to the control string format argument and returns the number of characters output.

printf sends its output to the port (current-output-port). fprintf sends its output to the port port. sprintf string-set!s locations of the non-constant string argument str to the output characters.

Two extensions of sprintf return new strings. If the first argument is #f, then the returned string’s length is as many characters as specified by the format and data; if the first argument is a non-negative integer k, then the length of the returned string is also bounded by k.

The string format contains plain characters which are copied to the output stream, and conversion specifications, each of which results in fetching zero or more of the arguments arg1 ... . The results are undefined if there are an insufficient number of arguments for the format. If format is exhausted while some of the arg1 ... arguments remain unused, the excess arg1 ... arguments are ignored.

The conversion specifications in a format string have the form:

```
% [ flags ] [ width ] [ . precision ] [ type ] conversion
```

An output conversion specifications consist of an initial ‘%’ character followed in sequence by:

- Zero or more flag characters that modify the normal behavior of the conversion specification.
  - `'-'` Left-justify the result in the field. Normally the result is right-justified.
  - `'+'` For the signed ‘%d’ and ‘%i’ conversions and all inexact conversions, prefix a plus sign if the value is positive.
  - `' '` For the signed ‘%d’ and ‘%i’ conversions, if the result doesn’t start with a plus or minus sign, prefix it with a space character instead. Since the ‘+’ flag ensures that the result includes a sign, this flag is ignored if both are specified.
For inexact conversions, ‘#’ specifies that the result should always include a decimal point, even if no digits follow it. For the ‘%g’ and ‘%G’ conversions, this also forces trailing zeros after the decimal point to be printed where they would otherwise be elided.

For the ‘%o’ conversion, force the leading digit to be ‘0’, as if by increasing the precision. For ‘%x’ or ‘%X’, prefix a leading ‘0x’ or ‘0X’ (respectively) to the result. This doesn’t do anything useful for the ‘%d’, ‘%i’, or ‘%u’ conversions. Using this flag produces output which can be parsed by the `scanf` functions with the ‘%i’ conversion (see Section 4.3.3 [Standard Formatted Input], page 56).

‘0’ Pad the field with zeros instead of spaces. The zeros are placed after any indication of sign or base. This flag is ignored if the ‘-’ flag is also specified, or if a precision is specified for an exact conversion.

- An optional decimal integer specifying the minimum field width. If the normal conversion produces fewer characters than this, the field is padded (with spaces or zeros per the ‘0’ flag) to the specified width. This is a minimum width; if the normal conversion produces more characters than this, the field is not truncated.

Alternatively, if the field width is ‘*’, the next argument in the argument list (before the actual value to be printed) is used as the field width. The width value must be an integer. If the value is negative it is as though the ‘-’ flag is set (see above) and the absolute value is used as the field width.

- An optional precision to specify the number of digits to be written for numeric conversions and the maximum field width for string conversions. The precision is specified by a period (‘.’) followed optionally by a decimal integer (which defaults to zero if omitted).

Alternatively, if the precision is ‘.*’, the next argument in the argument list (before the actual value to be printed) is used as the precision. The value must be an integer, and is ignored if negative. If you specify ‘*’ for both the field width and precision, the field width argument precedes the precision argument. The ‘. *’ precision is an enhancement. C library versions may not accept this syntax.

For the ‘%f’, ‘%e’, and ‘%E’ conversions, the precision specifies how many digits follow the decimal-point character. The default precision is 6. If the precision is explicitly 0, the decimal point character is suppressed.

For the ‘%g’ and ‘%G’ conversions, the precision specifies how many significant digits to print. Significant digits are the first digit before the decimal point, and all the digits after it. If the precision is 0 or not specified for ‘%g’ or ‘%G’, it is treated like a value of 1. If the value being printed cannot be expressed accurately in the specified number of digits, the value is rounded to the nearest number that fits.

For exact conversions, if a precision is supplied it specifies the minimum number of digits to appear; leading zeros are produced if necessary. If a precision is not supplied, the number is printed with as many digits as necessary. Converting an exact ‘0’ with an explicit precision of zero produces no characters.
• An optional one of ‘l’, ‘h’ or ‘L’, which is ignored for numeric conversions. It is an error to specify these modifiers for non-numeric conversions.

• A character that specifies the conversion to be applied.

4.3.2.1 Exact Conversions

‘b’, ‘B’ Print an integer as an unsigned binary number.

Note: ‘%b’ and ‘%B’ are SLIB extensions.

‘d’, ‘i’ Print an integer as a signed decimal number. ‘%d’ and ‘%i’ are synonymous for output, but are different when used with scanf for input (see Section 4.3.3 [Standard Formatted Input], page 56).

‘o’ Print an integer as an unsigned octal number.

‘u’ Print an integer as an unsigned decimal number.

‘x’, ‘X’ Print an integer as an unsigned hexadecimal number. ‘%x’ prints using the digits ‘0123456789abcdef’. ‘%X’ prints using the digits ‘0123456789ABCDEF’.

4.3.2.2 Inexact Conversions

‘f’ Print a floating-point number in fixed-point notation.

‘e’, ‘E’ Print a floating-point number in exponential notation. ‘%e’ prints ‘e’ between mantissa and exponent. ‘%E’ prints ‘E’ between mantissa and exponent.

‘g’, ‘G’ Print a floating-point number in either fixed or exponential notation, whichever is more appropriate for its magnitude. Unless an ‘#’ flag has been supplied, trailing zeros after a decimal point will be stripped off. ‘%g’ prints ‘e’ between mantissa and exponent. ‘%G’ prints ‘E’ between mantissa and exponent.

‘k’, ‘K’ Print a number like ‘%g’, except that an SI prefix is output after the number, which is scaled accordingly. ‘%K’ outputs a dot between number and prefix, ‘%k’ does not.

4.3.2.3 Other Conversions

‘c’ Print a single character. The ‘-’ flag is the only one which can be specified. It is an error to specify a precision.

‘s’ Print a string. The ‘-’ flag is the only one which can be specified. A precision specifies the maximum number of characters to output; otherwise all characters in the string are output.

‘a’, ‘A’ Print a scheme expression. The ‘-’ flag left-justifies the output. The ‘#’ flag specifies that strings and characters should be quoted as by write (which can be read using read); otherwise, output is as display prints. A precision specifies the maximum number of characters to output; otherwise as many characters as needed are output.

Note: ‘%a’ and ‘%A’ are SLIB extensions.
4.3.3 Standard Formatted Input

(require 'scanf)

scanf-read-list format [Function]
scanf-read-list format port [Function]
scanf-read-list format string [Function]

scanf format arg1 ... [Macro]
fscanf port format arg1 ... [Macro]
sscanf str format arg1 ... [Macro]

Each function reads characters, interpreting them according to the control string format argument.

scanf-read-list returns a list of the items specified as far as the input matches format. scanf, fscanf, and sscanf return the number of items successfully matched and stored. scanf, fscanf, and sscanf also set the location corresponding to arg1 ... using the methods:
symbol set!
car expression
   set-car!
cdr expression
   set-cdr!
vector-ref expression
   vector-set!
substring expression
   substring-move-left!

The argument to a substring expression in arg1 ... must be a non-constant string. Characters will be stored starting at the position specified by the second argument to substring. The number of characters stored will be limited by either the position specified by the third argument to substring or the length of the matched string, whichever is less.

The control string, format, contains conversion specifications and other characters used to direct interpretation of input sequences. The control string contains:

- White-space characters (blanks, tabs, newlines, or formfeeds) that cause input to be read (and discarded) up to the next non-white-space character.
- An ordinary character (not '%') that must match the next character of the input stream.
- Conversion specifications, consisting of the character '%', an optional assignment suppressing character '*', an optional numerical maximum-field width, an optional 'I', 'h' or 'L' which is ignored, and a conversion code.
Unless the specification contains the ‘n’ conversion character (described below), a conversion specification directs the conversion of the next input field. The result of a conversion specification is returned in the position of the corresponding argument points, unless ‘*’ indicates assignment suppression. Assignment suppression provides a way to describe an input field to be skipped. An input field is defined as a string of characters; it extends to the next inappropriate character or until the field width, if specified, is exhausted.

Note: This specification of format strings differs from the ANSI C and POSIX specifications. In SLIB, white space before an input field is not skipped unless white space appears before the conversion specification in the format string. In order to write format strings which work identically with ANSI C and SLIB, prepend whitespace to all conversion specifications except ‘[’ and ‘c’.

The conversion code indicates the interpretation of the input field; For a suppressed field, no value is returned. The following conversion codes are legal:

- ‘%’ A single % is expected in the input at this point; no value is returned.
- ‘d’, ‘D’ A decimal integer is expected.
- ‘u’, ‘U’ An unsigned decimal integer is expected.
- ‘o’, ‘O’ An octal integer is expected.
- ‘x’, ‘X’ A hexadecimal integer is expected.
- ‘i’ An integer is expected. Returns the value of the next input item, interpreted according to C conventions; a leading ‘0’ implies octal, a leading ‘0x’ implies hexadecimal; otherwise, decimal is assumed.
- ‘n’ Returns the total number of bytes (including white space) read by scanf. No input is consumed by %n.
- ‘f’, ‘F’, ‘e’, ‘E’, ‘g’, ‘G’ A floating-point number is expected. The input format for floating-point numbers is an optionally signed string of digits, possibly containing a radix character ‘.’, followed by an optional exponent field consisting of an ‘E’ or an ‘e’, followed by an optional ‘+’, ‘-’, or space, followed by an integer.
- ‘c’, ‘C’ Width characters are expected. The normal skip-over-white-space is suppressed in this case; to read the next non-space character, use ‘%Is’. If a field width is given, a string is returned; up to the indicated number of characters is read.
- ‘s’, ‘S’ A character string is expected The input field is terminated by a white-space character. scanf cannot read a null string.
- ‘[’ Indicates string data and the normal skip-over-leading-white-space is suppressed. The left bracket is followed by a set of characters, called the scanset, and a right bracket; the input field is the maximal sequence of input characters consisting entirely of characters in the scanset. ‘[’ when
it appears as the first character in the scanset, serves as a complement operator and redefines the scanset as the set of all characters not contained in the remainder of the scanset string. Construction of the scanset follows certain conventions. A range of characters may be represented by the construct first-last, enabling ‘[0123456789]’ to be expressed ‘[0-9]’. Using this convention, first must be lexically less than or equal to last; otherwise, the dash stands for itself. The dash also stands for itself when it is the first or the last character in the scanset. To include the right square bracket as an element of the scanset, it must appear as the first character (possibly preceded by a ‘^’) of the scanset, in which case it will not be interpreted syntactically as the closing bracket. At least one character must match for this conversion to succeed.

The `scanf` functions terminate their conversions at end-of-file, at the end of the control string, or when an input character conflicts with the control string. In the latter case, the offending character is left unread in the input stream.

## 4.4 Program and Arguments

### 4.4.1 Getopt

(require 'getopt)

This routine implements Posix command line argument parsing. Notice that returning values through global variables means that `getopt` is not reentrant.

Obedience to Posix format for the `getopt` calls sows confusion. Passing `argc` and `argv` as arguments while referencing `optind` as a global variable leads to strange behavior, especially when the calls to `getopt` are buried in other procedures.

Even in C, `argc` can be derived from `argv`; what purpose does it serve beyond providing an opportunity for `argv/argc` mismatch? Just such a mismatch existed for years in a SLIB `getopt` example.

I have removed the `argc` and `argv` arguments to getopt procedures; and replaced them with a global variable:

*argv* [Variable]

Define `argv` with a list of arguments before calling getopt procedures. If you don’t want the first (0th) element to be ignored, set `optind` to 0 (after requiring getopt).

*optind* [Variable]

Is the index of the current element of the command line. It is initially one. In order to parse a new command line or reparse an old one, `optind` must be reset.

*optarg* [Variable]

Is set by getopt to the (string) option-argument of the current option.

getopt `optstring`

Returns the next option letter in `argv` (starting from (vector-ref `argv` `optind*`)) that matches a letter in `optstring`. `argv` is a vector or list of strings, the 0th of which getopt usually ignores. `optstring` is a string of recognized option
characters; if a character is followed by a colon, the option takes an argument which may be immediately following it in the string or in the next element of *argv*.

*optind* is the index of the next element of the *argv* vector to be processed. It is initialized to 1 by 'getopt.scm', and getopt updates it when it finishes with each element of *argv*.

getopt returns the next option character from *argv* that matches a character in *optstring*, if there is one that matches. If the option takes an argument, getopt sets the variable *optarg* to the option-argument as follows:

- If the option was the last character in the string pointed to by an element of *argv*, then *optarg* contains the next element of *argv*, and *optind* is incremented by 2. If the resulting value of *optind* is greater than or equal to (length *argv*), this indicates a missing option argument, and getopt returns an error indication.
- Otherwise, *optarg* is set to the string following the option character in that element of *argv*, and *optind* is incremented by 1.

If, when getopt is called, the string (vector-ref argv *optind*) either does not begin with the character #\- or is just "-", getopt returns #f without changing *optind*. If (vector-ref argv *optind*) is the string "--", getopt returns #f after incrementing *optind*.

If getopt encounters an option character that is not contained in *optstring*, it returns the question-mark #\? character. If it detects a missing option argument, it returns the colon character #: if the first character of *optstring* was a colon, or a question-mark character otherwise. In either case, getopt sets the variable getopt:opt to the option character that caused the error.

The special option "--" can be used to delimit the end of the options; #f is returned, and "--" is skipped.

RETURN VALUE

getopt returns the next option character specified on the command line. A colon #: is returned if getopt detects a missing argument and the first character of *optstring* was a colon #:.

A question-mark #\? is returned if getopt encounters an option character not in *optstring* or detects a missing argument and the first character of *optstring* was not a colon #:.

Otherwise, getopt returns #f when all command line options have been parsed.

Example:

```scheme
#! /usr/local/bin/scm
(require 'program-arguments)
(require 'getopt)
(define argv (program-arguments))

(define opts ":a:b:cd")
(let loop ((opt (getopt (length argv) argv opts)))
  (case opt
    ...)
4.4.2 Getopt—

getopt-- optstring

The procedure getopt-- is an extended version of getopt which parses long option names of the form '--hold-the-onions' and '--verbosity-level=extreme'. Getopt-- behaves as getopt except for non-empty options beginning with '--'.

Options beginning with '--' are returned as strings rather than characters. If a value is assigned (using '=' ) to a long option, *optarg* is set to the value. The '=' and value are not returned as part of the option string.

No information is passed to getopt-- concerning which long options should be accepted or whether such options can take arguments. If a long option did not have an argument, *optarg* will be set to #f. The caller is responsible for detecting and reporting errors.

```
(define opts ":-:b:")
(define *argv* '("foo" "-b9" "--f1" "--2=" "--g3=35234.342" "--"))
(define *optind* 1)
(define *optarg* #f)
(require 'qp)
(do ((i 5 (+ -1 i)))
    ((zero? i))
    (let ((opt (getopt-- opts)))
        (print *optind* opt *optarg*))

| 2 #\b "9"
| 3 "f1" #f
| 4 "2" 
| 5 "g3" "35234.342"
| 5 #f "35234.342"
```

4.4.3 Command Line

(require 'read-command)
read-command port

read-command

read-command converts a command line into a list of strings suitable for parsing by getopt. The syntax of command lines supported resembles that of popular shells. read-command updates port to point to the first character past the command delimiter.

If an end of file is encountered in the input before any characters are found that can begin an object or comment, then an end of file object is returned.

The port argument may be omitted, in which case it defaults to the value returned by current-input-port.

The fields into which the command line is split are delimited by whitespace as defined by char-whitespace?. The end of a command is delimited by end-of-file or unescaped semicolon (;) or NEWLINE. Any character can be literally included in a field by escaping it with a backslash (\).

The initial character and types of fields recognized are:

'\' The next character has is taken literally and not interpreted as a field delimiter. If \ is the last character before a NEWLINE, that NEWLINE is just ignored. Processing continues from the characters after the NEWLINE as though the backslash and NEWLINE were not there.

'"' The characters up to the next unescaped " are taken literally, according to [R4RS] rules for literal strings (see Section “Strings” in Revised(4) Scheme).

'(', '%' One scheme expression is read starting with this character. The read expression is evaluated, converted to a string (using display), and replaces the expression in the returned field.

';' Semicolon delimits a command. Using semicolons more than one command can appear on a line. Escaped semicolons and semicolons inside strings do not delimit commands.

The comment field differs from the previous fields in that it must be the first character of a command or appear after whitespace in order to be recognized. # can be part of fields if these conditions are not met. For instance, ab#c is just the field ab#c.

'#echo' Introduces a comment. The comment continues to the end of the line on which the semicolon appears. Comments are treated as whitespace by read-command-line and backslashes before NEWLINEs in comments are also ignored.

read-options-file filename

read-options-file converts an options file into a list of strings suitable for parsing by getopt. The syntax of options files is the same as the syntax for command lines, except that NEWLINEs do not terminate reading (only ; or end of file).

If an end of file is encountered before any characters are found that can begin an object or comment, then an end of file object is returned.
4.4.4 Parameter lists

(require 'parameters)

Arguments to procedures in scheme are distinguished from each other by their position in the procedure call. This can be confusing when a procedure takes many arguments, many of which are not often used.

A parameter-list is a way of passing named information to a procedure. Procedures are also defined to set unused parameters to default values, check parameters, and combine parameter lists.

A parameter has the form (parameter-name value1 ...). This format allows for more than one value per parameter-name.

A parameter-list is a list of parameters, each with a different parameter-name.

make-parameter-list parameter-names

Returns an empty parameter-list with slots for parameter-names.

parameter-list-ref parameter-list parameter-name

parameter-name must name a valid slot of parameter-list. parameter-list-ref returns the value of parameter parameter-name of parameter-list.

remove-parameter parameter-name parameter-list

Removes the parameter parameter-name from parameter-list. remove-parameter does not alter the argument parameter-list.

If there are more than one parameter-name parameters, an error is signaled.

adjoin-parameters! parameter-list parameter1 ... 

Returns parameter-list with parameter1 ... merged in.

descriptions-

parameter-list-expand expanders parameter-list

expanders is a list of procedures whose order matches the order of the parameter-names in the call to make-parameter-list which created parameter-list. For each non-false element of expanders that procedure is mapped over the corresponding parameter value and the returned parameter lists are merged into parameter-list.

This process is repeated until parameter-list stops growing. The value returned from parameter-list-expand is unspecified.

fill-empty-parameters defaulters parameter-list

defaulters is a list of procedures whose order matches the order of the parameter-names in the call to make-parameter-list which created parameter-list. fill-empty-parameters returns a new parameter-list with each empty parameter replaced with the list returned by calling the corresponding defaulter with parameter-list as its argument.

check-parameters checks parameter-list

checks is a list of procedures whose order matches the order of the parameter-names in the call to make-parameter-list which created parameter-list. check-parameters returns parameter-list if each check of the corresponding parameter-list returns non-false. If some check returns #f a warning is signaled.
In the following procedures *arities* is a list of symbols. The elements of *arities* can be:

- **single**: Requires a single parameter.
- **optional**: A single parameter or no parameter is acceptable.
- **boolean**: A single boolean parameter or zero parameters is acceptable.
- **nary**: Any number of parameters are acceptable.
- **nary1**: One or more of parameters are acceptable.

**parameter-list->arglist**

*positions arities parameter-list*  
[Function]
Returns *parameter-list* converted to an argument list. Parameters of *arity* type *single* and *boolean* are converted to the single value associated with them. The other *arity* types are converted to lists of the value(s).

*positions* is a list of positive integers whose order matches the order of the *parameter-names* in the call to *make-parameter-list* which created *parameter-list*. The integers specify in which argument position the corresponding parameter should appear.

**4.4.5 Getopt Parameter lists**

(require 'getopt-parameters)

**getopt->parameter-list**

*optnames arities types aliases desc . . .*  
[Function]
Returns *argv* converted to a parameter-list. *optnames* are the parameter-names. *arities* and *types* are lists of symbols corresponding to *optnames*.

*aliases* is a list of lists of strings or integers paired with elements of *optnames*. Each one-character string will be treated as a single ‘-’ option by *getopt*. Longer strings will be treated as long-named options (see Section 4.4.1 [Getopt], page 58).

If the *aliases* association list has only strings as its *cars*, then all the option-arguments after an option (and before the next option) are adjoined to that option.

If the *aliases* association list has integers, then each (string) option will take at most one option-argument. Unoptioned arguments are collected in a list. A ‘-1’ alias will take the last argument in this list; ‘+1’ will take the first argument in the list. The aliases -2 then +2; -3 then +3; . . . are tried so long as a positive or negative consecutive alias is found and arguments remain in the list. Finally a ‘0’ alias, if found, absorbs any remaining arguments.

In all cases, if unclaimed arguments remain after processing, a warning is signaled and #f is returned.

**getopt->arglist**

*optnames positions arities types defaulters checks aliases desc . . .*  
[Function]
Like *getopt->parameter-list*, but converts *argv* to an argument-list as specified by *optnames*, *positions*, *arities*, *types*, *defaulters*, *checks*, and *aliases*. If the options supplied violate the *arities* or *checks* constraints, then a warning is signaled and #f is returned.

These *getopt* functions can be used with SLIB relational databases. For an example, See Section 6.1.1 [Using Databases], page 162.
If errors are encountered while processing options, directions for using the options (and argument strings desc . . .) are printed to current-error-port.

(begin
  (set! *optind* 1)
  (set! *argv* '("cmd" "-?"))
  (getopt->parameter-list
   '(flag number symbols string flag2 flag3 num2 num3)
   '(boolean optional nary1 nary single boolean boolean nary nary)
   '(boolean integer symbol symbol string boolean boolean integer integer)
   '(
     ("flag" flag)
     ("f" flag)
     ("Flag" flag2)
     ("B" flag3)
     ("optional" number)
     ("o" number)
     ("nary1" symbols)
     ("N" symbols)
     ("nary" symbols)
     ("n" symbols)
     ("single" string)
     ("s" string)
     ("a" num2)
     ("Abs" num3)))
)

-| Usage: cmd [OPTION ARGUMENT ...] ...

  -f, --flag
  -o, --optional=<number>
  -n, --nary=<symbols> ...
  -N, --nary1=<symbols> ...
  -s, --single=<string>
     --Flag
  -B
  -a <num2> ...
     --Abs=<num3> ...

ERROR: getopt->parameter-list "unrecognized option" "-?"

4.4.6 Filenames

(require 'filename)

filename:match?? pattern [Function]
filename:match-ci?? pattern [Function]

Returns a predicate which returns a non-false value if its string argument matches (the string) pattern, false otherwise. Filename matching is like glob expansion described the bash manpage, except that names beginning with ‘.’ are matched and ‘/’ characters are not treated specially.
These functions interpret the following characters specially in pattern strings:

- ‘*’ Matches any string, including the null string.
- ‘?’ Matches any single character.
- ‘[...]’ Matches any one of the enclosed characters. A pair of characters separated by a minus sign (−) denotes a range; any character lexically between those two characters, inclusive, is matched. If the first character following the ‘[’ is a ‘!’ or a ‘^’ then any character not enclosed is matched. A ‘−’ or ‘]’ may be matched by including it as the first or last character in the set.

```scheme
(filename:substitute?? pattern template) [Function]
(filename:substitute-ci?? pattern template) [Function]
```

Returns a function transforming a single string argument according to glob patterns pattern and template. pattern and template must have the same number of wildcard specifications, which need not be identical. pattern and template may have a different number of literal sections. If an argument to the function matches pattern in the sense of filename:match?? then it returns a copy of template in which each wildcard specification is replaced by the part of the argument matched by the corresponding wildcard specification in pattern. A * wildcard matches the longest leftmost string possible. If the argument does not match pattern then false is returned.

template may be a function accepting the same number of string arguments as there are wildcard specifications in pattern. In the case of a match the result of applying template to a list of the substrings matched by wildcard specifications will be returned, otherwise template will not be called and #f will be returned.

```scheme
((filename:substitute?? "scm_[0-9]*.html" "scm5c4_???.htm")
 "scm_10.html")
⇒ "scm5c4_10.htm"
((filename:substitute?? "??" "beg?mid?end") "AZ")
⇒ "begAmidZend"
((filename:substitute?? "*na*" "?NA?") "banana")
⇒ "banaNA"
((filename:substitute?? "*??" (lambda (s1 s2 s3) (string-append s3 s1)))
 "ABZ")
⇒ "ZA"
```

```scheme
replace-suffix str old new [Function]
```

str can be a string or a list of strings. Returns a new string (or strings) similar to str but with the suffix string old removed and the suffix string new appended. If the end of str does not match old, an error is signaled.

```scheme
(replace-suffix "/usr/local/lib/slib/batch.scm" ".scm" ".c")
⇒ "/usr/local/lib/slib/batch.scm"
```

```scheme
call-with-tmpnam proc k [Function]
call-with-tmpnam proc [Function]
```

Calls proc with k arguments, strings returned by successive calls to tmpnam. If proc returns, then any files named by the arguments to proc are deleted automatically and
the value(s) yielded by the proc is(are) returned. k may be omitted, in which case it defaults to 1.

\texttt{call-with-tpnam proc suffix1 \ldots}

[Function]

Calls proc with strings returned by successive calls to \texttt{tmpnam}, each with the corresponding suffix string appended. If proc returns, then any files named by the arguments to proc are deleted automatically and the value(s) yielded by the proc is(are) returned.

\subsection*{4.4.7 Batch}

\texttt{(require 'batch)}

The batch procedures provide a way to write and execute portable scripts for a variety of operating systems. Each \texttt{batch:} procedure takes as its first argument a parameter-list (see Section 4.4.4 [Parameter lists], page 62). This parameter-list argument \texttt{parms} contains named associations. Batch currently uses 2 of these:

\texttt{batch-port}

The port on which to write lines of the batch file.

\texttt{batch-dialect}

The syntax of batch file to generate. Currently supported are:

- \texttt{unix}
- \texttt{dos}
- \texttt{vms}
- \texttt{amigaos}
- \texttt{system}
- \texttt{*unknown*}

The \texttt{batch} module uses 2 enhanced relational tables (see Section 6.1.1 [Using Databases], page 162) to store information linking the names of \texttt{operating-systems} to \texttt{batch-dialects}.

\texttt{batch:initialize! database}

[Function]

Defines \texttt{operating-system} and \texttt{batch-dialect} tables and adds the domain \texttt{operating-system} to the enhanced relational database \texttt{database}.

\texttt{*operating-system*}

[Variable]

Is batch’s best guess as to which operating-system it is running under. \texttt{*operating-system*} is set to \texttt{(software-type)} (see Section 2.2 [Configuration], page 12) unless \texttt{(software-type)} is \texttt{unix}, in which case finer distinctions are made.

\texttt{batch:call-with-output-script parms file proc}

[Function]

proc should be a procedure of one argument. If file is an output-port, \texttt{batch:call-with-output-script} writes an appropriate header to file and then calls proc with file as the only argument. If file is a string, \texttt{batch:call-with-output-script} opens a output-file of name file, writes an appropriate header to file, and then calls proc with the newly opened port as the only argument. Otherwise, \texttt{batch:call-with-output-script} acts as if it was called with the result of \texttt{(current-output-port)} as its third argument.
The rest of the `batch:` procedures write (or execute if `batch-dialect` is `system`) commands to the batch port which has been added to `parms` or `(copy-tree parms)` by the code:

```lisp
(adjoin-parameters! parms (list 'batch-port port))
```

`batch:command parms string1 string2 ...`  
Calls `batch:try-command` (below) with arguments, but signals an error if `batch:try-command` returns `#f`.

These functions return a non-false value if the command was successfully translated into the batch dialect and `#f` if not. In the case of the `system` dialect, the value is non-false if the operation succeeded.

`batch:try-command parms string1 string2 ...`  
Writes a command to the `batch-port` in `parms` which executes the program named `string1` with arguments `string2` . . .

`batch:try-chopped-command parms arg1 arg2 ... list`  
breaks the last argument `list` into chunks small enough so that the command:

```
arg1 arg2 ... chunk
```
fits withing the platform's maximum command-line length.

`batch:try-chopped-command` calls `batch:try-command` with the command and returns non-false only if the commands all fit and `batch:try-command` of each command line returned non-false.

`batch:run-script parms string1 string2 ...`  
Writes a command to the `batch-port` in `parms` which executes the batch script named `string1` with arguments `string2` . . .

*Note:* `batch:run-script` and `batch:try-command` are not the same for some operating systems (VMS).

`batch:comment parms line1 ...`  
Writes comment lines `line1` . . . to the `batch-port` in `parms`.

`batch:lines->file parms file line1 ...`  
Writes commands to the `batch-port` in `parms` which create a file named `file` with contents `line1` . . .

`batch:delete-file parms file`  
Writes a command to the `batch-port` in `parms` which deletes the file named `file`.

`batch:rename-file parms old-name new-name`  
Writes a command to the `batch-port` in `parms` which renames the file `old-name` to `new-name`.

In addition, batch provides some small utilities very useful for writing scripts:

`truncate-up-to path char`  
`truncate-up-to path string`
**truncate-up-to** path charlist  
*Function* 
(path can be a string or a list of strings. Returns path sans any prefixes ending with a character of the second argument. This can be used to derive a filename moved locally from elsewhere.)*

```
(truncate-up-to "/usr/local/lib/slib/batch.scm" "/")
⇒ "batch.scm"
```

**string-join** joiner string1 ...  
*Function* 
(Returns a new string consisting of all the strings string1 ... in order appended together with the string joiner between each adjacent pair.)*

**must-be-first** list1 list2  
*Function* 
(Returns a new list consisting of the elements of list2 ordered so that if some elements of list1 are equal? to elements of list2, then those elements will appear first and in the order of list1.)*

**must-be-last** list1 list2  
*Function* 
(Returns a new list consisting of the elements of list1 ordered so that if some elements of list2 are equal? to elements of list1, then those elements will appear last and in the order of list2.)*

**os->batch-dialect** osname  
*Function* 
(Returns its best guess for the batch-dialect to be used for the operating-system named osname. os->batch-dialect uses the tables added to database by batch:initialize!.*

Here is an example of the use of most of batch’s procedures:

```
(require 'databases)
(require 'parameters)
(require 'batch)
(require 'filename)

(define batch (create-database #f 'alist-table))
(batch:initialize! batch)

(define my-parameters
  (list (list 'batch-dialect (os->batch-dialect *operating-system*))
        (list 'operating-system *operating-system*)
        (list 'batch-port (current-output-port)))) ;gets filled in later

(batch:call-with-output-script
  my-parameters
  "my-batch"
  (lambda (batch-port)
    (adjoin-parameters! my-parameters (list 'batch-port batch-port))
    (and
      (batch:comment my-parameters

```
"================ Write file with C program."

(batch:rename-file my-parameters "hello.c" "hello.c~")

(batch:lines->file my-parameters "hello.c"
  "#include <stdio.h>
  "int main(int argc, char **argv)"
  "{
    " printf("hello world\n");
    " return 0;
  }
"
)

(batch:command my-parameters "cc" "-c" "hello.c")

(batch:command my-parameters "cc" "-o" "hello"
  (replace-suffix "hello.c" "c" "o"))

(batch:command my-parameters "hello")

(batch:delete-file my-parameters "hello")

(batch:delete-file my-parameters "hello.c")

(batch:delete-file my-parameters "hello.o")

(batch:delete-file my-parameters "my-batch")
))

Produces the file 'my-batch':

```bash
#!/bin/sh
# "my-batch" script created by SLIB/batch Sun Oct 31 18:24:10 1999
# ================ Write file with C program.
mv -f hello.c hello.c~
rm -f hello.c
echo '#include <stdio.h>'>>hello.c
echo 'int main(int argc, char **argv)'
  '{
  printf("hello world\n");
  return 0;
  
}'>>hello.c
cc -c hello.c
cc -o hello hello.o
hello
rm -f hello
rm -f hello.c
rm -f hello.o
rm -f my-batch
```

When run, 'my-batch' prints:

```
bash$ my-batch
mv: hello.c: No such file or directory
hello world
```

4.5 HTML

(require 'html-form)
### html:atval

[Function]

```
html:atval txt
```

Returns a string with character substitutions appropriate to send `txt` as an attribute-value.

### html:plain

[Function]

```
html:plain txt
```

Returns a string with character substitutions appropriate to send `txt` as a plain-text.

### html:meta name content

[Function]

```
html:meta name content
```

Returns a tag of meta-information suitable for passing as the third argument to `html:head`. The tag produced is `<META NAME="name" CONTENT="content">`. The string or symbol `name` can be ‘author’, ‘copyright’, ‘keywords’, ‘description’, ‘date’, ‘robots’, ....

### html:http-equiv name content

[Function]

```
html:http-equiv name content
```

Returns a tag of HTTP information suitable for passing as the third argument to `html:head`. The tag produced is `<META HTTP-EQUIV="name" CONTENT="content">`. The string or symbol `name` can be ‘Expires’, ‘PICS-Label’, ‘Content-Type’, ‘Refresh’, ....

### html:meta-refresh delay uri
delay

[Function]

```
html:meta-refresh delay uri
```

Returns a tag suitable for passing as the third argument to `html:head`. If `uri` argument is supplied, then `delay` seconds after displaying the page with this tag, Netscape or IE browsers will fetch and display `uri`. Otherwise, `delay` seconds after displaying the page with this tag, Netscape or IE browsers will fetch and redisplay this page.

### 4.6 HTML Forms

### html:form method action body ...

[Function]

```
html:form method action body ...
```

The symbol `method` is either `get`, `head`, `post`, `put`, or `delete`. The strings `body` form the body of the form. `html:form` returns the HTML form.
**html:hidden**  name value  [Function]  
Returns HTML string which will cause name=value in form.

**html:checkbox**  pname default  [Function]  
Returns HTML string for check box.

**html:text**  pname default size . . .  [Function]  
Returns HTML string for one-line text box.

**html:textarea**  pname default-list  [Function]  
Returns HTML string for multi-line text box.

**html:select**  pname arity default-list foreign-values  [Function]  
Returns HTML string for pull-down menu selector.

**html:buttons**  pname arity default-list foreign-values  [Function]  
Returns HTML string for any-of selector.

**form:submit**  submit-label command  [Function]  
The string or symbol submit-label appears on the button which submits the form. 
If the optional second argument command is given, then *command*=command and 
*button*=submit-label are set in the query. Otherwise, *command*=submit-label 
is set in the query.

**form:image**  submit-label image-src  [Function]  
The image-src appears on the button which submits the form.

**form:reset**  [Function]  
Returns a string which generates a reset button.

**form:element**  pname arity default-list foreign-values  [Function]  
Returns a string which generates an INPUT element for the field named pname. The 
element appears in the created form with its representation determined by its arity 
and domain. For domains which are foreign-keys:

- **single** select menu
- **optional** select menu
- **nary** check boxes
- **nary1** check boxes

If the foreign-key table has a field named ‘visible-name’, then the contents of that 
field are the names visible to the user for those choices. Otherwise, the foreign-key 
itself is visible.

For other types of domains:

- **single** text area
- **optional** text area
boolean    check box
nary        text area
nary1       text area

form:delimited  pname  doc  aliat  arity  default-list  foreign-values  [Function]
   Returns a HTML string for a form element embedded in a line of a delimited list.
   Apply map form:delimited to the list returned by command->p-specs.

html:delimited-list  row . . .  [Function]
   Wraps its arguments with delimited-list (‘DL’ command.

get-foreign-choices  tab  [Function]
   Returns a list of the ‘visible-name’ or first fields of table tab.

command->p-specs  rdb  command-table  command  [Function]
   The symbol command-table names a command table in the rdb relational database.
   The symbol command names a key in command-table.
   command->p-specs returns a list of lists of pname, doc, aliat, arity, default-list, and
   foreign-values. The returned list has one element for each parameter of command
   command.

   This example demonstrates how to create a HTML-form for the ‘build’ command.
   (require (in-vicinity (implementation-vicinity) "build.scm"))
   (call-with-output-file "buildscm.html"
      (lambda (port)
         (display
          (string-append
           (html:head 'commands)
           (html:body
            (sprintf #f "<H2>%s:</H2><BLOCKQUOTE>%s</BLOCKQUOTE>\n" (html:plain 'build)
               (html:plain ((comtab 'get 'documentation) 'build)))
            (html:form
             'post
             (or "http://localhost:8081/buildscm" "/cgi-bin/build.cgi")
             (apply html:delimited-list
                (apply map form:delimited
                   (command->p-specs build ’*commands* ’build)))
             (form:submit ’build)
             (form:reset))))
         port)))

4.7 HTML Tables

   (require ’db->html)

html:table  options  row . . .  [Function]
html:caption  caption  align  [Function]
html:caption caption
  align can be `top' or `bottom'.

html:heading columns
  Outputs a heading row for the currently-started table.

html:href-heading columns uris
  Outputs a heading row with column-names columns linked to URIs uris.

html:linked-row-converter k foreigns
  The positive integer k is the primary-key-limit (number of primary-keys) of the table.
  foreigns is a list of the filenames of foreign-key field pages and #f for non foreign-key fields.

html:linked-row-converter returns a procedure taking a row for its single argument. This returned procedure returns the html string for that table row.

table-name->filename table-name
  Returns the symbol table-name converted to a filename.

table->linked-html caption db table-name match-key1 . . .
  Returns HTML string for db table table-name chopped into 50-row HTML tables.
  Every foreign-key value is linked to the page (of the table) defining that key.
  The optional match-key1 . . . arguments restrict actions to a subset of the table. See Section 6.1.2 [Table Operations], page 165.

table->linked-page db table-name index-filename arg . . .
  Returns a complete HTML page. The string index-filename names the page which refers to this one.
  The optional args . . . arguments restrict actions to a subset of the table. See Section 6.1.2 [Table Operations], page 165.

catalog->html db caption arg . . .
  Returns HTML string for the catalog table of db.

4.7.1 HTML editing tables

A client can modify one row of an editable table at a time. For any change submitted, these routines check if that row has been modified during the time the user has been editing the form. If so, an error page results.

The behavior of edited rows is:
  • If no fields are changed, then no change is made to the table.
  • If the primary keys equal null-keys (parameter defaults), and no other user has modified that row, then that row is deleted.
  • If only primary keys are changed, there are non-key fields, and no row with the new keys is in the table, then the old row is deleted and one with the new keys is inserted.
  • If only non-key fields are changed, and that row has not been modified by another user, then the row is changed to reflect the fields.
• If both keys and non-key fields are changed, and no row with the new keys is in the table, then a row is created with the new keys and fields.
• If fields are changed, all fields are primary keys, and no row with the new keys is in the table, then a row is created with the new keys.

After any change to the table, a `sync-database` of the database is performed.

```scheme
command:modify-table  table-name null-keys update delete retrieve  [Function]
command:modify-table  table-name null-keys update delete             [Function]
command:modify-table  table-name null-keys update                   [Function]
command:modify-table  table-name null-keys                         [Function]
```

Returns procedure (of `db`) which returns procedure to modify row of `table-name`. `null-keys` is the list of null keys indicating the row is to be deleted when any matches its corresponding primary key. Optional arguments `update`, `delete`, and `retrieve` default to the `row:update`, `row:delete`, and `row:retrieve` of `table-name` in `db`.

```scheme
command:make-editable-table  rdb table-name arg ...  [Function]
```

Given `table-name` in `rdb`, creates parameter and `*command*` tables for editing one row of `table-name` at a time. `command:make-editable-table` returns a procedure taking a row argument which returns the HTML string for editing that row.

Optional `args` are expressions (lists) added to the call to `command:modify-table`.

The domain name of a column determines the expected arity of the data stored in that column. Domain names ending in:

- `*` have arity `nary`;
- `+` have arity `nary1`.

```scheme
html:editable-row-converter  k names edit-point edit-converter  [Function]
```

The positive integer `k` is the primary-key-limit (number of primary-keys) of the table. `names` is a list of the field-names. `edit-point` is the list of primary-keys denoting the row to edit (or `#f`). `edit-converter` is the procedure called with `k`, `names`, and the row to edit.

`html:editable-row-converter` returns a procedure taking a row for its single argument. This returned procedure returns the html string for that table row.

Each HTML table constructed using `html:editable-row-converter` has first `k` fields (typically the primary key fields) of each row linked to a text encoding of these fields (the result of calling `row->anchor`). The page so referenced typically allows the user to edit fields of that row.

### 4.7.2 HTML databases

```scheme
db->html-files  db dir index-filename caption  [Function]
```

`db` must be a relational database. `dir` must be `#f` or a non-empty string naming an existing sub-directory of the current directory.

`db->html-files` creates an html page for each table in the database `db` in the sub-directory named `dir`, or the current directory if `dir` is `#f`. The top level page with the catalog of tables (captioned `caption`) is written to a file named `index-filename`. 
db->html-directory  db dir index-filename

db->html-directory  db dir
  db must be a relational database. dir must be a non-empty string naming an existing
  sub-directory of the current directory or one to be created. The optional string index-
  filename names the filename of the top page, which defaults to ‘index.html’.

  db->html-directory creates sub-directory dir if necessary, and calls (db->html-
  files db dir index-filename dir). The ‘file:’ URI of index-filename is
  returned.

db->netscape  db dir index-filename

4.8 HTTP and CGI

(require 'http) or (require 'cgi)

http:header  alist
  Returns a string containing lines for each element of alist; the car of which is followed
  by ‘: ’, then the cdr.

http:content  alist body ...
  Returns the concatenation of strings body with the (http:header alist) and the
  ‘Content-Length’ prepended.

*http:byline*
  String appearing at the bottom of error pages.

http:error-page  status-code reason-phrase html-string ...
  status-code and reason-phrase should be an integer and string as specified in RFC
  2068. The returned page (string) will show the status-code and reason-phrase and
  any additional html-strings ...; with *http:byline* or SLIB’s default at the bottom.

http:forwarding-page  title dly uri html-string ...
  The string or symbol title is the page title. dly is a non-negative integer. The html-
  strings ... are typically used to explain to the user why this page is being forwarded.
  http:forwarding-page returns an HTML string for a page which automatically for-
  wards to uri after dly seconds. The returned page (string) contains any html-strings
  ... followed by a manual link to uri, in case the browser does not forward automati-
  cally.

http:serve-query  serve-proc input-port output-port
  reads the URI and query-string from input-port. If the query is a valid ‘"POST"'
  or ‘"GET"’ query, then http:serve-query calls serve-proc with three arguments,
  the request-line, query-string, and header-alist. Otherwise, http:serve-query calls
  serve-proc with the request-line, #f, and header-alist.
  If serve-proc returns a string, it is sent to output-port. If serve-proc returns a list
  whose first element is an integer, then an error page with the status integer which
is the first element of the list and strings from the list. If serve-proc returns a list whose first element isn’t an number, then an error page with the status code 500 and strings from the list. If serve-proc returns #f, then a ‘Bad Request’ (400) page is sent to output-port.

Otherwise, http:serve-query replies (to output-port) with appropriate HTML describing the problem.

This example services HTTP queries from port-number:

```
(define socket (make-stream-socket AF_INET 0))
(and (socket:bind socket port-number) ; AF_INET INADDR_ANY
   (socket:listen socket 10) ; Queue up to 10 requests.
   (dynamic-wind
     (lambda () #f)
     (lambda ()
       (do ((port (socket:accept socket) (socket:accept socket)))
          (#f)
          (let ((iport (duplicate-port port "r")))
             (oport (duplicate-port port "w")))
             (http:serve-query build:serve iport oport)
             (close-port iport)
             (close-port oport))
             (close-port port)))
   (lambda () (close-port socket)))
```

cgi:serve-query serve-proc [Function]
reads the URI and query-string from (current-input-port). If the query is a valid ‘POST’ or ‘GET’ query, then cgi:serve-query calls serve-proc with three arguments, the request-line, query-string, and header-alist. Otherwise, cgi:serve-query calls serve-proc with the request-line, #f, and header-alist.

If serve-proc returns a string, it is sent to (current-output-port). If serve-proc returns a list whose first element is an integer, then an error page with the status integer which is the first element of the list and strings from the list. If serve-proc returns a list whose first element isn’t an number, then an error page with the status code 500 and strings from the list. If serve-proc returns #f, then a ‘Bad Request’ (400) page is sent to (current-output-port).

Otherwise, cgi:serve-query replies (to (current-output-port)) with appropriate HTML describing the problem.

make-query-alist-command-server rdb command-table [Function]
make-query-alist-command-server rdb command-table #t [Function]
Returns a procedure of one argument. When that procedure is called with a query-alist (as returned by uri:decode-query), the value of the ‘*command*’ association will be the command invoked in command-table. If ‘*command*’ is not in the query-alist then the value of ‘*suggest*’ is tried. If neither name is in the query-alist, then the literal value ‘*default*’ is tried in command-table.
If optional third argument is non-false, then the command is called with just the parameter-list; otherwise, command is called with the arguments described in its table.

4.9 Parsing HTML

(require 'html-for-each)

(html-for-each file word-proc markup-proc white-proc newline-proc)

File is an input port or a string naming an existing file containing HTML text. Word-proc is a procedure of one argument or #f. Markup-proc is a procedure of one argument or #f. White-proc is a procedure of one argument or #f. Newline-proc is a procedure of no arguments or #f.

(html-for-each) opens and reads characters from port file or the file named by string file. Sequential groups of characters are assembled into strings which are either

- enclosed by ‘<’ and ‘>’ (hypertext markups or comments);
- end-of-line;
- whitespace; or
- none of the above (words).

Procedures are called according to these distinctions in order of the string’s occurrence in file.

Newline-proc is called with no arguments for end-of-line not within a markup or comment.

White-proc is called with strings of non-newline whitespace.

Markup-proc is called with hypertext markup strings (including ‘<’ and ‘>’).

Word-proc is called with the remaining strings.

(html-for-each) returns an unspecified value.

(html:read-title file limit)

File is an input port or a string naming an existing file containing HTML text. If supplied, limit must be an integer. Limit defaults to 1000.

(html:read-title) opens and reads HTML from port file or the file named by string file, until reaching the (mandatory) ‘TITLE’ field. html:read-title returns the title string with adjacent whitespaces collapsed to one space. html:read-title returns #f if the title field is empty, absent, if the first character read from file is not ‘#<’, or if the end of title is not found within the first (approximately) limit words.

(htm-fields htm)

Htm is a hypertext markup string.

If htm is a (hypertext) comment or DTD, then htm-fields returns #f. Otherwise htm-fields returns the hypertext element string consed onto an association list of the attribute name-symbols and values. If the tag ends with "/>", then "/" is appended to the hypertext element string. The name-symbols are created by string-ci->symbol. Each value is a string; or #t if the name had no value assigned within the markup.
4.10 URI

(require 'uri)

Implements Uniform Resource Identifiers (URI) as described in RFC 2396.

(make-uri) [Function]

Returns a Uniform Resource Identifier string from component arguments.

(make-uri fragment) [Function]

Returns a URI string combining the components of list path.

(html:anchor name) [Function]

Returns a string which defines this location in the (HTML) file as name. The hypertext `<A HREF="#name">` will link to this point.

(html:link uri highlighted) [Function]

Returns a string which links the highlighted text to uri.

(html:base uri) [Function]

Returns a string specifying the base uri of a document, for inclusion in the HEAD of the document (see Section 4.5 [HTML], page 69).

(html:isindex prompt) [Function]

Returns a string specifying the search prompt of a document, for inclusion in the HEAD of the document (see Section 4.5 [HTML], page 69).

(uri->tree uri-reference base-tree) [Function]

Returns a list of 5 elements corresponding to the parts (scheme authority path query fragment) of string uri-reference. Elements corresponding to absent parts are #f.

The path is a list of strings. If the first string is empty, then the path is absolute; otherwise relative. The optional base-tree is a tree as returned by uri->tree; and is used as the base address for relative URIs.

If the authority component is a Server-based Naming Authority, then it is a list of the userinfo, host, and port strings (or #f). For other types of authority components the authority will be a string.
(uri->tree "http://www.ics.uci.edu/pub/ietf/uri/#Related")
⇒
(http "www.ics.uci.edu" ("" "pub" "ietf" "uri" ") #f "Related")

(uri:split-fields txt chr)
  Returns a list of txt split at each occurrence of chr. chr does not appear in the
  returned list of strings.

(uri:decode-query query-string)
  Converts a URI encoded query-string to a query-alist.

uric: prefixes indicate procedures dealing with URI-components.

(uri:encode uri-component allows)
  Returns a copy of the string uri-component in which all unsafe octets (as defined
  in RFC 2396) have been '% escaped. uri:decode decodes strings encoded by
  uri:encode.

(uri:decode uri-component)
  Returns a copy of the string uri-component in which each '%' escaped characters in
  uri-component is replaced with the character it encodes. This routine is useful for
  showing URI contents on error pages.

(uri:path->keys path-list ptypes)
  path-list is a path-list as returned by uri:split-fields. uri:path->keys returns
  a list of items returned by uri:decode-path, coerced to types ptypes.

File-system Locators and Predicates

(path->uri path)
  Returns a URI-string for path on the local host.

(absolute-uri? str)
  Returns #t if str is an absolute-URI as indicated by a syntactically valid (per RFC
  2396) scheme; otherwise returns #f.

(absolute-path? file-name)
  Returns #t if file-name is a fully specified pathname (does not depend on the current
  working directory); otherwise returns #f.

(null-directory? str)
  Returns #t if changing directory to str would leave the current directory unchanged;
  otherwise returns #f.

(glob-pattern? str)
  Returns #t if the string str contains characters used for specifying glob patterns,
  namely '*', '?', or '['.

Before RFC 2396, the File Transfer Protocol (FTP) served a similar purpose.
parse-ftp-address uri

[Function]

Returns a list of the decoded FTP uri; or #f if indecipherable. FTP Uniform Resource Locator, ange-ftp, and getit formats are handled. The returned list has four elements which are strings or #f:

0. username
1. password
2. remote-site
3. remote-directory

4.11 Parsing XML

(require 'xml-parse) or (require 'ssax)


The present frameworks fully supports the XML Namespaces Recommendation http://www.w3.org/TR/REC-xml-names.

4.11.1 String Glue

ssax:reverse-collect-str list-of-frags

[Function]

Given the list of fragments (some of which are text strings), reverse the list and concatenate adjacent text strings. If LIST-OF-FRAGS has zero or one element, the result of the procedure is equal? to its argument.

ssax:reverse-collect-str-drop-ws list-of-frags

[Function]

Given the list of fragments (some of which are text strings), reverse the list and concatenate adjacent text strings while dropping "insignificant" whitespace, that is, whitespace in front, behind and between elements. The whitespace that is included in character data is not affected.

Use this procedure to "intelligently" drop "insignificant" whitespace in the parsed SXML. If the strict compliance with the XML Recommendation regarding the whitespace is desired, use the ssax:reverse-collect-str procedure instead.

4.11.2 Character and Token Functions

The following functions either skip, or build and return tokens, according to inclusion or delimiting semantics. The list of characters to expect, include, or to break at may vary from one invocation of a function to another. This allows the functions to easily parse even context-sensitive languages.

Exceptions are mentioned specifically. The list of expected characters (characters to skip until, or break-characters) may include an EOF "character", which is coded as symbol *eof*

The input stream to parse is specified as a PORT, which is the last argument.
ssax:assert-current-char char-list string port  [Function]
Reads a character from the port and looks it up in the char-list of expected characters. If
the read character was found among expected, it is returned. Otherwise, the procedure
writes a message using string as a comment and quits.

ssax:skip-while char-list port  [Function]
Reads characters from the port and disregards them, as long as they are mentioned
in the char-list. The first character (which may be EOF) peeked from the stream that
is not a member of the char-list is returned.

ssax:init-buffer  [Function]
Returns an initial buffer for ssax:next-token* procedures. ssax:init-buffer may
allocate a new buffer at each invocation.

ssax:next-token prefix-char-list break-char-list comment-string port  [Function]
Skips any number of the prefix characters (members of the prefix-char-list), if any,
and reads the sequence of characters up to (but not including) a break character, one
of the break-char-list.

The string of characters thus read is returned. The break character is left on the
input stream. break-char-list may include the symbol *eof*; otherwise, EOF is fatal,
generating an error message including a specified comment-string.

ssax:next-token-of is similar to ssax:next-token except that it implements an inclusion
rather than delimiting semantics.

ssax:next-token-of inc-charset port  [Function]
Reads characters from the port that belong to the list of characters inc-charset. The
reading stops at the first character which is not a member of the set. This character
is left on the stream. All the read characters are returned in a string.

ssax:next-token-of pred port  [Function]
Reads characters from the port for which pred (a procedure of one argument) returns
non-#f. The reading stops at the first character for which pred returns #f. That
character is left on the stream. All the results of evaluating of pred up to #f are
returned in a string.

pred is a procedure that takes one argument (a character or the EOF object) and
returns a character or #f. The returned character does not have to be the same as
the input argument to the pred. For example,

```lisp
(ssax:next-token-of (lambda (c)
    (cond ((eof-object? c) #f)
          ((char-alphabetic? c) (char-downcase c))
          (else #f)))
    (current-input-port))
```

will try to read an alphabetic token from the current input port, and return it in
lower case.
**ssax:read-string**  len  port

[Function]
Reads  \textit{len} characters from the \textit{port}, and returns them in a string. If EOF is encountered before \textit{len} characters are read, a shorter string will be returned.

### 4.11.3 Data Types

**TAG-KIND**
A symbol ‘\texttt{START}’, ‘\texttt{END}’, ‘\texttt{PI}’, ‘\texttt{DECL}’, ‘\texttt{COMMENT}’, ‘\texttt{CDSECT}’, or ‘\texttt{ENTITY-REF}’ that identifies a markup token

**UNRES-NAME**
a name (called GI in the XML Recommendation) as given in an XML document for a markup token: start-tag, PI target, attribute name. If a GI is an NCName, UNRES-NAME is this NCName converted into a Scheme symbol. If a GI is a QName, ‘UNRES-NAME’ is a pair of symbols: \((\texttt{PREFIX} . \texttt{LOCALPART})\).

**RES-NAME**
An expanded name, a resolved version of an ‘UNRES-NAME’. For an element or an attribute name with a non-empty namespace URI, ‘RES-NAME’ is a pair of symbols, \((\texttt{URI-SYMB} . \texttt{LOCALPART})\). Otherwise, it’s a single symbol.

**ELEM-CONTENT-MODEL**
A symbol:
- ‘\texttt{ANY}’ anything goes, expect an END tag.
- ‘\texttt{EMPTY-TAG}’ no content, and no END-tag is coming
- ‘\texttt{EMPTY}’ no content, expect the END-tag as the next token
- ‘\texttt{PCDATA}’ expect character data only, and no children elements
- ‘\texttt{MIXED}’
- ‘\texttt{ELEM-CONTENT}’

**URI-SYMB**
A symbol representing a namespace URI – or other symbol chosen by the user to represent URI. In the former case, URI-SYMB is created by %-quoting of bad URI characters and converting the resulting string into a symbol.

**NAMESPACE**
A list representing namespaces in effect. An element of the list has one of the following forms:
- \((\texttt{prefix uri-symb} . \texttt{uri-symb})\) or
- \((\texttt{prefix user-prefix} . \texttt{uri-symb})\)
  \texttt{user-prefix} is a symbol chosen by the user to represent the URI.
- \((\texttt{#f user-prefix} . \texttt{uri-symb})\)
  Specification of the user-chosen prefix and a URI-SYMBOL.
- \((\texttt{*DEFAULT* user-prefix} . \texttt{uri-symb})\)
  Declaration of the default namespace
Un-declaration of the default namespace. This notation represents
overriding of the previous declaration

A NAMESPACES list may contain several elements for the same prefix. The
one closest to the beginning of the list takes effect.

**ATTLIST**

An ordered collection of (NAME . VALUE) pairs, where NAME is a RES-
NAME or an UNRES-NAME. The collection is an ADT.

**STR-HANDLER**

A procedure of three arguments: string1 string2 seed returning a new seed.
The procedure is supposed to handle a chunk of character data string1 followed
by a chunk of character data string2. string2 is a short string, often "\n" and
even "".

**ENTITIES**

An assoc list of pairs:

(named-entity-name . named-entity-body)

where named-entity-name is a symbol under which the entity was declared,
named-entity-body is either a string, or (for an external entity) a thunk that
will return an input port (from which the entity can be read). named-entity-
body may also be #f. This is an indication that a named-entity-name is cur-
rently being expanded. A reference to this named-entity-name will be an error:
violation of the WFC nonrecursion.

**XML-TOKEN**

This record represents a markup, which is, according to the XML Recommen-
dation, "takes the form of start-tags, end-tags, empty-element tags, entity ref-
erences, character references, comments, CDATA section delimiters, document
type declarations, and processing instructions."

kind a TAG-KIND

head an UNRES-NAME. For XML-TOKENs of kinds 'COMMENT and
'CDSECT, the head is #f.

For example,

\[
\begin{align*}
\langle P \rangle & \Rightarrow \text{kind=START, head=P} \\
\langle /P \rangle & \Rightarrow \text{kind=END, head=P} \\
\langle BR \rangle & \Rightarrow \text{kind=EMPTY-EL, head=BR} \\
\langle !DOCTYPE OMF ... \rangle & \Rightarrow \text{kind=DECL, head=DOCTYPE} \\
\langle ?xml version="1.0"?> & \Rightarrow \text{kind=PI, head=xml} \\
& \text{my-ent;} & \Rightarrow \text{kind=ENTITY-REF, head=my-ent}
\end{align*}
\]

Character references are not represented by xml-tokens as these references are
transparently resolved into the corresponding characters.

**XML-DECL**

The record represents a datatype of an XML document: the list of declared
elements and their attributes, declared notations, list of replacement strings or
loading procedures for parsed general entities, etc. Normally an XML-DECL
record is created from a DTD or an XML Schema, although it can be created and filled in in many other ways (e.g., loaded from a file).

elems  an (assoc) list of decl-elem or #f. The latter instructs the parser to do no validation of elements and attributes.

decl-elem  declaration of one element:
  \[(\text{elem-name} \ \text{elem-content} \ \text{decl-attrs})\]
  elem-name is an UNRES-NAME for the element.
  elem-content is an ELEM-CONTENT-MODEL. 
  decl-attrs is an ATTLIST, of (attr-name . value) associations.
  This element can declare a user procedure to handle parsing of an element (e.g., to do a custom validation, or to build a hash of IDs as they’re encountered).

decl-attr  an element of an ATTLIST, declaration of one attribute:
  \[(\text{attr-name} \ \text{content-type} \ \text{use-type} \ \text{default-value})\]
  attr-name is an UNRES-NAME for the declared attribute.
  content-type is a symbol: CDATA, NMTOKEN, NMTOKENS, ... or a list of strings for the enumerated type.
  use-type is a symbol: REQUIRED, IMPLIED, or FIXED.
  default-value is a string for the default value, or #f if not given.

4.11.4 Low-Level Parsers and Scanners

These procedures deal with primitive lexical units (Names, whitespaces, tags) and with pieces of more generic productions. Most of these parsers must be called in appropriate context. For example, \texttt{ssax:complete-start-tag} must be called only when the start-tag has been detected and its GI has been read.

\texttt{ssax:skip-s port} \hspace{1cm} \textbf{[Function]}

Skip the S (whitespace) production as defined by

[3] S ::= (#x20 | #x09 | #x0D | #x0A)

\texttt{ssax:skip-s} returns the first not-whitespace character it encounters while scanning the \texttt{port}. This character is left on the input stream.

\texttt{ssax:read-ncname port} \hspace{1cm} \textbf{[Function]}

Read a NCName starting from the current position in the \texttt{port} and return it as a symbol.

[4] NameChar ::= Letter | Digit | ’.’ | ’-’ | ’_’ | ‘:’ | CombiningChar | Extender

[5] Name ::= (Letter | ’_’ | ‘:’) (NameChar)*

This code supports the XML Namespace Recommendation REC-xml-names, which modifies the above productions as follows:

[4] NCNameChar ::= Letter | Digit | ’.’ | ’-’ | ’_’ | ‘:’ | CombiningChar | Extender
As the Rec-xml-names says, "An XML document conforms to this specification if all other tokens [other than element types and attribute names] in the document which are required, for XML conformance, to match the XML production for Name, match this specification's production for NCName."

Element types and attribute names must match the production QName, defined below.

**ssax:read-qname** *port*  
Read a (namespace-) Qualified Name, QName, from the current position in *port*; and return an UNRES-NAME.

From REC-xml-names:

- [6] QName ::= (Prefix ':')? LocalPart
- [7] Prefix ::= NCName
- [8] LocalPart ::= NCName

**ssax:read-markup-token** *port*  
This procedure starts parsing of a markup token. The current position in the stream must be '<'. This procedure scans enough of the input stream to figure out what kind of a markup token it is seeing. The procedure returns an XML-TOKEN structure describing the token. Note, generally reading of the current markup is not finished!

In particular, no attributes of the start-tag token are scanned.

Here's a detailed break out of the return values and the position in the PORT when that particular value is returned:

- **PI-token**
  - only PI-target is read. To finish the Processing-Instruction and disregard it, call **ssax:skip-pi**. **ssax:read-attributes** may be useful as well (for PIs whose content is attribute-value pairs).

- **END-token**
  - The end tag is read completely; the current position is right after the terminating '>' character.

- **COMMENT**
  - is read and skipped completely. The current position is right after '--->' that terminates the comment.

- **CDSECT**
  - The current position is right after '<!CDATA['. Use **ssax:read-cdata-body** to read the rest.

- **DECL**
  - We have read the keyword (the one that follows '<!') identifying this declaration markup. The current position is after the keyword (usually a whitespace character)
START-token

We have read the keyword (GI) of this start tag. No attributes are scanned yet. We don’t know if this tag has an empty content either. Use ssax:complete-start-tag to finish parsing of the token.

ssax:skip-pi port
The current position is inside a PI. Skip till the rest of the PI

ssax:read-pi-body-as-string port
The current position is right after reading the PITarget. We read the body of PI and return it as a string. The port will point to the character right after ‘?>’ combination that terminates PI.

\[16\] PI ::= ‘<?’ PITarget (S (Char* - (Char* ‘?>’ Char*))?) ‘?>’

ssax:skip-internal-dtd port
The current position in the port is inside an internal DTD subset (e.g., after reading ‘#\[' that begins an internal DTD subset) Skip until the ‘]>’ combination that terminates this DTD.

ssax:read-cdata-body port str-handler seed
This procedure must be called after we have read a string ‘<![CDATA[’ that begins a CDATA section. The current position must be the first position of the CDATA body. This function reads lines of the CDATA body and passes them to a str-handler, a character data consumer.

str-handler is a procedure taking arguments: string1, string2, and seed. The first string1 argument to str-handler never contains a newline; the second string2 argument often will. On the first invocation of str-handler, seed is the one passed to ssax:read-cdata-body as the third argument. The result of this first invocation will be passed as the seed argument to the second invocation of the line consumer, and so on. The result of the last invocation of the str-handler is returned by the ssax:read-cdata-body. Note a similarity to the fundamental fold iterator.

Within a CDATA section all characters are taken at their face value, with three exceptions:

- CR, LF, and CRLF are treated as line delimiters, and passed as a single ‘\n’ to str-handler
- ‘]>’ combination is the end of the CDATA section. ‘&gt;’ is treated as an embedded ‘>’ character.
- ‘&lt;’ and ‘&amp;’ are not specially recognized (and are not expanded)!

ssax:read-char-ref port

\[66\] CharRef ::= ‘&’ [0-9]+ ‘;’
| ‘&#’ [0-9a-fA-F]+ ‘;’

This procedure must be called after we have read ‘&’ that introduces a char reference. The procedure reads this reference and returns the corresponding char. The current position in PORT will be after the ‘;’ that terminates the char reference.
Faults detected:
WFC: XML-Spec.html#wf-Legalchar

According to Section 4.1 Character and Entity References of the XML Recommendation:

"[Definition: A character reference refers to a specific character in the ISO/IEC 10646 character set, for example one not directly accessible from available input devices.]

**ssax:handle-parsed-entity**  
**port name entities content-handler**  

*Function*

Expands and handles a parsed-entity reference.

*name* is a symbol, the name of the parsed entity to expand. *content-handler* is a procedure of arguments *port*, *entities*, and *seed* that returns a seed. *str-handler* is called if the entity in question is a pre-declared entity.

**ssax:handle-parsed-entity** returns the result returned by **content-handler** or **str-handler**.

Faults detected:
WFC: XML-Spec.html#wf-entdeclared
WFC: XML-Spec.html#norecursion

**attlist-add**  
**attlist name-value**  

*Function*

Add a *name-value* pair to the existing *attlist*, preserving its sorted ascending order; and return the new list. Return #f if a pair with the same name already exists in *attlist*

**attlist-remove-top**  
**attlist**  

*Function*

Given a non-null *attlist*, return a pair of values: the top and the rest.

**ssax:read-attributes**  
**port entities**  

*Function*

This procedure reads and parses a production *Attribute*.

\[
[41] \text{Attribute} ::= \text{Name Eq AttValue} \\
[10] \text{AttValue} ::= "", (\[^<&"] | \text{Reference}\)* "", \\
\text{Eq} ::= S? '=' S?
\]

The procedure returns an ATTLIST, of Name (as UNRES-NAME), Value (as string) pairs. The current character on the *port* is a non-whitespace character that is not an NCName-starting character.

Note the following rules to keep in mind when reading an *AttValue*:

Before the value of an attribute is passed to the application or checked for validity, the XML processor must normalize it as follows:

- A character reference is processed by appending the referenced character to the attribute value.
- An entity reference is processed by recursively processing the replacement text of the entity. The named entities ‘amp’, ‘lt’, ‘gt’, ‘quot’, and ‘apos’ are pre-declared.
• A whitespace character (#x20, #xD, #xA, #x09) is processed by appending #x20 to the normalized value, except that only a single #x20 is appended for a "#xD#xA" sequence that is part of an external parsed entity or the literal entity value of an internal parsed entity.

• Other characters are processed by appending them to the normalized value.

Faults detected:
WFC: XML-Spec.html#CleanAttrVals
WFC: XML-Spec.html#uniqattspec

**ssax:resolve-name**  port unres-name namespaces apply-default-ns?  [Function]
Convert an unres-name to a RES-NAME, given the appropriate namespaces declarations. The last parameter, apply-default-ns?, determines if the default namespace applies (for instance, it does not for attribute names).

Per REC-xml-names/#nsc-NSDeclared, the "xml" prefix is considered pre-declared and bound to the namespace name "http://www.w3.org/XML/1998/namespace".

**ssax:resolve-name** tests for the namespace constraints:
http://www.w3.org/TR/REC-xml-names/#nsc-NSDeclared

**ssax:complete-start-tag**  tag port elems entities namespaces  [Function]
Complete parsing of a start-tag markup. ssax:complete-start-tag must be called after the start tag token has been read. tag is an UNRES-NAME. elems is an instance of the ELEMS slot of XML-DECL; it can be #f to tell the function to do no validation of elements and their attributes.

**ssax:complete-start-tag** returns several values:

• ELEM-GI: a RES-NAME.

• ATTRIBUTES: element’s attributes, an ATTLIST of (RES-NAME . STRING) pairs. The list does NOT include xmlns attributes.

• NAMESPACES: the input list of namespaces amended with namespace (re-)declarations contained within the start-tag under parsing

• ELEM-CONTENT-MODEL

On exit, the current position in port will be the first character after ‘>’ that terminates the start-tag markup.

Faults detected:
VC: XML-Spec.html#enum
VC: XML-Spec.html#RequiredAttr
VC: XML-Spec.html#FixedAttr
VC: XML-Spec.html#ValueType
WFC: XML-Spec.html#uniqattspec (after namespaces prefixes are resolved)
VC: XML-Spec.html#elementvalid
WFC: REC-xml-names/#dt-NSName
Note: although XML Recommendation does not explicitly say it, xmlns and xmlns: attributes don’t have to be declared (although they can be declared, to specify their default value).

ssax:read-external-id port  
 Parses an ExternalID production:

[75] ExternalID ::= 'SYSTEM' S SystemLiteral  
   | 'PUBLIC' S PubidLiteral S SystemLiteral  
   | ("" [~"]* "")  
[12] PubidLiteral ::= "" PubidChar* ""  
   | "" (PubidChar - "")* ""  
[13] PubidChar ::= #x20 | #xD | #xA | [a-zA-Z0-9]  
   | [~']()+,./:=?;!*#@$_%  

Call ssax:read-external-id when an ExternalID is expected; that is, the current character must be either #S or #P that starts correspondingly a SYSTEM or PUBLIC token. ssax:read-external-id returns the SystemLiteral as a string. A PubidLiteral is disregarded if present.

4.11.5 Mid-Level Parsers and Scanners  
These procedures parse productions corresponding to the whole (document) entity or its higher-level pieces (prolog, root element, etc).

ssax:scan-misc port  
 Scan the Misc production in the context:

[1] document ::= prolog element Misc*  
[22] prolog ::= XMLDecl? Misc* (doctypedec l Misc*)?  
[27] Misc ::= Comment | PI | S  

Call ssax:scan-misc in the prolog or epilog contexts. In these contexts, whitespaces are completely ignored. The return value from ssax:scan-misc is either a PI-token, a DECL-token, a START token, or *EOF*. Comments are ignored and not reported.

ssax:read-char-data port expect-eof? str-handler iseed  
 Read the character content of an XML document or an XML element.

[43] content ::=  
   (element | CharData | Reference | CDSect | PI | Comment)*  

To be more precise, ssax:read-char-data reads CharData, expands CDSect and character entities, and skips comments. ssax:read-char-data stops at a named reference, EOF, at the beginning of a PI, or a start/end tag.

expect-eof? is a boolean indicating if EOF is normal; i.e., the character data may be terminated by the EOF. EOF is normal while processing a parsed entity.

iseed is an argument passed to the first invocation of str-handler.

ssax:read-char-data returns two results: seed and token. The seed is the result of the last invocation of str-handler, or the original iseed if str-handler was never called. token can be either an eof-object (this can happen only if expect-eof? was #t), or:
• an xml-token describing a START tag or an END-tag; For a start token, the caller has to finish reading it.
• an xml-token describing the beginning of a PI. It’s up to an application to read or skip through the rest of this PI;
• an xml-token describing a named entity reference.

CDATA sections and character references are expanded inline and never returned. Comments are silently disregarded.

As the XML Recommendation requires, all whitespace in character data must be preserved. However, a CR character (#x0D) must be disregarded if it appears before a LF character (#x0A), or replaced by a #x0A character otherwise. See Secs. 2.10 and 2.11 of the XML Recommendation. See also the canonical XML Recommendation.

**ssax:assert-token**

```lf
ssax:assert-token token kind gi error-cont
```

Make sure that *token* is of anticipated *kind* and has anticipated *gi*. Note that the *gi* argument may actually be a pair of two symbols, Namespace-URI or the prefix, and of the localname. If the assertion fails, *error-cont* is evaluated by passing it three arguments: *token kind gi*. The result of *error-cont* is returned.

### 4.11.6 High-level Parsers

These procedures are to instantiate a SSAX parser. A user can instantiate the parser to do the full validation, or no validation, or any particular validation. The user specifies which PI he wants to be notified about. The user tells what to do with the parsed character and element data. The latter handlers determine if the parsing follows a SAX or a DOM model.

**ssax:make-pi-parser**

```lf
ssax:make-pi-parser my-pi-handlers
```

Create a parser to parse and process one Processing Element (PI). *my-pi-handlers* is an association list of pairs (pi-tag . pi-handler) where pi-tag is an NCName symbol, the PI target; and pi-handler is a procedure taking arguments *port*, *pi-tag*, and *seed*.

*pi-handler* should read the rest of the PI up to and including the combination ‘?>’ that terminates the PI. The handler should return a new seed. One of the *pi-tags* may be the symbol *DEFAULT*. The corresponding handler will handle PIs that no other handler will. If the *DEFAULT* *pi-tag* is not specified, *ssax:make-pi-parser* will assume the default handler that skips the body of the PI.

*ssax:make-pi-parser* returns a procedure of arguments *port*, *pi-tag*, and *seed*; that will parse the current PI according to *my-pi-handlers*.

**ssax:make-elem-parser**

```lf
ssax:make-elem-parser my-new-level-seed my-finish-element
```

Create a parser to parse and process one element, including its character content or children elements. The parser is typically applied to the root element of a document.

*my-new-level-seed* is a procedure taking arguments:

```
  elem-gi attributes namespaces expected-content seed
```
where \textit{elem-gi} is a RES-NAME of the element about to be processed. \\
\textit{my-new-level-seed} is to generate the seed to be passed to handlers that process the content of the element.

\textit{my-finish-element} \\
is a procedure taking arguments: \\
\textit{elem-gi attributes namespaces parent-seed seed} \\
\textit{my-finish-element} is called when parsing of \textit{elem-gi} is finished. The \textit{seed} is the result from the last content parser (or from \textit{my-new-level-seed} if the element has the empty content). \textit{parent-seed} is the same seed as was passed to \textit{my-new-level-seed}. \textit{my-finish-element} is to generate a seed that will be the result of the element parser.

\textit{my-char-data-handler} \\
is a STR-HANDLER as described in Data Types above.

\textit{my-pi-handlers} \\
is as described for \texttt{ssax:make-pi-handler} above.

The generated parser is a procedure taking arguments: \\
\textit{start-tag-head port elems entities namespaces preserve-ws? seed} \\
The procedure must be called after the start tag token has been read. \textit{start-tag-head} is an UNRES-NAME from the start-element tag. ELEMS is an instance of ELEMS slot of XML-DECL.

Faults detected: \\
VC: XML-Spec.html\#elementvalid \\
WFC: XML-Spec.html\#GIMatch

\texttt{ssax:make-parser user-handler-tag user-handler \ldots} \hspace{1cm} [Function] \\
Create an XML parser, an instance of the XML parsing framework. This will be a SAX, a DOM, or a specialized parser depending on the supplied user-handlers.

\texttt{ssax:make-parser} takes an even number of arguments; \texttt{user-handler-tag} is a symbol that identifies a procedure (or association list for PROCESSING-INSTRUCTIONS) (\texttt{user-handler}) that follows the tag. Given below are tags and signatures of the corresponding procedures. Not all tags have to be specified. If some are omitted, reasonable defaults will apply.

\texttt{DOCTYPE} \\
\hspace{1cm} handler-procedure: \texttt{port docname systemid internal-subset? seed} \\
If \texttt{internal-subset?} is \#t, the current position in the port is right after we have read \{' that begins the internal DTD subset. We must finish reading of this subset before we return (or must call \texttt{skip-internal-dtd} if we aren’t interested in reading it). \texttt{port} at exit must be at the first symbol after the whole DOCTYPE declaration.

The handler-procedure must generate four values: \\
\texttt{elems entities namespaces seed} \\
\texttt{elems} is as defined for the ELEMS slot of XML-DECL. It may be \#f to switch off validation. \texttt{namespaces} will typically contain user-prefixes
for selected uri-syms. The default handler-procedure skips the internal subset, if any, and returns (values #f '() '() seed).

‘UNDECL-ROOT’
procedure:  
where elem-gi is an UNRES-NAME of the root element. This procedure is called when an XML document under parsing contains no DOCTYPE declaration.

The handler-procedure, as a DOCTYPE handler procedure above, must generate four values:

elems entities namespaces seed

The default handler-procedure returns (values #f '() '() seed)

‘DECL-ROOT’
procedure:  
where elem-gi is an UNRES-NAME of the root element. This procedure is called when an XML document under parsing does contains the DOCTYPE declaration. The handler-procedure must generate a new seed (and verify that the name of the root element matches the doctype, if the handler so wishes). The default handler-procedure is the identity function.

‘NEW-LEVEL-SEED’
procedure: see ssax:make-elem-parser, my-new-level-seed

‘FINISH-ELEMENT’
procedure: see ssax:make-elem-parser, my-finish-element

‘CHAR-DATA-HANDLER’
procedure: see ssax:make-elem-parser, my-char-data-handler

‘PROCESSING-INSTRUCTIONS’
association list as is passed to ssax:make-pi-parser. The default value is '(')

The generated parser is a procedure of arguments port and seed.

This procedure parses the document prolog and then exits to an element parser (created by ssax:make-elem-parser) to handle the rest.

[1] document ::= prolog element Misc*
[22] prolog ::= XMLDecl? Misc* (doctypedec | Misc*)?
[27] Misc ::= Comment | PI | S
[28] doctypedec ::= '<!DOCTYPE' S Name (S ExternalID)? S?
       (['!' (markupdecl | PEReference | S)* '] 'S)? '>
[29] markupdecl ::= elementdecl | AttlistDecl
             | EntityDecl
             | NotationDecl | PI
             | Comment
4.11.7 Parsing XML to SXML

ssax:xml->sxml port namespace-prefix-assig  [Function]
This is an instance of the SSAX parser that returns an SXML representation of
the XML document to be read from port. namespace-prefix-assig is a list of (user-
prefix . uri-string) that assigns user-prefixes to certain namespaces identified by
particular uri-strings. It may be an empty list. ssax:xml->sxml returns an SXML
tree. The port points out to the first character after the root element.

4.12 Printing Scheme

4.12.1 Generic-Write

(require 'generic-write)

generic-write is a procedure that transforms a Scheme data value (or Scheme pro-
gram expression) into its textual representation and prints it. The interface to the procedure
is sufficiently general to easily implement other useful formatting procedures such as pretty
printing, output to a string and truncated output.

generic-write obj display? width output  [Procedure]
 obj        Scheme data value to transform.
display?    Boolean, controls whether characters and strings are quoted.
width       Extended boolean, selects format:
            #f     single line format
            integer > 0
            pretty-print (value = max nb of chars per line)
 output      Procedure of 1 argument of string type, called repeatedly with successive
            substrings of the textual representation. This procedure can return #f to
            stop the transformation.

The value returned by generic-write is undefined.
Examples:
  (write obj)  ≡  (generic-write obj #f #f display-string)
  (display obj)  ≡  (generic-write obj #t #f display-string)

where

display-string  ≡
  (lambda (s) (for-each write-char (string->list s)) #t)

4.12.2 Object-To-String

(require 'object->string)

object->string obj  [Function]
 Returns the textual representation of obj as a string.

object->limited-string obj limit  [Function]
 Returns the textual representation of obj as a string of length at most limit.
4.12.3 Pretty-Print

(require 'pretty-print)

**pretty-print** `obj`  
**pretty-print** `obj port`  
Pretty-prints `obj` on `port`. If `port` is not specified, `current-output-port` is used.

Example:

```lisp
(pretty-print '(1 2 3 4 5) (6 7 8 9 10) (11 12 13 14 15) (16 17 18 19 20) (21 22 23 24 25))
```

⇒

```
((1 2 3 4 5)
 (6 7 8 9 10)
 (11 12 13 14 15)
 (16 17 18 19 20)
 (21 22 23 24 25))
```

**pretty-print->string** `obj`  
**pretty-print->string** `obj width`  
Returns the string of `obj` pretty-printed in `width` columns. If `width` is not specified, (`output-port-width`) is used.

Example:

```lisp
(pretty-print->string '(1 2 3 4 5) (6 7 8 9 10) (11 12 13 14 15) (16 17 18 19 20) (21 22 23 24 25))
```

⇒

```
"((1 2 3 4 5)
 (6 7 8 9 10)
 (11 12 13 14 15)
 (16 17 18 19 20)
 (21 22 23 24 25))
"
(pretty-print->string '((1 2 3 4 5) (6 7 8 9 10) (11 12 13 14 15) (16 17 18 19 20) (21 22 23 24 25))
⇒ "((1 2 3 4 5)
  (6 7 8 9 10)
  (11 12 13 14 15)
  (16 17 18 19 20)
  (21 22 23 24 25))"

(require 'pprint-file)

pprint-file infile
[Procedure]
pprint-file infile outfile
[Procedure]
  Pretty-prints all the code in infile. If outfile is specified, the output goes to outfile, otherwise it goes to (current-output-port).

pprint-filter-file infile proc outfile
[Function]
pprint-filter-file infile proc
[Function]
  infile is a port or a string naming an existing file. Scheme source code expressions and definitions are read from the port (or file) and proc is applied to them sequentially.
  outfile is a port or a string. If no outfile is specified then current-output-port is assumed. These expanded expressions are then pretty-printed to this port.
  Whitespace and comments (introduced by ;) which are not part of scheme expressions are reproduced in the output. This procedure does not affect the values returned by current-input-port, current-error-port, and current-output-port.

pprint-filter-file can be used to pre-compile macro-expansion and thus can reduce loading time. The following will write into 'exp-code.scm' the result of expanding all defmacros in 'code.scm'.

(require 'pprint-file)
(require 'defmacroexpand)
(defmacro:load "my-macros.scm")
(pprint-filter-file "code.scm" defmacro:expand* "exp-code.scm")
4.13 Time and Date

If (provided? 'current-time):

The procedures current-time, difftime, and offset-time deal with a calendar time datatype which may or may not be disjoint from other Scheme datatypes.

**current-time** [Function]

Returns the time since 00:00:00 GMT, January 1, 1970, measured in seconds. Note that the reference time is different from the reference time for get-universal-time in Section 4.13.3 [Common-Lisp Time], page 99.

**difftime caltime1 caltime0** [Function]

Returns the difference (number of seconds) between two calendar times: \( \text{caltime1} - \text{caltime0} \). caltime0 may also be a number.

**offset-time caltime offset** [Function]

Returns the calendar time of caltime offset by offset number of seconds \(+ \text{caltime offset}\).

4.13.1 Time Zone

(require 'time-zone)

**TZ-string** [Data Format]

POSIX standards specify several formats for encoding time-zone rules.

\[<\text{pathname}>\]

If the first character of <pathname> is ‘/’, then <pathname> specifies the absolute pathname of a tzfile(5) format time-zone file. Otherwise, <pathname> is interpreted as a pathname within tzfile:vicinity (/usr/lib/zoneinfo/) naming a tzfile(5) format time-zone file.

\[<\text{std}><\text{offset}>\]

The string <std> consists of 3 or more alphabetic characters. <offset> specifies the time difference from GMT. The <offset> is positive if the local time zone is west of the Prime Meridian and negative if it is east. <offset> can be the number of hours or hours and minutes (and optionally seconds) separated by ‘:’. For example, -4:30.

\[<\text{std}><\text{offset}><\text{dst}>\]

<dst> is the at least 3 alphabetic characters naming the local daylight-savings-time.

\[<\text{std}><\text{offset}><\text{dst}><\text{doffset}>\]

<doffset> specifies the offset from the Prime Meridian when daylight-savings-time is in effect.

The non-tzfile formats can optionally be followed by transition times specifying the day and time when a zone changes from standard to daylight-savings and back again.

\[,<\text{date}/> <\text{time}>,,<\text{date}/> <\text{time}>\]

The <time>s are specified like the <offset>s above, except that leading ‘+’ and ‘-’ are not allowed.
Each `<date>` has one of the formats:

- `<J<day>>` specifies the Julian day with `<day>` between 1 and 365. February 29 is never counted and cannot be referenced.
- `<<day>>` This specifies the Julian day with `n` between 0 and 365. February 29 is counted in leap years and can be specified.
- `<M<month>>.<<week>>.<<day>>` This specifies day `<day>` `(0 <= <day> <= 6)` of week `<week>` `(1 <= <week> <= 5)` of month `<month>` `(1 <= <month> <= 12)`. Week 1 is the first week in which day `d` occurs and week 5 is the last week in which day `<day>` occurs. Day 0 is a Sunday.

**time-zone**

is a datatype encoding how many hours from Greenwich Mean Time the local time is, and the Daylight Savings Time rules for changing it.

**time-zone TZ-string**

Creates and returns a time-zone object specified by the string `TZ-string`. If time-zone cannot interpret `TZ-string`, `#f` is returned.

**tz:params caltime tz**

`tz` is a time-zone object. `tz:params` returns a list of three items:

1. An integer. 0 if standard time is in effect for timezone `tz` at `caltime`; 1 if daylight savings time is in effect for timezone `tz` at `caltime`.
2. The number of seconds west of the Prime Meridian timezone `tz` is at `caltime`.
3. The name for timezone `tz` at `caltime`.

`tz:params` is unaffected by the default timezone; inquiries can be made of any timezone at any calendar time.

**tz:std-offset tz**

`tz` is a time-zone object. `tz:std-offset` returns the number of seconds west of the Prime Meridian timezone `tz` is.

The rest of these procedures and variables are provided for POSIX compatibility. Because of shared state they are not thread-safe.

**tzset**

Returns the default time-zone.

**tzset tz**

Sets (and returns) the default time-zone to `tz`.

**tzset TZ-string**

Sets (and returns) the default time-zone to that specified by `TZ-string`.

`tzset` also sets the variables `*timezone*`, `daylight?`, and `tzname`. This function is automatically called by the time conversion procedures which depend on the time zone (see Section 4.13 [Time and Date], page 96).
*timezone*  [Variable]
Contains the difference, in seconds, between Greenwich Mean Time and local standard time (for example, in the U.S. Eastern time zone (EST), timezone is 5*60*60). 
*timezone* is initialized by tzset.

daylight?  [Variable]
is #t if the default timezone has rules for Daylight Savings Time. Note: daylight? does not tell you when Daylight Savings Time is in effect, just that the default zone sometimes has Daylight Savings Time.

tzname  [Variable]
is a vector of strings. Index 0 has the abbreviation for the standard timezone; If daylight?, then index 1 has the abbreviation for the Daylight Savings timezone.

4.13.2 Posix Time

(require 'posix-time)

Calendar-Time  [Data Type]
is a datatype encapsulating time.

Coordinated Universal Time  [Data Type]
(abbreviated UTC) is a vector of integers representing time:

- 0. seconds (0 - 61)
- 1. minutes (0 - 59)
- 2. hours since midnight (0 - 23)
- 3. day of month (1 - 31)
- 4. month (0 - 11). Note difference from decode-universal-time.
- 5. the number of years since 1900. Note difference from decode-universal-time.
- 6. day of week (0 - 6)
- 7. day of year (0 - 365)
- 8. 1 for daylight savings, 0 for regular time

gmtime caltime  [Function]
Converts the calendar time caltime to UTC and returns it.

localtime caltime tz  [Function]
Returns caltime converted to UTC relative to timezone tz.

localtime caltime  [Function]
converts the calendar time caltime to a vector of integers expressed relative to the user's time zone. localtime sets the variable *timezone* with the difference between Coordinated Universal Time (UTC) and local standard time in seconds (see Section 4.13.1 [Time Zone], page 96).

gmktime univtime  [Function]
Converts a vector of integers in GMT Coordinated Universal Time (UTC) format to a calendar time.
**mktime univtime**

Converts a vector of integers in local Coordinated Universal Time (UTC) format to a calendar time.

**mktime univtime tz**

Converts a vector of integers in Coordinated Universal Time (UTC) format (relative to time-zone tz) to calendar time.

**asctime univtime**

Converts the vector of integers caltime in Coordinated Universal Time (UTC) format into a string of the form "Wed Jun 30 21:49:08 1993".

**gtime caltime**

**ctime caltime**

**ctime caltime tz**

Equivalent to (asctime (gmtime caltime)), (asctime (localtime caltime)), and (asctime (localtime caltime tz)), respectively.

### 4.13.3 Common-Lisp Time

**get-decoded-time**

Equivalent to (decode-universal-time (get-universal-time)).

**get-universal-time**

Returns the current time as *Universal Time*, number of seconds since 00:00:00 Jan 1, 1900 GMT. Note that the reference time is different from current-time.

**decode-universal-time univtime**

Converts univtime to *Decoded Time* format. Nine values are returned:

1. seconds (0 - 61)
2. minutes (0 - 59)
3. hours since midnight
4. day of month
5. month (1 - 12). Note difference from gmtime and localtime.
6. year (A.D.). Note difference from gmtime and localtime.
7. #t for daylight savings, #f otherwise
8. hours west of GMT (-24 - +24)

Notice that the values returned by decode-universal-time do not match the arguments to encode-universal-time.

**encode-universal-time second minute hour date month year**

**encode-universal-time second minute hour date month year time-zone**

Converts the arguments in Decoded Time format to Universal Time format. If time-zone is not specified, the returned time is adjusted for daylight saving time. Otherwise, no adjustment is performed.

Notice that the values returned by decode-universal-time do not match the arguments to encode-universal-time.
4.13.4 Time Infrastructure

(time:gmtime tm) [Function]
(time:invert decoder target) [Function]
(time:split t tmＴisdst tmＴgmtoff tm zone)
(require 'tzfile)

4.14 NCBI-DNA

(ncbi:read-dna-sequence port) [Function]
Reads the NCBI-format DNA sequence following the word ‘ORIGIN’ from port.

(ncbi:read-file file) [Function]
Reads the NCBI-format DNA sequence following the word ‘ORIGIN’ from file.

(mrna<cdna str) [Function]
Replaces ‘T’ with ‘U’ in str.

(codons<cdna cdna) [Function]
Returns a list of three-letter symbol codons comprising the protein sequence encoded by cdna starting with its first occurrence of ‘atg’.

(protein<cdna cdna) [Function]
Returns a list of three-letter symbols for the protein sequence encoded by cdna starting with its first occurrence of ‘atg’.

(p<cdna cdna) [Function]
Returns a string of one-letter amino acid codes for the protein sequence encoded by cdna starting with its first occurrence of ‘atg’.

These cDNA count routines provide a means to check the nucleotide sequence with the ‘BASE COUNT’ line preceding the sequence from NCBI.

(cdna:base-count cdna) [Function]
Returns a list of counts of ‘a’, ‘c’, ‘g’, and ‘t’ occurring in cdna.

(cdna:report-base-count cdna) [Function]
Prints the counts of ‘a’, ‘c’, ‘g’, and ‘t’ occurring in cdna.

4.15 Schmooz

Schmooz is a simple, lightweight markup language for interspersing Texinfo documentation with Scheme source code. Schmooz does not create the top level Texinfo file; it creates ‘txi’ files which can be imported into the documentation using the Texinfo command ‘@include’.

(require 'schmooz) defines the function schmooz, which is used to process files. Files containing schmooz documentation should not contain (require 'schmooz).
**Chapter 4: Textual Conversion Packages**

**Procedure**

`schmooz filename.scm` ...

`Filename.scm` should be a string ending with `.scm` naming an existing file containing Scheme source code. `schmooz` extracts top-level comments containing schmooz commands from `filename.scm` and writes the converted Texinfo source to a file named `filename.txi`.

**Procedure**

`schmooz filename.texi` ...

**Procedure**

`schmooz filename.tex` ...

**Procedure**

`schmooz filename.txi` ...

`Filename` should be a string naming an existing file containing Texinfo source code. For every occurrence of the string `@include filename.txi` within that file, `schmooz` calls itself with the argument `filename.scm`.

Schmooz comments are distinguished (from non-schmooz comments) by their first line, which must start with an at-sign (@) preceded by one or more semicolons (;). A schmooz comment ends at the first subsequent line which does not start with a semicolon. Currently schmooz comments are recognized only at top level.

Schmooz comments are copied to the Texinfo output file with the leading contiguous semicolons removed. Certain character sequences starting with at-sign are treated specially. Others are copied unchanged.

A schmooz comment starting with `@body` must be followed by a Scheme definition. All comments between the `@body` line and the definition will be included in a Texinfo definition, either a `@defun` or a `@defvar`, depending on whether a procedure or a variable is being defined.

Within the text of that schmooz comment, at-sign followed by `0` will be replaced by `@code{procedure-name}` if the following definition is of a procedure; or `@var{variable}` if defining a variable.

An at-sign followed by a non-zero digit will expand to the variable citation of that numbered argument: `@var{argument-name}`.

If more than one definition follows a `@body` comment line without an intervening blank or comment line, then those definitions will be included in the same Texinfo definition using `@defvarx` or `@defunx`, depending on whether the first definition is of a variable or of a procedure.

Schmooz can figure out whether a definition is of a procedure if it is of the form:

`'(define (<identifier> <arg> ...)) <expression>')`

or if the left hand side of the definition is some form ending in a lambda expression. Obviously, it can be fooled. In order to force recognition of a procedure definition, start the documentation with `@args` instead of `@body`. `@args` should be followed by the argument list of the function being defined, which may be enclosed in parentheses and delimited by whitespace, (as in Scheme), enclosed in braces and separated by commas, (as in Texinfo), or consist of the remainder of the line, separated by whitespace.

For example:

```scheme
;;@args arg1 args ...
;;@0 takes argument @1 and any number of @2
(define myfun (some-function-returning-magic))
```
Will result in:

@defun myfun arg1 args @dots{}

@end defun

@code{myfun} takes argument @var{arg1} and any number of @var{args}

‘@args’ may also be useful for indicating optional arguments by name. If ‘@args’ occurs inside a schmooz comment section, rather than at the beginning, then it will generate a ‘@defunx’ line with the arguments supplied.

If the first at-sign in a schmooz comment is immediately followed by whitespace, then the comment will be expanded to whatever follows that whitespace. If the at-sign is followed by a non-whitespace character then the at-sign will be included as the first character of the expansion. This feature is intended to make it easy to include Texinfo directives in schmooz comments.
5 Mathematical Packages

5.1 Bit-Twiddling

(require 'logical) or (require 'srfi-60)

The bit-twiddling functions are made available through the use of the logical package. logical is loaded by inserting (require 'logical) before the code that uses these functions. These functions behave as though operating on integers in two’s-complement representation.

5.1.1 Bitwise Operations

logand n1 ... 
bitwise-and n1 ... 
Returns the integer which is the bit-wise AND of the integer arguments.

Example:

(number->string (logand #b1100 #b1010) 2) 
⇒ "1000"

logior n1 ... 
bitwise-ior n1 ... 
Returns the integer which is the bit-wise OR of the integer arguments.

Example:

(number->string (logior #b1100 #b1010) 2) 
⇒ "1110"

logxor n1 ... 
bitwise-xor n1 ... 
Returns the integer which is the bit-wise XOR of the integer arguments.

Example:

(number->string (logxor #b1100 #b1010) 2) 
⇒ "110"

lognot n 
bitwise-not n 
Returns the integer which is the one’s-complement of the integer argument.

Example:

(number->string (lognot #b10000000) 2) 
⇒ "-10000001"
(number->string (lognot #b0) 2) 
⇒ "-1"
Chapter 5: Mathematical Packages

bitwise-if  

```
bitwise-if  mask n0 n1

Returns an integer composed of some bits from integer n0 and some from integer n1.
A bit of the result is taken from n0 if the corresponding bit of integer mask is 1 and
from n1 if that bit of mask is 0.
```

logtest  

```
logtest j k
any-bits-set? j k
(logtest j k) \equiv (not (zero? (logand j k)))
```

any-bits-set?  

```
(logtest #b0100 #b1011) \Rightarrow \#f
(logtest #b0100 #b0111) \Rightarrow \#t
```

5.1.2 Integer Properties

logcount  

```
logcount n

Returns the number of bits in integer n. If integer is positive, the 1-bits in its binary
representation are counted. If negative, the 0-bits in its two’s-complement binary
representation are counted. If 0, 0 is returned.
```

```
Example:

(logcount #b10101010)  \Rightarrow  4
(logcount 0)            \Rightarrow  0
(logcount -2)           \Rightarrow  1
```

On discuss@r6rs.org Ben Harris credits Simon Tatham with the idea to have bitwise-
bit-count return a negative count for negative inputs. Alan Bawden came up with the
succinct invariant.

bitwise-bit-count  

```
bitwise-bit-count n

If n is non-negative, this procedure returns the number of 1 bits in the two’s-
complement representation of n. Otherwise it returns the result of the following
computation:

(bitwise-not (bitwise-bit-count (bitwise-not n)))
```

integer-length  

```
integer-length n

Returns the number of bits neccessary to represent n.
```

```
Example:

(integer-length #b10101010) \Rightarrow 8
(integer-length 0)         \Rightarrow 0
(integer-length #b1111)    \Rightarrow 4
```
Chapter 5: Mathematical Packages

log2-binary-factors \( n \)  
Returns the number of factors of two of integer \( n \). This value is also the bit-index of the least-significant ‘1’ bit in \( n \).

```scheme
(require 'printf)
(do ((idx 0 (+ 1 idx)))
    ((> idx 16))
  (printf "\%s(\%3d) ==> %-5d \%s(\%2d) ==> %-5d\n"
      'log2-binary-factors
      (- idx) (log2-binary-factors (- idx))
      'log2-binary-factors
      idx (log2-binary-factors idx)))
```

- \( \text{log2-binary-factors}(0) \Rightarrow -1 \)
- \( \text{log2-binary-factors}(-1) \Rightarrow 0 \)
- \( \text{log2-binary-factors}(-2) \Rightarrow 1 \)
- \( \text{log2-binary-factors}(-3) \Rightarrow 0 \)
- \( \text{log2-binary-factors}(-4) \Rightarrow 2 \)
- \( \text{log2-binary-factors}(-5) \Rightarrow 0 \)
- \( \text{log2-binary-factors}(-6) \Rightarrow 1 \)
- \( \text{log2-binary-factors}(-7) \Rightarrow 0 \)
- \( \text{log2-binary-factors}(-8) \Rightarrow 3 \)
- \( \text{log2-binary-factors}(-9) \Rightarrow 0 \)
- \( \text{log2-binary-factors}(-10) \Rightarrow 1 \)
- \( \text{log2-binary-factors}(-11) \Rightarrow 0 \)
- \( \text{log2-binary-factors}(-12) \Rightarrow 2 \)
- \( \text{log2-binary-factors}(-13) \Rightarrow 0 \)
- \( \text{log2-binary-factors}(-14) \Rightarrow 1 \)
- \( \text{log2-binary-factors}(-15) \Rightarrow 0 \)
- \( \text{log2-binary-factors}(-16) \Rightarrow 4 \)

5.1.3 Bit Within Word

logbit? \( \text{index} \ n \)  
bit-set? \( \text{index} \ n \)

\( \text{logbit?} \text{index} \ n \) ≡ \( \text{logtest} \text{expt} \text{index} \) \( n \)

```scheme
(logbit? 0 #b1101) ⇒ #t
(logbit? 1 #b1101) ⇒ #f
(logbit? 2 #b1101) ⇒ #t
(logbit? 3 #b1101) ⇒ #t
(logbit? 4 #b1101) ⇒ #f
```

copy-bit \( \text{index from bit} \)

Returns an integer the same as from except in the indexth bit, which is 1 if \( \text{bit} \) is \#t and 0 if \( \text{bit} \) is \#f.

Example:
(number->string (copy-bit 0 0 #t) 2)  ⇒ "1"
(number->string (copy-bit 2 0 #t) 2)  ⇒ "100"
(number->string (copy-bit 2 #b1111 #f) 2)  ⇒ "1011"

5.1.4 Field of Bits

bit-field n start end  [Function]
Returns the integer composed of the start (inclusive) through end (exclusive) bits of n. The startth bit becomes the 0-th bit in the result.
Example:
(number->string (bit-field #b1101101010 0 4) 2)
⇒ "1010"
(number->string (bit-field #b1101101010 4 9) 2)
⇒ "10110"

copy-bit-field to from start end  [Function]
Returns an integer the same as to except possibly in the start (inclusive) through end (exclusive) bits, which are the same as those of from. The 0-th bit of from becomes the startth bit of the result.
Example:
(number->string (copy-bit-field #b1101101010 0 0 4) 2)
⇒ "1101100000"
(number->string (copy-bit-field #b1101101010 -1 0 4) 2)
⇒ "1101101111"
(number->string (copy-bit-field #b110100100010000 -1 5 9) 2)
⇒ "110100111110000"

ash n count  [Function]
arithmetic-shift n count  [Function]
Returns an integer equivalent to (inexact->exact (floor (* n (expt 2 count)))).
Example:
(number->string (ash #b1 3) 2)
⇒ "1000"
(number->string (ash #b1010 -1) 2)
⇒ "101"

rotate-bit-field n count start end  [Function]
Returns n with the bit-field from start to end cyclically permuted by count bits towards high-order.
Example:
(number->string (rotate-bit-field #b0100 3 0 4) 2)
⇒ "10"
(number->string (rotate-bit-field #b0100 -1 0 4) 2)
⇒ "10"
(number->string (rotate-bit-field #b110100100010000 -1 5 9) 2)
⇒ "110100010010000"
(number->string (rotate-bit-field #b110100100010000 1 5 9) 2)
⇒ "11010000110000"

reverse-bit-field n start end
[Function]
Returns n with the order of bits start to end reversed.

(number->string (reverse-bit-field #xa7 0 8) 16)
⇒ "e5"

5.1.5 Bits as Booleans

integer->list k len
[Function]
integer->list k
integer->list returns a list of len booleans corresponding to each bit of the non-negative integer k. #t is coded for each 1; #f for 0. The len argument defaults to (integer-length k).

list->integer list
[Function]
list->integer returns an integer formed from the booleans in the list list, which must be a list of booleans. A 1 bit is coded for each #t; a 0 bit for #f.

(list->integer (integer->list k)) ⇒ k

booleans->integer bool1 ...
[Function]
Returns the integer coded by the bool1 ... arguments.

5.2 Modular Arithmetic

(require 'modular)

extended-euclid n1 n2
[Function]
Returns a list of 3 integers (d x y) such that d = gcd(n1, n2) = n1 * x + n2 * y.

symmetric:modulus m
[Function]
For odd positive integer m, returns an object suitable for passing as the first argument to modular: procedures, directing them to return a symmetric modular number, ie. an n such that

(<= (quotient m -2) n (quotient m 2))

modular:characteristic modulus
[Function]
Returns the non-negative integer characteristic of the ring formed when modulus is used with modular: procedures.

modular:normalize modulus n
[Function]
Returns the integer (modulo n (modular:characteristic modulus)) in the representation specified by modulus.

The rest of these functions assume normalized arguments; That is, the arguments are constrained by the following table:
For all of these functions, if the first argument (modulus) is:
positive?
Integers mod \( mod\)ulus. The result is between 0 and \( mod\)ulus.

zero?
The arguments are treated as integers. An integer is returned.

Otherwise, if \( mod\)ulus is a value returned by \(\text{symmetric}\mod\)ulus \( rad\)ix\), then the arguments and result are treated as members of the integers modulo \( rad\)ix, but with \(\text{symmetric}\) representation; i.e.

\[
(\leq (\text{quotient} \ rad\)ix 2) n (\text{quotient} (- -1 \ rad\)ix 2))
\]

If all the arguments are fixnums the computation will use only fixnums.

**modular:invertable? \( mod\)ulus \( k\)**

Returns \#t if there exists an integer \( n\) such that \( k \times n \equiv 1 \mod mod\)ulus, and \#f otherwise.

**modular:invert \( mod\)ulus \( n2\)**

Returns an integer \( n\) such that \( 1 = (n \times n2) \mod mod\)ulus. If \( n2\) has no inverse mod \( mod\)ulus an error is signaled.

**modular:negate \( mod\)ulus \( n2\)**

Returns \((-n2) \mod mod\)ulus.

**modular:+ \( mod\)ulus \( n2\ n3\)**

Returns \((n2 + n3) \mod mod\)ulus.

**modular:- \( mod\)ulus \( n2\ n3\)**

Returns \((n2 - n3) \mod mod\)ulus.

**modular:* \( mod\)ulus \( n2\ n3\)**

Returns \((n2 \times n3) \mod mod\)ulus.

The Scheme code for modular:* with negative \( mod\)ulus is not completed for fixnum-only implementations.

**modular:expt \( mod\)ulus \( n2\ n3\)**

Returns \((n2 ^ n3) \mod mod\)ulus.

### 5.3 Irrational Integer Functions

(\textit{require} `math-integer`)

**integer-expt \( n1\ n2\)**

Returns \( n1 \) raised to the power \( n2\) if that result is an exact integer; otherwise signals an error.

\[
(\text{integer-expt} 0 n2)
\]

returns 1 for \( n2\) equal to 0; returns 0 for positive integer \( n2\); signals an error otherwise.

**integer-log \( base\ k\)**

Returns the largest exact integer whose power of \( base\) is less than or equal to \( k\). If \( base\) or \( k\) is not a positive exact integer, then integer-log signals an error.
integer-sqrt \( k \)

For non-negative integer \( k \) returns the largest integer whose square is less than or equal to \( k \); otherwise signals an error.

quotient \( n1 \ n2 \)
remainder \( n1 \ n2 \)
modulo \( n1 \ n2 \)

are redefined so that they accept only exact-integer arguments.

round-quotient \( n1 \ n2 \)

Returns the quotient of \( n1 \) and \( n2 \) rounded toward even.

\[
\text{(quotient 3 2)} \quad \Rightarrow 1 \\
\text{(round-quotient 3 2)} \quad \Rightarrow 2
\]

5.4 Irrational Real Functions

(require 'math-real)

Although this package defines real and complex functions, it is safe to load into an integer-only implementation; those functions will be defined to \#f.

real-exp \( x \)
real-ln \( x \)
real-log \( y \ x \)
real-sin \( x \)
real-cos \( x \)
real-tan \( x \)
real-asin \( x \)
real-acos \( x \)
real-atan \( x \)
atan \( y \ x \)

These procedures are part of every implementation that supports general real numbers; they compute the usual transcendental functions. `real-ln` computes the natural logarithm of \( x \); `real-log` computes the logarithm of \( x \) base \( y \), which is \((/ (real-ln x) (real-ln y)))\). If arguments \( x \) and \( y \) are not both real; or if the correct result would not be real, then these procedures signal an error.

real-sqrt \( x \)

For non-negative real \( x \) the result will be its positive square root; otherwise an error will be signaled.

real-expt \( x1 \ x2 \)

Returns \( x1 \) raised to the power \( x2 \) if that result is a real number; otherwise signals an error.

\[
\text{(real-expt 0.0 x2)}
\]

• returns 1.0 for \( x2 \) equal to 0.0;
• returns 0.0 for positive real \( x2 \);
• signals an error otherwise.
Chapter 5: Mathematical Packages

\texttt{quo \; x1 \; x2}  \quad \text{[Function]}
\texttt{rem \; x1 \; x2}  \quad \text{[Function]}
\texttt{mod \; x1 \; x2}  \quad \text{[Function]}

$x_2$ should be non-zero.

\begin{align*}
(\texttt{quo \; x1 \; x2}) & \quad \Rightarrow \; n_q \\
(\texttt{rem \; x1 \; x2}) & \quad \Rightarrow \; x_r \\
(\texttt{mod \; x1 \; x2}) & \quad \Rightarrow \; x_m
\end{align*}

where $n_q$ is $x_1/x_2$ rounded towards zero, $0 < |x_r| < |x_2|$, $0 < |x_m| < |x_2|$, $x_r$ and $x_m$ differ from $x_1$ by a multiple of $x_2$, $x_r$ has the same sign as $x_1$, and $x_m$ has the same sign as $x_2$.

From this we can conclude that for $x_2$ not equal to 0,

\begin{align*}
(= \; x1 \; (+ \; (*) \; x2 \; (\texttt{quo \; x1 \; x2})) \\
(\texttt{rem \; x1 \; x2})) & \quad \Rightarrow \; \#t
\end{align*}

provided all numbers involved in that computation are exact.

\begin{align*}
(\texttt{quo \; 2/3 \; 1/5}) & \quad \Rightarrow \; 3 \\
(\texttt{mod \; 2/3 \; 1/5}) & \quad \Rightarrow \; 1/15 \\
(\texttt{quo \; .666 \; 1/5}) & \quad \Rightarrow \; 3.0 \\
(\texttt{mod \; .666 \; 1/5}) & \quad \Rightarrow \; 65.99999999999995e-3
\end{align*}

\texttt{ln \; z}  \quad \text{[Function]}

These procedures are part of every implementation that supports general real numbers. ‘Ln’ computes the natural logarithm of $z$.

In general, the mathematical function $\ln z$ is multiply defined. The value of $\ln z$ is defined to be the one whose imaginary part lies in the range from $-\pi$ (exclusive) to $\pi$ (inclusive).

\texttt{abs \; x}  \quad \text{[Function]}

For real argument $x$, ‘Abs’ returns the absolute value of $x$’ otherwise it signals an error.

\begin{align*}
(\texttt{abs \; -7}) & \quad \Rightarrow \; 7
\end{align*}

\texttt{make-rectangular \; x1 \; x2}  \quad \text{[Function]}
\texttt{make-polar \; x3 \; x4}  \quad \text{[Function]}

These procedures are part of every implementation that supports general complex numbers. Suppose $x_1$, $x_2$, $x_3$, and $x_4$ are real numbers and $z$ is a complex number such that

$$z = x_1 + x_2 i = x_3 \cdot e^{i x_4}$$

Then

\begin{align*}
(\texttt{make-rectangular \; x1 \; x2}) & \quad \Rightarrow \; z \\
(\texttt{make-polar \; x3 \; x4}) & \quad \Rightarrow \; z
\end{align*}
where \(-\pi < x_{\text{angle}} \leq \pi\) with \(x_{\text{angle}} = x4 + 2\pi n\) for some integer \(n\).

If an argument is not real, then these procedures signal an error.

5.5 Prime Numbers

(require 'factor)

\texttt{prime:prngs}  

\texttt{prime:prngs} is the random-state (see Section 5.6 [Random Numbers], page 111) used by these procedures. If you call these procedures from more than one thread (or from interrupt), \texttt{random} may complain about reentrant calls.

\textit{Note:} The prime test and generation procedures implement (or use) the Solovay-Strassen primality test. See


\texttt{jacobi-symbol} \(p\) \(q\)  

Returns the value (\(+1\), \(-1\), or \(0\)) of the Jacobi-Symbol of exact non-negative integer \(p\) and exact positive odd integer \(q\).

\texttt{prime:trials}  

\texttt{prime:trials} the maximum number of iterations of Solovay-Strassen that will be done to test a number for primality.

\texttt{prime?} \(n\)  

Returns \#\(f\) if \(n\) is composite; \#\(t\) if \(n\) is prime. There is a slight chance \((\text{expt} 2 (- prime:trials))\) that a composite will return \#\(t\).

\texttt{primes<} \(\text{start count}\)  

Returns a list of the first \(\text{count}\) prime numbers less than \(\text{start}\). If there are fewer than \(\text{count}\) prime numbers less than \(\text{start}\), then the returned list will have fewer than \(\text{start}\) elements.

\texttt{primes>} \(\text{start count}\)  

Returns a list of the first \(\text{count}\) prime numbers greater than \(\text{start}\).

\texttt{factor} \(k\)  

Returns a list of the prime factors of \(k\). The order of the factors is unspecified. In order to obtain a sorted list do \((\text{sort!} (\text{factor} k) <)\).

5.6 Random Numbers

A pseudo-random number generator is only as good as the tests it passes. George Marsaglia of Florida State University developed a battery of tests named \texttt{DIEHARD} (http://stat.fsu.edu/~geo/diehard.html). \texttt{diehard.c} has a bug which the patch http://groups.csail.mit.edu/mac/ftpdir/users/jaffer/diehard.c.pat corrects.
SLIB’s PRNG generates 8 bits at a time. With the degenerate seed ‘0’, the numbers generated pass DIEHARD; but when bits are combined from sequential bytes, tests fail. With the seed ‘http://swissnet.ai.mit.edu/~jaffer/SLIB.html’, all of those tests pass.

5.6.1 Exact Random Numbers

(require 'random)

random n state
[Function]
random n
[Function]

n must be an exact positive integer. random returns an exact integer between zero (inclusive) and n (exclusive). The values returned by random are uniformly distributed from 0 to n.

The optional argument state must be of the type returned by (seed->random-state) or (make-random-state). It defaults to the value of the variable *random-state*. This object is used to maintain the state of the pseudo-random-number generator and is altered as a side effect of calls to random.

*random-state*
[Variable]

Holds a data structure that encodes the internal state of the random-number generator that random uses by default. The nature of this data structure is implementation-dependent. It may be printed out and successfully read back in, but may or may not function correctly as a random-number state object in another implementation.

copy-random-state state
[Function]

Returns a new copy of argument state.

copy-random-state
[Function]

Returns a new copy of *random-state*.

seed->random-state seed
[Function]

Returns a new object of type suitable for use as the value of the variable *random-state* or as a second argument to random. The number or string seed is used to initialize the state. If seed->random-state is called twice with arguments which are equal?, then the returned data structures will be equal?. Calling seed->random-state with unequal arguments will nearly always return unequal states.

make-random-state
[Function]

make-random-state obj
[Function]

Returns a new object of type suitable for use as the value of the variable *random-state* or as a second argument to random. If the optional argument obj is given, it should be a printable Scheme object; the first 50 characters of its printed representation will be used as the seed. Otherwise the value of *random-state* is used as the seed.

5.6.2 Inexact Random Numbers

(require 'random-inexact)
random:uniform
random:uniform state
Returns an uniformly distributed inexact real random number in the range between 0 and 1.

random:exp
random:exp state
Returns an inexact real in an exponential distribution with mean 1. For an exponential distribution with mean \( u \) use \((\ast u (\text{random:exp}))\).

random:normal
random:normal state
Returns an inexact real in a normal distribution with mean 0 and standard deviation 1. For a normal distribution with mean \( m \) and standard deviation \( d \) use \((\ast m (\ast d (\text{random:normal})))\).

random:normal-vector! vect
random:normal-vector! vect state
Fills vect with inexact real random numbers which are independent and standard normally distributed (i.e., with mean 0 and variance 1).

random:hollow-sphere! vect
random:hollow-sphere! vect state
Fills vect with inexact real random numbers the sum of whose squares is equal to 1.0. Thinking of vect as coordinates in space of dimension \( n = (\text{vector-length vect}) \), the coordinates are uniformly distributed over the surface of the unit \( n \)-sphere.

random:solid-sphere! vect
random:solid-sphere! vect state
Fills vect with inexact real random numbers the sum of whose squares is less than 1.0. Thinking of vect as coordinates in space of dimension \( n = (\text{vector-length vect}) \), the coordinates are uniformly distributed within the unit \( n \)-sphere. The sum of the squares of the numbers is returned.

5.7 Discrete Fourier Transform

(require 'dft) or (require 'Fourier-transform)

fft and fft-1 compute the Fast-Fourier-Transforms (O(n*log(n))) of arrays whose dimensions are all powers of 2.

sft and sft-1 compute the Discrete-Fourier-Transforms for all combinations of dimensions (O(n^2)).

sft array prot
sft array
array is an array of positive rank. sft returns an array of type prot (defaulting to array) of complex numbers comprising the Discrete Fourier Transform of array.
Chapter 5: Mathematical Packages

\( \text{sft-1 \ array \ prot} \) [Function]
\( \text{sft-1 \ array} \) [Function]
\( \text{array} \) is an array of positive rank. \( \text{sft-1} \) returns an array of type \( \text{prot} \) (defaulting to \( \text{array} \)) of complex numbers comprising the inverse Discrete Fourier Transform of \( \text{array} \).

\( \text{fft \ array \ prot} \) [Function]
\( \text{fft \ array} \) [Function]
\( \text{array} \) is an array of positive rank whose dimensions are all powers of 2. \( \text{fft} \) returns an array of type \( \text{prot} \) (defaulting to \( \text{array} \)) of complex numbers comprising the Discrete Fourier Transform of \( \text{array} \).

\( \text{fft-1 \ array \ prot} \) [Function]
\( \text{fft-1 \ array} \) [Function]
\( \text{array} \) is an array of positive rank whose dimensions are all powers of 2. \( \text{fft-1} \) returns an array of type \( \text{prot} \) (defaulting to \( \text{array} \)) of complex numbers comprising the inverse Discrete Fourier Transform of \( \text{array} \).

\( \text{dft} \) and \( \text{dft-1} \) compute the discrete Fourier transforms using the best method for decimating each dimension.

\( \text{dft \ array \ prot} \) [Function]
\( \text{dft \ array} \) [Function]
\( \text{dft} \) returns an array of type \( \text{prot} \) (defaulting to \( \text{array} \)) of complex numbers comprising the Discrete Fourier Transform of \( \text{array} \).

\( \text{dft-1 \ array \ prot} \) [Function]
\( \text{dft-1 \ array} \) [Function]
\( \text{dft-1} \) returns an array of type \( \text{prot} \) (defaulting to \( \text{array} \)) of complex numbers comprising the inverse Discrete Fourier Transform of \( \text{array} \).

\( \text{(fft-1 (fft array))} \) will return an array of values close to \( \text{array} \).

\( \text{(fft ' #(1 0+i -1 0-i 1 0+i -1 0-i))} \Rightarrow \)

\( \#(0.0 0.0 0.0+628.0783185208527e-18i 0.0 0.0 0.0 8.0-628.0783185208527e-18i 0.0) \)

\( \text{(fft-1 ' #(0 0 0 0 0 0 8 0))} \Rightarrow \)

\( \#(1.0 -61.23031769111886e-18+1.0i -1.0 61.23031769111886e-18-1.0i 1.0 -61.23031769111886e-18+1.0i -1.0 61.23031769111886e-18-1.0i) \)

5.8 Cyclic Checksum

\( \text{(require 'crc)} \) Cyclic Redundancy Checks using Galois field GF(2) polynomial arithmetic are used for error detection in many data transmission and storage applications.

The generator polynomials for various CRC protocols are available from many sources. But the polynomial is just one of many parameters which must match in order for a CRC implementation to interoperate with existing systems:
• the byte-order and bit-order of the data stream;
• whether the CRC or its inverse is being calculated;
• the initial CRC value; and
• whether and where the CRC value is appended (inverted or non-inverted) to the data stream.

The performance of a particular CRC polynomial over packets of given sizes varies widely. In terms of the probability of undetected errors, some uses of extant CRC polynomials are suboptimal by several orders of magnitude.

If you are considering CRC for a new application, consult the following article to find the optimum CRC polynomial for your range of data lengths:


http://www.ece.cmu.edu/~koopman/roses/dsn04/koopman04_crc_poly_embedded.pdf

There is even some controversy over the polynomials themselves.

**crc-32-polynomial**

For CRC-32, http://www2.sis.pitt.edu/~jkabara/tele-2100/lect08.html gives
\[x^{32}+x^{26}+x^{23}+x^{16}+x^{12}+x^{11}+x^{10}+x^8+x^7+x^5+x^4+x^2+x+1.\]


SLIB crc-32-polynomial uses the latter definition.

**crc-ccitt-polynomial**


**crc-16-polynomial**

Chapter 5: Mathematical Packages

**crc-12-polynomial** [Constant]
http://www.cs.ncl.ac.uk/people/harry.whitfield/home.formal/CRCs.html,
give CRC-12: \(x^{12} + x^{11} + x^3 + x^2 + 1\).

But http://www.ffldusoe.edu/Faculty/Denenberg/Topics/Networks/Error_Detection_Correction/crc.html,
http://duchon.umuc.edu/Web_Pages/duchon/99_f_cm435/ShiftRegister.htm,
http://www.gpfn.sk.ca/~rhg/csc8550s02/crc.html, and http://www.efg2.com/Lab/Mathematics/CRC.htm
give CRC-12: \(x^{12} + x^{11} + x^3 + x^2 + 1\).

These differ in bit 1 and calculations using them return different values. With citations near evenly split, it is hard to know which is correct. Thanks to Philip Koopman for breaking the tie in favor of the latter (#xC07).

**crc-10-polynomial** [Constant]
gives CRC-10: \(x^{10} + x^9 + x^5 + x^4 + 1\); but http://cell-relay.indiana.edu/cell-relay/publications/software/CRC/crc10.html,
give CRC-10: \(x^{10} + x^9 + x^5 + x^4 + 1\).

**crc-08-polynomial** [Constant]
http://www.cs.ncl.ac.uk/people/harry.whitfield/home.formal/CRCs.html,
give CRC-8: \(x^8 + x^2 + x^1 + 1\).

**atm-hec-polynomial** [Constant]
and http://www.gpfn.sk.ca/~rhg/csc8550s02/crc.html give ATM HEC: \(x^8 + x^2 + x^1\).

**dowcrc-polynomial** [Constant]
http://www.cs.ncl.ac.uk/people/harry.whitfield/home.formal/CRCs.html
gives DOWCRC: \(x^8 + x^5 + x^4 + 1\).

**usb-token-polynomial** [Constant]
give USB-token: \(x^5 + x^2 + 1\).

Each of these polynomial constants is a string of ‘1’s and ‘0’s, the exponent of each power of \(x\) in descending order.

**crc:make-table** [Function]
\(poly\) must be string of ‘1’s and ‘0’s beginning with ‘1’ and having length greater than 8. \(crc:make-table\) returns a vector of 256 integers, such that:

```
(set! crc
  (logxor (ash (logand (+ -1 (ash 1 (- deg 8))) crc) 8)
```
(vector-ref crc-table (logxor (ash crc (~ 8 deg)) byte))))

will compute the crc with the 8 additional bits in byte; where crc is the previous
accumulated CRC value, deg is the degree of poly, and crc-table is the vector returned
by crc:make-table.

If the implementation does not support deg-bit integers, then crc:make-table re-
turns #f.

cksum file
[Function]
Computes the P1003.2/D11.2 (POSIX.2) 32-bit checksum of file.

(require 'crc)
(cksum (in-vicinity (library-vicinity) "ratize.scm"))
⇒ 157103930

ccksum port
[Function]
Computes the checksum of the bytes read from port until the end-of-file.

ccksum-string, which returns the P1003.2/D11.2 (POSIX.2) 32-bit checksum of the bytes
in str, can be defined as follows:

(require 'string-port)
(define (ccksum-string str) (call-with-input-string str ccksum))

crc16 file
[Function]
Computes the USB data-packet (16-bit) CRC of file.

crc16 port
[Function]
Computes the USB data-packet (16-bit) CRC of the bytes read from port until the
end-of-file.

crc16 calculates the same values as the crc16.pl program given in

crc5 file
[Function]
Computes the USB token (5-bit) CRC of file.

crc5 port
[Function]
Computes the USB token (5-bit) CRC of the bytes read from port until the end-of-file.

crc5 calculates the same values as the crc5.pl program given in

5.9 Graphing

5.9.1 Character Plotting

(require 'charplot)

charplot:dimensions
[Variable]
A list of the maximum height (number of lines) and maximum width (number of
columns) for the graph, its scales, and labels.
The default value for `charplot:dimensions` is the `output-port-height` and `output-port-width` of current-output-port.

**plot** `coords x-label y-label`  

`coords` is a list or vector of coordinates, lists of x and y coordinates. `x-label` and `y-label` are strings with which to label the x and y axes.

Example:

```scheme
(require 'charplot)
(set! charplot:dimensions '(20 55))

(define (make-points n)
  (if (zero? n)
      ()
      (cons (list (/ n 6) (sin (/ n 6))) (make-points (1- n)))))

(plot (make-points 40) "x" "Sin(x)")
```

```
\begin{verbatim}
Sin(x) _________________________________________
\end{verbatim}
```

```
\begin{verbatim}
0.75|--- * .... *---
|  ** **   ** \\
0.5|---- * .. *----
|  * *    * \\
0.25|--- * . *---
|  * *    * \\
0|-------------------*------------------*--|
|  |* |
-0.25|-- * . *--
|  * . *   * \\
-0.5|-- * *---
|  *   *    * \\
-0.75|-- ** **---
|  **   **   ** \\
-1|-- ****---
\end{verbatim}
```

```
x 2 4 6
```

**plot** `func x1 x2`  
**plot** `func x1 x2 npts`

Plots the function of one argument `func` over the range `x1` to `x2`. If the optional integer argument `npts` is supplied, it specifies the number of points to evaluate `func` at.

```scheme
(plot sin 0 (* 2 pi))
```

```
\begin{verbatim}
(plot sin 0 (* 2 pi))
\end{verbatim}
```

```
\begin{verbatim}
(plot sin 0 (* 2 pi))
\end{verbatim}
```
histograph data label

[Procedure]
Creates and displays a histogram of the numerical values contained in vector or list data

(require 'random-inexact)
(histograph (do ((idx 99 (+ -1 idx))
    (lst '()) (cons (* .02 (random:normal)) lst)))
  ((negative? idx) lst))
  "normal")
-)
5.9.2 PostScript Graphing

This is a graphing package creating encapsulated-PostScript files. Its motivations and design choice are described in [http://people.csail.mit.edu/jaffer/Docupage/grapheps](http://people.csail.mit.edu/jaffer/Docupage/grapheps)

A dataset to be plotted is taken from a 2-dimensional array. Corresponding coordinates are in rows. Coordinates from any pair of columns can be plotted.

**create-postscript-graph** filename.eps size elt1 ...  
*Function*  
filename.eps should be a string naming an output file to be created. size should be an exact integer, a list of two exact integers, or #f. elt1, ... are values returned by graphing primitives described here.

create-postscript-graph creates an Encapsulated-PostScript file named filename.eps containing graphs as directed by the elt1, ... arguments.

The size of the graph is determined by the size argument. If a list of two integers, they specify the width and height. If one integer, then that integer is the width and the height is 3/4 of the width. If #f, the graph will be 800 by 600.

These graphing procedures should be called as arguments to create-postscript-graph. The order of these arguments is significant; PostScript graphics state is affected serially from the first elt argument to the last.

**whole-page**  
*Function*  
Pushes a rectangle for the whole encapsulated page onto the PostScript stack. This pushed rectangle is an implicit argument to partition-page or setup-plot.
5.9.2.1 Column Ranges

A range is a list of two numbers, the minimum and the maximum. Ranges can be given explicity or computed in PostScript by column-range.

```
column-range array k
  Returns the range of values in 2-dimensional array column k.
```

```
pad-range range p
  Expands range by p/100 on each end.
```

```
snap-range range
  Expands range to round number of ticks.
```

```
combine-ranges range1 range2 ...
  Returns the minimal range covering all range1, range2, ...
```

```
setup-plot x-range y-range pagerect
  x-range and y-range should each be a list of two numbers or the value returned by pad-range, snap-range, or combine-range. pagerect is the rectangle bounding the graph to be drawn; if missing, the rectangle from the top of the PostScript stack is popped and used.
  Based on the given ranges, setup-plot sets up scaling and margins for making a graph. The margins are sized proportional to the fontheight value at the time of the call to setup-plot. setup-plot sets two variables:
  plotrect The region where data points will be plotted.
  graphrect The pagerect argument to setup-plot. Includes plotrect, legends, etc.
```

5.9.2.2 Drawing the Graph

```
plot-column array x-column y-column proc3s
  Plots points with x coordinate in x-column of array and y coordinate y-column of array. The symbol proc3s specifies the type of glyph or drawing style for presenting these coordinates.
```

The glyphs and drawing styles available are:

```
line      Draws line connecting points in order.
mountain  Fill area below line connecting points.
cloud     Fill area above line connecting points.
impulse   Draw line from x-axis to each point.
bargraph  Draw rectangle from x-axis to each point.
disc      Solid round dot.
point     Minimal point – invisible if linewidth is 0.
square    Square box.
```
diamond Square box at 45.0
plus Plus sign.
cross X sign.
triup Triangle pointing upward
tridown Triangle pointing downward
pentagon Five sided polygon
circle Hollow circle

```plaintext
plot-text-column array x-column y-column t-column proc3s
```
Plots text in \textit{t-column} of \textit{array} at \textit{x} coordinate in \textit{x-column} of \textit{array} and \textit{y} coordinate in \textit{y-column} of \textit{array}. The symbol \textit{proc3s} specifies the offset of the text from the specified coordinates.

The offsets available are:

- **above** Draws the text centered above at the point.
- **center** Draws the text centered at the point.
- **below** Draws the text centered below the point.
- **left** Draws the text to the left of the point.
- **right** Draws the text to the right of the point.

All the offsets other than \texttt{center} are calculated to keep the text clear of a glyph drawn at the same coordinates. If you need more or less clearance, use \texttt{set-glyphsize}.

### 5.9.2.3 Graphics Context

```plaintext
in-graphic-context arg ...
```
Saves the current graphics state, executes \texttt{args}, then restores to saved graphics state.

```plaintext
set-color color
```
\textit{color} should be a string naming a Resene color, a saturate color, or a number between 0 and 100.

\texttt{set-color} sets the PostScript color to the color of the given string, or a grey value between black (0) and white (100).

```plaintext
set-font font height
```

```plaintext
set-font font encoding height
```
\textit{font} should be a (case-sensitive) string naming a PostScript font. \textit{height} should be a positive real number. \textit{encoding} should name a PostScript encoding such as `ISO88591Encoding`.

\texttt{set-font} Changes the current PostScript font to \textit{font} with the \textit{encoding} encoding, and \textit{height} equal to \textit{height}. The default font is `Helvetica' (12pt). The default encoding is `StandardEncoding'.
The base set of PostScript fonts is:

- Times
- Times-Italic
- Times-Bold
- Times-BoldItalic
- Helvetica
- Helvetica-Oblique
- Helvetica-Bold
- Helvetica-BoldOblique
- Courier
- Courier-Oblique
- Courier-Bold
- Courier-BoldOblique
- Symbol

The base set of PostScript encodings is:

- StandardEncoding
- ISOLatin1Encoding
- ExpertEncoding
- ExpertSubsetEncoding
- SymbolEncoding

Line parameters do not affect fonts; they do effect glyphs.

**set-linewidth w**

The default linewidth is 1. Setting it to 0 makes the lines drawn as skinny as possible. Linewidth must be much smaller than glyphsize for readable glyphs.

**set-linedash j k**

Lines are drawn j-on k-off.

**set-linedash j**

Lines are drawn j-on j-off.

**set-linedash**

Turns off dashing.

**set-glyphsize w**

Sets the (PostScript) variable glyphsize to w. The default glyphsize is 6.

The effects of `clip-to-rect` are also part of the graphic context.

### 5.9.2.4 Rectangles

A rectangle is a list of 4 numbers; the first two elements are the x and y coordinates of lower left corner of the rectangle. The other two elements are the width and height of the rectangle.

**whole-page**

Pushes a rectangle for the whole encapsulated page onto the PostScript stack. This pushed rectangle is an implicit argument to `partition-page` or `setup-plot`.

**partition-page xparts yparts**

Pops the rectangle currently on top of the stack and pushes `xparts * yparts` sub-rectangles onto the stack in decreasing y and increasing x order. If you are drawing just one graph, then you don’t need `partition-page`.

**plotrect**

The rectangle where data points should be plotted. `plotrect` is set by `setup-plot`.

**graphrect**

The `pagerect` argument of the most recent call to `setup-plot`. Includes `plotrect`, legends, etc.
5.9.2.5 Legending

**title-top** title subtitle  
**title-top** title  
Puts a title line and an optional subtitle line above the graphrect.

**title-bottom** title subtitle  
**title-bottom** title  
Puts a title line and an optional subtitle line below the graphrect.

**topedge**  
**bottomedge**  
These edge coordinates of graphrect are suitable for passing as the first argument to rule-horizontal.

**leftedge**  
**rightedge**  
These edge coordinates of graphrect are suitable for passing as the first argument to rule-vertical.

**set-margin-templates** left right  
The margin-templates are strings whose displayed width is used to reserve space for the left and right side numerical legends. The default values are ",-.0123456789".

**rule-vertical** x-coord text tick-width  
Draws a vertical ruler with X coordinate x-coord and labeled with string text. If tick-width is positive, then the ticks are tick-width long on the right side of x-coord; and text and numeric legends are on the left. If tick-width is negative, then the ticks are -tick-width long on the left side of x-coord; and text and numeric legends are on the right.

**rule-horizontal** y-coord text tick-height  
Draws a horizontal ruler with Y coordinate y-coord and labeled with string text. If tick-height is positive, then the ticks are tick-height long on the top side of y-coord; and text and numeric legends are on the bottom. If tick-height is negative, then the ticks are -tick-height long on the bottom side of y-coord; and text and numeric legends are on the top.

**y-axis**  
Draws the y-axis.
x-axis
  [Function]
  Draws the x-axis.

grid-verticals
  [Function]
  Draws vertical lines through graphrect at each tick on the vertical ruler.

grid-horizontals
  [Function]
  Draws horizontal lines through graphrect at each tick on the horizontal ruler.

5.9.2.6 Legacy Plotting

graph:dimensions
  [Variable]
  A list of the width and height of the graph to be plotted using plot.

plot func x1 x2 npts
  [Function]
  plot func x1 x2
  Creates and displays using (system "gv tmp.eps") an encapsulated PostScript graph
  of the function of one argument func over the range x1 to x2. If the optional integer
  argument npts is supplied, it specifies the number of points to evaluate func at.

plot x1 x2 npts func1 func2 . . .
  [Function]
  Creates and displays an encapsulated PostScript graph of the one-argument functions
  func1, func2, ... over the range x1 to x2 at npts points.

plot coords x-label y-label
  [Function]
  coords is a list or vector of coordinates, lists of x and y coordinates. x-label and
  y-label are strings with which to label the x and y axes.

5.9.2.7 Example Graph

The file ‘am1.5.html’, a table of solar irradiance, is fetched with ‘wget’ if it isn’t already
in the working directory. The file is read and stored into an array, irradiance.

create-postscript-graph is then called to create an encapsulated-PostScript file,
’solarad.eps’. The size of the page is set to 600 by 300. whole-page is called and leaves
the rectangle on the PostScript stack. setup-plot is called with a literal range for x and
computes the range for column 1.

Two calls to top-title are made so a different font can be used for the lower half.
in-graphic-context is used to limit the scope of the font change. The graphing area is
outlined and a rule drawn on the left side.

Because the X range was intentionally reduced, in-graphic-context is called and
clip-to-rect limits drawing to the plotting area. A black line is drawn from data column
1. That line is then overlayed with a mountain plot of the same column colored "Bright
Sun".

After returning from the in-graphic-context, the bottom ruler is drawn. Had it been
drawn earlier, all its ticks would have been painted over by the mountain plot.

The color is then changed to ‘seagreen’ and the same graphrect is setup again, this
time with a different Y scale, 0 to 1000. The graphic context is again clipped to plotrect,
linedash is set, and column 2 is plotted as a dashed line. Finally the rightedge is ruled.
Having the line and its scale both in green helps disambiguate the scales.
(require 'pdf-graph)
(require 'line-i/o)
(require 'string-port)

(define irradiance
  (let ((url "http://www.pv.unsw.edu.au/am1.5.html")
    (file "am1.5.html"))
    (define (read->list line)
      (define elts '())
      (call-with-input-string line
        (lambda (iprt) (do ((elt (read iprt) (read iprt)))
          ((eof-object? elt) elts)
            ((set! elts (cons elt elts))))))
    (if (not (file-exists? file))
      (system (string-append "wget -c -O" file " " url)))
    (call-with-input-file file
      (lambda (iprt)
        (define lines '())
        (do ((line (read-line iprt) (read-line iprt)))
            ((eof-object? line)
              (let ((nra (make-array (A:floR64b)
                (length lines)
                (length (car lines)))))
                (do ((lns lines (cdr lns))
                    (idx (+ -1 (length lines)) (+ -1 idx)))
                    ((null? lns) nra)
                  (do ((kdx (+ -1 (length (car lines))) (+ -1 kdx))
                      (lst (car lns) (cdr lst)))
                    ((null? lst))
                      (array-set! nra (car lst) idx kdx))))))))
    (if (and (positive? (string-length line))
                (char-numeric? (string-ref line 0)))
      (set! lines (cons (read->list line) lines))))))

(let ((xrange '(.25 2.5)))
  (create-postscript-graph
    "solarad.eps" '(600 300)
    (whole-page)
    (setup-plot xrange (column-range irradiance 1))
    (title-top
      "Solar Irradiance http://www.pv.unsw.edu.au/am1.5.html")
    (in-graphic-context
      (set-font "Helvetica-Oblique" 12)
      (title-top
        "Key Centre for Photovoltaic Engineering UNSW - Air Mass 1.5 Global Spectrum")
      (outline-rect plotrect)
(rule-vertical leftedge "\text{\textsuperscript{\textdegree}}W/(\text{m}^2.\text{\textmu}m)" 10)
(in-graphic-context (clip-to-rect plotrect)
  (plot-column irradiance 0 1 'line)
  (set-color "Bright Sun")
  (plot-column irradiance 0 1 'mountain))
)

(rule-horizontal bottomedge "Wavelength in .\text{\textmu}m" 5)
(set-color 'seagreen)

(setup-plot xrange '(0 1000) graphrect)
(in-graphic-context (clip-to-rect plotrect)
  (set-linedash 5 2)
  (plot-column irradiance 0 2 'line))

(rule-vertical rightedge "Integrated .\text{\textmu}W/(\text{m}^2)" -10)
))

(system "gv solarad.eps")

5.10 Solid Modeling

(require 'solid)

http://people.csail.mit.edu/jaffer/Solid/#Example gives an example use of this package.

vrml node ... [Function]
Returns the VRML97 string (including header) of the concatenation of strings nodes, ...

vrml-append node1 node2 ... [Function]
Returns the concatenation with interdigitated newlines of strings node1, node2, ...

vrml-to-file file node ... [Function]
Writes to file named file the VRML97 string (including header) of the concatenation of strings nodes, ...

world:info title info ... [Function]
Returns a VRML97 string setting the title of the file in which it appears to title. Additional strings info, ... are comments.

VRML97 strings passed to vrml and vrml-to-file as arguments will appear in the resulting VRML code. This string turns off the headlight at the viewpoint:

" NavigationInfo \{headlight FALSE\}"

scene:panorama front right back left top bottom [Function]
Specifies the distant images on the inside faces of the cube enclosing the virtual world.

scene:sphere colors angles [Function]
colors is a list of color objects. Each may be of type Section 5.11.1 [Color Data-Type], page 134, a 24-bit sRGB integer, or a list of 3 numbers between 0.0 and 1.0.
angles is a list of non-increasing angles the same length as colors. Each angle is between 90 and -90 degrees. If 90 or -90 are not elements of angles, then the color at the zenith and nadir are taken from the colors paired with the angles nearest them.

scene:sphere fills horizontal bands with interpolated colors on the background sphere encasing the world.

scene:sky-and-dirt
[Function]
Returns a blue and brown background sphere encasing the world.

scene:sky-and-grass
[Function]
Returns a blue and green background sphere encasing the world.

scene:sun latitude julian-day hour turbidity strength
[Function]
scene:sun latitude julian-day hour turbidity
[Function]
latitude is the virtual place’s latitude in degrees. julian-day is an integer from 0 to 366, the day of the year. hour is a real number from 0 to 24 for the time of day; 12 is noon. turbidity is the degree of fogginess described in See Section 5.11.7 [Daylight], page 149.

scene:sun returns a bright yellow, distant sphere where the sun would be at hour on julian-day at latitude. If strength is positive, included is a light source of strength (default 1).

scene:overcast latitude julian-day hour turbidity strength
[Function]
scene:overcast latitude julian-day hour turbidity
[Function]
latitude is the virtual place’s latitude in degrees. julian-day is an integer from 0 to 366, the day of the year. hour is a real number from 0 to 24 for the time of day; 12 is noon. turbidity is the degree of cloudiness described in See Section 5.11.7 [Daylight], page 149.

scene:overcast returns an overcast sky as it might look at hour on julian-day at latitude. If strength is positive, included is an ambient light source of strength (default 1).

Viewpoints are objects in the virtual world, and can be transformed individually or with solid objects.

scene:viewpoint name distance compass pitch
[Function]
scene:viewpoint name distance compass
[Function]
Returns a viewpoint named name facing the origin and placed distance from it. compass is a number from 0 to 360 giving the compass heading. pitch is a number from -90 to 90, defaulting to 0, specifying the angle from the horizontal.

scene:viewpoints proximity
[Function]
Returns 6 viewpoints, one at the center of each face of a cube with sides 2 * proximity, centered on the origin.
Light Sources

In VRML97, lights shine only on objects within the same children node and descendants of that node. Although it would have been convenient to let light direction be rotated by `solid:rotation`, this restricts a rotated light’s visibility to objects rotated with it.

To workaround this limitation, these directional light source procedures accept either Cartesian or spherical coordinates for direction. A spherical coordinate is a list \((\theta, \text{azimuth})\); where \(\theta\) is the angle in degrees from the zenith, and azimuth is the angle in degrees due west of south.

It is sometimes useful for light sources to be brighter than ‘1’. When `intensity` arguments are greater than 1, these functions gang multiple sources to reach the desired strength.

```scheme
light:ambient  color intensity  [Function]
light:ambient  color  [Function]

Ambient light shines on all surfaces with which it is grouped.

`color` is an object of type Section 5.11.1 [Color Data-Type], page 134, a 24-bit sRGB integer, or a list of 3 numbers between 0.0 and 1.0. If `color` is #f, then the default color will be used. `intensity` is a real non-negative number defaulting to ‘1’.

`light:ambient` returns a light source or sources of `color` with total strength of `intensity` (or 1 if omitted).

light:directional  color direction intensity  [Function]
light:directional  color direction  [Function]
light:directional  color  [Function]

Directional light shines parallel rays with uniform intensity on all objects with which it is grouped.

`color` is an object of type Section 5.11.1 [Color Data-Type], page 134, a 24-bit sRGB integer, or a list of 3 numbers between 0.0 and 1.0. If `color` is #f, then the default color will be used.

`direction` must be a list or vector of 2 or 3 numbers specifying the direction to this light. If `direction` has 2 numbers, then these numbers are the angle from zenith and the azimuth in degrees; if `direction` has 3 numbers, then these are taken as a Cartesian vector specifying the direction to the light source. The default direction is upwards; thus its light will shine down.

`intensity` is a real non-negative number defaulting to ‘1’.

`light:directional` returns a light source or sources of `color` with total strength of `intensity`, shining from `direction`.

light:beam  attenuation radius aperture peak  [Function]
light:beam  attenuation radius aperture  [Function]
light:beam  attenuation radius  [Function]
light:beam  attenuation  [Function]

`attenuation` is a list or vector of three nonnegative real numbers specifying the reduction of intensity, the reduction of intensity with distance, and the reduction of intensity as the square of distance. `radius` is the distance beyond which the light does not shine. `radius` defaults to ‘100’.
**aperture** is a real number between 0 and 180, the angle centered on the light’s axis through which it sheds some light. **peak** is a real number between 0 and 90, the angle of greatest illumination.

light:point  location color intensity beam  [Function]
light:point  location color intensity  [Function]
light:point  location color  [Function]
light:point  location  [Function]

Point light radiates from location, intensity decreasing with distance, towards all objects with which it is grouped.

color is a an object of type Section 5.11.1 [Color Data-Type], page 134, a 24-bit sRGB integer, or a list of 3 numbers between 0.0 and 1.0. If color is #f, then the default color will be used. intensity is a real non-negative number defaulting to ‘1’. beam is a structure returned by light:beam or #f.

light:point returns a light source or sources at location of color with total strength intensity and beam properties. Note that the pointlight itself is not visible. To make it so, place an object with emissive appearance at location.

light:spot  location direction color intensity beam  [Function]
light:spot  location direction color intensity  [Function]
light:spot  location direction color  [Function]
light:spot  location direction  [Function]
light:spot  location  [Function]

Spot light radiates from location towards direction, intensity decreasing with distance, illuminating objects with which it is grouped.

direction must be a list or vector of 2 or 3 numbers specifying the direction to this light. If direction has 2 numbers, then these numbers are the angle from zenith and the azimuth in degrees; if direction has 3 numbers, then these are taken as a Cartesian vector specifying the direction to the light source. The default direction is upwards; thus its light will shine down.

color is a an object of type Section 5.11.1 [Color Data-Type], page 134, a 24-bit sRGB integer, or a list of 3 numbers between 0.0 and 1.0. If color is #f, then the default color will be used.

intensity is a real non-negative number defaulting to ‘1’.

light:spot returns a light source or sources at location of direction with total strength color. Note that the spotlight itself is not visible. To make it so, place an object with emissive appearance at location.

**Object Primitives**

solid:box  geometry appearance  [Function]
solid:box  geometry  [Function]

gometry must be a number or a list or vector of three numbers. If geometry is a number, the solid:box returns a cube with sides of length geometry centered on the origin. Otherwise, solid:box returns a rectangular box with dimensions geometry
centered on the origin. appearance determines the surface properties of the returned object.

**solid:lumber** geometry appearance
Returns a box of the specified geometry, but with the y-axis of a texture specified in appearance being applied along the longest dimension in geometry.

**solid:cylinder** radius height appearance
**solid:cylinder** radius height
Returns a right cylinder with dimensions \((\text{abs radius})\) and \((\text{abs height})\) centered on the origin. If height is positive, then the cylinder ends will be capped. If radius is negative, then only the ends will appear. appearance determines the surface properties of the returned object.

**solid:disk** radius thickness appearance
**solid:disk** radius thickness
thickness must be a positive real number. solid:disk returns a circular disk with dimensions radius and thickness centered on the origin. appearance determines the surface properties of the returned object.

**solid:cone** radius height appearance
**solid:cone** radius height
Returns an isosceles cone with dimensions radius and height centered on the origin. appearance determines the surface properties of the returned object.

**solid:pyramid** side height appearance
**solid:pyramid** side height
Returns an isosceles pyramid with dimensions side and height centered on the origin. appearance determines the surface properties of the returned object.

**solid:sphere** radius appearance
**solid:sphere** radius
Returns a sphere of radius radius centered on the origin. appearance determines the surface properties of the returned object.

**solid:ellipsoid** geometry appearance
**solid:ellipsoid** geometry
geometry must be a number or a list or vector of three numbers. If geometry is a number, the solid:ellipsoid returns a sphere of diameter geometry centered on the origin. Otherwise, solid:ellipsoid returns an ellipsoid with diameters geometry centered on the origin. appearance determines the surface properties of the returned object.

**solid:polyline** coordinates appearance
**solid:polyline** coordinates
coordinates must be a list or vector of coordinate lists or vectors specifying the x, y, and z coordinates of points. solid:polyline returns lines connecting successive pairs of points. If called with one argument, then the polyline will be white. If appearance
is given, then the polyline will have its emissive color only; being black if *appearance* does not have an emissive color.

The following code will return a red line between points at (1 2 3) and (4 5 6):

```
(solid:polyline '((1 2 3) (4 5 6)) (solid:color #f 0 #f 0 '(1 0 0)))
```

- **solid:prism** `xz-array y appearance`  
  - **Function**
  - `xz-array` must be an n-by-2 array holding a sequence of coordinates tracing a non-intersecting clockwise loop in the x-z plane. *solid:prism* will close the sequence if the first and last coordinates are not the same.
  
  *solid:prism* returns a capped prism `y` long.

- **solid:basrelief** `width height depth colorray appearance`  
  - **Function**
  - `width`, `height`, or `depth` must be a 2-dimensional array; the others must be real numbers giving the length of the basrelief in those dimensions. The rest of this description assumes that `height` is an array of heights.
  
  *solid:basrelief* returns a `width` by `depth` basrelief solid with heights per array `height` with the bottom surface centered on the origin.

  If present, `appearance` determines the surface properties of the returned object. If present, `colorray` must be an array of objects of type Section 5.11.1 [Color Data-Type], page 134, 24-bit sRGB integers or lists of 3 numbers between 0.0 and 1.0.

  If `colorray`'s dimensions match `height`, then each element of `colorray` paints its corresponding vertex of `height`. If `colorray` has all dimensions one smaller than `height`, then each element of `colorray` paints the corresponding face of `height`. Other dimensions for `colorray` are in error.

- **solid:text** `fontstyle str len appearance`  
  - **Function**

  *solid:text* returns a two-sided, flat text object positioned in the Z=0 plane of the local coordinate system

### Surface Attributes

- **solid:color** `diffuseColor ambientIntensity specularColor shininess`  
  - **Function**

  *solid:color* returns a color with the given diffuse, ambient, specular, and shininess properties.

- **solid:color** `diffuseColor ambientIntensity specularColor shininess emissiveColor`  
  - **Function**

- **solid:color** `diffuseColor ambientIntensity specularColor shininess emissiveColor transparency`  
  - **Function**

- **solid:color** `diffuseColor ambientIntensity specularColor shininess emissiveColor`  
  - **Function**

- **solid:color** `diffuseColor ambientIntensity specularColor`  
  - **Function**
solid:color  diffuseColor ambientIntensity [Function]
solid:color  diffuseColor [Function]

Returns an appearance, the optical properties of the objects with which it is associated. ambientIntensity, shininess, and transparency must be numbers between 0 and 1. diffuseColor, specularColor, and emissiveColor are objects of type Section 5.11.1 [Color Data-Type], page 134, 24-bit sRGB integers or lists of 3 numbers between 0.0 and 1.0. If a color argument is omitted or #f, then the default color will be used.

solid:texture  image color scale rotation center translation [Function]
solid:texture  image color scale rotation center [Function]
solid:texture  image color scale rotation [Function]
solid:texture  image color scale [Function]
solid:texture  image color [Function]
solid:texture  image [Function]

Returns an appearance, the optical properties of the objects with which it is associated. image is a string naming a JPEG or PNG image resource. color is #f, a color, or the string returned by solid:color. The rest of the optional arguments specify 2-dimensional transforms applying to the image.

scale must be #f, a number, or list or vector of 2 numbers specifying the scale to apply to image. rotation must be #f or the number of degrees to rotate image. center must be #f or a list or vector of 2 numbers specifying the center of image relative to the image dimensions. translation must be #f or a list or vector of 2 numbers specifying the translation to apply to image.

solid:font  family style justify size spacing language direction [Function]

Returns a fontstyle object suitable for passing as an argument to solid:text. Any of the arguments may be #f, in which case its default value, which is first in each list of allowed values, is used.

family is a case-sensitive string naming a font; ‘SERIF’, ‘SANS’, and ‘TYPEWRITER’ are supported at the minimum.

style is a case-sensitive string ‘PLAIN’, ‘BOLD’, ‘ITALIC’, or ‘BOLDITALIC’.

justify is a case-sensitive string ‘FIRST’, ‘BEGIN’, ‘MIDDLE’, or ‘END’; or a list of one or two case-sensitive strings (same choices). The mechanics of justify get complicated; it is explained by tables 6.2 to 6.7 of http://www.web3d.org/x3d/specifications/vrml/ISO-IEC-14772-IS-VRML97WithAmendment1/part1/nodesRef.html#Table6.2

size is the extent, in the non-advancing direction, of the text. size defaults to 1.

spacing is the ratio of the line (or column) offset to size. spacing defaults to 1.

language is the RFC-1766 language name.

direction is a list of two numbers: (x y). If (> (abs x) (abs y)), then the text will be arrayed horizontally; otherwise vertically. The direction in which characters are arrayed is determined by the sign of the major axis: positive x being left-to-right; positive y being top-to-bottom.
Aggregating Objects

**solid:center-row-of** number solid spacing

Returns a row of number solid objects spaced evenly spacing apart.

**solid:center-array-of** number-a number-b solid spacing-a spacing-b

Returns number-b rows, spacing-b apart, of number-a solid objects spacing-a apart.

**solid:center-pile-of** number-a number-b number-c solid spacing-a spacing-b spacing-c

Returns number-c planes, spacing-c apart, of number-b rows, spacing-b apart, of number-a solid objects spacing-a apart.

**solid:arrow** center

center must be a list or vector of three numbers. Returns an upward pointing metallic arrow centered at center.

**solid:arrow**

Returns an upward pointing metallic arrow centered at the origin.

Spatial Transformations

**solid:translation** center solid . . .

center must be a list or vector of three numbers. solid:translation Returns an aggregate of solids, . . . with their origin moved to center.

**solid:scale** scale solid . . .

scale must be a number or a list or vector of three numbers. solid:scale Returns an aggregate of solids, . . . scaled per scale.

**solid:rotation** axis angle solid . . .

axis must be a list or vector of three numbers. solid:rotation Returns an aggregate of solids, . . . rotated angle degrees around the axis axis.

5.11 Color

http://people.csail.mit.edu/jaffer/Color

The goals of this package are to provide methods to specify, compute, and transform colors in a core set of additive color spaces. The color spaces supported should be sufficient for working with the color data encountered in practice and the literature.

5.11.1 Color Data-Type

(require 'color)

color? object

Returns #t if object is a color.

color? object typ

Returns #t if object is a color of color-space typ. The symbol typ must be one of:

- CIEXYZ
- RGB709
- L*a*b*
- L*u*v*
- sRGB
- e-sRGB
- L*C*h

\textbf{make-color \space \textit{arg} ...} \quad \text{[Function]}

Returns a color of type \textit{space}.

- For \textit{space} arguments CIEXYZ, RGB709, and sRGB, the sole \textit{arg} is a list of three numbers.
- For \textit{space} arguments L*a*b*, L*u*v*, and L*C*h, \textit{arg} is a list of three numbers optionally followed by a whitepoint.
- For xRGB, \textit{arg} is an integer.
- For e-sRGB, the arguments are as for e-sRGB->color.

\textbf{color-space \textit{color}} \quad \text{[Function]}

Returns the symbol for the color-space in which \textit{color} is embedded.

\textbf{color-precision \textit{color}} \quad \text{[Function]}

For colors in digital color-spaces, \textbf{color-precision} returns the number of bits used for each of the R, G, and B channels of the encoding. Otherwise, \textbf{color-precision} returns #f.

\textbf{color-white-point \textit{color}} \quad \text{[Function]}

Returns the white-point of \textit{color} in all color-spaces except CIEXYZ.

\textbf{convert-color \textit{color \space \textit{space \space \textit{white-point}}} \quad \text{[Function]}

\textbf{convert-color \textit{color \space \textit{space}}} \quad \text{[Function]}

\textbf{convert-color \textit{color e-sRGB \textit{precision}}} \quad \text{[Function]}

Converts \textit{color} into \textit{space} at optional \textit{white-point}.

### 5.11.1.1 External Representation

Each color encoding has an external, case-insensitive representation. To ensure portability, the white-point for all color strings is D65.\footnote{Readers may recognize these color string formats from Xlib. X11’s color management system was doomed by its fiction that CRT monitors’ (and X11 default) color-spaces were linear RGBi. Unable to shed this legacy, the only practical way to view pictures on X is to ignore its color management system and use an sRGB monitor. In this implementation the device-independent RGB709 and sRGB spaces replace the device-dependent RGBi and RGB spaces of Xlib.}

\begin{tabular}{|l|l|}
\hline
Color Space & External Representation \\
CIEXYZ & CIEXYZ:\textless X\textgreater /\textless Y\textgreater /\textless Z\textgreater \\
RGB709 & RGBi:\textless R\textgreater /\textless G\textgreater /\textless B\textgreater \\
L*a*b* & CIELAB:\textless L\textgreater /\textless a\textgreater /\textless b\textgreater \\
L*u*v* & CIELuv:\textless L\textgreater /\textless u\textgreater /\textless v\textgreater \\
L*C*h & CIELCh:\textless L\textgreater /\textless C\textgreater /\textless h\textgreater \\
\hline
\end{tabular}
The X, Y, Z, L, a, b, u, v, C, h, R, G, and B fields are (Scheme) real numbers within the appropriate ranges.

<table>
<thead>
<tr>
<th>Color Space</th>
<th>External Representation</th>
</tr>
</thead>
<tbody>
<tr>
<td>sRGB</td>
<td>sRGB:&lt;R&gt;/&lt;G&gt;/&lt;B&gt;</td>
</tr>
<tr>
<td>e-sRGB10</td>
<td>e-sRGB10:&lt;R&gt;/&lt;G&gt;/&lt;B&gt;</td>
</tr>
<tr>
<td>e-sRGB12</td>
<td>e-sRGB12:&lt;R&gt;/&lt;G&gt;/&lt;B&gt;</td>
</tr>
<tr>
<td>e-sRGB16</td>
<td>e-sRGB16:&lt;R&gt;/&lt;G&gt;/&lt;B&gt;</td>
</tr>
</tbody>
</table>

The R, G, and B, fields are non-negative exact decimal integers within the appropriate ranges.

Several additional syntaxes are supported by string->color:

<table>
<thead>
<tr>
<th>Color Space</th>
<th>External Representation</th>
</tr>
</thead>
<tbody>
<tr>
<td>sRGB</td>
<td>sRGB:&lt;RRGGBB&gt;</td>
</tr>
<tr>
<td>sRGB</td>
<td>#&lt;RRGGBB&gt;</td>
</tr>
<tr>
<td>sRGB</td>
<td>0x&lt;RRGGBB&gt;</td>
</tr>
<tr>
<td>sRGB</td>
<td>#x&lt;RRGGBB&gt;</td>
</tr>
</tbody>
</table>

Where RRGGBB is a non-negative six-digit hexadecimal number.

**color->string** color

Returns a string representation of color.

**string->color** string

Returns the color represented by string. If string is not a syntactically valid notation for a color, then string->color returns #f.

### 5.11.1.2 White

We experience color relative to the illumination around us. CIEXYZ coordinates, although subject to uniform scaling, are objective. Thus other color spaces are specified relative to a white point in CIEXYZ coordinates.

The white point for digital color spaces is set to D65. For the other spaces a white-point argument can be specified. The default if none is specified is the white-point with which the color was created or last converted; and D65 if none has been specified.

**D65** [Constant]

Is the color of 6500.K (blackbody) illumination. D65 is close to the average color of daylight.

**D50** [Constant]

Is the color of 5000.K (blackbody) illumination. D50 is the color of indoor lighting by incandescent bulbs, whose filaments have temperatures around 5000.K.

### 5.11.2 Color Spaces

**Measurement-based Color Spaces**

The *tristimulus* color spaces are those whose component values are proportional measurements of light intensity. The CIEXYZ(1931) system provides 3 sets of spectra to dot-product with a spectrum of interest. The result of those dot-products is coordinates in CIEXYZ.
space. All tristimulus color spaces are related to CIEXYZ by linear transforms, namely matrix multiplication. Of the color spaces listed here, CIEXYZ and RGB709 are tristimulus spaces.

**CIEXYZ**

[Color Space]

The CIEXYZ color space covers the full gamut. It is the basis for color-space conversions.

CIEXYZ is a list of three inexact numbers between 0.0 and 1.1. '(0. 0. 0.) is black; '(1. 1. 1.) is white.

ciexyz->color xyz

xyz must be a list of 3 numbers. If xyz is valid CIEXYZ coordinates, then ciexyz->color returns the color specified by xyz; otherwise returns #f.

color:ciexyz x y z

Returns the CIEXYZ color composed of x, y, z. If the coordinates do not encode a valid CIEXYZ color, then an error is signaled.

color->ciexyz color

Returns the list of 3 numbers encoding color in CIEXYZ.

**RGB709**

[Color Space]

BT.709-4 (03/00) Parameter values for the HDTV standards for production and international programme exchange specifies parameter values for chromaticity, sampling, signal format, frame rates, etc., of high definition television signals.

An RGB709 color is represented by a list of three inexact numbers between 0.0 and 1.0. '(0. 0. 0.) is black '(1. 1. 1.) is white.

rgb709->color rgb

rgb must be a list of 3 numbers. If rgb is valid RGB709 coordinates, then rgb709->color returns the color specified by rgb; otherwise returns #f.

color:rgb709 r g b

Returns the RGB709 color composed of r, g, b. If the coordinates do not encode a valid RGB709 color, then an error is signaled.

color->rgb709 color

Returns the list of 3 numbers encoding color in RGB709.

**Perceptual Uniformity**

Although properly encoding the chromaticity, tristimulus spaces do not match the logarithmic response of human visual systems to intensity. Minimum detectable differences between colors correspond to a smaller range of distances (6:1) in the L*a*b* and L*u*v* spaces than in tristimulus spaces (80:1). For this reason, color distances are computed in L*a*b* (or L*C*h).

**L*a*b***

[Color Space]

Is a CIE color space which better matches the human visual system’s perception of color. It is a list of three numbers:
• 0 \leq L^* \leq 100 (CIE Lightness)
• -500 \leq a^* \leq 500
• -200 \leq b^* \leq 200

\texttt{l*a*b*->color L^*a^*b^* white-point} \quad \text{[Function]}
\begin{itemize}
\item \texttt{l*a*b*} must be a list of 3 numbers. If \texttt{L*a*b*} is valid \texttt{L*a*b*} coordinates, then
\item \texttt{l*a*b*->color} returns the color specified by \texttt{L*a*b*}; otherwise returns \#f.
\end{itemize}

\texttt{color:l*a*b* L^* a^* b^* white-point} \quad \text{[Function]}
Returns the \texttt{L*a*b*} color composed of \texttt{L*}, \texttt{a*}, \texttt{b*} with \texttt{white-point}.

\texttt{color:l*a*b* L^* a^* b^*} \quad \text{[Function]}
Returns the \texttt{L*a*b*} color composed of \texttt{L*}, \texttt{a*}, \texttt{b*}. If the coordinates do not encode a valid \texttt{L*a*b*} color, then an error is signaled.

\texttt{color->l*a*b* color white-point} \quad \text{[Function]}
Returns the list of 3 numbers encoding \texttt{color} in \texttt{L*a*b*} with \texttt{white-point}.

\texttt{L*u*v* \quad [Color Space]}
Is another CIE encoding designed to better match the human visual system’s perception of color.

\texttt{l*u*v*->color L^*u^*v^* white-point} \quad \text{[Function]}
\begin{itemize}
\item \texttt{l*u*v*} must be a list of 3 numbers. If \texttt{L*u*v*} is valid \texttt{L*u*v*} coordinates, then
\item \texttt{l*u*v*->color} returns the color specified by \texttt{L*u*v*}; otherwise returns \#f.
\end{itemize}

\texttt{color:l*u*v* L^* u^* v^* white-point} \quad \text{[Function]}
Returns the \texttt{L*u*v*} color composed of \texttt{L*}, \texttt{u*}, \texttt{v*} with \texttt{white-point}.

\texttt{color:l*u*v* L^* u^* v^*} \quad \text{[Function]}
Returns the \texttt{L*u*v*} color composed of \texttt{L*}, \texttt{u*}, \texttt{v*}. If the coordinates do not encode a valid \texttt{L*u*v*} color, then an error is signaled.

\texttt{color->l*u*v* color white-point} \quad \text{[Function]}
Returns the list of 3 numbers encoding \texttt{color} in \texttt{L*u*v*} with \texttt{white-point}.

\texttt{color->l*u*v* color} \quad \text{[Function]}
Returns the list of 3 numbers encoding \texttt{color} in \texttt{L*u*v*}.

\textbf{Cylindrical Coordinates}

HSL (Hue Saturation Lightness), HSV (Hue Saturation Value), HSI (Hue Saturation Intensity) and HCI (Hue Chroma Intensity) are cylindrical color spaces (with angle hue). But these spaces are all defined in terms device-dependent RGB spaces.

One might wonder if there is some fundamental reason why intuitive specification of color must be device-dependent. But take heart! A cylindrical system can be based on \texttt{L*a*b*} and is used for predicting how close colors seem to observers.
L*C*h

Expresses the *a and b* of L*a*b* in polar coordinates. It is a list of three numbers:

- 0 <= L* <= 100 (CIE Lightness)
- C* (CIE Chroma) is the distance from the neutral (gray) axis.
- 0 <= h <= 360 (CIE Hue) is the angle.

The colors by quadrant of h are:

<table>
<thead>
<tr>
<th>h</th>
<th>Colors</th>
<th>L</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>red, orange, yellow</td>
<td>90</td>
</tr>
<tr>
<td>90</td>
<td>yellow, yellow-green, green</td>
<td>180</td>
</tr>
<tr>
<td>180</td>
<td>green, cyan (blue-green), blue</td>
<td>270</td>
</tr>
<tr>
<td>270</td>
<td>blue, purple, magenta</td>
<td>360</td>
</tr>
</tbody>
</table>

l*c*h->color L*C*h white-point

L*C*h must be a list of 3 numbers. If L*C*h is valid L*C*h coordinates, then l*c*h->color returns the color specified by L*C*h; otherwise returns #f.

color:l*c*h L* C* h white-point

Returns the L*C*h color composed of L*, C*, h with white-point.

color:l*c*h L* C* h

Returns the L*C*h color composed of L*, C*, h. If the coordinates do not encode a valid L*C*h color, then an error is signaled.

color->l*c*h color white-point

Returns the list of 3 numbers encoding color in L*C*h with white-point.

color->l*c*h color

Returns the list of 3 numbers encoding color in L*C*h.

Digital Color Spaces

The color spaces discussed so far are impractical for image data because of numerical precision and computational requirements. In 1998 the IEC adopted A Standard Default Color Space for the Internet - sRGB (http://www.w3.org/Graphics/Color/sRGB). sRGB was cleverly designed to employ the 24-bit (256x256x256) color encoding already in widespread use; and the 2.2 gamma intrinsic to CRT monitors.

Conversion from CIEXYZ to digital (sRGB) color spaces is accomplished by conversion first to a RGB709 tristimulus space with D65 white-point; then each coordinate is individually subjected to the same non-linear mapping. Inverse operations in the reverse order create the inverse transform.
sRGB  
[Color Space]
Is "A Standard Default Color Space for the Internet". Most display monitors will work fairly well with sRGB directly. Systems using ICC profiles \(^2\) should work very well with sRGB.

\textbf{srgb->color \ rgb}
[Function]
\(\text{rgb} \text{ must be a list of 3 numbers. If } \text{rgb} \text{ is valid sRGB coordinates, then } \text{srgb->color} \text{ returns the color specified by } \text{rgb}; \text{ otherwise returns } \text{#f}.\)

\textbf{color:srgb \ r \ g \ b}
[Function]
Returns the sRGB color composed of \(r, g, b\). If the coordinates do not encode a valid sRGB color, then an error is signaled.

xRGB  
[Color Space]
Represents the equivalent sRGB color with a single 24-bit integer. The most significant 8 bits encode red, the middle 8 bits blue, and the least significant 8 bits green.

\textbf{color->srgb \ color}
[Function]
Returns the list of 3 integers encoding \(\text{color}\) in sRGB.

\textbf{color->xrgb \ color}
[Function]
Returns the 24-bit integer encoding \(\text{color}\) in sRGB.

\textbf{xrgb->color \ k}
[Function]
Returns the sRGB color composed of the 24-bit integer \(k\).

e-sRGB  
[Color Space]
Is "Photography - Electronic still picture imaging - Extended sRGB color encoding" (PIMA 7667:2001). It extends the gamut of sRGB; and its higher precision numbers provide a larger dynamic range.

A triplet of integers represent e-sRGB colors. Three precisions are supported:

\(\text{e-sRGB10} \quad 0 \text{ to } 1023\)
\(\text{e-sRGB12} \quad 0 \text{ to } 4095\)
\(\text{e-sRGB16} \quad 0 \text{ to } 65535\)

\textbf{e-srgb->color \ precision \ rgb}
[Function]
\(\text{precision} \text{ must be the integer } 10, 12, \text{ or } 16. \text{ rgb} \text{ must be a list of 3 numbers. If } \text{rgb} \text{ is valid e-sRGB coordinates, then } \text{e-srgb->color} \text{ returns the color specified by } \text{rgb}; \text{ otherwise returns } \text{#f}.\)

\(^2\) A comprehensive encoding of transforms between CIEXYZ and device color spaces is the International Color Consortium profile format, ICC.1:1998-09:

The intent of this format is to provide a cross-platform device profile format. Such device profiles can be used to translate color data created on one device into another device's native color space.
Chapter 5: Mathematical Packages 141

**color:e-srgb 10 r g b**
Returns the e-sRGB10 color composed of integers r, g, b.

**color:e-srgb 12 r g b**
Returns the e-sRGB12 color composed of integers r, g, b.

**color:e-srgb 16 r g b**
Returns the e-sRGB16 color composed of integers r, g, b. If the coordinates do not encode a valid e-sRGB color, then an error is signaled.

**color->e-srgb precision color**
*precision* must be the integer 10, 12, or 16. **color->e-srgb** returns the list of 3 integers encoding **color** in sRGB10, sRGB12, or sRGB16.

### 5.11.3 Spectra

The following functions compute colors from spectra, scale color luminance, and extract chromaticity. XYZ is used in the names of procedures for unnormalized colors; the coordinates of CIEXYZ colors are constrained as described in Section 5.11.2 [Color Spaces], page 136.

```scheme
(require 'color-space)
```

A spectrum may be represented as:

- A procedure of one argument accepting real numbers from 380e-9 to 780e-9, the wavelength in meters; or
- A vector of real numbers representing intensity samples evenly spaced over some range of wavelengths overlapping the range 380e-9 to 780e-9.

CIEXYZ values are calculated as dot-product with the X, Y (Luminance), and Z Spectral Tristimulus Values. The files ‘cie1931.xyz’ and ‘cie1964.xyz’ in the distribution contain these CIE-defined values.

**cie1964**

**cie1931**

**ciexyz**
Requires Spectral Tristimulus Values, defaulting to cie1931, defining cie:x-bar, cie:y-bar, and cie:z-bar.

```scheme
(require 'cie1964) or (require 'cie1931)
```
will **load-ciexyz** specific values used by the following spectrum conversion procedures. The spectrum conversion procedures (**require 'ciexyz**) to assure that a set is loaded.

**read-cie-illuminant path**
*path* must be a string naming a file consisting of 107 numbers for 5.nm intervals from 300.nm to 830.nm. **read-cie-illuminant** reads (using Scheme **read**) these numbers and returns a length 107 vector filled with them.
Chapter 5: Mathematical Packages

(define CIE:SI-D65
  (read-CIE-illuminant (in-vicinity (library-vicinity) "ciesid65.dat")))
(spectrum->XYZ CIE:SI-D65 300e-9 830e-9)
⇒ (25.10856942237499 26.41801346562500 28.764075683374993)

Function
read-normalized-illuminant path
path must be a string naming a file consisting of 107 numbers for 5.nm intervals from 300.nm to 830.nm. read-normalized-illuminant reads (using Scheme read) these numbers and returns a length 107 vector filled with them, normalized so that spectrum->XYZ of the illuminant returns its whitepoint.

CIE Standard Illuminants A and D65 are included with SLIB:

(define CIE:SI-A
  (read-normalized-illuminant (in-vicinity (library-vicinity) "ciesia.dat")))
(define CIE:SI-D65
  (read-normalized-illuminant (in-vicinity (library-vicinity) "ciesid65.dat")))
(spectrum->XYZ CIE:SI-A 300e-9 830e-9)
⇒ (1.098499460820401 999.9999999999998e-3 355.8173930654951e-3)
(CIEXYZ->sRGB (spectrum->XYZ CIE:SI-A 300e-9 830e-9))
⇒ (255 234 133)
(spectrum->XYZ CIE:SI-D65 300e-9 830e-9)
⇒ (950.4336673552745e-3 1.0000000000000002 1.0888053986649182)
(CIEXYZ->sRGB (spectrum->XYZ CIE:SI-D65 300e-9 830e-9))
⇒ (255 255 255)

Function
illuminant-map proc siv
siv must be a one-dimensional array or vector of 107 numbers. illuminant-map returns a vector of length 107 containing the result of applying proc to each element of siv.

Function
illuminant-map->XYZ proc siv
(spectrum->XYZ (illuminant-map proc siv) 300e-9 830e-9)

Function
spectrum->XYZ proc
proc must be a function of one argument. spectrum->XYZ computes the CIEXYZ(1931) values for the spectrum returned by proc when called with arguments from 380e-9 to 780e-9, the wavelength in meters.

Function
spectrum->XYZ spectrum x1 x2
x1 and x2 must be positive real numbers specifying the wavelengths (in meters) corresponding to the zeroth and last elements of vector or list spectrum. spectrum->XYZ returns the CIEXYZ(1931) values for a light source with spectral values proportional to the elements of spectrum at evenly spaced wavelengths between x1 and x2.

Compute the colors of 6500.K and 5000.K blackbody radiation:

(require 'color-space)
(define xyz (spectrum->XYZ (blackbody-spectrum 6500)))
(define y_n (cadr xyz))
(map (lambda (x) (/ x y_n)) xyz)
(define xyz (spectrum->XYZ (blackbody-spectrum 5000)))
(map (lambda (x) (/ x y_n)) xyz)
⇒ (0.2933441826889158 0.2988931825387761 0.25783646831201573)

Chapter 5: Mathematical Packages

⇒ (0.9687111145512467 1.0 1.1210875945303613)

Chapter 5: Mathematical Packages

⇒ (0.3135191660557008 0.3236456786200268)

⇒ (0.34508082841161052 0.3516084965163377)

⇒ (0.3135191660557008 0.3236456786200268)
(temperature->chromaticity 5000)
⇒ (0.34508082841161052 0.3516084965163377)

\[ \text{XYZ->chromaticity}\text{ } xyz \]
Returns a two element list: the x and y components of xyz normalized to 1 (\(= x + y + z\)).

\[ \text{chromaticity->CIEXYZ}\text{ } x \text{ } y \]
Returns the list of x, and y, \(1 - y - x\).

\[ \text{chromaticity->whitepoint}\text{ } x \text{ } y \]
Returns the CIEXYZ(1931) values having luminosity 1 and chromaticity x and y.

Many color datasets are expressed in xyY format; chromaticity with CIE luminance (Y). But xyY is not a CIE standard like CIEXYZ, CIELAB, and CIELUV. Although chrominance is well defined, the luminance component is sometimes scaled to 1, sometimes to 100, but usually has no obvious range. With no given whitepoint, the only reasonable course is to ascertain the luminance range of a dataset and normalize the values to lie from 0 to 1.

\[ \text{XYZ->xyY}\text{ } xyz \]
Returns a three element list: the x and y components of XYZ normalized to 1, and CIE luminance Y.

\[ \text{xyY->XYZ}\text{ } xyY \]
\[ \text{xyY:normalize-colors}\text{ } colors \]
\(colors\) is a list of xyY triples. \text{xyY:normalize-colors} scales each chromaticity so it sums to 1 or less; and divides the Y values by the maximum Y in the dataset, so all lie between 0 and 1.

\[ \text{xyY:normalize-colors}\text{ } colors \text{ } n \]
If \(n\) is positive real, then \text{xyY:normalize-colors} divides the Y values by \(n\) times the maximum Y in the dataset.

If \(n\) is an exact non-positive integer, then \text{xyY:normalize-colors} divides the Y values by the maximum of the Ys in the dataset excepting the \(-n\) largest Y values.

In all cases, returned Y values are limited to lie from 0 to 1.

Why would one want to normalize to other than 1? If the sun or its reflection is the brightest object in a scene, then normalizing to its luminance will tend to make the rest of the scene very dark. As with photographs, limiting the specular highlights looks better than darkening everything else.

The results of measurements being what they are, \text{xyY:normalize-colors} is extremely tolerant. Negative numbers are replaced with zero, and chromaticities with sums greater than one are scaled to sum to one.

5.11.4 Color Difference Metrics
(\text{require} \ 'color-space)

The low-level metric functions operate on lists of 3 numbers, lab1, lab2, lch1, or lch2.
The wrapped functions operate on objects of type color, color1 and color2 in the function entries.

\texttt{(require 'color)}

\texttt{L*a*b*:DE* lab1 lab2} \hspace{1cm} \texttt{[Function]}

Returns the Euclidean distance between \texttt{lab1} and \texttt{lab2}.

\texttt{CIE:DE* color1 color2 white-point} \hspace{1cm} \texttt{[Function]}

\texttt{CIE:DE* color1 color2} \hspace{1cm} \texttt{[Function]}

Returns the Euclidean distance in L*a*b* space between \texttt{color1} and \texttt{color2}.

\texttt{L*a*b*:DE*94 lab1 lab2 parametric-factors} \hspace{1cm} \texttt{[Function]}

\texttt{L*a*b*:DE*94 lab1 lab2} \hspace{1cm} \texttt{[Function]}

\texttt{CIE:DE*94 color1 color2 parametric-factors} \hspace{1cm} \texttt{[Function]}

\texttt{CIE:DE*94 color1 color2} \hspace{1cm} \texttt{[Function]}

Measures distance in the L*a*b* color-space. The three axes are individually scaled in their contributions to the total distance.

\texttt{DE*94} is not symmetrical in its arguments. \texttt{lab1} is the “reference” color and \texttt{lab2} is the “sample” color.

The CIE has defined reference conditions under which the metric with default parameters can be expected to perform well. These are:

- The specimens are homogeneous in colour.
- The colour difference (CIELAB) is \( \leq 5 \) units.
- They are placed in direct edge contact.
- Each specimen subtends an angle of \( >4 \) degrees to the assessor, whose colour vision is normal.
- They are illuminated at 1000 lux, and viewed against a background of uniform grey, with L* of 50, under illumination simulating D65.

The \texttt{parametric-factors} argument is a list of 3 quantities kL, kC and kH. \texttt{parametric-factors} independently adjust each colour-difference term to account for any deviations from the reference viewing conditions. Under the reference conditions explained above, the default is \( kL = kC = kH = 1 \).

The Color Measurement Committee of The Society of Dyers and Colorists in Great Britain created a more sophisticated color-distance function for use in judging the consistency of dye lots. With CMC:DE* it is possible to use a single value pass/fail tolerance for all shades.

\texttt{CMC-DE lch1 lch2 parametric-factors} \hspace{1cm} \texttt{[Function]}

\texttt{CMC-DE lch1 lch2 l c} \hspace{1cm} \texttt{[Function]}

\texttt{CMC-DE lch1 lch2 l} \hspace{1cm} \texttt{[Function]}

\texttt{CMC-DE lch1 lch2} \hspace{1cm} \texttt{[Function]}

\texttt{CMC:DE* color1 color2 l c} \hspace{1cm} \texttt{[Function]}

\texttt{CMC:DE* color1 color2} \hspace{1cm} \texttt{[Function]}

\texttt{CMC:DE} is a L*C*h metric. The \texttt{parametric-factors} argument is a list of 2 numbers \( l \) and \( c \). \( l \) and \( c \) parameterize this metric. 1 and 1 are recommended for perceptibility; the default, 2 and 1, for acceptability.
5.11.5 Color Conversions

This package contains the low-level color conversion and color metric routines operating on lists of 3 numbers. There is no type or range checking.

```
(require 'color-space)
```

**CIEXYZ:D65**

Is the color of 6500.K (blackbody) illumination. D65 is close to the average color of daylight.

**CIEXYZ:D50**

Is the color of 5000.K (blackbody) illumination. D50 is the color of indoor lighting by incandescent bulbs.

**CIEXYZ:A**

**CIEXYZ:B**

**CIEXYZ:C**

**CIEXYZ:E**

CIE 1931 illuminants normalized to 1 = y.

**color:linear-transform**

**matrix row**

**CIEXYZ->RGB709**

**xyz**

**RGB709->CIEXYZ**

**srgb**

**CIEXYZ->L*u*v***

**xyz**

white-point

**CIEXYZ->L*u*v***

**xyz**

**L*u*v***

white-point

**L*u*v***

**CIEXYZ**

**L*u*v***

white-point

**The white-point defaults to CIEXYZ:D65.**

**CIEXYZ->L*a*b***

**xyz**

white-point

**CIEXYZ->L*a*b***

**xyz**

**L*a*b***

white-point

**L*a*b***

**CIEXYZ**

**L*a*b***

white-point

**The XYZ white-point defaults to CIEXYZ:D65.**

**L*a*b***

**L*C*h**

**L*a*b***

**L*C*h**

**CIEXYZ->sRGB**

**xyz**

**sRGB->CIEXYZ**

**srgb**

**CIEXYZ->xRGB**

**xyz**

**xRGB->CIEXYZ**

**srgb**

**sRGB->xRGB**

**xyz**

**xRGB->sRGB**

**srgb**

**CIEXYZ->e-sRGB**

**n**

**xyz**

**e-sRGB->CIEXYZ**

**n**

**sRGB->e-sRGB**

**n**

**srgb**
e-sRGB->sRGB  n  srgb

The integer n must be 10, 12, or 16. Because sRGB and e-sRGB use the same RGB709 chromaticities, conversion between them is simpler than conversion through CIEXYZ.

Do not convert e-sRGB precision through e-sRGB->sRGB then sRGB->e-sRGB – values would be truncated to 8-bits!

e-sRGB->e-sRGB  n1  srgb  n2

The integers n1 and n2 must be 10, 12, or 16. e-sRGB->e-sRGB converts srgb to e-sRGB of precision n2.

5.11.6 Color Names

(require 'color-names)

Rather than ballast the color dictionaries with numbered grays, file->color-dictionary discards them. They are provided through the grey procedure:

grey  k
    Returns (inexact->exact (round (* k 2.55))), the X11 color grey<k>.

A color dictionary is a database table relating canonical color-names to color-strings (see Section 5.11.1 [Color Data-Type], page 134).

The column names in a color dictionary are unimportant; the first field is the key, and the second is the color-string.

color-name:canonicalize  name
    Returns a downcased copy of the string or symbol name with ‘_’, ‘-’, and whitespace removed.

color-name->color  name  table1  table2  ...
    table1, table2, ... must be color-dictionary tables. color-name->color searches for the canonical form of name in table1, table2, ... in order; returning the color-string of the first matching record; #f otherwise.

color-dictionaries->lookup  table1  table2  ...
    table1, table2, ... must be color-dictionary tables. color-dictionaries->lookup returns a procedure which searches for the canonical form of its string argument in table1, table2, ...; returning the color-string of the first matching record; and #f otherwise.

color-dictionary  name  rdb  base-table-type
    rdb must be a string naming a relational database file; and the symbol name a table therein. The database will be opened as base-table-type. color-dictionary returns the read-only table name in database name if it exists; #f otherwise.

color-dictionary  name  rdb
    rdb must be an open relational database or a string naming a relational database file; and the symbol name a table therein. color-dictionary returns the read-only table name in database name if it exists; #f otherwise.
load-color-dictionary name rdb base-table-type  [Function]
load-color-dictionary name rdb                [Function]
   rdb must be a string naming a relational database file; and the symbol name a table therein. If the symbol base-table-type is provided, the database will be opened as base-table-type. load-color-dictionary creates a top-level definition of the symbol name to a lookup procedure for the color dictionary name in rdb.

   The value returned by load-color-dictionary is unspecified.

Dictionary Creation

Dictionary Creation

(file->color-dictionary file table-name rdb base-table-type  [Function]
file->color-dictionary file table-name rdb                [Function]
   rdb must be an open relational database or a string naming a relational database file, table-name a symbol, and the string file must name an existing file with color-names and their corresponding xRGB (6-digit hex) values. file->color-dictionary creates a table table-name in rdb and enters the associations found in file into it.

url->color-dictionary url table-name rdb base-table-type  [Function]
url->color-dictionary url table-name rdb                [Function]
   rdb must be an open relational database or a string naming a relational database file and table-name a symbol. url->color-dictionary retrieves the resource named by the string url using the wget program; then calls file->color-dictionary to enter its associations in table-name in url.

This section has detailed the procedures for creating and loading color dictionaries. So where are the dictionaries to load?


Describes and evaluates several color-name dictionaries on the web. The following procedure creates a database containing two of these dictionaries.

make-slib-color-name-db  [Function]
   Creates an alist-table relational database in library-vicinity containing the Resene and saturate color-name dictionaries.

   If the files ‘resenecolours.txt’, ‘nbs-iscc.txt’, and ‘saturate.txt’ exist in the library-vicinity, then they used as the source of color-name data. Otherwise, make-slib-color-name-db calls url->color-dictionary with the URLs of appropriate source files.

The Short List

The Short List

(saturate name  [Function]
   Looks for name among the 19 saturated colors from Approximate Colors on CIE Chromaticity Diagram:
   reddish orange     orange     yellowish orange     yellow
greenish yellow  yellow green  yellowish green  green  
bluish green  blue green  greenish blue  blue  
purplish blue  bluish purple  purple  reddish purple  
red purple  purplish red  red

(http://people.csail.mit.edu/jaffer/Color/saturate.pdf). If \texttt{name} is found, the corresponding color is returned. Otherwise \#f is returned. Use \texttt{saturate} only for light source colors.

Resene Paints Limited, New Zealand’s largest privately-owned and operated paint manufacturing company, has generously made their \textit{Resene RGB Values List} available.

(require 'resene)

\texttt{resene name} \hfill \textbf{[Function]}

Looks for \texttt{name} among the 1300 entries in the Resene color-name dictionary (http://people.csail.mit.edu/jaffer/Color/resene.pdf). If \texttt{name} is found, the corresponding color is returned. Otherwise \#f is returned. The \textit{Resene RGB Values List} is an excellent source for surface colors.

If you include the \textit{Resene RGB Values List} in binary form in a program, then you must include its license with your program:

Resene RGB Values List
For further information refer to http://www.resene.co.nz
Copyright Resene Paints Ltd 2001

Permission to copy this dictionary, to modify it, to redistribute it, to distribute modified versions, and to use it for any purpose is granted, subject to the following restrictions and understandings.

1. Any text copy made of this dictionary must include this copyright notice in full.
2. Any redistribution in binary form must reproduce this copyright notice in the documentation or other materials provided with the distribution.
3. Resene Paints Ltd makes no warranty or representation that this dictionary is error-free, and is under no obligation to provide any services, by way of maintenance, update, or otherwise.
4. There shall be no use of the name of Resene or Resene Paints Ltd in any advertising, promotional, or sales literature without prior written consent in each case.
5. These RGB colour formulations may not be used to the detriment of Resene Paints Ltd.

5.11.7 Daylight

(require 'daylight)

This package calculates the colors of sky as detailed in:


\textit{A Practical Analytic Model for Daylight}

A. J. Preetham, Peter Shirley, Brian Smits
**solar-hour** \textit{julian-day hour} \hfill [Function]

Returns the solar-time in hours given the integer \textit{julian-day} in the range 1 to 366, and the local time in hours.

To be meticulous, subtract 4 minutes for each degree of longitude west of the standard meridian of your time zone.

**solar-declination** \textit{julian-day} \hfill [Function]

**solar-polar** \textit{declination latitude solar-hour} \hfill [Function]

Returns a list of \( \theta_s \), the solar angle from the zenith, and \( \phi_s \), the solar azimuth.

\( 0 \leq \theta_s \) measured in degrees. \( \phi_s \) is measured in degrees from due south; west of south being positive.

In the following procedures, the number \( 0 \leq \theta_s \leq 90 \) is the solar angle from the zenith in degrees.

Turbidity is a measure of the fraction of scattering due to haze as opposed to molecules. This is a convenient quantity because it can be estimated based on visibility of distant objects. This model fails for turbidity values less than 1.3.

\begin{verbatim}
512|:-|   | | * pure-air |
256|:-**|   | | : ** exceptionally-clear |
128|--: *|   | | : ** |
64|--: *|   | | : ** very-clear |
32|--: **|   | | : ** |
16|--: *** clear |   | | : **** |
8|--: **** |   | | : **** light-haze |
4|--: **** |   | | : **** |
2|--: ****** haze thin- |   | | : ****** fog |
1|------------------------------------------|   | 1 2 4 8 16 32 64 |
   | Meterorological range (km) versus Turbidity |
\end{verbatim}

**sunlight-spectrum** \textit{turbidity theta_s} \hfill [Function]

Returns a vector of 41 values, the spectrum of sunlight from 380.nm to 790.nm for a given \textit{turbidity} and \textit{theta_s}. 
sunlight-chromaticity turbidity theta_s  [Function]
   Given turbidity and theta_s, sunlight-chromaticity returns the CIEXYZ triple for
color of sunlight scaled to be just inside the RGB709 gamut.

zenith-xyy turbidity theta_s  [Function]
   Returns the xyY (chromaticity and luminance) at the zenith. The Luminance has
   units kcd/m^2.

overcast-sky-color-xyy turbidity theta_s  [Function]
   turbidity is a positive real number expressing the amount of light scattering. The
   real number theta_s is the solar angle from the zenith in degrees.

   overcast-sky-color-xyy returns a function of one angle theta, the angle from the
   zenith of the viewing direction (in degrees); and returning the xyY value for light
   coming from that elevation of the sky.

clear-sky-color-xyy turbidity theta_s phi_s  [Function]

sky-color-xyy turbidity theta_s phi_s  [Function]
   turbidity is a positive real number expressing the amount of light scattering. The
   real number theta_s is the solar angle from the zenith in degrees. The real number
   phi_s is the solar angle from south.

clear-sky-color-xyy returns a function of two angles, theta and phi which specify
the angles from the zenith and south meridian of the viewing direction (in degrees);
returning the xyY value for light coming from that direction of the sky.

   sky-color-xyY calls overcast-sky-color-xyY for turbidity <= 20; otherwise the
clear-sky-color-xyy function.

5.12 Root Finding

   (require 'root)
   In the Newton method, divide the df/dx argument by the multiplicity of the desired
root in order to preserve quadratic convergence.

newton:find-integer-root f df/dx x0  [Function]
   Given integer valued procedure f, its derivative (with respect to its argument) df/dx,
and initial integer value x0 for which df/dx(x0) is non-zero, returns an integer x for
which f(x) is closer to zero than either of the integers adjacent to x; or returns #f if
such an integer can’t be found.

   To find the closest integer to a given integer’s square root:

   (define (integer-sqrt y)
      (newton:find-integer-root
         (lambda (x) (- (* x x) y))
         (lambda (x) (* 2 x))
         (ash 1 (quotient (integer-length y) 2)))))

   (integer-sqrt 15) ⇒ 4
newton:find-root \( f \, df/dx \, x_0 \, \text{prec} \)  
Given real valued procedures \( f \), \( df/dx \) of one (real) argument, initial real value \( x_0 \) for which \( df/dx(x_0) \) is non-zero, and positive real number \( \text{prec} \), returns a real \( x \) for which \( \abs{f(x)} \) is less than \( \text{prec} \); or returns \#f if such a real can't be found.

If \( \text{prec} \) is instead a negative integer, newton:find-root returns the result of -\( \text{prec} \) iterations.


There are 2 errors in Orchard's Table II. Line \( k=2 \) for starting value of \( 1000+j0 \) should have \( Z_k \) of \( 1.0475 + j4.1036 \) and line \( k=2 \) for starting value of \( 0+j1000 \) should have \( Z_k \) of \( 1.0988 + j4.0833 \).

laguerre:find-root \( f \, df/dz \, ddf/dz^2 \, z_0 \, \text{prec} \)  
Given complex valued procedure \( f \) of one (complex) argument, its derivative (with respect to its argument) \( df/dx \), its second derivative \( ddf/dz^2 \), initial complex value \( z_0 \), and positive real number \( \text{prec} \), returns a complex number \( z \) for which \( \text{magnitude}(f(z)) \) is less than \( \text{prec} \); or returns \#f if such a number can't be found.

If \( \text{prec} \) is instead a negative integer, laguerre:find-root returns the result of -\( \text{prec} \) iterations.

laguerre:find-polynomial-root \( \text{deg} \, f \, df/dz \, ddf/dz^2 \, z_0 \, \text{prec} \)  
Given polynomial procedure \( f \) of integer degree \( \text{deg} \) of one argument, its derivative (with respect to its argument) \( df/dx \), its second derivative \( ddf/dz^2 \), initial complex value \( z_0 \), and positive real number \( \text{prec} \), returns a complex number \( z \) for which \( \text{magnitude}(f(z)) \) is less than \( \text{prec} \); or returns \#f if such a number can't be found.

If \( \text{prec} \) is instead a negative integer, laguerre:find-polynomial-root returns the result of -\( \text{prec} \) iterations.

secant:find-root \( f \, x_0 \, x_1 \, \text{prec} \)  
secant:find-bracketed-root \( f \, x_0 \, x_1 \, \text{prec} \)
Given a real valued procedure \( f \) and two real valued starting points \( x_0 \) and \( x_1 \), returns a real \( x \) for which \( \abs{f(x)} \) is less than \( \text{prec} \); or returns \#f if such a real can't be found.

If \( x_0 \) and \( x_1 \) are chosen such that they bracket a root, that is
\[
\text{or} \quad (\neg (f \, x_0) \, 0 \, (f \, x_1)) \\
(\neg (f \, x_1) \, 0 \, (f \, x_0))
\]
then the root returned will be between \( x_0 \) and \( x_1 \), and \( f \) will not be passed an argument outside of that interval.

secant:find-bracketed-root will return \#f unless \( x_0 \) and \( x_1 \) bracket a root.

The secant method is used until a bracketing interval is found, at which point a modified *regula falsi* method is used.

If \( \text{prec} \) is instead a negative integer, secant:find-root returns the result of -\( \text{prec} \) iterations.
If `prec` is a procedure it should accept 5 arguments: `x0 f0 x1 f1` and `count`, where `f0` will be `(f x0)`, `f1 (f x1)`, and `count` the number of iterations performed so far. `prec` should return non-false if the iteration should be stopped.

### 5.13 Minimizing

```scheme
(require 'minimize)
```

The Golden Section Search algorithm finds minima of functions which are expensive to compute or for which derivatives are not available. Although optimum for the general case, convergence is slow, requiring nearly 100 iterations for the example \(x^3 - 2x - 5\).

If the derivative is available, Newton-Raphson is probably a better choice. If the function is inexpensive to compute, consider approximating the derivative.

```scheme
(golden-section-search f x0 x1 prec)
```

*Function*

`x_0` are `x_1` real numbers. The (single argument) procedure `f` is unimodal over the open interval \((x_0, x_1)\). That is, there is exactly one point in the interval for which the derivative of `f` is zero.

`golden-section-search` returns a pair \((x, f(x))\) where `f(x)` is the minimum. The `prec` parameter is the stop criterion. If `prec` is a positive number, then the iteration continues until `x` is within `prec` from the true value. If `prec` is a negative integer, then the procedure will iterate `-prec` times or until convergence. If `prec` is a procedure of seven arguments, `x0`, `x1`, `a`, `b`, `fa`, `fb`, and `count`, then the iterations will stop when the procedure returns `#t`.

Analytically, the minimum of \(x^3 - 2x - 5\) is 0.816497.

```scheme
(define func (lambda (x) (+ (* x (+ (* x x) -2)) -5)))
(golden-section-search func 0 1 (/ 10000))
==> (816.4883855245578e-3 . -6.0886621077391165)
(golden-section-search func 0 1 -5)
==> (819.6601125010515e-3 . -6.088637561916407)
(golden-section-search func 0 1
  (lambda (a b c d e f g) (= g 500)))
==> (816.4965933140557e-3 . -6.088662107903635)
```

### 5.14 The Limit

```scheme
(limit proc x1 x2 k)
(limit proc x1 x2)
```

*Proc* must be a procedure taking a single inexact real argument. `K` is the number of points on which `proc` will be called; it defaults to 8.

If `x1` is finite, then `Proc` must be continuous on the half-open interval:

```scheme
(x1 .. x1+x2]
```

---

And \( x_2 \) should be chosen small enough so that \( \text{proc} \) is expected to be monotonic or constant on arguments between \( x_1 \) and \( x_1 + x_2 \).

\( \text{Limit} \) computes the limit of \( \text{proc} \) as its argument approaches \( x_1 \) from \( x_1 + x_2 \). \( \text{Limit} \) returns a real number or real infinity or ‘\#f’.

If \( x_1 \) is not finite, then \( x_2 \) must be a finite nonzero real with the same sign as \( x_1 \); in which case \( \text{limit} \) returns:

\[
\text{(limit (lambda (x) (proc (/ x))) 0.0 (/ x2) k)}
\]

\( \text{Limit} \) examines the magnitudes of the differences between successive values returned by \( \text{proc} \) called with a succession of numbers from \( x_1 + x_2/k \) to \( x_1 \).

If the magnitudes of differences are monotonically decreasing, then then the limit is extrapolated from the degree \( n \) polynomial passing through the samples returned by \( \text{proc} \).

If the magnitudes of differences are increasing as fast or faster than a hyperbola matching at \( x_1 + x_2 \), then a real infinity with sign the same as the differences is returned.

If the magnitudes of differences are increasing more slowly than the hyperbola matching at \( x_1 + x_2 \), then the limit is extrapolated from the quadratic passing through the three samples closest to \( x_1 \).

If the magnitudes of differences are not monotonic or are not completely within one of the above categories, then \#f is returned.

;;; constant
\[
\text{(limit (lambda (x) (/ x x)) 0 1.0e-9)} \quad \Rightarrow \quad 1.0
\]
\[
\text{(limit (lambda (x) (expt 0 x)) 0 1.0e-9)} \quad \Rightarrow \quad 0.0
\]
\[
\text{(limit (lambda (x) (expt 0 x)) 0 -1.0e-9)} \quad \Rightarrow \quad +\text{inf.}0
\]

;;; linear
\[
\text{(limit + 0 976.5625e-6)} \quad \Rightarrow \quad 0.0
\]
\[
\text{(limit - 0 976.5625e-6)} \quad \Rightarrow \quad 0.0
\]

;;; vertical point of inflection
\[
\text{(limit sqrt 0 1.0e-18)} \quad \Rightarrow \quad 0.0
\]
\[
\text{(limit (lambda (x) (* x (log x))) 0 1.0e-9)} \quad \Rightarrow \quad -102.70578127633066e-12
\]
\[
\text{(limit (lambda (x) (/ (log x) x)) 0 1.0e-9)} \quad \Rightarrow \quad 96.12123142321669e-15
\]

;;; limits tending to infinity
\[
\text{(limit + +inf.0 1.0e9)} \quad \Rightarrow \quad +\text{inf.0}
\]
\[
\text{(limit + -inf.0 -1.0e9)} \quad \Rightarrow \quad -\text{inf.0}
\]
\[
\text{(limit / 0 1.0e-9)} \quad \Rightarrow \quad +\text{inf.0}
\]
\[
\text{(limit / 0 -1.0e-9)} \quad \Rightarrow \quad -\text{inf.0}
\]
\[
\text{(limit (lambda (x) (/ (log x) x)) 0 1.0e-9)} \quad \Rightarrow \quad -\text{inf.0}
\]
\[
\text{(limit (lambda (x) (/ (magnitude (log x)) x)) 0 -1.0e-9)} \quad \Rightarrow \quad -\text{inf.0}
\]

;;; limit doesn’t exist
\[
\text{(limit sin +inf.0 1.0e9)} \quad \Rightarrow \quad \#f
\]
\[
\text{(limit (lambda (x) (sin (/ x))) 0 1.0e-9)} \quad \Rightarrow \quad \#f
\]
\[
\text{(limit (lambda (x) (sin (/ x))) 0 -1.0e-9)} \quad \Rightarrow \quad \#f
\]
\[
\text{(limit (lambda (x) (/ (log x) x)) 0 -1.0e-9)} \quad \Rightarrow \quad \#f
\]
;; conditionally convergent - return #f
(limit (lambda (x) (/ (sin x) x)) +inf.0 1.0e222)

==> #f

;; asymptotes
(limit / -inf.0 -1.0e222) ==> 0.0
(limit / +inf.0 1.0e222) ==> 0.0
(limit (lambda (x) (expt x x)) 0 1.0e-18) ==> 1.0
(limit (lambda (x) (sin (/ x))) +inf.0 1.0e222) ==> 0.0
(limit (lambda (x) (/ (+ (exp (/ x)) 1))) 0 1.0e-9)

==> 0.0
(limit (lambda (x) (/ (+ (exp (/ x)) 1))) 0 -1.0e-9)

==> 1.0
(limit (lambda (x) (real-part (expt (tan x) (cos x)))) (/ pi 2) 1.0e-9)

==> 1.0

;; This example from the 1979 Macsyma manual grows so rapidly
;; that x2 must be less than 41. It correctly returns e^2.
(limit (lambda (x) (expt (+ x (exp x) (exp (* 2 x))) (/ x))) +inf.0 40)

==> 7.3890560989306504

;; LIMIT can calculate the proper answer when evaluation
;; of the function at the limit point does not:
(tan (atan +inf.0)) ==> 16.331778728383844e15
(limit tan (atan +inf.0) -1.0e-15) ==> +inf.0
(tan (atan +inf.0)) ==> 16.331778728383844e15
(limit tan (atan +inf.0) 1.0e-15) ==> -inf.0
((lambda (x) (expt (exp (/ -1 x)) x)) 0) ==> 1.0
(limit (lambda (x) (expt (exp (/ -1 x)) x)) 0 1.0e-9)

==> 0.0

5.15 Commutative Rings

Scheme provides a consistent and capable set of numeric functions. Inexacts implement a
field; integers a commutative ring (and Euclidean domain). This package allows one to use
basic Scheme numeric functions with symbols and non-numeric elements of commutative
rings.

(require 'commutative-ring)

The commutative-ring package makes the procedures +, -, *, /, and ^ careful in the
sense that any non-numeric arguments they do not reduce appear in the expression output.
In order to see what working with this package is like, self-set all the single letter identifiers
(to their corresponding symbols).

(define a 'a)
...
(define z 'z)

Or just (require 'self-set). Now try some sample expressions:
(+ (+ a b) (- a b)) ⇒ (* a 2)
(* (+ a b) (+ a b)) ⇒ (^ (+ a b) 2)
(* (+ a b) (- a b)) ⇒ (* (+ a b) (- a b))
(* (- a b) (- a b)) \Rightarrow (\^\ (- a b) 2)
(* (- a b) (+ a b)) \Rightarrow (* (+ a b) (- a b))
(/ (+ a b) (+ c d)) \Rightarrow (/ (+ a b) (+ c d))
(\^\ (+ a b) 3) \Rightarrow (\^\ (+ a b) 3)
(\^\ (+ a 2) 3) \Rightarrow (\^\ (+ 2 a) 3)

Associative rules have been applied and repeated addition and multiplication converted to multiplication and exponentiation.

We can enable distributive rules, thus expanding to sum of products form:

(set! *ruleset* (combined-rulesets distribute* distribute/))

(* (+ a b) (+ a b)) \Rightarrow (+ (* 2 a b) (^ a 2) (^ b 2))
(* (+ a b) (- a b)) \Rightarrow (- (^ a 2) (^ b 2))
(* (- a b) (- a b)) \Rightarrow (- (+ (^ a 2) (^ b 2)) (* 2 a b))
(* (- a b) (+ a b)) \Rightarrow (- (^ a 2) (^ b 2))
(/ (+ a b) (+ c d)) \Rightarrow (+ (/ a (+ c d)) (/ b (+ c d)))
(/ (+ a b) (- c d)) \Rightarrow (+ (/ a (- c d)) (/ b (- c d)))
(/ (- a b) (+ c d)) \Rightarrow (+ (/ a (- c d)) (/ b (+ c d)))
(/ (- a b) (- c d)) \Rightarrow (- (/ a (+ c d)) (/ b (+ c d)))
(\^\ (+ a b) 3) \Rightarrow (+ (* 3 a (- b 2)) (* 3 b (- a 2)) (- a 3) (- b 3))
(\^\ (+ a 2) 3) \Rightarrow (+ 8 (* a 12) (* (- a 2) 6) (- a 3))

Use of this package is not restricted to simple arithmetic expressions:

(require 'determinant)

(determinant '((a b c) (d e f) (g h i))) \Rightarrow
(- (+ (* a e i) (* b f g) (* c d h)) (* a f h) (* b d i) (* c e g))

Currently, only +, -, *, /, and ^ support non-numeric elements. Expressions with - are converted to equivalent expressions without -, so behavior for - is not defined separately. / expressions are handled similarly.

This list might be extended to include quotient, modulo, remainder, lcm, and gcd; but these work only for the more restrictive Euclidean (Unique Factorization) Domain.

## 5.16 Rules and Rulesets

The commutative-ring package allows control of ring properties through the use of rulesets.

*ruleset*  [Variable]
Contains the set of rules currently in effect. Rules defined by cring:define-rule are stored within the value of *ruleset* at the time cring:define-rule is called. If *ruleset* is #f, then no rules apply.

make-ruleset rule1 . . .  [Function]
make-ruleset name rule1 . . .  [Function]

Returns a new ruleset containing the rules formed by applying cring:define-rule to each 4-element list argument rule. If the first argument to make-ruleset is a symbol, then the database table created for the new ruleset will be named name. Calling make-ruleset with no rule arguments creates an empty ruleset.
combined-rulesets ruleset1 ...

combined-rulesets name ruleset1 ...

Returns a new ruleset containing the rules contained in each ruleset argument ruleset.
If the first argument to combined-ruleset is a symbol, then the database table created for the new ruleset will be named name. Calling combined-ruleset with no ruleset arguments creates an empty ruleset.

Two rulesets are defined by this package.

distribute*

Contains the ruleset to distribute multiplication over addition and subtraction.

distribute/

Contains the ruleset to distribute division over addition and subtraction.

Take care when using both distribute* and distribute/ simultaneously. It is possible to put / into an infinite loop.

You can specify how sum and product expressions containing non-numeric elements simplify by specifying the rules for + or * for cases where expressions involving objects reduce to numbers or to expressions involving different non-numeric elements.

cring:define-rule op sub-op1 sub-op2 reduction

Defines a rule for the case when the operation represented by symbol op is applied to lists whose cars are sub-op1 and sub-op2, respectively. The argument reduction is a procedure accepting 2 arguments which will be lists whose cars are sub-op1 and sub-op2.

cring:define-rule op sub-op1 'identity reduction

Defines a rule for the case when the operation represented by symbol op is applied to a list whose car is sub-op1, and some other argument. Reduction will be called with the list whose car is sub-op1 and some other argument.

If reduction returns #f, the reduction has failed and other reductions will be tried. If reduction returns a non-false value, that value will replace the two arguments in arithmetic (+, -, and *) calculations involving non-numeric elements.

The operations + and * are assumed commutative; hence both orders of arguments to reduction will be tried if necessary.

The following rule is the definition for distributing * over +.

(cring:define-rule
 ' * ' + 'identity
 (lambda (exp1 exp2)
   (apply + (map (lambda (trm) (* trm exp2)) (cdr exp1))))))

5.17 How to Create a Commutative Ring

The first step in creating your commutative ring is to write procedures to create elements of the ring. A non-numeric element of the ring must be represented as a list whose first element is a symbol or string. This first element identifies the type of the object. A convenient and
clear convention is to make the type-identifying element be the same symbol whose top-level value is the procedure to create it.

```
(define (n . list1)
  (cond ((and (= 2 (length list1))
      (eq? (car list1) (cadr list1)))
    0)
  ((not (term< (first list1) (last1 list1)))
   (apply n (reverse list1)))
  (else (cons 'n list1)))))

(define (s x y) (n x y))

(define (m . list1)
  (cond ((neq? (first list1) (term_min list1))
        (apply m (cyclicrotate list1)))
        ((term< (last1 list1) (cadr list1))
        (apply m (reverse (cyclicrotate list1))))
    (else (cons 'm list1)))))
```

Define a procedure to multiply 2 non-numeric elements of the ring. Other multiplicators are handled automatically. Objects for which rules have not been defined are not changed.

```
(define (n*n ni nj)
  (let ((list1 (cdr ni)) (list2 (cdr nj)))
    (cond ((null? (intersection list1 list2)) #f)
      ((and (eq? (last1 list1) (first list2))
           (neq? (first list1) (last1 list2)))
       (apply n (splice list1 list2)))
      ((and (eq? (first list1) (first list2))
           (neq? (last1 list1) (last1 list2)))
       (apply n (splice (reverse list1) list2)))
      ((and (eq? (last1 list1) (last1 list2))
           (neq? (first list1) (first list2)))
       (apply n (splice list1 (reverse list2))))
      ((and (eq? (last1 list1) (first list2))
           (eq? (first list1) (last1 list2)))
       (apply m (cyclicsplice list1 list2)))
      ((and (eq? (first list1) (first list2))
           (eq? (last1 list1) (last1 list2)))
       (apply m (cyclicsplice (reverse list1) list2)))
      (else #f)))))
```

Test the procedures to see if they work.

```scheme
;;; where cyclicrotate(list) is cyclic rotation of the list one step
;;; by putting the first element at the end
(define (cyclicrotate list1)
  (append (rest list1) (list (first list1))))
```
and where term_min(list) is the element of the list which is first in the term ordering.

(define (term_min list1)
  (car (sort list1 term<)))
(define (term< sym1 sym2)
  (string<? (symbol->string sym1) (symbol->string sym2)))
(define first car)
(define rest cdr)
(define (last1 list1) (car (last-pair list1)))
(define (neq? obj1 obj2) (not (eq? obj1 obj2)))

where splice is the concatenation of list1 and list2 except that their common element is not repeated.

(define (splice list1 list2)
  (cond ((eq? (last1 list1) (first list2))
         (append list1 (cdr list2)))
        (else (slib:error 'splice list1 list2))))

where cyclicsplice is the result of leaving off the last element of splice(list1,list2).

(define (cyclicsplice list1 list2)
  (cond ((and (eq? (last1 list1) (first list2))
              (eq? (first list1) (last1 list2)))
         (butlast (splice list1 list2) 1))
        (else (slib:error 'cyclicsplice list1 list2))))

\[
\begin{aligned}
  (N\times N (S\ a\ b) (S\ a\ b)) & \Rightarrow (m\ a\ b)
\end{aligned}
\]

Then register the rule for multiplying type N objects by type N objects.

\[
\begin{aligned}
  \text{(cring:define-rule }& \ast\ 'N\ 'N\ N\times N)\text{)}
\end{aligned}
\]

Now we are ready to compute!

(define (t)
  (define detM
    (+ (* (S g b)
        (+ (* (S f d)
            (- (* (S a f) (S d g)) (* a g) (S d f)))
            (* (S f f))
            (- (* (S a g) (S d d)) (* a d) (S d g)))
            (* (S f g)
                (- (* (S a d) (S d f)) (* a f) (S d d)))))
        (* (S g d)
            (+ (* (S f b)
                (- (* (S a g) (S d f)) (* a f) (S d g)))
                (* (S f f))
                (- (* (S a b) (S d g)) (* a g) (S d b)))
                (* (S f g)
                    (- (* (S a f) (S d b)) (* a b) (S d f)))))))

(* (S g f)
(+ (* (S f b)
    (- (* (S a d) (S d g)) (* (S a g) (S d d))))
  (* (S f d)
    (- (* (S a g) (S d b)) (* (S a b) (S d g))))
  (* (S f g)
    (- (* (S a b) (S d d)) (* (S a d) (S d b))))
  (* (S g g)
    (+ (* (S f b)
         (- (* (S a f) (S d d)) (* (S a d) (S d f))))
       (* (S f d)
          (- (* (S a b) (S d f)) (* (S a f) (S d b))))
       (* (S f f)
          (- (* (S a d) (S d b)) (* (S a b) (S d d))))
       (* (S b e) (S c a) (S e c) detM))))
(pretty-print (t))
-|
(- (+ (m a c e b d f g)
    (m a c e b d g f)
    (m a c e b f d g)
    (m a c e b f g d)
    (m a c e b g d f)
    (m a c e b g f d))
  (* 2 (m a b e c) (m d f g))
  (* (m a c e b d) (m f g))
  (* (m a c e b f) (m d g))
  (* (m a c e b g) (m d f)))

5.18 Matrix Algebra

(require 'determinant)

A Matrix can be either a list of lists (rows) or an array. Unlike linear-algebra texts, this package uses 0-based coordinates.

matrix->lists matrix
  Returns the list-of-lists form of matrix.

matrix->array matrix
  Returns the array form of matrix.

determinant matrix
  matrix must be a square matrix. determinant returns the determinant of matrix.
  (require 'determinant)
  (determinant '((1 2) (3 4))) \(\rightarrow\) -2
  (determinant '(((1 2 3) (4 5 6) (7 8 9))) \(\rightarrow\) 0
transpose matrix

Returns a copy of matrix flipped over the diagonal containing the 1,1 element.

matrix:sum m1 m2

Returns the element-wise sum of matrices m1 and m2.

matrix:difference m1 m2

Returns the element-wise difference of matrices m1 and m2.

matrix:product m1 m2

Returns the product of matrices m1 and m2.

matrix:product m1 z

Returns matrix m1 times scalar z.

matrix:product z m1

Returns matrix m1 times scalar z.

matrix:inverse matrix

matrix must be a square matrix. If matrix is singular, then matrix:inverse returns #f; otherwise matrix:inverse returns the matrix:product inverse of matrix.
6 Database Packages

6.1 Relational Database

(require 'relational-database)

This package implements a database system inspired by the Relational Model (E. F. Codd, A Relational Model of Data for Large Shared Data Banks). An SLIB relational database implementation can be created from any Section 6.2.1 [Base Table], page 179 implementation.


6.1.1 Using Databases

(require 'databases)

This enhancement wraps a utility layer on relational-database which provides:

- Identification of open databases by filename.
- Automatic sharing of open (immutable) databases.
- Automatic loading of base-table package when creating a database.
- Detection and automatic loading of the appropriate base-table package when opening a database.
- Table and data definition from Scheme lists.

Database Sharing

Auto-sharing refers to a call to the procedure open-database returning an already open database (procedure), rather than opening the database file a second time.

Note: Databases returned by open-database do not include wrappers applied by packages like Section 6.1.4 [Embedded Commands], page 169. But wrapped databases do work as arguments to these functions.

When a database is created, it is mutable by the creator and not auto-sharable. A database opened mutably is also not auto-sharable. But any number of readers can (open) share a non-mutable database file.

This next set of procedures mirror the whole-database methods in Section 6.2.4 [Database Operations], page 186. Except for create-database, each procedure will accept either a filename or database procedure for its first argument.

create-database filename base-table-type

filename should be a string naming a file; or #f. base-table-type must be a symbol naming a feature which can be passed to require. create-database returns a new, open relational database (with base-table type base-table-type) associated with filename, or a new ephemeral database if filename is #f.

create-database is the only run-time use of require in SLIB which crosses module boundaries. When base-table-type is required by create-database; it adds an association of base-table-type with its relational-system procedure to mdbm:*databases*.
alist-table is the default base-table type:

```
(require 'databases)
(define my-rdb (create-database "my.db" 'alist-table))
```

Only `alist-table` and base-table modules which have been `require`d will dispatch correctly from the `open-database` procedures. Therefore, either pass two arguments to `open-database`, or require the base-table of your database file uses before calling `open-database` with one argument.

```
(open-database! rdb base-table-type)  ; [Procedure]
  Returns mutable open relational database or #f.

(open-database rdb base-table-type)  ; [Function]
  Returns an open relational database associated with `rdb`. The database will be opened with base-table type `base-table-type`).

(open-database rdb)  ; [Function]
  Returns an open relational database associated with `rdb`. `open-database` will attempt to deduce the correct base-table-type.

(write-database rdb filename)  ; [Function]
  Writes the mutable relational-database `rdb` to `filename`.

(sync-database rdb)  ; [Function]
  Writes the mutable relational-database `rdb` to the filename it was opened with.

(solidify-database rdb)  ; [Function]
  Syncs `rdb` and makes it immutable.

(close-database rdb)  ; [Function]
  `rdb` will only be closed when the count of `open-database` - `close-database` calls for `rdb` (and its filename) is 0. `close-database` returns #t if successful; and #f otherwise.

(mdbm:report)  ; [Function]
  Prints a table of open database files. The columns are the base-table type, number of opens, ‘!’ for mutable, the filename, and the lock certificate (if locked).

```
(mdbm:report)
|alist-table 003 /usr/local/lib/slib/clrnamdb.scm
|alist-table 001 ! sdram.db jaffer@aubrey.jaffer.3166:1038628199|
```

**Opening Tables**

```
(open-table rdb table-name)  ; [Function]
  `rdb` must be a relational database and `table-name` a symbol.

(open-table rdb table-name) returns a "methods" procedure for an existing relational table in `rdb` if it exists and can be opened for reading, otherwise returns #f.
```
open-table! rdb table-name  

[Procedure]

open-table! rdb table-name 

rdb must be a relational database and table-name a symbol.

open-table! returns a "methods" procedure for an existing relational table in rdb if it exists and can be opened in mutable mode, otherwise returns #f.

Defining Tables

define-domains rdb row5 . . .  

[Function]

Adds the domain rows row5 . . . to the ‘domains-data’ table in rdb. The format of the row is given in Section 6.2.2 [Catalog Representation], page 184.

(define-domains rdb '(permittivity #f complex? c64 #f))

add-domain rdb row5  

[Function]

Use define-domains instead.

define-tables rdb spec-0 . . .  

[Function]

Adds tables as specified in spec-0 . . . to the open relational-database rdb. Each spec has the form:

((name) (descriptor-name) (primary-key-fields) (other-fields) (rows))

or

((name) (primary-key-fields) (other-fields) (rows))

where <name> is the table name, <descriptor-name> is the symbol name of a descriptor table, <primary-key-fields> and <other-fields> describe the primary keys and other fields respectively, and <rows> is a list of data rows to be added to the table.

<primary-key-fields> and <other-fields> are lists of field descriptors of the form:

((column-name) (domain))

or

((column-name) (domain) (column-integrity-rule))

where <column-name> is the column name, <domain> is the domain of the column, and <column-integrity-rule> is an expression whose value is a procedure of one argument (which returns #f to signal an error).

If <domain> is not a defined domain name and it matches the name of this table or an already defined (in one of spec-0 . . .) single key field table, a foreign-key domain will be created for it.

Listing Tables

list-table-definition rdb table-name  

[Function]

If symbol table-name exists in the open relational-database rdb, then returns a list of the table-name, its primary key names and domains, its other key names and domains, and the table’s records (as lists). Otherwise, returns #f.

The list returned by list-table-definition, when passed as an argument to define-tables, will recreate the table.
6.1.2 Table Operations

These are the descriptions of the methods available from an open relational table. A method is retrieved from a table by calling the table with the symbol name of the operation. For example:

$$((\text{plat } 'get 'processor) 'djgpp) \Rightarrow i386$$

Some operations described below require primary key arguments. Primary keys arguments are denoted $key1 key2 \ldots$. It is an error to call an operation for a table which takes primary key arguments with the wrong number of primary keys for that table.

get column-name

[Operation on relational-table]

Returns a procedure of arguments $key1 key2 \ldots$ which returns the value for the column-name column of the row associated with primary keys $key1, key2 \ldots$ if that row exists in the table, or $\#f$ otherwise.

$$((\text{plat } 'get 'processor) 'djgpp) \Rightarrow i386$$
$$((\text{plat } 'get 'processor) 'be-os) \Rightarrow \#f$$

6.1.2.1 Single Row Operations

The term row used below refers to a Scheme list of values (one for each column) in the order specified in the descriptor (table) for this table. Missing values appear as $\#f$. Primary keys must not be missing.

row:insert

[Operation on relational-table]

Adds the row row to this table. If a row for the primary key(s) specified by row already exists in this table an error is signaled. The value returned is unspecified.

```scheme
(define telephone-table-desc
  ((my-database 'create-table) 'telephone-table-desc))
(define ndrp (telephone-table-desc 'row:insert))
(ndrp '(1 #t name #f string))
(ndrp '(2 #f telephone
  (lambda (d)
    (and (string? d) (> (string-length d) 2)
      (every
        (lambda (c)
          (memv c '('#0 '#1 '#2 '#3 '#4 '#5 '#6 '#7 '#8 '#9
            #\+ #\- #\space #\-))))
        (string->list d))))))```

row:update

[Operation on relational-table]

Returns a procedure of one argument, row, which adds the row, row, to this table. If a row for the primary key(s) specified by row already exists in this table, it will be overwritten. The value returned is unspecified.

row:retrieve

[Operation on relational-table]

Returns a procedure of arguments $key1 key2 \ldots$ which returns the row associated with primary keys $key1, key2 \ldots$ if it exists, or $\#f$ otherwise.
((plat 'row:retrieve) 'linux) ⇒ (linux i386 linux gcc)
((plat 'row:retrieve) 'multics) ⇒ #f

row:remove  [Operation on relational-table]
Returns a procedure of arguments key1 key2 . . . which removes and returns the row associated with primary keys key1, key2 . . . if it exists, or #f otherwise.

row:delete  [Operation on relational-table]
Returns a procedure of arguments key1 key2 . . . which deletes the row associated with primary keys key1, key2 . . . if it exists. The value returned is unspecified.

6.1.2.2 Match-Keys
The (optional) match-key1 . . . arguments are used to restrict actions of a whole-table operation to a subset of that table. Those procedures (returned by methods) which accept match-key arguments will accept any number of match-key arguments between zero and the number of primary keys in the table. Any unspecified match-key arguments default to #f.

The match-key1 . . . restrict the actions of the table command to those records whose primary keys each satisfy the corresponding match-key argument. The arguments and their actions are:

#f  The false value matches any key in the corresponding position.

an object of type procedure
This procedure must take a single argument, the key in the corresponding position. Any key for which the procedure returns a non-false value is a match; Any key for which the procedure returns a #f is not.

other values
Any other value matches only those keys equal? to it.

get* column-name  [Operation on relational-table]
Returns a procedure of optional arguments match-key1 . . . which returns a list of the values for the specified column for all rows in this table. The optional match-key1 . . . arguments restrict actions to a subset of the table.

((plat 'get* 'processor)) ⇒
(i386 i8086 i386 i8086 i386 i8086 m68000 m68000 m68000 m68000 powerpc)

((plat 'get* 'processor) #f) ⇒
(i386 i8086 i386 i8086 i386 i8086 m68000 m68000 m68000 m68000 powerpc)

(define (a-key? key)
  (char=? #\a (string-ref (symbol->string key) 0)))

((plat 'get* 'processor) a-key?) ⇒
(m68000 m68000 m68000 m68000 m68000 powerpc)

((plat 'get* 'name) a-key?) ⇒
(atari-st-turbo-c atari-st-gcc amiga-sas/c-5.10
  amiga-aztec amiga-dice-c aix)

6.1.2.3 Multi-Row Operations

row:retrieve*
[Operation on relational-table]
Returns a procedure of optional arguments match-key1 . . . which returns a list of all rows in this table. The optional match-key1 . . . arguments restrict actions to a subset of the table. For details see See Section 6.1.2.2 [Match-Keys], page 166.

((plat 'row:retrieve*) a-key?) ⇒
((atari-st-turbo-c m68000 atari turbo-c)
  (atari-st-gcc m68000 atari gcc)
  (amiga-sas/c-5.10 m68000 amiga sas/c)
  (amiga-aztec m68000 amiga aztec)
  (amiga-dice-c m68000 amiga dice-c)
  (aix powerpc aix -))

row:remove*
[Operation on relational-table]
Returns a procedure of optional arguments match-key1 . . . which removes and returns a list of all rows in this table. The optional match-key1 . . . arguments restrict actions to a subset of the table.

row:delete*
[Operation on relational-table]
Returns a procedure of optional arguments match-key1 . . . which Deletes all rows from this table. The optional match-key1 . . . arguments restrict deletions to a subset of the table. The value returned is unspecified. The descriptor table and catalog entry for this table are not affected.

for-each-row
[Operation on relational-table]
Returns a procedure of arguments proc match-key1 . . . which calls proc with each row in this table. The optional match-key1 . . . arguments restrict actions to a subset of the table. For details see See Section 6.1.2.2 [Match-Keys], page 166.

Note that row:insert* and row:update* do not use match-keys.

row:insert*
[Operation on relational-table]
Returns a procedure of one argument, rows, which adds each row in the list of rows, rows, to this table. If a row for the primary key specified by an element of rows already exists in this table, an error is signaled. The value returned is unspecified.

row:update*
[Operation on relational-table]
Returns a procedure of one argument, rows, which adds each row in the list of rows, rows, to this table. If a row for the primary key specified by an element of rows already exists in this table, it will be overwritten. The value returned is unspecified.
6.1.2.4 Indexed Sequential Access Methods

Indexed Sequential Access Methods are a way of arranging database information so that records can be accessed both by key and by key sequence (ordering). ISAM is not part of Codd’s relational model.

Associative memory in B-Trees is an example of a database implementation which can support a native key ordering. SLIB’s alist-table implementation uses sort to implement for-each-row-in-order, but does not support isam-next and isam-prev.

The multi-primary-key ordering employed by these operations is the lexicographic collation of those primary-key fields in their given order. For example:

\[(12 \text{ a } 34) < (12 \text{ a } 36) < (12 \text{ b } 1) < (13 \text{ a } 0)\]

6.1.2.5 Sequential Index Operations

The following procedures are individually optional depending on the base-table implementation. If an operation is not supported, then calling the table with that operation symbol will return false.

**for-each-row-in-order**  
[Operation on relational-table]  
Returns a procedure of arguments proc match-key1 . . . which calls proc with each row in this table in the (implementation-dependent) natural, repeatable ordering for rows. The optional match-key1 . . . arguments restrict actions to a subset of the table. For details see See Section 6.1.2.2 [Match-Keys], page 166.

**isam-next**  
[Operation on relational-table]  
Returns a procedure of arguments key1 key2 . . . which returns the key-list identifying the lowest record higher than key1 key2 . . . which is stored in the relational-table; or false if no higher record is present.

**isam-next column-name**  
[Operation on relational-table]  
The symbol column-name names a key field. In the list returned by isam-next, that field, or a field to its left, will be changed. This allows one to skip over less significant key fields.

**isam-prev**  
[Operation on relational-table]  
Returns a procedure of arguments key1 key2 . . . which returns the key-list identifying the highest record less than key1 key2 . . . which is stored in the relational-table; or false if no lower record is present.

**isam-prev column-name**  
[Operation on relational-table]  
The symbol column-name names a key field. In the list returned by isam-next, that field, or a field to its left, will be changed. This allows one to skip over less significant key fields.

For example, if a table has key fields:

\[(\text{col1 col2})\]
\[(9 \text{ 5})\]
\[(9 \text{ 6})\]
\[(9 \text{ 7})\]
Then:

\[
\begin{align*}
((\text{table } '\text{isam-next} ) ' (9 \ 5)) & \Rightarrow (9 \ 6) \\
((\text{table } '\text{isam-next } \text{col2} ) ' (9 \ 5)) & \Rightarrow (9 \ 6) \\
((\text{table } '\text{isam-next } \text{col1} ) ' (9 \ 5)) & \Rightarrow (12 \ 5) \\
((\text{table } '\text{isam-prev} ) ' (12 \ 7)) & \Rightarrow (12 \ 6) \\
((\text{table } '\text{isam-prev } \text{col2} ) ' (12 \ 7)) & \Rightarrow (12 \ 6) \\
((\text{table } '\text{isam-prev } \text{col1} ) ' (12 \ 7)) & \Rightarrow (9 \ 8)
\end{align*}
\]

### 6.1.2.6 Table Administration

- **column-names**  
  [Operation on relational-table]  
  Return a list of the column names, foreign-key table names, domain names, or type names respectively for this table. These 4 methods are different from the others in that the list is returned, rather than a procedure to obtain the list.

- **primary-limit**  
  [Operation on relational-table]  
  Returns the number of primary keys fields in the relations in this table.

- **close-table**  
  [Operation on relational-table]  
  Subsequent operations to this table will signal an error.

### 6.1.3 Database Interpolation

(\texttt{require 'database-interpolate})

Indexed sequential access methods allow finding the keys (having associations) closest to a given value. This facilitates the interpolation of associations between those in the table.

\texttt{interpolate-from-table \texttt{table} column}

**Function**

\texttt{Table} should be a relational table with one numeric primary key field which supports the \texttt{isam-prev} and \texttt{isam-next} operations. \texttt{column} should be a symbol or exact positive integer designating a numerically valued column of \texttt{table}.

\texttt{interpolate-from-table} calculates and returns a value proportionally intermediate between its values in the next and previous key records contained in \texttt{table}. For keys larger than all the stored keys the value associated with the largest stored key is used. For keys smaller than all the stored keys the value associated with the smallest stored key is used.

### 6.1.4 Embedded Commands

(\texttt{require 'database-commands})

This enhancement wraps a utility layer on \texttt{relational-database} which provides:
• Automatic execution of initialization commands stored in database.
• Transparent execution of database commands stored in *commands* table in database.

When an enhanced relational-database is called with a symbol which matches a name in the *commands* table, the associated procedure expression is evaluated and applied to the enhanced relational-database. A procedure should then be returned which the user can invoke on (optional) arguments.

The command *initialize* is special. If present in the *commands* table, open-database or open-database! will return the value of the *initialize* command. Notice that arbitrary code can be run when the *initialize* procedure is automatically applied to the enhanced relational-database.

Note also that if you wish to shadow or hide from the user relational-database methods described in Section 6.2.4 [Database Operations], page 186, this can be done by a dispatch in the closure returned by the *initialize* expression rather than by entries in the *commands* table if it is desired that the underlying methods remain accessible to code in the *commands* table.

### 6.1.4.1 Database Extension

**wrap-command-interface rdb**  
Returns relational database rdb wrapped with additional commands defined in its *commands* table.

**add-command-tables rdb**  
The relational database rdb must be mutable. add-command-tables adds a *command* table to rdb; then returns (wrap-command-interface rdb).

**define-*commands* rdb spec-0 . . .**  
Adds commands to the *commands* table as specified in spec-0 . . . to the open relational-database rdb. Each spec has the form:

```lambda ( (<name> <rdb>) "comment" <expression1> <expression2> ... )```

or

```((<name> <rdb>) <expression1> <expression2> ... )```

where <name> is the command name, <rdb> is a formal passed the calling relational database, "comment" describes the command, and <expression1>, <expression1>, . . . are the body of the procedure.

**define-*commands* adds to the *commands* table a command <name>:**

```(lambda ( (<name> <rdb>) <expression1> <expression2> ... )```

**open-command-database filename**  
Returns an open enhanced relational database associated with filename. The database will be opened with base-table type base-table-type) if supplied. If base-table-type is not supplied, open-command-database will attempt to deduce the correct base-table-type. If the database can not be opened or if it lacks the *commands* table, #f is returned.
open-command-database! filename [Function]

open-command-database! filename base-table-type [Function]

Returns mutable open enhanced relational database . . .

open-command-database database [Function]

Returns database if it is an immutable relational database; #f otherwise.

open-command-database! database [Function]

Returns database if it is a mutable relational database; #f otherwise.

6.1.4.2 Command Intrinsics

Some commands are defined in all extended relational-databases. The are called just like Section 6.2.4 [Database Operations], page 186.

add-domain domain-row [Operation on relational-database]

Adds domain-row to the domains table if there is no row in the domains table associated with key (car domain-row) and returns #t. Otherwise returns #f.

For the fields and layout of the domain table, See Section 6.2.2 [Catalog Representation], page 184. Currently, these fields are

- domain-name
- foreign-table
- domain-integrity-rule
- type-id
- type-param

The following example adds 3 domains to the 'build' database. 'Optstring' is either a string or #f. filename is a string and build-whats is a symbol.

(for-each (build 'add-domain)
  '((optstring #f
    (lambda (x) (or (not x) (string? x)))
      string
      #f)
    (filename #f #f string #f)
    (build-whats #f #f symbol #f)))

delete-domain domain-name [Operation on relational-database]

Removes and returns the domain-name row from the domains table.

domain-checker domain [Operation on relational-database]

Returns a procedure to check an argument for conformance to domain domain.

6.1.4.3 Define-tables Example

The following example shows a new database with the name of 'foo.db' being created with tables describing processor families and processor/os/compiler combinations. The database is then solidified; saved and changed to immutable.

(require 'databases)
(define my-rdb (create-database "foo.db" 'alist-table))
6.1.4.4 The *commands* Table

The table *commands* in an enhanced relational-database has the fields (with domains):

- **PRI name symbol**
- **parameters** parameter-list
- **procedure** expression
- **documentation** string

The **parameters** field is a foreign key (domain parameter-list) of the *catalog-data* table and should have the value of a table described by *parameter-columns*. This parameter-list table describes the arguments suitable for passing to the associated command. The intent of this table is to be of a form such that different user-interfaces (for instance, pull-down menus or plain-text queries) can operate from the same table. A parameter-list table has the following fields:
The **arity** field can take the values:

- **single**: Requires a single parameter of the specified domain or zero parameters is acceptable.
- **optional**: A single parameter of the specified domain or zero parameters (in which case #f is substituted) is acceptable.
- **boolean**: A single boolean parameter or zero parameters (in which case #f is substituted) is acceptable.
- **nary**: Any number of parameters of the specified domain are acceptable. The argument passed to the command function is always a list of the parameters.
- **nary1**: One or more of parameters of the specified domain are acceptable. The argument passed to the command function is always a list of the parameters.

The **domain** field specifies the domain which a parameter or parameters in the **index**th field must satisfy.

The **defaulter** field is an expression whose value is either #f or a procedure of one argument (the parameter-list) which returns a list of the default value or values as appropriate. Note that since the **defaulter** procedure is called every time a default parameter is needed for this column, *sticky* defaults can be implemented using shared state with the domain-integrity-rule.

### 6.1.4.5 Command Service

**make-command-server**  

```
make-command-server  rdb  table-name
```

[Function]

Returns a procedure of 2 arguments, a (symbol) command and a call-back procedure. When this returned procedure is called, it looks up **command** in table **table-name** and calls the call-back procedure with arguments:

- **command**: The command
- **command-value**: The result of evaluating the expression in the **procedure** field of **table-name** and calling it with rdb.
- **parameter-name**: A list of the official name of each parameter. Corresponds to the **name** field of the command’s parameter-table.
- **positions**: A list of the positive integer index of each parameter. Corresponds to the **index** field of the command’s parameter-table.
- **arities**: A list of the arities of each parameter. Corresponds to the **arity** field of the command’s parameter-table. For a description of **arity** see table above.
types A list of the type name of each parameter. Corresponds to the type-id field of the contents of the domain of the command’s parameter-table.

defaulters A list of the defaulters for each parameter. Corresponds to the defaulters field of the command’s parameter-table.

domain-integrity-rules A list of procedures (one for each parameter) which tests whether a value for a parameter is acceptable for that parameter. The procedure should be called with each datum in the list for nary arity parameters.

aliases A list of lists of (alias parameter-name). There can be more than one alias per parameter-name.

For information about parameters, See Section 4.4.4 [Parameter lists], page 62.

6.1.4.6 Command Example

Here is an example of setting up a command with arguments and parsing those arguments from a getopt style argument list (see Section 4.4.1 [Getopt], page 58).

```
(define my-rdb (add-command-tables (create-database #f 'alist-table)))
(define-tables my-rdb
  '(foo-params
    *parameter-columns*
    *parameter-columns*
    ((1 single-string single string
      (lambda (pl) '("str")) #f "single string")
     (2 nary-symbols nary symbol
      (lambda (pl) '()) #f "zero or more symbols")
     (3 nary1-symbols nary1 symbol
      (lambda (pl) '(symb)) #f "one or more symbols")
     (4 optional-number optional ordinal
      (lambda (pl) '()) #f "zero or one number")
     (5 flag boolean boolean
      (lambda (pl) '(#f)) #f "a boolean flag")))
  '(foo-pnames
    ((name string))
    ((parameter-index ordinal))
    ("s" 1)
    ("single-string" 1))
```

Chapter 6: Database Packages

("n" 2)
("nary-symbols" 2)
("N" 3)
("nary1-symbols" 3)
("o" 4)
("optional-number" 4)
("f" 5)
("flag" 5))

'(my-commands
  ((name symbol))
  ((parameters parameter-list)
    (parameter-names parameter-name-translation)
    (procedure expression)
    (documentation string))
  ((foo
    foo-params
    foo-pnames
    (lambda (rdb) (lambda args (print args)))
    "test command arguments")))

(define (dbutil:serve-command-line rdb command-table command argv)
  (set! *argv* (if (vector? argv) (vector->list argv) argv))
  ((make-command-server rdb command-table)
   command
   (lambda (comname comval options positions
              arities types defaulters dirs aliases)
     (apply comval (getopt->arglist options positions
                    arities types defaulters dirs aliases)))))

(define (cmd . opts)
  (fluid-let (((*optind* 1))
               (printf "%-34s ⇒ "
               (call-with-output-string
                (lambda (pt) (write (cons 'cmd opts) pt))))
               (set! opts (cons "cmd" opts))
               (force-output)
               (dbutil:serve-command-line
                my-rdb 'my-commands 'foo (length opts) opts)))

  (cmd) ⇒ ("str" () (symb) () #f)
  (cmd "-f") ⇒ ("str" () (symb) () #t)
  (cmd "--flag") ⇒ ("str" () (symb) () #t)
  (cmd "-o177") ⇒ ("str" () (symb) (177) #f)
  (cmd "-o" "177") ⇒ ("str" () (symb) (177) #f)
  (cmd "--optional" "621") ⇒ ("str" () (symb) (621) #f)
  (cmd "--optional=621") ⇒ ("str" () (symb) (621) #f)
  (cmd "-s" "speciality") ⇒ ("speciality" () (symb) () #f)
6.1.5 Database Macros

The object-oriented programming interface to SLIB relational databases has failed to support clear, understandable, and modular code-writing for database applications.

This seems to be a failure of the object-oriented paradigm where the type of an object is not manifest (or even traceable) in source code.

within-database, along with the databases package, reorganizes high-level database functions toward a more declarative style. Using this package, one can tag database table and command declarations for emacs:

```
etags -lscheme -r'/ *(define-(command|table) \([^; \t]+/\2/)' 
  source1.scm ...
```

### 6.1.5.1 Within-database

within-database creates a lexical scope in which the commands define-table and define-command create tables and *commands*-table entries respectively in open relational database database. The expressions in 'within-database' form are executed in order.
within-database Returns database.

define-command (\(<name> \<rdb>\)) "comment" \(<expression1> \<expression2>\) [Syntax]
... 

define-command (\(<name> \<rdb>\)) \(<expression1> \<expression2>\) ... [Syntax]
Add to the *commands* table a command \(<name>:\n
    (lambda (\(<name> \<rdb>\)) \(<expression1> \<expression2>\) ...)

define-table \(<name> <descriptor-name> <descriptor-name> \<rows>\) [Syntax]
define-table \(<name> \<primary-key-fields> \<other-fields> \<rows>\) [Syntax]
where \(<name>\) is the table name, \(<descriptor-name>\) is the symbol name of a descriptor table, \(<primary-key-fields>\) and \(<other-fields>\) describe the primary keys and other fields respectively, and \(<rows>\) is a list of data rows to be added to the table.

\(<primary-key-fields>\) and \(<other-fields>\) are lists of field descriptors of the form:

    (\(<column-name> \<domain>\))

or

    (\(<column-name> \<domain> \<column-integrity-rule>\))

where \(<column-name>\) is the column name, \(<domain>\) is the domain of the column, and \(<column-integrity-rule>\) is an expression whose value is a procedure of one argument (which returns #f to signal an error).

If \(<domain>\) is not a defined domain name and it matches the name of this table or an already defined (in one of spec-0 ...) single key field table, a foreign-key domain will be created for it.

add-macro-support database [Function]
The relational database \(<database>\) must be mutable. add-macro-support adds a *macros* table and define-macro macro to \(<database>\); then \(<database>\) is returned.

define-macro (\(<name> \arg1 \ldots\)) "comment" \(<expression1> \<expression2>\) [Syntax]
... 

define-macro (\(<name> \arg1 \ldots\)) \(<expression1> \<expression2>\) ... [Syntax]
Adds a macro \(<name>\) to the *macros*.

Note: within-database creates lexical scope where not only define-command and define-table, but every command and macro are defined, ie.:

    (within-database my-rdb
        (define-command (message rdb)
            (lambda (msg)
                (display "message: ")
                (display msg)
                (newline)))
        (message "Defining FOO...")
        ;; ... defining FOO ...
        (message "Defining BAR...")
        ;; ... defining BAR ... )
)
6.1.5.2 Within-database Example

Here is an example of within-database macros:

```scheme
(require 'within-database)

(define my-rdb
  (add-command-tables
    (create-database "foo.db" 'alist-table)))

(within-database my-rdb
  (define-command (*initialize* rdb)
    "Print Welcome"
    (display "Welcome")
    (newline)
    rdb)

  (define-command (without-documentation rdb)
    (display "without-documentation called")
    (newline))

  (define-table (processor-family
      ((family atom))
      ((also-ran processor-family)))
    (m68000  #f)
    (m68030  m68000)
    (i386    i8086)
    (i8086   #f)
    (powerpc #f))

  (define-table (platform
      ((name symbol))
      ((processor processor-family)
        (os   symbol)
        (compiler symbol)))
    (aix    powerpc aix   -)
    ;; ...
    (amiga-aztec  m68000  amiga  aztec)
    (amiga-sas/c-5.10 m68000  amiga  sas/c)
    (atari-st-gcc  m68000  atari  gcc)
    ;; ...
    (watcom-9.0    i386    ms-dos watcom))

  (define-command (get-processor rdb)
    "Get processor for given platform."
    (((rdb 'open-table) 'platform #f) 'get 'processor)))

(close-database my-rdb)

(set! my-rdb (open-command-database! "foo.db"))

-|
Welcome
```
(my-rdb 'without-documentation)
-| without-documentation called

((my-rdb 'get-processor) 'amiga-sas/c-5.10)
⇒ m68000

(close-database my-rdb)

6.1.6 Database Browser

(require 'database-browse)

(browse database) [Procedure]
Prints the names of all the tables in database and sets browse’s default to database.

(browse) [Procedure]
Prints the names of all the tables in the default database.

(browse table-name) [Procedure]
For each record of the table named by the symbol table-name, prints a line composed
of all the field values.

(browse pathname) [Procedure]
Opens the database named by the string pathname, prints the names of all its tables,
and sets browse’s default to the database.

(browse database table-name) [Procedure]
Sets browse’s default to database and prints the records of the table named by the
symbol table-name.

(browse pathname table-name) [Procedure]
Opens the database named by the string pathname and sets browse’s default to it;
browse prints the records of the table named by the symbol table-name.

6.2 Relational Infrastructure

6.2.1 Base Table

A base-table is the primitive database layer upon which SLIB relational databases are
built. At the minimum, it must support the types integer, symbol, string, and boolean.
The base-table may restrict the size of integers, symbols, and strings it supports.

A base table implementation is available as the value of the identifier naming it (eg.
alist-table) after requiring the symbol of that name.

(alist-table) [Feature]

Association-list base tables support all Scheme types and are suitable for small data-
bases. In order to be retrieved after being written to a file, the data stored should
include only objects which are readable and writeable in the Scheme implementation.
The alist-table base-table implementation is included in the SLIB distribution.
WB is a B-tree database package with SCM interfaces. Being disk-based, WB databases readily store and access hundreds of megabytes of data. WB comes with two base-table embeddings.

wb-table

(require 'wb-table)

wb-table supports scheme expressions for keys and values whose text representations are less than 255 characters in length. See Section "wb-table" in WB.

rwb-isam

(require 'rwb-isam)

rwb-isam is a sophisticated base-table implementation built on WB and SCM which uses binary numerical formats for key and non-key fields. It supports IEEE floating-point and fixed-precision integer keys with the correct numerical collation order.

This rest of this section documents the interface for a base table implementation from which the Section 6.1 [Relational Database], page 162 package constructs a Relational system. It will be of interest primarily to those wishing to port or write new base-table implementations.

*base-table-implementations*

To support automatic dispatch for open-database, each base-table module adds an association to *base-table-implementations* when loaded. This association is the list of the base-table symbol and the value returned by (make-relational-system base-table).

6.2.1.1 The Base

All of these functions are accessed through a single procedure by calling that procedure with the symbol name of the operation. A procedure will be returned if that operation is supported and #f otherwise. For example:

(referw 'alist-table)
(define my-base (alist-table 'make-base))
my-base ⇒ *a procedure*
(define foo (alist-table 'foo))
foo ⇒ #f

make-base filename key-dimension column-types

[Operation on base-table]
Returns a new, open, low-level database (collection of tables) associated with filename. This returned database has an empty table associated with catalog-id. The positive integer key-dimension is the number of keys composed to make a primary-key for the catalog table. The list of symbols column-types describes the types of each column for that table. If the database cannot be created as specified, #f is returned.

Calling the close-base method on this database and possibly other operations will cause filename to be written to. If filename is #f a temporary, non-disk based database will be created if such can be supported by the base table implementation.
open-base filename mutable

Returns an open low-level database associated with filename. If mutable is #t, this database will have methods capable of effecting change to the database. If mutable is #f, only methods for inquiring the database will be available. If the database cannot be opened as specified #f is returned.

Calling the close-base (and possibly other) method on a mutable database will cause filename to be written to.

write-base lldb filename

Causes the low-level database lldb to be written to filename. If the write is successful, also causes lldb to henceforth be associated with filename. Calling the close-database (and possibly other) method on lldb may cause filename to be written to. If filename is #f this database will be changed to a temporary, non-disk based database if such can be supported by the underlying base table implementation. If the operations completed successfully, #t is returned. Otherwise, #f is returned.

close-base lldb

Causes the low-level database lldb to be written to its associated file (if any). If the write is successful, subsequent operations to lldb will signal an error. If the operations complete successfully, #t is returned. Otherwise, #f is returned.

6.2.1.2 Base Tables

make-table lldb key-dimension column-types

Returns the ordinal base-id for a new base table, otherwise returns #f. The base table can then be opened using (open-table lldb base-id). The positive integer key-dimension is the number of keys composed to make a primary-key for this table. The list of symbols column-types describes the types of each column.

open-table lldb base-id key-dimension column-types

Returns a handle for an existing base table in the low-level database lldb if that table exists and can be opened in the mode indicated by mutable, otherwise returns #f.

As with make-table, the positive integer key-dimension is the number of keys composed to make a primary-key for this table. The list of symbols column-types describes the types of each column.

kill-table lldb base-id key-dimension column-types

Returns #t if the base table associated with base-id was removed from the low level database lldb, and #f otherwise.

catalog-id

A constant base-id ordinal suitable for passing as a parameter to open-table. catalog-id will be used as the base table for the system catalog.
6.2.1.3 Base Field Types

**supported-type? symbol**  
[Operation on base-table]  
Returns `#t` if `symbol` names a type allowed as a column value by the implementa-

tion, and `#f` otherwise. At a minimum, an implementation must support the types

integer, ordinal, symbol, string, and boolean.

**supported-key-type? symbol**  
[Operation on base-table]  
Returns `#t` if `symbol` names a type allowed as a key value by the implementa-

tion, and `#f` otherwise. At a minimum, an implementation must support the types ordinal, and symbol.

An ordinal is an exact positive integer. The other types are standard Scheme.

6.2.1.4 Composite Keys

**make-keyifier-1 type**  
[Operation on base-table]  
Returns a procedure which accepts a single argument which must be of type `type`.  
This returned procedure returns an object suitable for being a key argument in the

functions whose descriptions follow.

Any 2 arguments of the supported type passed to the returned function which are

not `equal?` must result in returned values which are not `equal?`.

**make-list-keyifier key-dimension types**  
[Operation on base-table]  
The list of symbols `types` must have at least `key-dimension` elements. Returns a proce-

dure which accepts a list of length `key-dimension` and whose types must corresopond  
to the types named by `types`. This returned procedure combines the elements of its  
list argument into an object suitable for being a key argument in the functions whose  
descriptions follow.

Any 2 lists of supported types (which must at least include symbols and non-negative  
integers) passed to the returned function which are not `equal?` must result in returned  
values which are not `equal?`.

**make-key-extractor key-dimension types**  
[Operation on base-table]  
`column-number`  
Returns a procedure which accepts objects produced by application of the result of  
(make-list-keyifier `key-dimension types`). This procedure returns a key which is  
`equal?` to the `column-number`th element of the list which was passed to create  
`composite-key`. The list `types` must have at least `key-dimension` elements.

**make-key->list key-dimension types**  
[Operation on base-table]  
Returns a procedure which accepts objects produced by application of the result of  
(make-list-keyifier `key-dimension types`). This procedure returns a list of keys  
which are elementwise `equal?` to the list which was passed to create `composite-key`.

6.2.1.5 Base Record Operations

In the following functions, the key argument can always be assumed to be the value returned  
by a call to a keyify routine.
present? handle key  
[Operation on base-table]
Returns a non-#f value if there is a row associated with key in the table opened in handle and #f otherwise.

make-getter key-dimension types  
[Operation on base-table]
Returns a procedure which takes arguments handle and key. This procedure returns a list of the non-primary values of the relation (in the base table opened in handle) whose primary key is key if it exists, and #f otherwise.

make-getter-1 is a new operation. The relational-database module works with older base-table implementations by using make-getter.

make-getter-1 key-dimension types index  
[Operation on base-table]
Returns a procedure which takes arguments handle and key. This procedure returns the value of the indexth field (in the base table opened in handle) whose primary key is key if it exists, and #f otherwise.
index must be larger than key-dimension.

make-putter key-dimension types  
[Operation on base-table]
Returns a procedure which takes arguments handle and key and value-list. This procedure associates the primary key key with the values in value-list (in the base table opened in handle) and returns an unspecified value.

delete handle key  
[Operation on base-table]
Removes the row associated with key from the table opened in handle. An unspecified value is returned.

6.2.1.6 Match Keys
A match-keys argument is a list of length equal to the number of primary keys. The match-keys restrict the actions of the table command to those records whose primary keys all satisfy the corresponding element of the match-keys list. The elements and their actions are:

#f The false value matches any key in the corresponding position.

an object of type procedure
This procedure must take a single argument, the key in the corresponding position. Any key for which the procedure returns a non-false value is a match; Any key for which the procedure returns a #f is not.

other values
Any other value matches only those keys equal? to it.

6.2.1.7 Aggregate Base Operations
The key-dimension and column-types arguments are needed to decode the composite-keys for matching with match-keys.
**delete** handle key-dimension column-types match-keys  
[Operation on base-table]
Removes all rows which satisfy match-keys from the table opened in handle. An unspecified value is returned.

**for-each-key** handle procedure key-dimension  
column-types match-keys  
[Operation on base-table]
Calls procedure once with each key in the table opened in handle which satisfy match-keys in an unspecified order. An unspecified value is returned.

**map-key** handle procedure key-dimension column-types  
match-keys  
[Operation on base-table]
Returns a list of the values returned by calling procedure once with each key in the table opened in handle which satisfy match-keys in an unspecified order.

### 6.2.1.8 Base ISAM Operations

These operations are optional for a Base-Table implementation.

**ordered-for-each-key** handle procedure  
key-dimension column-types match-keys  
[Operation on base-table]
Calls procedure once with each key in the table opened in handle which satisfy match-keys in the natural order for the types of the primary key fields of that table. An unspecified value is returned.

**make-nexer** handle key-dimension column-types index  
[Operation on base-table]
Returns a procedure of arguments key1 key2 ... which returns the key-list identifying the lowest record higher than key1 key2 ... which is stored in the base-table and which differs in column index or a lower indexed key; or false if no higher record is present.

**make-prever** handle key-dimension column-types index  
[Operation on base-table]
Returns a procedure of arguments key1 key2 ... which returns the key-list identifying the highest record less than key1 key2 ... which is stored in the base-table and which differs in column index or a lower indexed key; or false if no higher record is present.

### 6.2.2 Catalog Representation

Each database (in an implementation) has a system catalog which describes all the user accessible tables in that database (including itself).

The system catalog base table has the following fields. PRI indicates a primary key for that table.

**PRI** table-name  
column-limit the highest column number  
coltab-name descriptor table name  
bastab-id data base table identifier  
user-integrity-rule  
view-procedure A scheme thunk which, when called, produces a handle for the view. coltab and bastab are specified if and only if view-procedure is not.
Descriptors for base tables (not views) are tables (pointed to by system catalog). Descriptor (base) tables have the fields:

- **PRI column-number**: sequential integers from 1
- **primary-key?**: boolean TRUE for primary key components
- **column-name**
- **column-integrity-rule**
- **domain-name**

A primary key is any column marked as **primary-key?** in the corresponding descriptor table. All the **primary-key?** columns must have lower column numbers than any non-**primary-key?** columns. Every table must have at least one primary key. Primary keys must be sufficient to distinguish all rows from each other in the table. All of the system defined tables have a single primary key.

A **domain** is a category describing the allowable values to occur in a column. It is described by a (base) table with the fields:

- **PRI domain-name**
- **foreign-table**
- **domain-integrity-rule**
- **type-id**
- **type-param**

The **type-id** field value is a symbol. This symbol may be used by the underlying base table implementation in storing that field.

If the **foreign-table** field is non-**#f** then that field names a table from the catalog. The values for that domain must match a primary key of the table referenced by the **type-param** (or **#f**, if allowed). This package currently does not support composite foreign-keys.

The types for which support is planned are:

- **atom**
- **symbol**
- **string** [length]
- **number** [base]
- **money** <currency>
- **date-time**
- **boolean**

- **foreign-key** <table-name>
- **expression**
- **virtual** <expression>

### 6.2.3 Relational Database Objects

This object-oriented interface is deprecated for typical database applications; Section 6.1.1 [Using Databases], page 162 provides an application programmer interface which is easier to understand and use.
**make-relational-system**  
*base-table-implementation*  
[Function]  
Returns a procedure implementing a relational database using the *base-table-implementation*.

All of the operations of a base table implementation are accessed through a procedure defined by requiring that implementation. Similarly, all of the operations of the relational database implementation are accessed through the procedure returned by **make-relational-system**. For instance, a new relational database could be created from the procedure returned by **make-relational-system** by:

```
(define relational-alist-system
  (make-relational-system alist-table))
(define create-alist-database
  (relational-alist-system 'create-database))
(define my-database
  (create-alist-database "mydata.db"))
```

What follows are the descriptions of the methods available from relational system returned by a call to **make-relational-system**.

**create-database**  
*filename*  
[Operation on relational-system]  
Returns an open, nearly empty relational database associated with *filename*. The only tables defined are the system catalog and domain table. Calling the **close-database** method on this database and possibly other operations will cause *filename* to be written to. If *filename* is #f a temporary, non-disk based database will be created if such can be supported by the underlying base table implementation. If the database cannot be created as specified #f is returned. For the fields and layout of descriptor tables, Section 6.2.2 [Catalog Representation], page 184

**open-database**  
*filename*  
[Operation on relational-system]  
Returns an open relational database associated with *filename*. If *mutable?* is #t, this database will have methods capable of effecting change to the database. If *mutable?* is #f, only methods for inquiring the database will be available. Calling the **close-database** (and possibly other) method on a *mutable?* database will cause *filename* to be written to. If the database cannot be opened as specified #f is returned.

### 6.2.4 Database Operations

This object-oriented interface is deprecated for typical database applications; Section 6.1.1 [Using Databases], page 162 provides an application programmer interface which is easier to understand and use.

These are the descriptions of the methods available from an open relational database. A method is retrieved from a database by calling the database with the symbol name of the operation. For example:

```
(define my-database
  (create-alist-database "mydata.db"))
(define telephone-table-desc
  ((my-database 'create-table) 'telephone-table-desc))
```
close-database  
Causes the relational database to be written to its associated file (if any). If the write is successful, subsequent operations to this database will signal an error. If the operations completed successfully, \#t is returned. Otherwise, \#f is returned.

write-database filename  
Causes the relational database to be written to filename. If the write is successful, also causes the database to henceforth be associated with filename. Calling the close-database (and possibly other) method on this database will cause filename to be written to. If filename is \#f this database will be changed to a temporary, non-disk based database if such can be supported by the underlying base table implementation. If the operations completed successfully, \#t is returned. Otherwise, \#f is returned.

sync-database  
Causes any pending updates to the database file to be written out. If the operations completed successfully, \#t is returned. Otherwise, \#f is returned.

solidify-database  
Causes any pending updates to the database file to be written out. If the writes completed successfully, then the database is changed to be immutable and \#t is returned. Otherwise, \#f is returned.

table-exists? table-name  
Returns \#t if table-name exists in the system catalog, otherwise returns \#f.

open-table table-name mutable?  
Returns a methods procedure for an existing relational table in this database if it exists and can be opened in the mode indicated by mutable?, otherwise returns \#f.

These methods will be present only in mutable databases.

delete-table table-name  
Removes and returns the table-name row from the system catalog if the table or view associated with table-name gets removed from the database, and \#f otherwise.

create-table table-desc-name  
Returns a methods procedure for a new (open) relational table for describing the columns of a new base table in this database, otherwise returns \#f. For the fields and layout of descriptor tables, See Section 6.2.2 [Catalog Representation], page 184.

create-table table-name table-desc-name  
Returns a methods procedure for a new (open) relational table with columns as described by table-desc-name, otherwise returns \#f.

create-view ??  
project-table ??  
restrict-table ??  
cart-prod-tables ??  
Not yet implemented.
6.3 Weight-Balanced Trees

(require 'wt-tree)

Balanced binary trees are a useful data structure for maintaining large sets of ordered objects or sets of associations whose keys are ordered. MIT Scheme has an comprehensive implementation of weight-balanced binary trees which has several advantages over the other data structures for large aggregates:

- In addition to the usual element-level operations like insertion, deletion and lookup, there is a full complement of collection-level operations, like set intersection, set union and subset test, all of which are implemented with good orders of growth in time and space. This makes weight balanced trees ideal for rapid prototyping of functionally derived specifications.

- An element in a tree may be indexed by its position under the ordering of the keys, and the ordinal position of an element may be determined, both with reasonable efficiency.

- Operations to find and remove minimum element make weight balanced trees simple to use for priority queues.

- The implementation is functional rather than imperative. This means that operations like ‘inserting’ an association in a tree do not destroy the old tree, in much the same way that (+ 1 x) modifies neither the constant 1 nor the value bound to x. The trees are referentially transparent thus the programmer need not worry about copying the trees. Referential transparency allows space efficiency to be achieved by sharing subtrees.

These features make weight-balanced trees suitable for a wide range of applications, especially those that require large numbers of sets or discrete maps. Applications that have a few global databases and/or concentrate on element-level operations like insertion and lookup are probably better off using hash-tables or red-black trees.

The size of a tree is the number of associations that it contains. Weight balanced binary trees are balanced to keep the sizes of the subtrees of each node within a constant factor of each other. This ensures logarithmic times for single-path operations (like lookup and insertion). A weight balanced tree takes space that is proportional to the number of associations in the tree. For the current implementation, the constant of proportionality is six words per association.

Weight balanced trees can be used as an implementation for either discrete sets or discrete maps (associations). Sets are implemented by ignoring the datum that is associated with the key. Under this scheme if an associations exists in the tree this indicates that the key of the association is a member of the set. Typically a value such as (), #t or #f is associated with the key.

Many operations can be viewed as computing a result that, depending on whether the tree arguments are thought of as sets or maps, is known by two different names. An example is wt-tree/member?, which, when regarding the tree argument as a set, computes the set membership operation, but, when regarding the tree as a discrete map, wt-tree/member? is the predicate testing if the map is defined at an element in its domain. Most names in this package have been chosen based on interpreting the trees as sets, hence the name wt-tree/member? rather than wt-tree/defined-at?. 
Chapter 6: Database Packages

The weight balanced tree implementation is a run-time-loadable option. To use weight balanced trees, execute

(load-option 'wt-tree)

once before calling any of the procedures defined here.

6.3.1 Construction of Weight-Balanced Trees

Binary trees require there to be a total order on the keys used to arrange the elements in the tree. Weight balanced trees are organized by types, where the type is an object encapsulating the ordering relation. Creating a tree is a two-stage process. First a tree type must be created from the predicate which gives the ordering. The tree type is then used for making trees, either empty or singleton trees or trees from other aggregate structures like association lists. Once created, a tree ‘knows’ its type and the type is used to test compatibility between trees in operations taking two trees. Usually a small number of tree types are created at the beginning of a program and used many times throughout the program’s execution.

make-wt-tree-type key<?

This procedure creates and returns a new tree type based on the ordering predicate key<?. Key<? must be a total ordering, having the property that for all key values a, b and c:

(key<? a a) ⇒ #f
(and (key<? a b) (key<? b a)) ⇒ #f
(if (and (key<? a b) (key<? b c)) (key<? a c) #t) ⇒ #t

Two key values are assumed to be equal if neither is less than the other by key<?.

Each call to make-wt-tree-type returns a distinct value, and trees are only compatible if their tree types are eq?. A consequence is that trees that are intended to be used in binary tree operations must all be created with a tree type originating from the same call to make-wt-tree-type.

number-wt-type

A standard tree type for trees with numeric keys. Number-wt-type could have been defined by

(define number-wt-type (make-wt-tree-type <))

string-wt-type

A standard tree type for trees with string keys. String-wt-type could have been defined by

(define string-wt-type (make-wt-tree-type string<?))

make-wt-tree wt-tree-type

This procedure creates and returns a newly allocated weight balanced tree. The tree is empty, i.e. it contains no associations. Wt-tree-type is a weight balanced tree type obtained by calling make-wt-tree-type; the returned tree has this type.
**singleton-wt-tree** *wt-tree-type key datum*  
This procedure creates and returns a newly allocated weight balanced tree. The tree contains a single association, that of *datum* with *key*. *Wt-tree-type* is a weight balanced tree type obtained by calling `make-wt-tree-type`; the returned tree has this type.

**alist->wt-tree** *tree-type alist*  
Returns a newly allocated weight-balanced tree that contains the same associations as *alist*. This procedure is equivalent to:

```scheme
(lambda (type alist)
  (let ((tree (make-wt-tree type)))
    (for-each (lambda (association)
      (wt-tree/add! tree
        (car association)
        (cdr association)))
      alist)
    tree))
```

### 6.3.2 Basic Operations on Weight-Balanced Trees

This section describes the basic tree operations on weight balanced trees. These operations are the usual tree operations for insertion, deletion and lookup, some predicates and a procedure for determining the number of associations in a tree.

**wt-tree/empty?** *wt-tree*  
Returns `#t` if *wt-tree* contains no associations, otherwise returns `#f`.

**wt-tree/size** *wt-tree*  
Returns the number of associations in *wt-tree*, an exact non-negative integer. This operation takes constant time.

**wt-tree/add** *wt-tree key datum*  
Returns a new tree containing all the associations in *wt-tree* and the association of *datum* with *key*. If *wt-tree* already had an association for *key*, the new association overrides the old. The average and worst-case times required by this operation are proportional to the logarithm of the number of associations in *wt-tree*.

**wt-tree/add!** *wt-tree key datum*  
Associates *datum* with *key* in *wt-tree* and returns an unspecified value. If *wt-tree* already has an association for *key*, that association is replaced. The average and worst-case times required by this operation are proportional to the logarithm of the number of associations in *wt-tree*.

**wt-tree/member?** *key wt-tree*  
Returns `#t` if *wt-tree* contains an association for *key*, otherwise returns `#f`. The average and worst-case times required by this operation are proportional to the logarithm of the number of associations in *wt-tree*. 
**wt-tree/lookup** `wt-tree key default`  
Returns the datum associated with `key` in `wt-tree`. If `wt-tree` doesn’t contain an association for `key`, `default` is returned. The average and worst-case times required by this operation are proportional to the logarithm of the number of associations in `wt-tree`.

**wt-tree/delete** `wt-tree key`  
Returns a new tree containing all the associations in `wt-tree`, except that if `wt-tree` contains an association for `key`, it is removed from the result. The average and worst-case times required by this operation are proportional to the logarithm of the number of associations in `wt-tree`.

**wt-tree/delete!** `wt-tree key`  
If `wt-tree` contains an association for `key` the association is removed. Returns an unspecified value. The average and worst-case times required by this operation are proportional to the logarithm of the number of associations in `wt-tree`.

### 6.3.3 Advanced Operations on Weight-Balanced Trees

In the following the `size` of a tree is the number of associations that the tree contains, and a `smaller` tree contains fewer associations.

**wt-tree/split<** `wt-tree bound`  
Returns a new tree containing all and only the associations in `wt-tree` which have a key that is less than `bound` in the ordering relation of the tree type of `wt-tree`. The average and worst-case times required by this operation are proportional to the logarithm of the size of `wt-tree`.

**wt-tree/split>** `wt-tree bound`  
Returns a new tree containing all and only the associations in `wt-tree` which have a key that is greater than `bound` in the ordering relation of the tree type of `wt-tree`. The average and worst-case times required by this operation are proportional to the logarithm of size of `wt-tree`.

**wt-tree/union** `wt-tree-1 wt-tree-2`  
Returns a new tree containing all the associations from both trees. This operation is asymmetric: when both trees have an association for the same key, the returned tree associates the datum from `wt-tree-2` with the key. Thus if the trees are viewed as discrete maps then `wt-tree/union` computes the map override of `wt-tree-1` by `wt-tree-2`. If the trees are viewed as sets the result is the set union of the arguments. The worst-case time required by this operation is proportional to the sum of the sizes of both trees. If the minimum key of one tree is greater than the maximum key of the other tree then the time required is at worst proportional to the logarithm of the size of the larger tree.

**wt-tree/intersection** `wt-tree-1 wt-tree-2`  
Returns a new tree containing all and only those associations from `wt-tree-1` which have keys appearing as the key of an association in `wt-tree-2`. Thus the associated data in the result are those from `wt-tree-1`. If the trees are being used as sets the
result is the set intersection of the arguments. As a discrete map operation, \texttt{wt-tree/intersection} computes the domain restriction of \texttt{wt-tree-1} to (the domain of) \texttt{wt-tree-2}. The time required by this operation is never worse that proportional to the sum of the sizes of the trees.

\texttt{wt-tree/difference \texttt{wt-tree-1 \texttt{wt-tree-2}}} \hspace{1cm} [\text{procedure}^*]

Returns a new tree containing all and only those associations from \texttt{wt-tree-1} which have keys that \textit{do not} appear as the key of an association in \texttt{wt-tree-2}. If the trees are viewed as sets the result is the asymmetric set difference of the arguments. As a discrete map operation, it computes the domain restriction of \texttt{wt-tree-1} to the complement of (the domain of) \texttt{wt-tree-2}. The time required by this operation is never worse that proportional to the sum of the sizes of the trees.

\texttt{wt-tree/subset? \texttt{wt-tree-1 \texttt{wt-tree-2}}} \hspace{1cm} [\text{procedure}^*]

Returns \texttt{#t} iff the key of each association in \texttt{wt-tree-1} is the key of some association in \texttt{wt-tree-2}, otherwise returns \texttt{#f}. Viewed as a set operation, \texttt{wt-tree/subset?} is the improper subset predicate. A proper subset predicate can be constructed:

\begin{verbatim}
(define (proper-subset? s1 s2)
  (and (wt-tree/subset? s1 s2)
       (< (wt-tree/size s1) (wt-tree/size s2))))
\end{verbatim}

As a discrete map operation, \texttt{wt-tree/subset?} is the subset test on the domain(s) of the map(s). In the worst-case the time required by this operation is proportional to the size of \texttt{wt-tree-1}.

\texttt{wt-tree/set-equal? \texttt{wt-tree-1 \texttt{wt-tree-2}}} \hspace{1cm} [\text{procedure}^*]

Returns \texttt{#t} iff for every association in \texttt{wt-tree-1} there is an association in \texttt{wt-tree-2} that has the same key, and \textit{vice versa}.

Viewing the arguments as sets \texttt{wt-tree/set-equal?} is the set equality predicate. As a map operation it determines if two maps are defined on the same domain.

This procedure is equivalent to

\begin{verbatim}
(lambda (wt-tree-1 wt-tree-2)
  (and (wt-tree/subset? wt-tree-1 wt-tree-2
        (wt-tree/subset? wt-tree-2 wt-tree-1)))
\end{verbatim}

In the worst-case the time required by this operation is proportional to the size of the smaller tree.

\texttt{wt-tree/fold \texttt{combiner initial \texttt{wt-tree}}} \hspace{1cm} [\text{procedure}^*]

This procedure reduces \texttt{wt-tree} by combining all the associations, using an reverse in-order traversal, so the associations are visited in reverse order. \texttt{Combiner} is a procedure of three arguments: a key, a datum and the accumulated result so far. Provided \texttt{combiner} takes time bounded by a constant, \texttt{wt-tree/fold} takes time proportional to the size of \texttt{wt-tree}.

A sorted association list can be derived simply:

\begin{verbatim}
(wt-tree/fold (lambda (key datum list)
   (cons (cons key datum) list))
\end{verbatim}
The data in the associations can be summed like this:

\[
\text{wt-tree/fold} \quad \text{(lambda (key datum sum) (+ sum datum))} \\
0 \\
\text{wt-tree)
}\]

**wt-tree/for-each action wt-tree**

This procedure traverses the tree in-order, applying `action` to each association. The associations are processed in increasing order of their keys. `Action` is a procedure of two arguments which take the key and datum respectively of the association. Provided `action` takes time bounded by a constant, `wt-tree/for-each` takes time proportional to in the size of `wt-tree`. The example prints the tree:

\[
\text{(wt-tree/for-each (lambda (key value)
    (display (list key value)))
  wt-tree))
\]

**wt-tree/union-merge wt-tree-1 wt-tree-2 merge**

Returns a new tree containing all the associations from both trees. If both trees have an association for the same key, the datum associated with that key in the result tree is computed by applying the procedure `merge` to the key, the value from `wt-tree-1` and the value from `wt-tree-2`. `Merge` is of the form

\[
\text{(merge (key datum-1 datum-2) ...)
}\]

If some key occurs only in one tree, that association will appear in the result tree without being processed by `merge`, so for this operation to make sense, either `merge` must have both a right and left identity that correspond to the association being absent in one of the trees, or some guarantee must be made, for example, all the keys in one tree are known to occur in the other.

These are all reasonable procedures for `merge`

\[
\text{(merge (key val1 val2) (+ val1 val2))}
\]
\[
\text{(merge (key val1 val2) (append val1 val2))}
\]
\[
\text{(merge (key val1 val2) (wt-tree/union val1 val2))}
\]

However, a procedure like

\[
\text{(merge (key val1 val2) (- val1 val2))}
\]

would result in a subtraction of the data for all associations with keys occuring in both trees but associations with keys occuring in only the second tree would be copied, not negated, as is presumably be intent. The programmer might ensure that this never happens.

This procedure has the same time behavior as `wt-tree/union` but with a slightly worse constant factor. Indeed, `wt-tree/union` might have been defined like this:

\[
\text{(define (wt-tree/union tree1 tree2)
  (wt-tree/union-merge tree1 tree2
    (lambda (key val1 val2) val2)))
}\]
The merge procedure takes the key as a parameter in case the data are not independent of the key.

### 6.3.4 Indexing Operations on Weight-Balanced Trees

Weight balanced trees support operations that view the tree as sorted sequence of associations. Elements of the sequence can be accessed by position, and the position of an element in the sequence can be determined, both in logarithmic time.

```scheme
(wt-tree/index wt-tree index)  [procedure+]
(wt-tree/index-datum wt-tree index)  [procedure+]
(wt-tree/index-pair wt-tree index)  [procedure+]
```

Returns the 0-based indexth association of wt-tree in the sorted sequence under the tree’s ordering relation on the keys. `wt-tree/index` returns the indexth key; `wt-tree/index-datum` returns the datum associated with the indexth key and `wt-tree/index-pair` returns a new pair `(key . datum)` which is the cons of the indexth key and its datum. The average and worst-case times required by this operation are proportional to the logarithm of the number of associations in the tree.

These operations signal an error if the tree is empty, if index<0, or if index is greater than or equal to the number of associations in the tree.

Indexing can be used to find the median and maximum keys in the tree as follows:

```scheme
median: (wt-tree/index wt-tree (quotient (wt-tree/size wt-tree) 2))
maximum: (wt-tree/index wt-tree (-1+ (wt-tree/size wt-tree)))
```

```scheme
(wt-tree/rank wt-tree key)  [procedure+]
```

Determines the 0-based position of key in the sorted sequence of the keys under the tree’s ordering relation, or #f if the tree has no association with for key. This procedure returns either an exact non-negative integer or #f. The average and worst-case times required by this operation are proportional to the logarithm of the number of associations in the tree.

```scheme
(wt-tree/min wt-tree)  [procedure+]
(wt-tree/min-datum wt-tree)  [procedure+]
(wt-tree/min-pair wt-tree)  [procedure+]
```

Returns the association of wt-tree that has the least key under the tree’s ordering relation. `wt-tree/min` returns the least key, `wt-tree/min-datum` returns the datum associated with the least key and `wt-tree/min-pair` returns a new pair `(key . datum)` which is the cons of the minimum key and its datum. The average and worst-case times required by this operation are proportional to the logarithm of the number of associations in the tree.

These operations signal an error if the tree is empty. They could be written

```scheme
(define (wt-tree/min tree) (wt-tree/index tree 0))
(define (wt-tree/min-datum tree) (wt-tree/index-datum tree 0))
(define (wt-tree/min-pair tree) (wt-tree/index-pair tree 0))
```
**wt-tree/delete-min**  
wt-tree  
[procedure+]  
Returns a new tree containing all of the associations in `wt-tree` except the association with the least key under the `wt-tree`’s ordering relation. An error is signalled if the tree is empty. The average and worst-case times required by this operation are proportional to the logarithm of the number of associations in the tree. This operation is equivalent to

\[
(wt-tree/delete wt-tree (wt-tree/min wt-tree))
\]

**wt-tree/delete-min!**  
wt-tree  
[procedure+]  
Removes the association with the least key under the `wt-tree`’s ordering relation. An error is signalled if the tree is empty. The average and worst-case times required by this operation are proportional to the logarithm of the number of associations in the tree. This operation is equivalent to

\[
(wt-tree/delete! wt-tree (wt-tree/min wt-tree))
\]
7 Other Packages

7.1 Data Structures

7.1.1 Arrays

(require 'array) or (require 'srfi-63)

array? obj
[Function]

Returns #t if the obj is an array, and #f if not.

Note: Arrays are not disjoint from other Scheme types. Vectors and possibly strings also satisfy array?. A disjoint array predicate can be written:

(define (strict-array? obj)
  (and (array? obj) (not (string? obj)) (not (vector? obj))))

equal? obj1 obj2
[Function]

Returns #t if obj1 and obj2 have the same rank and dimensions and the corresponding elements of obj1 and obj2 are equal?.

equal? recursively compares the contents of pairs, vectors, strings, and arrays, applying eqv? on other objects such as numbers and symbols. A rule of thumb is that objects are generally equal? if they print the same. equal? may fail to terminate if its arguments are circular data structures.

(array-rank obj)
[Function]

Returns the number of dimensions of obj. If obj is not an array, 0 is returned.

(array-dimensions array)
[Function]

Returns a list of dimensions.

(array-dimensions (make-array '() 3 5))
⇒ (3 5)
**make-array**  
Prototype: k1 . . .  
Function  
Creates and returns an array of type prototype with dimensions k1, . . . and filled with elements from prototype. prototype must be an array, vector, or string. The implementation-dependent type of the returned array will be the same as the type of prototype; except if that would be a vector or string with rank not equal to one, in which case some variety of array will be returned.

If the prototype has no elements, then the initial contents of the returned array are unspecified. Otherwise, the returned array will be filled with the element at the origin of prototype.

**create-array**  
Prototype: k1 . . .  
Function  
create-array is an alias for make-array.

**make-shared-array**  
Prototype: array mapper k1 . . .  
Function  
make-shared-array can be used to create shared subarrays of other arrays. The mapper is a function that translates coordinates in the new array into coordinates in the old array. A mapper must be linear, and its range must stay within the bounds of the old array, but it can be otherwise arbitrary. A simple example:

```scheme
(define fred (make-array '#(#f) 8 8))
(define freds-diagonal
  (make-shared-array fred (lambda (i) (list i i)) 8))
(array-set! freds-diagonal 'foo 3)
(array-ref fred 3 3)  
⇒ F00
(define freds-center
  (make-shared-array fred (lambda (i j) (list (+ 3 i) (+ 3 j)))
   2 2))
(array-ref freds-center 0 0)  
⇒ F00
```

**list->array**  
Prototype: rank proto list  
Function  
list->array returns an array of rank rank and type proto consisting of all the elements, in row-major order, of list. When rank is 0, list is the lone array element; not necessarily a list.

```scheme
(list->array 2 '#() '((1 2) (3 4)))  
⇒ #2A((1 2) (3 4))
(list->array 0 '#() 3)  
⇒ #0A 3
```

**array->list**  
Prototype: array  
Function  
array->list returns the single element. In the case of a rank-0 array, array->list returns the single element.

```scheme
(array->list #2A((ho ho ho) (ho oh oh)))  
⇒ ((ho ho ho) (ho oh oh))
```
(array->list #0A ho)
  ⇒ ho

vector->array vect proto dim1 . . . [Function]
  vect must be a vector of length equal to the product of exact nonnegative integers
  dim1, . . .

vector->array returns an array of type proto consisting of all the elements, in row-
  major order, of vect. In the case of a rank-0 array, vect has a single element.
  (vector->array #(1 2 3 4) #() 2 2)
  ⇒ #2A((1 2) (3 4))
  (vector->array '#(3) '#())
  ⇒ #0A 3

array->vector array [Function]
  Returns a new vector consisting of all the elements of array in row-major order.
  (array->vector #2A ((1 2) ( 3 4)))
  ⇒ #(1 2 3 4)
  (array->vector #0A ho)
  ⇒ #(ho)

array-in-bounds? array index1 . . . [Function]
  Returns #t if its arguments would be acceptable to array-ref.

array-ref array k1 . . . [Function]
  Returns the (k1, . . .) element of array.

array-set! array obj k1 . . . [Procedure]
  Stores obj in the (k1, . . .) element of array. The value returned by array-set! is
  unspecified.

These functions return a prototypical uniform-array enclosing the optional argument (which
  must be of the correct type). If the uniform-array type is supported by the implementation,
  then it is returned; defaulting to the next larger precision type; resorting finally to vector.

A:floC128b z [Function]
  Returns an inexact 128.bit flonum complex uniform-array prototype.

A:floC128b [Function]

A:floC64b z [Function]
  Returns an inexact 64.bit flonum complex uniform-array prototype.

A:floC64b [Function]

A:floC32b z [Function]
  Returns an inexact 32.bit flonum complex uniform-array prototype.

A:floC32b [Function]

A:floC16b z [Function]
  Returns an inexact 16.bit flonum complex uniform-array prototype.

A:floC16b [Function]
Chapter 7: Other Packages

A:floR128b x  
A:floR128b  
Returns an inexact 128.bit flonum real uniform-array prototype.

A:floR64b x  
A:floR64b  
Returns an inexact 64.bit flonum real uniform-array prototype.

A:floR32b x  
A:floR32b  
Returns an inexact 32.bit flonum real uniform-array prototype.

A:floR16b x  
A:floR16b  
Returns an inexact 16.bit flonum real uniform-array prototype.

A:floR128d q  
A:floR128d  
Returns an exact 128.bit decimal flonum rational uniform-array prototype.

A:floR64d q  
A:floR64d  
Returns an exact 64.bit decimal flonum rational uniform-array prototype.

A:floR32d q  
A:floR32d  
Returns an exact 32.bit decimal flonum rational uniform-array prototype.

A:fixZ64b n  
A:fixZ64b  
Returns an exact binary fixnum uniform-array prototype with at least 64 bits of precision.

A:fixZ32b n  
A:fixZ32b  
Returns an exact binary fixnum uniform-array prototype with at least 32 bits of precision.

A:fixZ16b n  
A:fixZ16b  
Returns an exact binary fixnum uniform-array prototype with at least 16 bits of precision.

A:fixZ8b n  
A:fixZ8b  
Returns an exact binary fixnum uniform-array prototype with at least 8 bits of precision.
A:fixN64b k  
A:fixN64b k

Returns an exact non-negative binary fixnum uniform-array prototype with at least 64 bits of precision.

A:fixN32b k  
A:fixN32b k

Returns an exact non-negative binary fixnum uniform-array prototype with at least 32 bits of precision.

A:fixN16b k  
A:fixN16b k

Returns an exact non-negative binary fixnum uniform-array prototype with at least 16 bits of precision.

A:fixN8b k  
A:fixN8b k

Returns an exact non-negative binary fixnum uniform-array prototype with at least 8 bits of precision.

A:bool bool  
A:bool bool

Returns a boolean uniform-array prototype.

### 7.1.2 Subarrays

(require 'subarray)

subarray array select ...  

selects a subset of an array. For 0 ≤ j < n, selectj is either an integer, a list of two integers within the range for the jth index, or #f.

When selectj is a list of two integers, then the jth index is restricted to that subrange in the returned array.

When selectj is #f, then the full range of the jth index is accessible in the returned array. An elided argument is equivalent to #f.

When selectj is an integer, then the rank of the returned array is less than array, and only elements whose jth index equals selectj are shared.

```scheme
> (define ra '#2A((a b c) (d e f)))
#<unspecified>
> (subarray ra 0 #f)
#1A(a b c)
> (subarray ra 1 #f)
#1A(d e f)
> (subarray ra #f 1)
#1A(b e)
> (subarray ra '(0 1) #f)
#2A((a b c) (d e f))
> (subarray ra #f '(0 1))
```
#2A((a b) (d e))
> (subarray ra #f '(1 2))
#2A((b c) (e f))
> (subarray ra #f '(2 1))
#2A((c b) (f e))

Arrays can be reflected (reversed) using subarray:

> (subarray '1A(a b c d e) '(4 0))
#1A(e d c b a)

\textbf{array-trim} array trim . . .

[Function]
Returns a subarray sharing contents with array except for slices removed from either side of each dimension. Each of the \textit{trims} is an exact integer indicating how much to trim. A positive \textit{s} trims the data from the lower end and reduces the upper bound of the result; a negative \textit{s} trims from the upper end and increases the lower bound.

For example:

(array-trim '#(0 1 2 3 4) 1) \Rightarrow #1A(1 2 3 4)
(array-trim '#(0 1 2 3 4) -1) \Rightarrow #1A(0 1 2 3)

(requires 'array-for-each)
(define (centered-difference ra)
  (array-map ra - (array-trim ra 1) (array-trim ra -1)))

(centered-difference '#(0 1 3 5 9 22))
\Rightarrow #(1 2 2 4 13)

\textbf{7.1.3 Array Mapping}

(requires 'array-for-each)

\textbf{array-map!} array0 proc array1 . . .

[Procedure]
array1, . . . must have the same number of dimensions as array0 and have a range for each index which includes the range for the corresponding index in array0. proc is applied to each tuple of elements of array1 . . . and the result is stored as the corresponding element in array0. The value returned is unspecified. The order of application is unspecified.

\textbf{array-map} prototype proc array1 array2 . . .

[Function]
array2, . . . must have the same number of dimensions as array1 and have a range for each index which includes the range for the corresponding index in array1. proc is applied to each tuple of elements of array1, array2, . . . and the result is stored as the corresponding element in a new array of type prototype. The new array is returned. The order of application is unspecified.

\textbf{array-for-each} proc array0 . . .

[Function]
proc is applied to each tuple of elements of array0 . . . in row-major order. The value returned is unspecified.
array-indexes array  [Function]
Returns an array of lists of indexes for array such that, if li is a list of indexes for which array is defined, (equal? li (apply array-ref (array-indexes array) li)).

array-index-for-each array proc  [Function]
applies proc to the indices of each element of array in turn. The value returned and the order of application are unspecified.

One can implement array-index-map! as

(define (array-index-map! ra fun)
  (array-index-for-each
   ra
   (lambda is (apply array-set! ra (apply fun is) is))))

array-index-map! array proc  [Procedure]
applies proc to the indices of each element of array in turn, storing the result in the corresponding element. The value returned and the order of application are unspecified.

One can implement array-indexes as

(define (array-indexes array)
  (let ((ra (apply make-array '#() (array-dimensions array))))
    (array-index-map! ra (lambda x x))
    ra))

Another example:

(define (apl:index-generator n)
  (let ((v (make-vector n 1)))
    (array-index-map! v (lambda (i) i))
    v))

array:copy! destination source  [Procedure]
copies every element from vector or array source to the corresponding element of destination. destination must have the same rank as source, and be at least as large in each dimension. The order of copying is unspecified.

7.1.4 Array Interpolation

(require 'array-interpolate)

interpolate-array-ref ra x1 ... xj  [Function]
ra must be an array of rank j containing numbers. interpolate-array-ref returns a value interpolated from the nearest j-dimensional cube of elements of ra.

(interpolate-array-ref '#2A:fixZ32b((1 2 3) (4 5 6)) 1 0.1)
  ==> 4.1

(interpolate-array-ref '#2A:fixZ32b((1 2 3) (4 5 6)) 0.5 0.25)
  ==> 2.75
**resample-array!**  
*ra1 ra2*  
**[Procedure]**  
ra1 and ra2 must be numeric arrays of equal rank. resample-array! sets ra1 to values interpolated from ra2 such that the values of elements at the corners of ra1 and ra2 are equal.

```
(define ra (make-array (A:fixZ32b) 2 2))
(resample-array! ra '#2A:fixZ32b((1 2 3) (4 5 6)))
ra ==> #2A:fixZ32b((1 3) (4 6))
```

```
(define ra (make-array (A:floR64b) 3 2))
(resample-array! ra '#2A:fixZ32b((1 2 3) (4 5 6)))
ra ==> #2A:floR64b((1.0 3.0) (2.5 4.5) (4.0 6.0))
```

### 7.1.5 Association Lists

(require 'alist)

Alist functions provide utilities for treating a list of key-value pairs as an associative database. These functions take an equality predicate, *pred*, as an argument. This predicate should be repeatable, symmetric, and transitive.

Alist functions can be used with a secondary index method such as hash tables for improved performance.

**predicate->asso**  
*pred*  
**[Function]**  
Returns an association function (like assq, assv, or assoc) corresponding to *pred*. The returned function returns a key-value pair whose key is *pred*-equal to its first argument or #f if no key in the alist is *pred*-equal to the first argument.

**alist-inquirer**  
*pred*  
**[Function]**  
Returns a procedure of 2 arguments, *alist* and *key*, which returns the value associated with *key* in *alist* or #f if *key* does not appear in *alist*.

**alist-associator**  
*pred*  
**[Function]**  
Returns a procedure of 3 arguments, *alist*, *key*, and *value*, which returns an alist with *key* and *value* associated. Any previous value associated with *key* will be lost. This returned procedure may or may not have side effects on its *alist* argument. An example of correct usage is:

```
(define put (alist-associator string-ci=?))
(define alist '())
(set! alist (put alist "Foo" 9))
```

**alist-remover**  
*pred*  
**[Function]**  
Returns a procedure of 2 arguments, *alist* and *key*, which returns an alist with an association whose *key* is key removed. This returned procedure may or may not have side effects on its *alist* argument. An example of correct usage is:

```
(define rem (alist-remover string-ci=?))
(set! alist (rem alist "foo"))
```

**alist-map**  
*proc*  
*alist*  
**[Function]**  
Returns a new association list formed by mapping *proc* over the keys and values of *alist*. *proc* must be a function of 2 arguments which returns the new value part.
alist-for-each proc alist  
Applies proc to each pair of keys and values of alist. proc must be a function of 2 arguments. The returned value is unspecified.

7.1.6 Byte

(require 'byte)

Some algorithms are expressed in terms of arrays of small integers. Using Scheme strings to implement these arrays is not portable vis-a-vis the correspondence between integers and characters and non-ascii character sets. These functions abstract the notion of a byte.

byte-ref bytes k  
k must be a valid index of bytes. byte-ref returns byte k of bytes using zero-origin indexing.

byte-set! bytes k byte  
k must be a valid index of bytes, and byte must be a small nonnegative integer. byte-set! stores byte in element k of bytes and returns an unspecified value.

make-bytes k byte  
make-bytes returns a newly allocated byte-array of length k. If byte is given, then all elements of the byte-array are initialized to byte, otherwise the contents of the byte-array are unspecified.

bytes-length bytes  
bytes-length returns length of byte-array bytes.

bytes byte ...  
Returns a newly allocated byte-array composed of the small nonnegative arguments.

list->bytes bytes  
list->bytes returns a newly allocated byte-array formed from the small nonnegative integers in the list bytes.

bytes->list bytes  
bytes->list returns a newly allocated list of the bytes that make up the given byte-array.

Bytes->list and list->bytes are inverses so far as equal? is concerned.

bytes->string bytes  
Returns a new string formed from applying integer->char to each byte in bytes->string. Note that this may signal an error for bytes having values between 128 and 255.

string->bytes string  
Returns a new byte-array formed from applying char->integer to each character in string->bytes. Note that this may signal an error if an integer is larger than 255.
Chapter 7: Other Packages

bytes-copy bytes

Returns a newly allocated copy of the given bytes.

subbytes bytes start end

bytes must be a bytes, and start and end must be exact integers satisfying
0 <= start <= end <= (bytes-length bytes).

subbytes returns a newly allocated bytes formed from the bytes of bytes beginning
with index start (inclusive) and ending with index end (exclusive).

bytes-reverse! bytes

Reverses the order of byte-array bytes.

bytes-reverse bytes

Returns a newly allocated bytes-array consisting of the elements of bytes in reverse
order.

Input and output of bytes should be with ports opened in binary mode (see Section 2.3
[Input/Output], page 13). Calling open-file with 'rb or 'wb modes argument will return
a binary port if the Scheme implementation supports it.

write-byte byte port

write-byte byte

Writes the byte byte (not an external representation of the byte) to the given port
and returns an unspecified value. The port argument may be omitted, in which case
it defaults to the value returned by current-output-port.

read-byte port

read-byte

Returns the next byte available from the input port, updating the port to point to
the following byte. If no more bytes are available, an end-of-file object is returned.
port may be omitted, in which case it defaults to the value returned by current-
input-port.

When reading and writing binary numbers with read-bytes and write-bytes, the sign
of the length argument determines the endianness (order) of bytes. Positive treats them
as big-endian, the first byte input or output is highest order. Negative treats them as
little-endian, the first byte input or output is the lowest order.

Once read in, SLIB treats byte sequences as big-endian. The multi-byte sequences produced
and used by number conversion routines see Section 7.1.7 [Byte/Number Conversions],
page 206 are always big-endian.

read-bytes n port

read-bytes n

read-bytes returns a newly allocated bytes-array filled with (abs n) bytes read from
port. If n is positive, then the first byte read is stored at index 0; otherwise the last
byte read is stored at index 0. Note that the length of the returned byte-array will
be less than (abs n) if port reaches end-of-file.

port may be omitted, in which case it defaults to the value returned by current-
input-port.
write-bytes bytes n port  

write-bytes writes \((\text{abs } n)\) bytes to output-port \(port\). If \(n\) is positive, then the first byte written is index 0 of \(bytes\); otherwise the last byte written is index 0 of \(bytes\). write-bytes returns an unspecified value.

\(port\) may be omitted, in which case it defaults to the value returned by current-output-port.

subbytes-read! and subbytes-write provide lower-level procedures for reading and writing blocks of bytes. The relative size of \(start\) and \(end\) determines the order of writing.

subbytes-read! bts start end port  

subbytes-read! bts start end  

Fills \(bts\) with up to \((\text{abs } (\text{-start end}))\) bytes read from \(port\). The first byte read is stored at index \(bts\). subbytes-read! returns the number of bytes read.

\(port\) may be omitted, in which case it defaults to the value returned by current-input-port.

subbytes-write bts start end port  

subbytes-write bts start end  

subbytes-write writes \((\text{abs } (\text{-start end}))\) bytes to output-port \(port\). The first byte written is index \(start\) of \(bts\). subbytes-write returns the number of bytes written.

\(port\) may be omitted, in which case it defaults to the value returned by current-output-port.

7.1.7 Byte/Number Conversions

(require 'byte-number)

The multi-byte sequences produced and used by numeric conversion routines are always big-endian. Endianness can be changed during reading and writing bytes using read-bytes and write-bytes See Section 7.1.6 [Byte], page 204.

The sign of the length argument to bytes/integer conversion procedures determines the signedness of the number.

bytes->integer bytes n  

Converts the first \((\text{abs } n)\) bytes of big-endian \(bytes\) array to an integer. If \(n\) is negative then the integer coded by the bytes are treated as two's-complement (can be negative).

\[
\begin{align*}
\text{(bytes->integer (bytes 0 0 0 15) -4)} & \Rightarrow 15 \\
\text{(bytes->integer (bytes 0 0 0 15) 4)} & \Rightarrow 15 \\
\text{(bytes->integer (bytes 255 255 255 255) -4)} & \Rightarrow -1 \\
\text{(bytes->integer (bytes 255 255 255 255) 4)} & \Rightarrow 4294967295 \\
\text{(bytes->integer (bytes 128 0 0 0) -4)} & \Rightarrow -2147483648 \\
\text{(bytes->integer (bytes 128 0 0 0) 4)} & \Rightarrow 2147483648
\end{align*}
\]
integer->bytes n len

Converts the integer \( n \) to a byte-array of \( \text{abs}(n) \) bytes. If \( n \) and \( len \) are both negative, then the bytes in the returned array are coded two's-complement.

\[
\begin{align*}
\text{(bytes->list (integer->bytes 15 -4))} & \Rightarrow (0 \ 0 \ 0 \ 15) \\
\text{(bytes->list (integer->bytes 15 4))} & \Rightarrow (0 \ 0 \ 0 \ 15) \\
\text{(bytes->list (integer->bytes -1 -4))} & \Rightarrow (255 \ 255 \ 255 \ 255) \\
\text{(bytes->list (integer->bytes 4294967295 4))} & \Rightarrow (255 \ 255 \ 255 \ 255) \\
\text{(bytes->list (integer->bytes -2147483648 -4))} & \Rightarrow (128 \ 0 \ 0 \ 0) \\
\text{(bytes->list (integer->bytes 2147483648 4))} & \Rightarrow (128 \ 0 \ 0 \ 0)
\end{align*}
\]

bytes->ieee-float bytes

\( bytes \) must be a 4-element byte-array. bytes->ieee-float calculates and returns the value of \( bytes \) interpreted as a big-endian IEEE 4-byte (32-bit) number.

\[
\begin{align*}
\text{(bytes->ieee-float (bytes 0 0 0 0))} & \Rightarrow 0.0 \\
\text{(bytes->ieee-float (bytes #x80 0 0 0))} & \Rightarrow -0.0 \\
\text{(bytes->ieee-float (bytes #x40 0 0 0))} & \Rightarrow 2.0 \\
\text{(bytes->ieee-float (bytes #x40 #xd0 0 0))} & \Rightarrow 6.5 \\
\text{(bytes->ieee-float (bytes #xc0 #xd0 0 0))} & \Rightarrow -6.5 \\
\text{(bytes->ieee-float (bytes 0 #x80 0 0))} & \Rightarrow 11.754943508222875e-39 \\
\text{(bytes->ieee-float (bytes 0 #x40 0 0))} & \Rightarrow 5.87747175411437e-39 \\
\text{(bytes->ieee-float (bytes 0 0 0 1))} & \Rightarrow 1.401298464324817e-45 \\
\text{(bytes->ieee-float (bytes #xff #x80 0 0))} & \Rightarrow -inf.0 \\
\text{(bytes->ieee-float (bytes #x7f #x80 0 0))} & \Rightarrow +inf.0 \\
\text{(bytes->ieee-float (bytes #x7f #x80 0 1))} & \Rightarrow 0/0 \\
\text{(bytes->ieee-float (bytes #x7f #xc0 0 0))} & \Rightarrow 0/0 
\end{align*}
\]

bytes->ieee-double bytes

\( bytes \) must be a 8-element byte-array. bytes->ieee-double calculates and returns the value of \( bytes \) interpreted as a big-endian IEEE 8-byte (64-bit) number.

\[
\begin{align*}
\text{(bytes->ieee-double (bytes 0 0 0 0 0 0 0 0))} & \Rightarrow 0.0 \\
\text{(bytes->ieee-double (bytes #x80 0 0 0 0 0 0 0))} & \Rightarrow -0.0 \\
\text{(bytes->ieee-double (bytes #x40 0 0 0 0 0 0 0))} & \Rightarrow 2.0 \\
\text{(bytes->ieee-double (bytes #x40 #x1A 0 0 0 0 0 0))} & \Rightarrow 6.5 \\
\text{(bytes->ieee-double (bytes #xc0 #x1A 0 0 0 0 0 0))} & \Rightarrow -6.5 \\
\text{(bytes->ieee-double (bytes 0 8 0 0 0 0 0 0))} & \Rightarrow 11.125369292536006e-309 \\
\text{(bytes->ieee-double (bytes 0 4 0 0 0 0 0 0))} & \Rightarrow 5.5628464626803e-309 \\
\text{(bytes->ieee-double (bytes 0 0 0 0 0 0 0 1))} & \Rightarrow 4.0e-324 \\
\text{(bytes->ieee-double (list->bytes '(127 239 255 255 255 255 255 255)))} & \Rightarrow 179.7693134862319
\end{align*}
\]
**ieee-float->bytes** x

Returns a 4-element byte-array encoding the IEEE single-precision floating-point of x.

\[
\begin{align*}
\text{bytes->list (ieee-float->bytes 0.0))} & \Rightarrow (0 \ 0 \ 0 \ 0) \\
\text{bytes->list (ieee-float->bytes -0.0))} & \Rightarrow (128 \ 0 \ 0 \ 0) \\
\text{bytes->list (ieee-float->bytes 2.0))} & \Rightarrow (64 \ 0 \ 0 \ 0) \\
\text{bytes->list (ieee-float->bytes 6.5))} & \Rightarrow (64 \ 208 \ 0 \ 0) \\
\text{bytes->list (ieee-float->bytes -6.5))} & \Rightarrow (192 \ 208 \ 0 \ 0)
\end{align*}
\]

\[
\begin{align*}
\text{bytes->list (ieee-float->bytes 11.754943508222875e-39))} & \Rightarrow (0 \ 128 \ 0 \ 0) \\
\text{bytes->list (ieee-float->bytes 5.877471754111438e-39))} & \Rightarrow (0 \ 64 \ 0 \ 0) \\
\text{bytes->list (ieee-float->bytes 1.401298464324817e-45))} & \Rightarrow (0 \ 0 \ 0 \ 1)
\end{align*}
\]

\[
\begin{align*}
\text{bytes->list (ieee-float->bytes -inf.0))} & \Rightarrow (255 \ 128 \ 0 \ 0) \\
\text{bytes->list (ieee-float->bytes +inf.0))} & \Rightarrow (127 \ 128 \ 0 \ 0) \\
\text{bytes->list (ieee-float->bytes 0/0))} & \Rightarrow (127 \ 192 \ 0 \ 0)
\end{align*}
\]

**ieee-double->bytes** x

Returns a 8-element byte-array encoding the IEEE double-precision floating-point of x.

\[
\begin{align*}
\text{bytes->list (ieee-double->bytes 0.0))} & \Rightarrow (0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0) \\
\text{bytes->list (ieee-double->bytes -0.0))} & \Rightarrow (128 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0) \\
\text{bytes->list (ieee-double->bytes 2.0))} & \Rightarrow (64 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0) \\
\text{bytes->list (ieee-double->bytes 6.5))} & \Rightarrow (64 \ 26 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0) \\
\text{bytes->list (ieee-double->bytes -6.5))} & \Rightarrow (192 \ 26 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0)
\end{align*}
\]

\[
\begin{align*}
\text{bytes->list (ieee-double->bytes 11.125369292536006e-309))} & \Rightarrow (0 \ 8 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0) \\
\text{bytes->list (ieee-double->bytes 5.562684646268003e-309))} & \Rightarrow (0 \ 4 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0) \\
\text{bytes->list (ieee-double->bytes 4.0e-324))} & \Rightarrow (0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 1)
\end{align*}
\]

\[
\begin{align*}
\text{bytes->list (ieee-double->bytes -inf.0))} & \Rightarrow (255 \ 240 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0) \\
\text{bytes->list (ieee-double->bytes +inf.0))} & \Rightarrow (127 \ 240 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0) \\
\text{bytes->list (ieee-double->bytes 0/0))} & \Rightarrow (127 \ 248 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0)
\end{align*}
\]

**Byte Collation Order**

The `string<? ordering of big-endian byte-array representations of fixed and IEEE floating-point numbers agrees with the numerical ordering only when those numbers are non-negative.

Straightforward modification of these formats can extend the byte-collating order to work for their entire ranges. This agreement enables the full range of numbers as keys in indexed-sequential-access-method databases.
integer-byte-collate! byte-vector  [Procedure]
Modifies sign bit of byte-vector so that string<? ordering of two’s-complement byte-vectors matches numerical order. integer-byte-collate! returns byte-vector and is its own functional inverse.

integer-byte-collate byte-vector  [Function]
Returns copy of byte-vector with sign bit modified so that string<? ordering of two’s-complement byte-vectors matches numerical order. integer-byte-collate is its own functional inverse.

ieee-byte-collate! byte-vector  [Procedure]
Modifies byte-vector so that string<? ordering of IEEE floating-point byte-vectors matches numerical order. ieee-byte-collate! returns byte-vector.

ieee-byte-decollate! byte-vector  [Procedure]
Given byte-vector modified by ieee-byte-collate!, reverses the byte-vector modifications.

ieee-byte-collate byte-vector  [Function]
Returns copy of byte-vector encoded so that string<? ordering of IEEE floating-point byte-vectors matches numerical order.

ieee-byte-decollate byte-vector  [Function]
Given byte-vector returned by ieee-byte-collate, reverses the byte-vector modifications.

7.1.8 MAT-File Format

(receive 'matfile)

This package reads MAT-File Format version 4 (MATLAB) binary data files. MAT-files written from big-endian or little-endian computers having IEEE format numbers are currently supported. Support for files written from VAX or Cray machines could also be added.

The numeric and text matrix types handled; support for sparse matrices awaits a sample file.

matfile:read filename  [Function]
filename should be a string naming an existing file containing a MATLAB Version 4 MAT-File. The matfile:read procedure reads matrices from the file and returns a list of the results; a list of the name string and array for each matrix.

matfile:load filename  [Function]
filename should be a string naming an existing file containing a MATLAB Version 4 MAT-File. The matfile:load procedure reads matrices from the file and defines the string-ci->symbol for each matrix to its corresponding array. matfile:load returns a list of the symbols defined.
7.1.9 Portable Image Files

(require 'pnm)

Function

\texttt{pnm}:type-dimensions \texttt{path}

The string \texttt{path} must name a \textit{portable bitmap graphics} file. \texttt{pnm}:type-dimensions returns a list of 4 items:
1. A symbol describing the type of the file named by \texttt{path}.
2. The image width in pixels.
3. The image height in pixels.
4. The maximum value of pixels assume in the file.

The current set of file-type symbols is:

\begin{itemize}
  \item \texttt{pbm} \texttt{pbm-raw} Black-and-White image; pixel values are 0 or 1.
  \item \texttt{pgm} \texttt{pgm-raw} Gray (monochrome) image; pixel values are from 0 to \texttt{maxval} specified in file header.
  \item \texttt{ppm} \texttt{ppm-raw} RGB (full color) image; red, green, and blue interleaved pixel values are from 0 to \texttt{maxval}
\end{itemize}

Function

\texttt{pnm}:image-file->array \texttt{path} \texttt{array}

Reads the \textit{portable bitmap graphics} file named by \texttt{path} into \texttt{array}. \texttt{array} must be the correct size and type for \texttt{path}. \texttt{array} is returned.

Function

\texttt{pnm}:image-file->array \texttt{path}

\texttt{pnm}:image-file->array creates and returns an array with the \textit{portable bitmap graphics} file named by \texttt{path} read into it.

Function

\texttt{pnm}:array-write \texttt{type} \texttt{array} \texttt{maxval} \texttt{path} \texttt{comment} \ldots

Writes the contents of \texttt{array} to a \texttt{type} image file named \texttt{path}. The file will have pixel values between 0 and \texttt{maxval}, which must be compatible with \texttt{type}. For ‘\texttt{pbm}’ files, \texttt{maxval} must be ‘1’. comments are included in the file header.

7.1.10 Collections

(require 'collect)

Routines for managing collections. Collections are aggregate data structures supporting iteration over their elements, similar to the Dylan(TM) language, but with a different interface. They have elements indexed by corresponding keys, although the keys may be implicit (as with lists).

New types of collections may be defined as YASOS objects (see Section 3.14 [Yasos], page 35). They must support the following operations:

- \texttt{(collection? \texttt{self})} (always returns \texttt{#t});
- \texttt{(size \texttt{self})} returns the number of elements in the collection;
• (print self port) is a specialized print operation for the collection which prints a suitable representation on the given port or returns it as a string if port is #t;

• (gen-elts self) returns a thunk which on successive invocations yields elements of self in order or gives an error if it is invoked more than (size self) times;

• (gen-keys self) is like gen-elts, but yields the collection’s keys in order.

They might support specialized for-each-key and for-each-elt operations.

collection? obj

A predicate, true initially of lists, vectors and strings. New sorts of collections must answer #t to collection?.

map-elts proc collection1 ...

do-elts proc collection1 ...

proc is a procedure taking as many arguments as there are collections (at least one). The collections are iterated over in their natural order and proc is applied to the elements yielded by each iteration in turn. The order in which the arguments are supplied corresponds to the order in which the collections appear. do-elts is used when only side-effects of proc are of interest and its return value is unspecified. map-elts returns a collection (actually a vector) of the results of the applications of proc.

Example:

(map-elts + (list 1 2 3) (vector 1 2 3))
⇒ #(2 4 6)

map-keys proc collection1 ...

do-keys proc collection1 ...

These are analogous to map-elts and do-elts, but each iteration is over the collections’ keys rather than their elements.

Example:

(map-keys + (list 1 2 3) (vector 1 2 3))
⇒ #(0 2 4)

for-each-key collection proc

for-each-elt collection proc

These are like do-keys and do-elts but only for a single collection; they are potentially more efficient.

reduce proc seed collection1 ...

A generalization of the list-based reduce-init (see Section 7.2.1.3 [Lists as sequences], page 226) to collections.

Examples:

(reduce + 0 (vector 1 2 3))
⇒ 6

(reduce union '() '((a b c) (b c d) (d a)))
⇒ (c b d a).

Reduce called with two arguments will work as does the procedure of the same name from See Section 7.2.1 [Common List Functions], page 221.
any?  pred collection1 . . .  [Function]
A generalization of the list-based some (see Section 7.2.1.3 [Lists as sequences], page 226) to collections.
Example:

(any? odd? (list 2 3 4 5))  
⇒ #t

every?  pred collection1 . . .  [Function]
A generalization of the list-based every (see Section 7.2.1.3 [Lists as sequences], page 226) to collections.
Example:

(every? collection? '((1 2) #(1 2)))  
⇒ #t

empty?  collection  [Function]
Returns #t iff there are no elements in collection.

(empty? collection)  ≡  (zero? (size collection))

size  collection  [Function]
Returns the number of elements in collection.

setter list-ref  [Function]
See Section 3.14.3 [Setters], page 36 for a definition of setter. N.B. (setter list-ref) doesn’t work properly for element 0 of a list.

Here is a sample collection: simple-table which is also a table.

(define-predicate TABLE?)
(define-operation (LOOKUP table key failure-object))
(define-operation (ASSOCIATE! table key value))  ;; returns key
(define-operation (REMOVE! table key))  ;; returns value

(define (MAKE-SIMPLE-TABLE)
  (let ((table (list))))
    (object
      ;; table behaviors
      ((TABLE? self) #t)
      ((SIZE self) (size table))
      ((PRINT self port) (format port "#<SIMPLE-TABLE>"))
      ((LOOKUP self key failure-object)
        (cond
          ((assq key table) => cdr)
          (else failure-object)
        ))
      ((ASSOCIATE! self key value)
        (cond
          ((assq key table)
          )))
)
=> (lambda (bucket) (set-cdr! bucket value) key))
(else
 (set! table (cons (cons key value) table))
 key))

((REMOVE! self key);; returns old value
 (cond
 ((null? table) (slib:error "TABLE:REMOVE! Key not found: " key))
 ((eq? key (caar table))
  (let ((value (cdar table)))
   (set! table (cdr table))
   value))
 (else
  (let loop ((last table) (this (cdr table)))
   (cond
    ((null? this)
     (slib:error "TABLE:REMOVE! Key not found: " key))
    ((eq? key (caar this))
     (let ((value (cdar this)))
      (set-cdr! last (cdr this))
      value))
    (else
     (loop (cdr last) (cdr this))))
   )))

));; collection behaviors
 ((COLLECTION? self) #t)
 ((GEN-KEYS self) (collect:list-gen-elts (map car table)))
 ((GEN-ELTS self) (collect:list-gen-elts (map cdr table)))
 ((FOR-EACH-KEY self proc)
  (for-each (lambda (bucket) (proc (car bucket))) table))
 ((FOR-EACH-ELT self proc)
  (for-each (lambda (bucket) (proc (cdr bucket))) table))

7.1.11 Dynamic Data Type

(require 'dynamic)

(make-dynamic obj)  [Function]
Create and returns a new dynamic whose global value is obj.

dynamic? obj  [Function]
Returns true if and only if obj is a dynamic. No object satisfying dynamic? satisfies
any of the other standard type predicates.
Chapter 7: Other Packages

**dynamic-ref dyn**  
[Function]  
Return the value of the given dynamic in the current dynamic environment.

**dynamic-set! dyn obj**  
[Procedure]  
Change the value of the given dynamic to `obj` in the current dynamic environment. The returned value is unspecified.

**call-with-dynamic-binding dyn obj thunk**  
[Function]  
Invoke and return the value of the given thunk in a new, nested dynamic environment in which the given dynamic has been bound to a new location whose initial contents are the value `obj`. This dynamic environment has precisely the same extent as the invocation of the thunk and is thus captured by continuations created within that invocation and re-established by those continuations when they are invoked.

The **dynamic-bind** macro is not implemented.

### 7.1.12 Hash Tables

```lisp
(require 'hash-table)
```

**predicate->hash pred**  
[Function]  
Returns a hash function (like `hashq`, `hashv`, or `hash`) corresponding to the equality predicate `pred`. `pred` should be `eq?`, `eqv?`, `equal?`, `=`, `char=?`, `char-ci=?`, `string=?`, or `string-ci=?`.

A hash table is a vector of association lists.

**make-hash-table k**  
[Function]  
Returns a vector of `k` empty (association) lists.

Hash table functions provide utilities for an associative database. These functions take an equality predicate, `pred`, as an argument. `pred` should be `eq?`, `eqv?`, `equal?`, `=`, `char=?`, `char-ci=?`, `string=?`, or `string-ci=?`.

**predicate->hash-asso pred**  
[Function]  
Returns a hash association function of 2 arguments, `key` and `hashtab`, corresponding to `pred`. The returned function returns a key-value pair whose key is `pred`-equal to its first argument or `#f` if no key in `hashtab` is `pred`-equal to the first argument.

**hash-inquirer pred**  
[Function]  
Returns a procedure of 2 arguments, `hashtab` and `key`, which returns the value associated with `key` in `hashtab` or `#f` if `key` does not appear in `hashtab`.

**hash-associator pred**  
[Function]  
Returns a procedure of 3 arguments, `hashtab`, `key`, and `value`, which modifies `hashtab` so that `key` and `value` associated. Any previous value associated with `key` will be lost.

**hash-remover pred**  
[Function]  
Returns a procedure of 2 arguments, `hashtab` and `key`, which modifies `hashtab` so that the association whose key is `key` is removed.
hash-map proc hash-table  
Returns a new hash table formed by mapping proc over the keys and values of hash-table. proc must be a function of 2 arguments which returns the new value part.

hash-for-each proc hash-table  
Applies proc to each pair of keys and values of hash-table. proc must be a function of 2 arguments. The returned value is unspecified.

hash-rehasher pred  
hash-rehasher accepts a hash table predicate and returns a function of two arguments hashtab and new-k which is specialized for that predicate.
This function is used for nondestructively resizing a hash table. hashtab should be an existing hash-table using pred, new-k is the size of a new hash table to be returned. The new hash table will have all of the associations of the old hash table.

7.1.13 Macroless Object System
(require 'object)

This is the Macroless Object System written by Wade Humeniuk (whumeniu@datap.ca). Conceptual Tributes: Section 3.14 [Yasos], page 35, MacScheme’s %object, CLOS, Lack of R4RS macros.

7.1.14 Concepts

OBJECT  An object is an ordered association-list (by eq?) of methods (procedures). Methods can be added (make-method!), deleted (unmake-method!) and retrieved (get-method). Objects may inherit methods from other objects. The object binds to the environment it was created in, allowing closures to be used to hide private procedures and data.

GENERIC-METHOD  
A generic-method associates (in terms of eq?) object’s method. This allows scheme function style to be used for objects. The calling scheme for using a generic method is (generic-method object param1 param2 ...).

METHOD  A method is a procedure that exists in the object. To use a method get-method must be called to look-up the method. Generic methods implement the get-method functionality. Methods may be added to an object associated with any scheme obj in terms of eq?

GENERIC-PREDICATE  
A generic method that returns a boolean value for any scheme obj.

PREDICATE  
A object’s method associated with a generic-predicate. Returns #t.

7.1.15 Procedures

make-object ancestor ...  
Returns an object. Current object implementation is a tagged vector. ancestors are optional and must be objects in terms of object?. ancestors methods are included
in the object. Multiple ancestors might associate the same generic-method with a method. In this case the method of the ancestor first appearing in the list is the one returned by get-method.

**object? obj**
Returns boolean value whether obj was created by make-object.

**make-generic-method exception-procedure**
Returns a procedure which be associated with an object’s methods. If exception-procedure is specified then it is used to process non-objects.

**make-generic-predicate**
Returns a boolean procedure for any scheme object.

**make-method! object generic-method method**
Associates method to the generic-method in the object. The method overrides any previous association with the generic-method within the object. Using unmake-method! will restore the object’s previous association with the generic-method. method must be a procedure.

**make-predicate! object generic-predicate**
Makes a predicate method associated with the generic-predicate.

**unmake-method! object generic-method**
Removes an object’s association with a generic-method.

**get-method object generic-method**
Returns the object’s method associated (if any) with the generic-method. If no associated method exists an error is flagged.

### 7.1.16 Examples

```scheme
(define instantiate (make-generic-method))

(define (make-instance-object . ancestors)
  (define self (apply make-object
    (map (lambda (obj) (instantiate obj)) ancestors)))
  (make-method! self instantiate (lambda (self) self))
  self)

(define who (make-generic-method))
(define imigrate! (make-generic-method))
(define emigrate! (make-generic-method))
(define describe (make-generic-method))
(define name (make-generic-method))
(define address (make-generic-method))
(define members (make-generic-method))
```
(define society
  (let ()
    (define self (make-instance-object))
    (define population '())
    (make-method! self imigrate!
      (lambda (new-person)
        (if (not (eq? new-person self))
          (set! population (cons new-person population))))))
  (make-method! self emigrate!
    (lambda (person)
      (if (not (eq? person self))
        (set! population
          (comlist:remove-if (lambda (member)
                          (eq? member person))
          population)))))
  (make-method! self describe
    (lambda (self)
      (map (lambda (person) (describe person)) population)))
  (make-method! self who
    (lambda (self) (map (lambda (person) (name person)) population)))
  (make-method! self members (lambda (self) population))
  self))

(define (make-person %name %address)
  (define self (make-instance-object society))
  (make-method! self name (lambda (self) %name))
  (make-method! self address (lambda (self) %address))
  (make-method! self who (lambda (self) (name self)))
  (make-method! self instantiate
    (lambda (self)
      (make-person (string-append (name self) "-son-of")
        %address)))
  (make-method! self describe
    (lambda (self) (list (name self) (address self))))
  (imigrate! self)
  self)

7.1.16.1 Inverter Documentation

Inheritance:

<inverter>::(<number> <description>)

Generic-methods

<inverter>::value ⇒ <number>::value
<inverter>::set-value! ⇒ <number>::set-value!
<inverter>::describe ⇒ <description>::describe
<inverter>::help
<inverter>::invert
<inverter>::inverter?

7.1.16.2 Number Documentation
Inheritance
<number>::()
Slots
<number>::<x>
Generic Methods
<number>::value
<number>::set-value!

7.1.16.3 Inverter code
(define invert (make-generic-method
  (lambda (val)
    (if (number? val)
      (/ 1 val)
      (error "Method not supported:" val))))
(define noop (make-generic-method))
(define inverter? (make-generic-predicate))
(define describe (make-generic-method))
(define help (make-generic-method))

(define (make-number x)
  (define self (make-object))
  (make-method! self value (lambda (this) x))
  (make-method! self set-value!
    (lambda (this new-value) (set! x new-value)))
  self)

(define (make-description str)
  (define self (make-object))
  (make-method! self describe (lambda (this) str))
  (make-method! self help (lambda (this) "Help not available"))
  self)

(define (make-inverter)
  (let* ((self (make-object
    (make-number 1)
    (make-description "A number which can be inverted")))
(\(<\text{value}\>\) (get-method self value)))
(make-method! self invert (lambda (self) (/ 1 (\(<\text{value}\>\) self))))
(make-predicate! self inverter?)
(unmake-method! self help)
(make-method! self help
  (lambda (self)
    (display "Inverter Methods:") (newline)
    (display " (value inverter) ==> n") (newline)))

self)))

;;; Try it out
(define invert! (make-generic-method))

(define x (make-inverter))

(make-method! x invert! (lambda (x) (set-value! x (/ 1 (value x)))))

(value x) ⇒ 1
(set-value! x 33) ⇒ undefined
(invert! x) ⇒ undefined
(value x) ⇒ 1/33

(unmake-method! x invert!) ⇒ undefined

(invert! x) ⇒ error ERROR: Method not supported: x

7.1.17 Priority Queues
(require 'priority-queue)

This algorithm for priority queues is due to *Introduction to Algorithms* by T. Cormen, C. Leiserson, R. Rivest. 1989 MIT Press.

make-heap pred<? [Function]
Returns a binary heap suitable which can be used for priority queue operations.

heap-length heap [Function]
Returns the number of elements in heap.

heap-insert! heap item [Procedure]
Inserts item into heap. item can be inserted multiple times. The value returned is unspecified.

heap-extract-max! heap [Procedure]
Returns the item which is larger than all others according to the pred<? argument to make-heap. If there are no items in heap, an error is signaled.

7.1.18 Queues
(require 'queue)
A *queue* is a list where elements can be added to both the front and rear, and removed from the front (i.e., they are what are often called *dequeues*). A queue may also be used like a stack.

**make-queue**

Returns a new, empty queue.

**queue? obj**

Returns #t if obj is a queue.

**queue-empty? q**

Returns #t if the queue q is empty.

**queue-push! q datum**

Adds datum to the front of queue q.

**enqueue! q datum**

Adds datum to the rear of queue q.

**dequeue! q**

**queue-pop! q**

Both of these procedures remove and return the datum at the front of the queue. **queue-pop!** is used to suggest that the queue is being used like a stack.

All of the following functions raise an error if the queue q is empty.

**dequeue-all! q**

Removes and returns (the list) of all contents of queue q.

**queue-front q**

Returns the datum at the front of the queue q.

**queue-rear q**

Returns the datum at the rear of the queue q.

### 7.1.19 Records

(requires 'record)

The Record package provides a facility for user to define their own record data types.

**make-record-type type-name field-names**

Returns a *record-type descriptor*, a value representing a new data type disjoint from all others. The *type-name* argument must be a string, but is only used for debugging purposes (such as the printed representation of a record of the new type). The *field-names* argument is a list of symbols naming the *fields* of a record of the new type. It is an error if the list contains any duplicates. It is unspecified how record-type descriptors are represented.

**record-constructor rtd [field-names]**

Returns a procedure for constructing new members of the type represented by rtd. The returned procedure accepts exactly as many arguments as there are symbols in the
given list, field-names; these are used, in order, as the initial values of those fields in a
new record, which is returned by the constructor procedure. The values of any fields
not named in that list are unspecified. The field-names argument defaults to the list
of field names in the call to make-record-type that created the type represented by
rtd; if the field-names argument is provided, it is an error if it contains any duplicates
or any symbols not in the default list.

record-predicate rtd
  [Function]
  Returns a procedure for testing membership in the type represented by rtd. The
  returned procedure accepts exactly one argument and returns a true value if the
  argument is a member of the indicated record type; it returns a false value otherwise.

record-accessor rtd field-name
  [Function]
  Returns a procedure for reading the value of a particular field of a member of the type
  represented by rtd. The returned procedure accepts exactly one argument which must
  be a record of the appropriate type; it returns the current value of the field named by
  the symbol field-name in that record. The symbol field-name must be a member of the
  list of field-names in the call to make-record-type that created the type represented
  by rtd.

record-modifier rtd field-name
  [Function]
  Returns a procedure for writing the value of a particular field of a member of the type
  represented by rtd. The returned procedure accepts exactly two arguments: first, a
  record of the appropriate type, and second, an arbitrary Scheme value; it modifies
  the field named by the symbol field-name in that record to contain the given value.
  The returned value of the modifier procedure is unspecified. The symbol field-name
  must be a member of the list of field-names in the call to make-record-type that
  created the type represented by rtd.

In May of 1996, as a product of discussion on the rrrs-authors mailing list, I rewrote
‘record.scm’ to portably implement type disjointness for record data types.

As long as an implementation’s procedures are opaque and the record code is loaded
before other programs, this will give disjoint record types which are unforgeable and incor-
ruptible by R4RS procedures.

As a consequence, the procedures record?, record-type-descriptor, record-type-
name,and record-type-field-names are no longer supported.

7.2 Sorting and Searching

7.2.1 Common List Functions

(require 'common-list-functions)

The procedures below follow the Common LISP equivalents apart from optional argu-
ments in some cases.

7.2.1.1 List construction

make-list k
  [Function]
**make-list**  
*`k init`*  
**[Function]**  

`make-list` creates and returns a list of *k* elements. If *init* is included, all elements in the list are initialized to *init*.

Example:

```
(make-list 3)  
⇒ (#<unspecified> #<unspecified> #<unspecified>)

(make-list 5 'foo)  
⇒ (foo foo foo foo foo)
```

**list***  
*`obj1 obj2 ...`  
**[Function]**

Works like `list` except that the cdr of the last pair is the last argument unless there is only one argument, when the result is just that argument. Sometimes called `cons*`.

E.g.:

```
(list* 1)  
⇒ 1

(list* 1 2 3)  
⇒ (1 2 . 3)

(list* 1 2 '(3 4))  
⇒ (1 2 3 4)

(list* args '())  
≡ (list args)
```

**copy-list**  
*`lst`*  
**[Function]**

`copy-list` makes a copy of *lst* using new pairs and returns it. Only the top level of the list is copied, i.e., pairs forming elements of the copied list remain `eq?` to the corresponding elements of the original; the copy is, however, not `eq?` to the original, but is `equal?` to it.

Example:

```
(coppy-list '(foo foo foo))  
⇒ (foo foo foo)

(define q '(foo bar baz bang))

(define p q)

(eq? p q)  
⇒ #t

(define r (copy-list q))

(eq? q r)  
⇒ #f

(equal? q r)  
⇒ #t

(define bar '(bar))

(eq? bar (car (copy-list (list bar 'foo))))  
⇒ #t
```

### 7.2.1.2 Lists as sets

eqv? is used to test for membership by procedures which treat lists as sets.
adjoin \(e \ l\)  
\(\text{adjoin}\) returns the adjoint of the element \(e\) and the list \(l\). That is, if \(e\) is in \(l\), \(\text{adjoin}\) returns \(l\), otherwise, it returns \((\text{cons} \ e \ l)\).

Example:

\[
(\text{adjoin} \ '\text{baz} \ '(\text{bar baz bang}))
\Rightarrow (\text{bar baz bang})
\]

\[
(\text{adjoin} \ '\text{foo} \ '(\text{bar baz bang}))
\Rightarrow (\text{foo bar baz bang})
\]

union \(l1 \ l2\)  
\(\text{union}\) returns a list of all elements that are in \(l1\) or \(l2\). Duplicates between \(l1\) and \(l2\) are culled. Duplicates within \(l1\) or within \(l2\) may or may not be removed.

Example:

\[
(\text{union} \ '(1 \ 2 \ 3 \ 4) \ '(5 \ 6 \ 7 \ 8)) 
\Rightarrow (1 \ 2 \ 3 \ 4 \ 5 \ 6 \ 7 \ 8)
\]

\[
(\text{union} \ '(0 \ 1 \ 2 \ 3 \ 4) \ '(3 \ 4 \ 5 \ 6)) 
\Rightarrow (5 \ 6 \ 0 \ 1 \ 2 \ 3 \ 4)
\]

intersection \(l1 \ l2\)  
\(\text{intersection}\) returns a list of all elements that are in both \(l1\) and \(l2\).

Example:

\[
(\text{intersection} \ '(1 \ 2 \ 3 \ 4) \ '(3 \ 4 \ 5 \ 6)) 
\Rightarrow (3 \ 4)
\]

\[
(\text{intersection} \ '(1 \ 2 \ 3 \ 4) \ '(5 \ 6 \ 7 \ 8)) 
\Rightarrow ()
\]

set-difference \(l1 \ l2\)  
\(\text{set-difference}\) returns a list of all elements that are in \(l1\) but not in \(l2\).

Example:

\[
(\text{set-difference} \ '(1 \ 2 \ 3 \ 4) \ '(3 \ 4 \ 5 \ 6)) 
\Rightarrow (1 \ 2)
\]

\[
(\text{set-difference} \ '(1 \ 2 \ 3 \ 4) \ '(1 \ 2 \ 3 \ 4 \ 5 \ 6)) 
\Rightarrow ()
\]

subset? \(list1 \ list2\)  
\(\text{subset}\) returns \#t if every element of \(list1\) is eqv? an element of \(list2\); otherwise returns \#f.

Example:

\[
(\text{subset?} \ '(1 \ 2 \ 3 \ 4) \ '(3 \ 4 \ 5 \ 6)) 
\Rightarrow \#f
\]

\[
(\text{subset?} \ '(1 \ 2 \ 3 \ 4) \ '(6 \ 5 \ 4 \ 3 \ 2 \ 1 \ 0)) 
\Rightarrow \#t
\]

member-if \(\text{pred} \ lst\)  
\(\text{member-if}\) returns the list headed by the first element of \(lst\) to satisfy \(\text{pred element}\). Member-if returns \#f if \(\text{pred}\) returns \#f for every element in \(lst\).
Example:

\[
\begin{align*}
\text{(member-if vector? '(a 2 b 4))} \\
& \Rightarrow \text{#f} \\
\text{(member-if number? '(a 2 b 4))} \\
& \Rightarrow \text{(2 b 4)}
\end{align*}
\]

**some** \textit{pred lst1 lst2 ...} [Function]

\textit{pred} is a boolean function of as many arguments as there are list arguments to \textit{some}
i.e., \textit{lst} plus any optional arguments. \textit{pred} is applied to successive elements of the list
arguments in order. \textit{some} returns \texttt{#t} as soon as one of these applications returns \texttt{#t},
and is \texttt{#f} if none returns \texttt{#t}. All the lists should have the same length.

Example:

\[
\begin{align*}
\text{(some odd? '(1 2 3 4))} & \Rightarrow \texttt{#t} \\
\text{(some odd? '(2 4 6 8))} & \Rightarrow \texttt{#f} \\
\text{(some > '(1 3) '(2 4))} & \Rightarrow \texttt{#f}
\end{align*}
\]

**every** \textit{pred lst1 lst2 ...} [Function]

every is analogous to \textit{some} except it returns \texttt{#t} if every application of \textit{pred} is \texttt{#t} and
\texttt{#f} otherwise.

Example:

\[
\begin{align*}
\text{(every even? '(1 2 3 4))} & \Rightarrow \texttt{#f} \\
\text{(every even? '(2 4 6 8))} & \Rightarrow \texttt{#t} \\
\text{(every > '(2 3) '(1 4))} & \Rightarrow \texttt{#f}
\end{align*}
\]

**notany** \textit{pred lst1 ...} [Function]

\textit{notany} is analogous to \textit{some} but returns \texttt{#t} if no application of \textit{pred} returns \texttt{#t} or \texttt{#f}
as soon as any one does.

**notevery** \textit{pred lst1 ...} [Function]

\textit{notevery} is analogous to \textit{some} but returns \texttt{#t} as soon as an application of \textit{pred} returns
\texttt{#f}, and \texttt{#f} otherwise.

Example:

\[
\begin{align*}
\text{(notevery even? '(1 2 3 4))} & \Rightarrow \texttt{#t} \\
\text{(notevery even? '(2 4 6 8))} & \Rightarrow \texttt{#t}
\end{align*}
\]
list-of??  predicate  
Returns a predicate which returns true if its argument is a list every element of which satisfies predicate.

list-of??  predicate  low-bound  high-bound  
low-bound and high-bound are non-negative integers. list-of?? returns a predicate which returns true if its argument is a list of length between low-bound and high-bound (inclusive); every element of which satisfies predicate.

list-of??  predicate  bound  
bound is an integer. If bound is negative, list-of?? returns a predicate which returns true if its argument is a list of length greater than (- bound); every element of which satisfies predicate. Otherwise, list-of?? returns a predicate which returns true if its argument is a list of length less than or equal to bound; every element of which satisfies predicate.

find-if  pred  lst  
find-if searches for the first element in lst such that (pred element) returns #t. If it finds any such element in lst, element is returned. Otherwise, #f is returned.

Example:
\[(\text{find-if number? } '(\text{foo 1 bar 2}))\]  
\[\Rightarrow 1\]

\[(\text{find-if number? } '(\text{foo bar baz bang}))\]  
\[\Rightarrow #f\]

\[(\text{find-if symbol? } '(1 2 \text{ foo bar}))\]  
\[\Rightarrow \text{foo}\]

remove  elt  lst  
remove removes all occurrences of elt from lst using eqv? to test for equality and returns everything that’s left. N.B.: other implementations (Chez, Scheme-C and T, at least) use equal? as the equality test.

Example:
\[(\text{remove 1 } '(1 2 1 3 1 4 1 5))\]  
\[\Rightarrow (2 3 4 5)\]

\[(\text{remove } \text{foo } '(\text{bar baz bang}))\]  
\[\Rightarrow (\text{bar baz bang})\]

remove-if  pred  lst  
remove-if removes all elements from lst where (pred element) is #t and returns everything that’s left.

Example:
\[(\text{remove-if number? } '(1 2 3 4))\]
(remove-if even? '(1 2 3 4 5 6 7 8))
⇒ (1 3 5 7)

(remove-if-not pred lst)
remove-if-not removes all elements from lst for which (pred element) is #f and returns everything that’s left.
Example:
   (remove-if-not number? '(foo bar baz))
⇒ ()
   (remove-if-not odd? '(1 2 3 4 5 6 7 8))
⇒ (1 3 5 7)

(has-duplicates? lst)
returns #t if 2 members of lst are equal?, #f otherwise.
Example:
   (has-duplicates? '(1 2 3 4))
⇒ #f
   (has-duplicates? '(2 4 3 4))
⇒ #t

The procedure remove-duplicates uses member (rather than memv).

(remove-duplicates lst)
returns a copy of lst with its duplicate members removed. Elements are considered duplicate if they are equal?.
Example:
   (remove-duplicates '(1 2 3 4))
⇒ (1 2 3 4)
   (remove-duplicates '(2 4 3 4))
⇒ (2 4 3)

7.2.1.3 Lists as sequences

(position obj lst)
position returns the 0-based position of obj in lst, or #f if obj does not occur in lst.
Example:
   (position 'foo '(foo bar baz bang))
⇒ 0
   (position 'baz '(foo bar baz bang))
⇒ 2
   (position 'oops '(foo bar baz bang))
⇒ #f
**reduce** \( p \) \( lst \)  

**reduce** combines all the elements of a sequence using a binary operation (the combination is left-associative). For example, using \(+\), one can add up all the elements. **reduce** allows you to apply a function which accepts only two arguments to more than 2 objects. Functional programmers usually refer to this as *foldl*. **collect:reduce** (see Section 7.1.10 [Collections], page 210) provides a version of **collect** generalized to collections.

Example:

\[
\begin{align*}
(reduce + '(1 2 3 4)) & \Rightarrow 10 \\
(define (bad-sum . l) (reduce + l)) & (bad-sum 1 2 3 4) \\
& \equiv (reduce + (1 2 3 4)) \\
& \equiv (+ (+ (+ 1 2) 3) 4) \\
& \Rightarrow 10 \\
(bad-sum) & \equiv (reduce + ()) \\
& \Rightarrow () \\
(reduce string-append '("hello" "cruel" "world")) & \Rightarrow "hellocruelworld" \\
(reduce anything '()) & \Rightarrow () \\
(reduce anything '(x)) & \Rightarrow x
\end{align*}
\]

What follows is a rather non-standard implementation of **reverse** in terms of **reduce** and a combinator elsewhere called \( C \).

\[
\text{;;; Contributed by Jussi Piitulainen (jpiitula @ ling.helsinki.fi)}
\]

\[
\begin{align*}
(define commute & (lambda (f)) \\
& (lambda (x y) \\
& (f y x)))))
\end{align*}
\]

\[
\begin{align*}
(define reverse & (lambda (args) \\
& (reduce-init (commute cons) '() args)))
\end{align*}
\]

**reduce-init** \( p \) \( init \) \( lst \)  

**reduce-init** is the same as **reduce**, except that it implicitly inserts \( init \) at the start of the list. **reduce-init** is preferred if you want to handle the null list, the one-element, and lists with two or more elements consistently. It is common to use the operator’s idempotent as the initializer. Functional programmers usually call this *foldl*.

Example:

\[
\begin{align*}
(define (sum . l) (reduce-init + 0 l))
\end{align*}
\]
(sum 1 2 3 4)
≡ (reduce-init + 0 (1 2 3 4))
≡ (+ (+ (+ (+ 0 1) 2) 3) 4)
⇒ 10

(sum)
≡ (reduce-init + 0 '())
⇒ 0

(reduce-init string-append "@" '("hello" "cruel" "world"))
≡ (string-append (string-append "@" "hello")
"cruel")
"world")
⇒ "@hellocruelworld"

Given a differentiation of 2 arguments, diff, the following will differentiate by any number of variables.

(define (diff* exp . vars)
  (reduce-init diff exp vars))

Example:

;;; Real-world example: Insertion sort using reduce-init.

(define (insert l item)
  (if (null? l)
      (list item)
      (if (< (car l) item)
          (cons (car l) (insert (cdr l) item))
          (cons item l))))

(define (insertion-sort l) (reduce-init insert '() l))

(insertion-sort '(3 1 4 1 5)
≡ (reduce-init insert () (3 1 4 1 5))
≡ (insert (insert (insert (insert (insert () 3) 1) 4) 1) 5)
≡ (insert (insert (insert (insert (insert (insert (insert (insert 3)) 1) 4) 1) 5)
≡ (insert (insert (insert (insert (1 3 4) 1) 5)
≡ (insert (insert (1 3 4) 1) 5)
≡ (insert (1 1 3 4) 5)
⇒ (1 1 3 4 5)

last lst n

last returns the last n elements of lst. n must be a non-negative integer.

Example:

(last '(foo bar baz bang) 2)
⇒ (baz bang)

(last '(1 2 3) 0)
⇒ ()

**butlast** *lst n*  
butlast returns all but the last *n* elements of *lst.*

Example:

```
(butlast '(a b c d) 3)  ⇒ (a)
(butlast '(a b c d) 4)  ⇒ ()
```

**last** and **butlast** split a list into two parts when given identical arguments.

```
(last '(a b c d e) 2)  ⇒ (d e)
(butlast '(a b c d e) 2)  ⇒ (a b c)
```

**nthcdr** *n lst*  
nthcdr takes *n* cdrs of *lst* and returns the result. Thus (*nthcdr* 3 *lst*) ≡ (*cdddr* *lst*)

Example:

```
(nthcdr 2 '(a b c d))  ⇒ (c d)
(nthcdr 0 '(a b c d))  ⇒ (a b c d)
```

**butnthcdr** *n lst*  
butnthcdr returns all but the nthcdr *n* elements of *lst.*

Example:

```
(butnthcdr 3 '(a b c d))  ⇒ (a b c)
(butnthcdr 4 '(a b c d))  ⇒ (a b c d)
```

**nthcdr** and **butnthcdr** split a list into two parts when given identical arguments.

```
(nthcdr 2 '(a b c d e))  ⇒ (c d e)
(butnthcdr 2 '(a b c d e))  ⇒ (a b)
```

### 7.2.1.4 Destructive list operations

These procedures may mutate the list they operate on, but any such mutation is undefined.

**nconc** *args*  
nconc destructively concatenates its arguments. (Compare this with **append**, which copies arguments rather than destroying them.) Sometimes called **append!** (see Section 7.4.4 [Rev2 Procedures], page 248).
Example: You want to find the subsets of a set. Here’s the obvious way:

```
(define (subsets set)
  (if (null? set)
      '(()
      (append (map (lambda (sub) (cons (car set) sub))
                  (subsets (cdr set)))
              (subsets (cdr set)))))
```

But that does way more consing than you need. Instead, you could replace the `append` with `nconc`, since you don’t have any need for all the intermediate results.

Example:

```
(define x '(a b c))
(define y '(d e f))
(nconc x y)
⇒ (a b c d e f)
x
⇒ (a b c d e f)
```

`nconc` is the same as `append` in ‘sc2.scm’.

\[\text{nreverse \ list} \quad \text{[Procedure]}\]
\(\text{nreverse}\) reverses the order of elements in \(\text{lst}\) by mutating \(\text{cdrs}\) of the list. Sometimes called `reverse!`.

Example:

```
(define foo '(a b c))
(nreverse foo)
⇒ (c b a)
foo
⇒ (a)
```

Some people have been confused about how to use `nreverse`, thinking that it doesn’t return a value. It needs to be pointed out that

```
(set! lst (nreverse lst))
```

is the proper usage, not

```
(nreverse lst)
```

The example should suffice to show why this is the case.

\[\text{delete \ elt \ lst} \quad \text{[Procedure]}\]
\[\text{delete-if \ pred \ lst} \quad \text{[Procedure]}\]
\[\text{delete-if-not \ pred \ lst} \quad \text{[Procedure]}\]

Destructive versions of `remove`, `remove-if`, and `remove-if-not`.

Example:

```
(define lst (list 'foo 'bar 'baz 'bang))
(delete 'foo lst)
⇒ (bar baz bang)
lst
```
⇒ (foo bar baz bang)

(define lst (list 1 2 3 4 5 6 7 8 9))
(delel-if odd? lst)
⇒ (2 4 6 8)
lst
⇒ (1 2 4 6 8)

Some people have been confused about how to use delete, delete-if, and delete-if, thinking that they don’t return a value. It needs to be pointed out that

(set! lst (delete el lst))

is the proper usage, not

(delete el lst)

The examples should suffice to show why this is the case.

7.2.1.5 Non-List functions

and? arg1 . . . [Function]
and? checks to see if all its arguments are true. If they are, and? returns #t, otherwise, #f. (In contrast to and, this is a function, so all arguments are always evaluated and in an unspecified order.)

Example:

(and? 1 2 3)
⇒ #t
(and #f 1 2)
⇒ #f

or? arg1 . . . [Function]
or? checks to see if any of its arguments are true. If any is true, or? returns #t, and #f otherwise. (To or as and? is to and.)

Example:

(or? 1 2 #f)
⇒ #t
(or? #f #f #f)
⇒ #f

atom? object [Function]
 Returns #t if object is not a pair and #f if it is pair. (Called atom in Common LISP.)

(atom? 1)
⇒ #t
(atom? '(1 2))
⇒ #f
(atom? #(1 2)) ; dubious!
⇒ #t
7.2.2 Tree operations

(require 'tree)

These are operations that treat lists as representations of trees.

**Function**  subst new old tree

Makes a copy of tree, substituting new for every subtree or leaf of tree which is equal? to old and returns a modified tree. The original tree is unchanged, but may share parts with the result.

**Function**  substq new old tree

Substitutes new for every subtree of tree which is eq? to old and returns a modified tree. The original tree is unchanged, but may share parts with the result.

**Function**  substv new old tree

Substitutes new for every subtree of tree which is eqv? to old and returns a modified tree. The original tree is unchanged, but may share parts with the result.

Examples:

```
(substq 'tempest 'hurricane '(shakespeare wrote (the hurricane)))
⇒ (shakespeare wrote (tempest))
(substq 'foo () '(shakespeare wrote (twelfth night)))
⇒ (shakespeare wrote (twelfth night . foo) . foo)
```

**Function**  copy-tree tree

Makes a copy of the nested list structure tree using new pairs and returns it. All levels are copied, so that none of the pairs in the tree are eq? to the original ones – only the leaves are.

Example:

```
(define bar '(bar))
(copy-tree (list bar 'foo))
⇒ ((bar) foo)
```

7.2.3 Chapter Ordering

(require 'chapter-order)

The 'chap:' functions deal with strings which are ordered like chapter numbers (or letters) in a book. Each section of the string consists of consecutive numeric or consecutive alphabetic characters of like case.

**Function**  chap:string<? string1 string2

Returns #t if the first non-matching run of alphabetic upper-case or the first non-matching run of alphabetic lower-case or the first non-matching run of numeric characters of string1 is string<? than the corresponding non-matching run of characters of string2.

```
(chap:string<? "a.9" "a.10")
⇒ #t
```
(chap:string<? "4c" "4aa")  ⇒ #t
(chap:string<? "Revised~{3.99}" "Revised~{4}")  ⇒ #t

chap:string>=? string1 string2  [Function]
chap:string<=? string1 string2  [Function]
chap:string>? string1 string2  [Function]

Implement the corresponding chapter-order predicates.

chap:next-string string  [Function]

Returns the next string in the chapter order. If string has no alphabetic or numeric characters, (string-append string "0") is returned. The argument to chap:next-string will always be chap:string<? than the result.

(chap:next-string "a.9")  ⇒ "a.10"
(chap:next-string "4c")  ⇒ "4d"
(chap:next-string "4z")  ⇒ "4aa"
(chap:next-string "Revised~{4}" )  ⇒ "Revised~{5}"

7.2.4 Sorting

(require 'sort) or (require 'srfi-95)

[by Richard A. O’Keefe, 1991]

I am providing this source code with no restrictions at all on its use (but please retain D.H.D.Warren’s credit for the original idea).

The code of merge and merge! could have been quite a bit simpler, but they have been coded to reduce the amount of work done per iteration. (For example, we only have one null? test per iteration.)

I gave serious consideration to producing Common-LISP-compatible functions. However, Common LISP’s sort is our sort! (well, in fact Common LISP’s stable-sort is our sort!; merge sort is fast as well as stable!) so adapting CL code to Scheme takes a bit of work anyway. I did, however, appeal to CL to determine the order of the arguments.

The standard functions <, >, char<?, char>? , char-ci<?, char-ci>?, string<?, string-ci<? and string-ci>? are suitable for use as comparison functions. Think of (less? x y) as saying when x must not precede y.

[Addendum by Aubrey Jaffer, 2006]

These procedures are stable when called with predicates which return #f when applied to identical arguments.

The sorted?, merge, and merge! procedures consume asymptotic time and space no larger than \(O(N)\), where \(N\) is the sum of the lengths of the sequence arguments. The sort and sort! procedures consume asymptotic time and space no larger than \(O(N*\log(N))\), where \(N\) is the length of the sequence argument.

All five functions take an optional key argument corresponding to a CL-style ‘&key’ argument. A less? predicate with a key argument behaves like:

(lambda (x y) (less? (key x) (key y)))

All five functions will call the key argument at most once per element.
The ‘!’ variants sort in place; \texttt{sort!} returns its \texttt{sequence} argument.

\begin{verbatim}
(= sorted? sequence less?) \textbf{[Function]}
(= sorted? sequence less? key) \textbf{[Function]}
\end{verbatim}

Returns \texttt{#t} when the sequence argument is in non-decreasing order according to \texttt{less?} (that is, there is no adjacent pair \ldots \texttt{y x} \ldots for which \texttt{(less? y x)}).

Returns \texttt{#f} when the sequence contains at least one out-of-order pair. It is an error if the sequence is not a list or array (including vectors and strings).

\begin{verbatim}
(merge list1 list2 less?) \textbf{[Function]}
(merge list1 list2 less? key) \textbf{[Function]}
\end{verbatim}

Merges two sorted lists, returning a freshly allocated list as its result.

\begin{verbatim}
(merge! list1 list2 less?) \textbf{[Function]}
(merge! list1 list2 less? key) \textbf{[Function]}
\end{verbatim}

Merges two sorted lists, re-using the pairs of \texttt{list1} and \texttt{list2} to build the result. The result will be either \texttt{list1} or \texttt{list2}.

\begin{verbatim}
(sort sequence less?) \textbf{[Function]}
(sort sequence less? key) \textbf{[Function]}
\end{verbatim}

Accepts a list or array (including vectors and strings) for \texttt{sequence}; and returns a completely new sequence which is sorted according to \texttt{less?}. The returned sequence is the same type as the argument \texttt{sequence}. Given valid arguments, it is always the case that:

\[
(= (sorted? (sort sequence less?) less?) \Rightarrow \texttt{#t})
\]

\begin{verbatim}
(sort! sequence less?) \textbf{[Function]}
(sort! sequence less? key) \textbf{[Function]}
\end{verbatim}

Returns list, array, vector, or string \texttt{sequence} which has been mutated to order its elements according to \texttt{less?}. Given valid arguments, it is always the case that:

\[
(= (sorted? (sort! sequence less?) less?) \Rightarrow \texttt{#t})
\]

7.2.5 Topological Sort

\texttt{(require 'topological-sort)} or \texttt{(require 'tsort)}

The algorithm is inspired by Cormen, Leiserson and Rivest (1990) \textit{Introduction to Algorithms}, chapter 23.

\begin{verbatim}
(tsort dag pred) \textbf{[Function]}
topological-sort dag pred \textbf{[Function]}
\end{verbatim}

\texttt{where}

\begin{verbatim}
dag \texttt{is a list of sublists. The car of each sublist is a vertex. The cdr is the adjacency list of that vertex, i.e. a list of all vertices to which there exists an edge from the car vertex.}

pred \texttt{is one of eq?, eqv?, equal?, =, char=?, char-ci=?, string=?, or string-ci=?.}
\end{verbatim}
Sort the directed acyclic graph $dag$ so that for every edge from vertex $u$ to $v$, $u$ will come before $v$ in the resulting list of vertices.

Time complexity: $O(|V| + |E|)$

Example (from Cormen):

Prof. Bumstead topologically sorts his clothing when getting dressed. The first argument to `tsort` describes which garments he needs to put on before others. (For example, Prof Bumstead needs to put on his shirt before he puts on his tie or his belt.) `tsort` gives the correct order of dressing:

```
(require 'tsort)
(tsort '((shirt tie belt)
  (tie jacket)
  (belt jacket)
  (watch)
  (pants shoes belt)
  (undershorts pants shoes)
  (socks shoes))
eqv)
⇒
(socks undershorts pants shoes watch shirt belt tie jacket)
```

### 7.2.6 Hashing

```
(require 'hash)
```

These hashing functions are for use in quickly classifying objects. Hash tables use these functions.

- **hashq** $obj$ $k$  
  Returns an exact non-negative integer less than $k$. For each non-negative integer less than $k$ there are arguments $obj$ for which the hashing functions applied to $obj$ and $k$ returns that integer.

  For `hashq`, $(\text{eq? obj1 obj2})$ implies $(= (\text{hashq obj1} k) (\text{hashq obj2})).$

  For `hashv`, $(\text{eqv? obj1 obj2})$ implies $(= (\text{hashv obj1} k) (\text{hashv obj2})).$

  For `hash`, $(\text{equal? obj1 obj2})$ implies $(= (\text{hash obj1} k) (\text{hash obj2})).$

- **hashv** $obj$ $k$  
  Returns the same as `hashq`.

- **hash** $obj$ $k$  
  Returns the same as `hashq`.

`hash`, `hashv`, and `hashq` return in time bounded by a constant. Notice that items having the same `hash` implies the items have the same `hashv` implies the items have the same `hashq`.

### 7.2.7 Space-Filling Curves

#### 7.2.7.1 Hilbert Space-Filling Curve

```
(require 'hilbert-fill)
```
The *Hilbert Space-Filling Curve* is a one-to-one mapping between a unit line segment and an \( n \)-dimensional unit cube. This implementation treats the nonnegative integers either as fractional bits of a given width or as nonnegative integers.

The integer procedures map the non-negative integers to an arbitrarily large \( n \)-dimensional cube with its corner at the origin and all coordinates are non-negative.

For any exact nonnegative integer \( \text{scalar} \) and exact integer \( \text{rank} > 2 \),
\[
(= \text{scalar} \ (\text{hilbert-coordinates->integer} \\
(\text{integer->hilbert-coordinates} \ \text{scalar} \ \text{rank}))) \Rightarrow \#t
\]

When treating integers as \( k \) fractional bits,
\[
(= \text{scalar} \ (\text{hilbert-coordinates->integer} \\
(\text{integer->hilbert-coordinates} \ \text{scalar} \ \text{rank} \ k)) \ k) \Rightarrow \#t
\]

**integer->hilbert-coordinates** \( \text{scalar} \ \text{rank} \)

Returns a list of \( \text{rank} \) integer coordinates corresponding to exact non-negative integer \( \text{scalar} \). The lists returned by **integer->hilbert-coordinates** for \( \text{scalar} \) arguments 0 and 1 will differ in the first element.

**integer->hilbert-coordinates** \( \text{scalar} \ \text{rank} \ k \)

\( \text{scalar} \) must be a nonnegative integer of no more than \( \text{rank} \ast k \) bits.

**integer->hilbert-coordinates** Returns a list of \( \text{rank} \ k \)-bit nonnegative integer coordinates corresponding to exact non-negative integer \( \text{scalar} \). The curves generated by **integer->hilbert-coordinates** have the same alignment independent of \( k \).

**hilbert-coordinates->integer** \( \text{coords} \)

Returns an exact non-negative integer corresponding to \( \text{coords} \), a list of non-negative integer coordinates.

### 7.2.7.2 Gray code

A *Gray code* is an ordering of non-negative integers in which exactly one bit differs between each pair of successive elements. There are multiple Gray codings. An \( n \)-bit Gray code corresponds to a Hamiltonian cycle on an \( n \)-dimensional hypercube.

Gray codes find use communicating incrementally changing values between asynchronous agents. De-laminated Gray codes comprise the coordinates of Hilbert space-filling curves.

**integer->gray-code** \( k \)

Converts \( k \) to a Gray code of the same integer-length as \( k \).

**gray-code->integer** \( k \)

Converts the Gray code \( k \) to an integer of the same integer-length as \( k \).

For any non-negative integer \( k \),
\[
(= \text{eqv?} \ k \ (\text{gray-code->integer} \ (\text{integer->gray-code} \ k))) = k1 \ k2
\]

**gray-code<?** \( k1 \ k2 \)


gray-code>? k1 k2  
gray-code<=? k1 k2  
gray-code>=? k1 k2  
These procedures return #t if their Gray code arguments are (respectively): equal, monotonically increasing, monotonically decreasing, monotonically nondecreasing, or monotonically nonincreasing.

For any non-negative integers \( k1 \) and \( k2 \), the Gray code predicate of \((\text{integer->gray-code } k1)\) and \((\text{integer->gray-code } k2)\) will return the same value as the corresponding predicate of \( k1 \) and \( k2 \).

### 7.2.7.3 Bitwise Lamination

\texttt{delaminate-list count ks}

Returns a list of \( count \) integers comprised of the \( j \)th bit of the integers \( ks \) where \( j \) ranges from \( count \)-1 to 0.

\[
\text{(map (lambda (k) (number->string k 2)) (delaminate-list 4 '(7 6 5 4 0 0 0 0)))}
\Rightarrow ("0" "11110000" "11000000" "10100000")
\]

\texttt{delaminate-list} is its own inverse:

\[
\text{(delaminate-list 8 (delaminate-list 4 '(7 6 5 4 0 0 0 0 )))}
\Rightarrow (7 6 5 4 0 0 0 0)
\]

### 7.2.7.4 Peano Space-Filling Curve

\texttt{(require 'peano-fill)}

\texttt{natural->peano-coordinates scalar rank}

Returns a list of \( rank \) nonnegative integer coordinates corresponding to exact nonnegative integer \( scalar \). The lists returned by \texttt{natural->peano-coordinates} for \( scalar \) arguments 0 and 1 will differ in the first element.

\texttt{peano-coordinates->natural coords}

Returns an exact nonnegative integer corresponding to \( coords \), a list of nonnegative integer coordinates.

\texttt{integer->peano-coordinates scalar rank}

Returns a list of \( rank \) integer coordinates corresponding to exact integer \( scalar \). The lists returned by \texttt{integer->peano-coordinates} for \( scalar \) arguments 0 and 1 will differ in the first element.

\texttt{peano-coordinates->integer coords}

Returns an exact integer corresponding to \( coords \), a list of integer coordinates.

### 7.2.7.5 Sierpinski Curve

\texttt{(require 'sierpinski)}

\texttt{make-sierpinski-indexer max-coordinate}

Returns a procedure (eg hash-function) of 2 numeric arguments which preserves nearness in its mapping from \( \text{NxN} \) to \( N \).
max-coordinate is the maximum coordinate (a positive integer) of a population of points. The returned procedures is a function that takes the x and y coordinates of a point, (non-negative integers) and returns an integer corresponding to the relative position of that point along a Sierpinski curve. (You can think of this as computing a (pseudo-) inverse of the Sierpinski spacefilling curve.)

Example use: Make an indexer (hash-function) for integer points lying in square of integer grid points [0,99]x[0,99]:

```
(define space-key (make-sierpinski-indexer 100))
```

Now let's compute the index of some points:

```
(space-key 24 78)  ⇒ 9206
(space-key 23 80)  ⇒ 9172
```

Note that locations (24, 78) and (23, 80) are near in index and therefore, because the Sierpinski spacefilling curve is continuous, we know they must also be near in the plane. Nearness in the plane does not, however, necessarily correspond to nearness in index, although it tends to be so.

Example applications:


- Use Sierpinski index as key by which to store 2-dimensional data in a 1-dimensional data structure (such as a table). Then locations that are near each other in 2-d space will tend to be near each other in 1-d data structure; and locations that are near in 1-d data structure will be near in 2-d space. This can significantly speed retrieval from secondary storage because contiguous regions in the plane will tend to correspond to contiguous regions in secondary storage. (This is a standard technique for managing CAD/CAM or geographic data.)

7.2.8 Soundex

```
(require 'soundex)
```

```
soundex name
```

Computes the soundex hash of name. Returns a string of an initial letter and up to three digits between 0 and 6. Soundex supposedly has the property that names that sound similar in normal English pronunciation tend to map to the same key.

Soundex was a classic algorithm used for manual filing of personal records before the advent of computers. It performs adequately for English names but has trouble with other languages.

See Knuth, Vol. 3 Sorting and searching, pp 391–2

To manage unusual inputs, soundex omits all non-alphabetic characters. Consequently, in this implementation:

```
(soundex <string of blanks>)  ⇒ ""
(soundex "")  ⇒ ""
```
Chapter 7: Other Packages

Examples from Knuth:

\[
\begin{align*}
\text{(map soundex } & \text{('"Euler" "Gauss" "Hilbert" "Knuth" \\
& "Lloyd" "Lukasiewicz")}) \\
\quad \Rightarrow & \text{('"E460" "G200" "H416" "K530" "L300" "L222"})}
\end{align*}
\]

\[
\begin{align*}
\text{(map soundex } & \text{('"Ellery" "Ghosh" "Heilbronn" "Kant" \\
& "Ladd" "Lissajous")}) \\
\quad \Rightarrow & \text{('"E460" "G200" "H416" "K530" "L300" "L222"})}
\end{align*}
\]

Some cases in which the algorithm fails (Knuth):

\[
\begin{align*}
\text{(map soundex } & \text{('"Rogers" "Rodgers")}) \Rightarrow \text{('"R262" "R326")}
\end{align*}
\]

\[
\begin{align*}
\text{(map soundex } & \text{('"Sinclair" "St. Clair")}) \Rightarrow \text{('"S524" "S324")}
\end{align*}
\]

\[
\begin{align*}
\text{(map soundex } & \text{('"Tchebysheff" "Chebyshev")}) \Rightarrow \text{('"T212" "C121")}
\end{align*}
\]

7.2.9 String Search

(require 'string-search)

\text{string-index} \; \text{string char} \quad \text{[Procedure]}

\text{string-index-ci} \; \text{string char} \quad \text{[Procedure]}

Returns the index of the first occurrence of \text{char} within \text{string}, or \text{#f} if the \text{string} does not contain a character \text{char}.

\text{string-reverse-index} \; \text{string char} \quad \text{[Procedure]}

\text{string-reverse-index-ci} \; \text{string char} \quad \text{[Procedure]}

Returns the index of the last occurrence of \text{char} within \text{string}, or \text{#f} if the \text{string} does not contain a character \text{char}.

\text{substring?} \; \text{pattern string} \quad \text{[Procedure]}

\text{substring-ci?} \; \text{pattern string} \quad \text{[Procedure]}

Searches \text{string} to see if some substring of \text{string} is equal to \text{pattern}. \text{substring?} returns the index of the first character of the first substring of \text{string} that is equal to \text{pattern}; or \text{#f} if \text{string} does not contain \text{pattern}.

\[
\begin{align*}
\text{(substring? } & \text{"rat" "pirate"}) \Rightarrow 2 \\
\text{(substring? } & \text{"rat" "outrage"}) \Rightarrow \text{#f}
\end{align*}
\]

\[
\begin{align*}
\text{(substring? } & \text{"" any-string}) \Rightarrow 0
\end{align*}
\]

\text{find-string-from-port?} \; \text{str in-port max-no-chars} \quad \text{[Procedure]}

Looks for a string \text{str} within the first \text{max-no-chars} chars of the input port \text{in-port}.

\text{find-string-from-port?} \; \text{str in-port} \quad \text{[Procedure]}

When called with two arguments, the search span is limited by the end of the input stream.

\text{find-string-from-port?} \; \text{str in-port char} \quad \text{[Procedure]}

Searches up to the first occurrence of character \text{char} in \text{str}.
find-string-from-port?  str in-port proc  \[Procedure\]
Searches up to the first occurrence of the procedure proc returning non-false when called with a character (from in-port) argument.

When the str is found, find-string-from-port? returns the number of characters it has read from the port, and the port is set to read the first char after that (that is, after the str) The function returns #f when the str isn’t found.

find-string-from-port? reads the port \emph{strictly} sequentially, and does not perform any buffering. So find-string-from-port? can be used even if the in-port is open to a pipe or other communication channel.

string-subst  txt old1 new1 \ldots  \[Function\]
Returns a copy of string txt with all occurrences of string old1 in txt replaced with new1; then old2 replaced with new2 \ldots . Matches are found from the left. Matches do not overlap.

count-newlines  str  \[Function\]
Returns the number of \textquoteleft#
ewline\textquoteright characters in string str.

7.2.10 Sequence Comparison

(require 'diff)
diff:edit-length implements the algorithm:
The values returned by diff:edit-length can be used to gauge the degree of match between two sequences.

diff:edits and diff:longest-common-subsequence combine the algorithm with the divide-and-conquer method outlined in:

If the items being sequenced are text lines, then the computed edit-list is equivalent to the output of the diff utility program. If the items being sequenced are words, then it is like the lesser known spiff program.

diff:longest-common-subsequence  array1 array2 p-lim  \[Function\]
array1 and array2 are one-dimensional arrays.
The non-negative integer p-lim, if provided, is maximum number of deletions of the shorter sequence to allow. diff:longest-common-subsequence will return #f if more deletions would be necessary.

diff:longest-common-subsequence returns a one-dimensional array of length (quotient (- (+ len1 len2) (diff:edit-length array1 array2)) 2) holding the longest sequence common to both arrays.

diff:edits  array1 array2 p-lim  \[Function\]
diff:edits  array1 array2  \[Function\]
array1 and array2 are one-dimensional arrays.
The non-negative integer p-lim, if provided, is maximum number of deletions of the shorter sequence to allow. diff:edits will return #f if more deletions would be necessary.
diff:edits returns a vector of length (diff:edit-length array1 array2) composed of a shortest sequence of edits transforming array1 to array2.

Each edit is an integer:

- $k > 0$: Inserts (array-ref array1 (+ -1 j)) into the sequence.
- $k < 0$: Deletes (array-ref array2 (- -1 k)) from the sequence.

**diff:edit-length**

*array1* array2 *p-lim*  
**[Function]**

array1 and array2 are one-dimensional arrays.

The non-negative integer *p-lim*, if provided, is maximum number of deletions of the shorter sequence to allow. diff:edit-length will return #f if more deletions would be necessary.

**diff:edit-length** returns the length of the shortest sequence of edits transforming array1 to array2.

(diff:longest-common-subsequence "fghiejcklm" "fgehijkpqrmlm")  
⇒ "fghijklm"

(diff:edit-length "fghiejcklm" "fgehijkpqrmlm")  
⇒ 6

(diff:edits "fghiejcklm" "fgehijkpqrmlm")  
⇒ #A:fixZ32b(3 -5 -7 8 9 10)  
; e c h p q r

### 7.3 Procedures

Anything that doesn’t fall neatly into any of the other categories winds up here.

#### 7.3.1 Type Coercion

(require 'coerce)

**type-of**

*obj*  
**[Function]**

Returns a symbol name for the type of *obj*.

**coerce**

*obj result-type*  
**[Function]**

Converts and returns *obj* of type char, number, string, symbol, list, or vector to result-type (which must be one of these symbols).

#### 7.3.2 String-Case

(require 'string-case)

**string-upcase**

*str*  
**[Procedure]**

**string-downcase**

*str*  
**[Procedure]**

**string-capitalize**

*str*  
**[Procedure]**

The obvious string conversion routines. These are non-destructive.
### Chapter 7: Other Packages

#### 7.3.3 String Ports

**string-upcase! str**  
[Function]

**string-downcase! str**  
[Function]

**string-capitalise! str**  
[Function]

The destructive versions of the functions above.

**string-ci->symbol str**  
[Function]

Converts string str to a symbol having the same case as if the symbol had been **read**.

**symbol-append obj1 . . .**  
[Function]

Converts obj1 . . . to strings, appends them, and converts to a symbol which is returned. Strings and numbers are converted to read's symbol case; the case of symbol characters is not changed. #f is converted to the empty string (symbol).

**StudlyCapsExpand str delimiter**  
[Function]

**StudlyCapsExpand str** must be a string or character. If absent, delimiter defaults to ‘-’. **StudlyCapsExpand** returns a copy of str where delimiter is inserted between each lower-case character immediately followed by an upper-case character; and between two upper-case characters immediately followed by a lower-case character.

(StudlyCapsExpand "aX" " ") ⇒ "a X"
(StudlyCapsExpand "aX" "..") ⇒ "a..X"
(StudlyCapsExpand "AX") ⇒ "AX"
(StudlyCapsExpand "Ax") ⇒ "Ax"
(StudlyCapsExpand "AXLE") ⇒ "AXLE"
(StudlyCapsExpand "aAXCz") ⇒ "a-AXA-Cz"
(StudlyCapsExpand "aAXCz") ⇒ "a-AXA-Cz"
(StudlyCapsExpand "AAxACz") ⇒ "A-AXA-Cz"
(StudlyCapsExpand "AAxACz") ⇒ "A-AXA-Cz"

### 7.3.4 Line I/O

**(require ’line-i/o)**
read-line  [Function]
read-line port  [Function]
Returns a string of the characters up to, but not including a newline or end of file, updating port to point to the character following the newline. If no characters are available, an end of file object is returned. The port argument may be omitted, in which case it defaults to the value returned by current-input-port.

read-line! string  [Procedure]
read-line! string port  [Procedure]
Fills string with characters up to, but not including a newline or end of file, updating the port to point to the last character read or following the newline if it was read. If no characters are available, an end of file object is returned. If a newline or end of file was found, the number of characters read is returned. Otherwise, #f is returned. The port argument may be omitted, in which case it defaults to the value returned by current-input-port.

write-line string  [Function]
write-line string port  [Function]
Writes string followed by a newline to the given port and returns an unspecified value. The Port argument may be omitted, in which case it defaults to the value returned by current-input-port.

system->line command tmp  [Function]
system->line command  [Function]
command must be a string. The string tmp, if supplied, is a path to use as a temporary file. system->line calls system with command as argument, redirecting stdout to file tmp. system->line returns a string containing the first line of output from tmp. system->line is intended to be a portable method for getting one-line results from programs like pwd, whoami, hostname, which, identify, and cksum. Its behavior when called with programs which generate lots of output is unspecified.

7.3.5 Multi-Processing

(require 'process)

This module implements asynchronous (non-polled) time-sliced multi-processing in the SCM Scheme implementation using procedures alarm and alarm-interrupt. Until this is ported to another implementation, consider it an example of writing schedulers in Scheme.

add-process! proc  [Procedure]
Add proc, which must be a procedure (or continuation) capable of accepting accepting one argument, to the process:queue. The value returned is unspecified. The argument to proc should be ignored. If proc returns, the process is killed.

process:schedule!  [Procedure]
Saves the current process on process:queue and runs the next process from process:queue. The value returned is unspecified.
 kill-process!  [Procedure]
Kills the current process and runs the next process from process:queue. If there are no more processes on process:queue, (slib:exit) is called (see Section 2.4 [System], page 15).

7.3.6 Metric Units
(require 'metric-units)

http://people.csail.mit.edu/jaffer/MIXF

Metric Interchange Format is a character string encoding for numerical values and units which:
• is unambiguous in all locales;
• uses only [TOG] "Portable Character Set" characters matching "Basic Latin" characters in Plane 0 of the Universal Character Set [UCS];
• is transparent to [UTF-7] and [UTF-8] UCS transformation formats;
• is human readable and writable;
• is machine readable and writable;
• incorporates SI prefixes and units;
• incorporates ISO 6093 numbers; and
• incorporates IEC 60027-2 binary prefixes.

In the expression for the value of a quantity, the unit symbol is placed after the numerical value. A dot (PERIOD, ‘.’) is placed between the numerical value and the unit symbol.

Within a compound unit, each of the base and derived symbols can optionally have an attached SI prefix.

Unit symbols formed from other unit symbols by multiplication are indicated by means of a dot (PERIOD, ‘.’) placed between them.

Unit symbols formed from other unit symbols by division are indicated by means of a SOLIDUS (‘/’) or negative exponents. The SOLIDUS must not be repeated in the same compound unit unless contained within a parenthesized subexpression.

The grouping formed by a prefix symbol attached to a unit symbol constitutes a new inseparable symbol (forming a multiple or submultiple of the unit concerned) which can be raised to a positive or negative power and which can be combined with other unit symbols to form compound unit symbols.

The grouping formed by surrounding compound unit symbols with parentheses (‘(’ and ‘)’) constitutes a new inseparable symbol which can be raised to a positive or negative power and which can be combined with other unit symbols to form compound unit symbols.

Compound prefix symbols, that is, prefix symbols formed by the juxtaposition of two or more prefix symbols, are not permitted.

Prefix symbols are not used with the time-related unit symbols min (minute), h (hour), d (day). No prefix symbol may be used with dB (decibel). Only submultiple prefix symbols may be used with the unit symbols L (liter), Np (neper), o (degree), oC (degree Celsius),
rad (radian), and sr (steradian). Submultiple prefix symbols may not be used with the unit symbols t (metric ton), r (revolution), or Bd (baud).

A unit exponent follows the unit, separated by a CIRCUMFLEX (‘^’). Exponents may be positive or negative. Fractional exponents must be parenthesized.

### 7.3.6.1 SI Prefixes

<table>
<thead>
<tr>
<th>Factor</th>
<th>Name</th>
<th>Symbol</th>
<th>Factor</th>
<th>Name</th>
<th>Symbol</th>
</tr>
</thead>
<tbody>
<tr>
<td>1e24</td>
<td>yotta</td>
<td>Y</td>
<td>1e-1</td>
<td>deci</td>
<td>d</td>
</tr>
<tr>
<td>1e21</td>
<td>zetta</td>
<td>Z</td>
<td>1e-2</td>
<td>centi</td>
<td>c</td>
</tr>
<tr>
<td>1e18</td>
<td>exa</td>
<td>E</td>
<td>1e-3</td>
<td>milli</td>
<td>m</td>
</tr>
<tr>
<td>1e15</td>
<td>peta</td>
<td>P</td>
<td>1e-6</td>
<td>micro</td>
<td>u</td>
</tr>
<tr>
<td>1e12</td>
<td>tera</td>
<td>T</td>
<td>1e-9</td>
<td>nano</td>
<td>n</td>
</tr>
<tr>
<td>1e9</td>
<td>giga</td>
<td>G</td>
<td>1e-12</td>
<td>pico</td>
<td>p</td>
</tr>
<tr>
<td>1e6</td>
<td>mega</td>
<td>M</td>
<td>1e-15</td>
<td>femto</td>
<td>f</td>
</tr>
<tr>
<td>1e3</td>
<td>kilo</td>
<td>k</td>
<td>1e-18</td>
<td>atto</td>
<td>a</td>
</tr>
<tr>
<td>1e2</td>
<td>hecto</td>
<td>h</td>
<td>1e-21</td>
<td>zepto</td>
<td>z</td>
</tr>
<tr>
<td>1e1</td>
<td>deka</td>
<td>da</td>
<td>1e-24</td>
<td>yocto</td>
<td>y</td>
</tr>
</tbody>
</table>

### 7.3.6.2 Binary Prefixes

These binary prefixes are valid only with the units B (byte) and bit. However, decimal prefixes can also be used with bit; and decimal multiple (not submultiple) prefixes can also be used with B (byte).

<table>
<thead>
<tr>
<th>Factor (power-of-2)</th>
<th>Name</th>
<th>Symbol</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.152921504606846976e18 (2^-60)</td>
<td>exbi</td>
<td>Ei</td>
</tr>
<tr>
<td>1.125899906842624e15 (2^-50)</td>
<td>pebi</td>
<td>Pi</td>
</tr>
<tr>
<td>1.099511627748e12 (2^-40)</td>
<td>tebi</td>
<td>Ti</td>
</tr>
<tr>
<td>1.073741824e9 (2^-30)</td>
<td>gibi</td>
<td>Gi</td>
</tr>
<tr>
<td>1.048576e6 (2^-20)</td>
<td>mebi</td>
<td>Mi</td>
</tr>
<tr>
<td>1.024e3 (2^-10)</td>
<td>kibi</td>
<td>Ki</td>
</tr>
</tbody>
</table>

### 7.3.6.3 Unit Symbols

<table>
<thead>
<tr>
<th>Type of Quantity</th>
<th>Name</th>
<th>Symbol</th>
<th>Equivalent</th>
</tr>
</thead>
<tbody>
<tr>
<td>time</td>
<td>second</td>
<td>s</td>
<td></td>
</tr>
<tr>
<td>time</td>
<td>minute</td>
<td>min</td>
<td>= 60.s</td>
</tr>
<tr>
<td>time</td>
<td>hour</td>
<td>h</td>
<td>= 60.min</td>
</tr>
<tr>
<td>time</td>
<td>day</td>
<td>d</td>
<td>= 24.h</td>
</tr>
<tr>
<td>frequency</td>
<td>hertz</td>
<td>Hz</td>
<td>s^-1</td>
</tr>
<tr>
<td>signaling rate</td>
<td>baud</td>
<td>Bd</td>
<td>s^-1</td>
</tr>
<tr>
<td>length</td>
<td>meter</td>
<td>m</td>
<td></td>
</tr>
<tr>
<td>volume</td>
<td>liter</td>
<td>L</td>
<td>dm^-3</td>
</tr>
<tr>
<td>plane angle</td>
<td>radian</td>
<td>rad</td>
<td></td>
</tr>
<tr>
<td>solid angle</td>
<td>steradian</td>
<td>sr</td>
<td>rad^-2</td>
</tr>
<tr>
<td>plane angle</td>
<td>revolution</td>
<td>* r</td>
<td>= 6.283185307179586.rad</td>
</tr>
</tbody>
</table>
plane angle  
information capacity  
information capacity  
mass  
mass  
mass  
amount of substance  
catalytic activity  
thermodynamic temperature  
centigrade temperature  
luminous intensity  
luminous flux  
illuminance  
force  
pressure, stress  
energy, work, heat  
energy  
power, radiant flux  
logarithm of power ratio  
logarithm of power ratio  
electric current  
electric charge  
electric potential, EMF  
capacitance  
electric resistance  
electric conductance  
magnetic flux  
magnetic flux density  
inductance  
radionuclide activity  
absorbed dose energy  
dose equivalent  

* The formulas are:

- \( r/\text{rad} = 8 \times \text{atan}(1) \)
- \( o/r = 1 / 360 \)
- \( \text{dB}/\text{Np} = \ln(10) / 20 \)

si:conversion-factor \( \text{to-unit \ from-unit} \)

> If the strings \( \text{from-unit} \) and \( \text{to-unit} \) express valid unit expressions for quantities of the same unit-dimensions, then the value returned by \text{si:conversion-factor} will be such that multiplying a numerical value expressed in \( \text{from-units} \) by the returned conversion factor yields the numerical value expressed in \( \text{to-units} \).

Otherwise, \text{si:conversion-factor} returns:

-3 if neither \( \text{from-unit} \) nor \( \text{to-unit} \) is a syntactically valid unit.
-2 if \( \text{from-unit} \) is not a syntactically valid unit.
-1 if to-unit is not a syntactically valid unit.

0 if linear conversion (by a factor) is not possible.

\[
\begin{align*}
\text{(si:conversion-factor "km/s" "m/s" ) } & \Rightarrow 0.001 \\
\text{(si:conversion-factor "N" "m/s" ) } & \Rightarrow 0 \\
\text{(si:conversion-factor "m\degree C" "\degree C" ) } & \Rightarrow 1000 \\
\text{(si:conversion-factor "m\degree K" "\degree C" ) } & \Rightarrow 0 \\
\text{(si:conversion-factor "rad" "\degree" ) } & \Rightarrow 0.0174533 \\
\text{(si:conversion-factor "K" "\degree C" ) } & \Rightarrow 0 \\
\text{(si:conversion-factor "K" "K" ) } & \Rightarrow 1 \\
\text{(si:conversion-factor "\degree K" "\degree K" ) } & \Rightarrow -3 \\
\text{(si:conversion-factor "" "s/s" ) } & \Rightarrow 1 \\
\text{(si:conversion-factor "km/h" "mph" ) } & \Rightarrow -2
\end{align*}
\]

7.4 Standards Support

7.4.1 RnRS

The r2rs, r3rs, r4rs, and r5rs features attempt to provide procedures and macros to bring a Scheme implementation to the desired version of Scheme.

r2rs [Feature]

Requires features implementing procedures and optional procedures specified by Revised^2 Report on the Algorithmic Language Scheme; namely rev3-procedures and rev2-procedures.

r3rs [Feature]

Requires features implementing procedures and optional procedures specified by Revised^3 Report on the Algorithmic Language Scheme; namely rev3-procedures.

Note: SLIB already mandates the r3rs procedures which can be portably implemented in r4rs implementations.

r4rs [Feature]

Requires features implementing procedures and optional procedures specified by Revised^4 Report on the Algorithmic Language Scheme; namely rev4-optional-procedures.

r5rs [Feature]

Requires features implementing procedures and optional procedures specified by Revised^5 Report on the Algorithmic Language Scheme; namely values, macro, and eval.

7.4.2 With-File

(require 'with-file)

\[
\begin{align*}
\text{with-input-from-file file thunk } & \quad \text{[Function]} \\
\text{with-output-to-file file thunk } & \quad \text{[Function]}
\end{align*}
\]

Description found in R4RS.
7.4.3 Transcripts

(require 'transcript)

transcript-on filename  [Function]
transcript-off filename  [Function]

Redefines read-char, read, write-char, write, display, and newline.

7.4.4 Rev2 Procedures

(require 'rev2-procedures)

The procedures below were specified in the Revised^2 Report on Scheme. N.B.: The symbols 1+ and -1+ are not R4RS syntax. Scheme->C, for instance, chokes on this module.

substring-move-left! string1 start1 end1 string2 start2  [Procedure]
substring-move-right! string1 start1 end1 string2 start2  [Procedure]
string1 and string2 must be a string, and start1, start2 and end1 must be exact integers satisfying

\[
0 \leq start1 \leq end1 \leq (\text{string-length string1}) \\
0 \leq start2 \leq end1 - start1 + start2 \leq (\text{string-length string2})
\]

substring-move-left! and substring-move-right! store characters of string1 beginning with index start1 (inclusive) and ending with index end1 (exclusive) into string2 beginning with index start2 (inclusive).

substring-move-left! stores characters in time order of increasing indices. substring-move-right! stores characters in time order of increasing indeces.

substring-fill! string start end char  [Procedure]
Fills the elements start–end of string with the character char.

string-null? str  [Function]
\equiv (= 0 (\text{string-length str}))

append! pair1 ...  [Procedure]
Destructively appends its arguments. Equivalent to nconc.

1+ n  [Function]
Adds 1 to n.

-1+ n  [Function]
Subtracts 1 from n.

<?  [Function]
<=?  [Function]
=?  [Function]
>?  [Function]
>=?  [Function]

These are equivalent to the procedures of the same name but without the trailing ‘?’. 
7.4.5 Rev4 Optional Procedures

(require 'rev4-optional-procedures)

For the specification of these optional procedures, See Section “Standard procedures” in Revised(4) Scheme.

list-tail l p          [Function]
string-copy            [Function]
string-fill! s obj     [Procedure]
vector-fill! s obj     [Procedure]

7.4.6 Multi-argument / and -

(require 'multiarg/and-)

For the specification of these optional forms, See Section “Numerical operations” in Revised(4) Scheme.

/ dividend divisor1 . . .     [Function]
- minuend subtrahend1 . . .    [Function]

7.4.7 Multi-argument Apply

(require 'multiarg-apply)

For the specification of this optional form, See Section “Control features” in Revised(4) Scheme.

apply proc arg1 . . .        [Function]

7.4.8 Rationalize

(require 'rationalize)

rationalize x e             [Function]

Computes the correct result for exact arguments (provided the implementation supports exact rational numbers of unlimited precision); and produces a reasonable answer for inexact arguments when inexact arithmetic is implemented using floating-point.

Rationalize has limited use in implementations lacking exact (non-integer) rational numbers. The following procedures return a list of the numerator and denominator.

find-ratio x e             [Function]

find-ratio returns the list of the simplest numerator and denominator whose quotient differs from x by no more than e.

(find-ratio 3/97 .0001) ⇒ (3 97)
(find-ratio 3/97 .001)  ⇒ (1 32)
function \texttt{find-ratio-between}\ x\ y

\texttt{find-ratio-between} returns the list of the \textit{simplest} numerator and denominator between \texttt{x} and \texttt{y}.

\[
\begin{align*}
\text{(find-ratio-between } & \frac{2}{7} \ 3/5) \Rightarrow (1 \ 2) \\
\text{(find-ratio-between } & \frac{-3}{5} -\frac{2}{7}) \Rightarrow (-1 \ 2)
\end{align*}
\]

7.4.9 Promises

\begin{verbatim}
(require 'promise)
\end{verbatim}

\texttt{make-promise} proc

\texttt{force} promise

\begin{verbatim}
(require 'delay)
\end{verbatim}

\texttt{force} and \texttt{delay}:

\texttt{delay} obj

Change occurrences of \texttt{(delay expression)} to

\[
\text{(make-promise (lambda () expression))}
\]

\begin{verbatim}
(see Section “Control features” in Revised(4) Scheme).
\end{verbatim}

7.4.10 Dynamic-Wind

\begin{verbatim}
(require 'dynamic-wind)
\end{verbatim}

This facility is a generalization of Common LISP \texttt{unwind-protect}, designed to take into account the fact that continuations produced by \texttt{call-with-current-continuation} may be reentered.

\texttt{dynamic-wind} thunk1 thunk2 thunk3

The arguments \texttt{thunk1}, \texttt{thunk2}, and \texttt{thunk3} must all be procedures of no arguments (thunks).

\texttt{dynamic-wind} calls \texttt{thunk1}, \texttt{thunk2}, and then \texttt{thunk3}. The value returned by \texttt{thunk2} is returned as the result of \texttt{dynamic-wind}. \texttt{thunk3} is also called just before control leaves the dynamic context of \texttt{thunk2} by calling a continuation created outside that context. Furthermore, \texttt{thunk1} is called before reentering the dynamic context of \texttt{thunk2} by calling a continuation created inside that context. (Control is inside the context of \texttt{thunk2} if \texttt{thunk2} is on the current return stack).

\textbf{Warning:} There is no provision for dealing with errors or interrupts. If an error or interrupt occurs while using \texttt{dynamic-wind}, the dynamic environment will be that in effect at the time of the error or interrupt.

7.4.11 Eval

\begin{verbatim}
(require 'eval)
\end{verbatim}

\texttt{eval} expression environment-specifier

Evaluates \texttt{expression} in the specified environment and returns its value. \texttt{Expression} must be a valid Scheme expression represented as data, and \texttt{environment-specifier}
must be a value returned by one of the three procedures described below. Implementations may extend eval to allow non-expression programs (definitions) as the first argument and to allow other values as environments, with the restriction that eval is not allowed to create new bindings in the environments associated with null-environment or scheme-report-environment.

\[
\text{(eval '(* 7 3) (scheme-report-environment 5))} \quad \Rightarrow 21
\]

\[
\text{(let ((f (eval '(lambda (f x) (f x x)) (null-environment)))) (f + 10))} \quad \Rightarrow 20
\]

\text{scheme-report-environment version \quad [Function]}
\text{null-environment version \quad [Function]}
\text{null-environment \quad [Function]}

Version must be an exact non-negative integer \( n \) corresponding to a version of one of the Revised\(^n\) Reports on Scheme. Scheme-report-environment returns a specifier for an environment that contains the set of bindings specified in the corresponding report that the implementation supports. Null-environment returns a specifier for an environment that contains only the (syntactic) bindings for all the syntactic keywords defined in the given version of the report.

Not all versions may be available in all implementations at all times. However, an implementation that conforms to version \( n \) of the Revised\(^n\) Reports on Scheme must accept version \( n \). An error is signalled if the specified version is not available.

The effect of assigning (through the use of eval) a variable bound in a scheme-report-environment (for example car) is unspecified. Thus the environments specified by scheme-report-environment may be immutable.

\text{interaction-environment \quad [Function]}

This optional procedure returns a specifier for the environment that contains implementation-defined bindings, typically a superset of those listed in the report. The intent is that this procedure will return the environment in which the implementation would evaluate expressions dynamically typed by the user.

Here are some more eval examples:

\[
\text{(require 'eval)} \Rightarrow \text{#<unspecified>}
\text{(define car 'volvo)} \Rightarrow \text{#<unspecified>}
\text{car} \Rightarrow \text{volvo}
\text{(eval 'car (interaction-environment))} \Rightarrow \text{volvo}
\text{(eval 'car (scheme-report-environment 5))} \Rightarrow \text{#<primitive-procedure car>}
\]
(eval '(eval 'car (interaction-environment))
  (scheme-report-environment 5))
⇒ volvo
(eval '(eval '(set! car 'buick) (interaction-environment))
  (scheme-report-environment 5))
⇒ #\<unspecified>
car
⇒ buick
(eval 'car (scheme-report-environment 5))
⇒ #\<primitive-procedure car>
(eval '(eval 'car (interaction-environment))
  (scheme-report-environment 5))
⇒ buick

7.4.12 Values
(require 'values)

values obj ... [Function]
values takes any number of arguments, and passes (returns) them to its continuation.

call-with-values thunk proc [Function]
thunk must be a procedure of no arguments, and proc must be a procedure. call-with-values calls thunk with a continuation that, when passed some values, calls proc with those values as arguments.

Except for continuations created by the call-with-values procedure, all continuations take exactly one value, as now; the effect of passing no value or more than one value to continuations that were not created by the call-with-values procedure is unspecified.

7.4.13 SRFI
(require 'srfi)

Implements Scheme Request For Implementation (SRFI) as described at http://srfi.schemers.org/

cond-expand <clause1> <clause2> ... [Macro]
Syntax: Each <clause> should be of the form
  (<feature> <expression1> ...)
where <feature> is a boolean expression composed of symbols and ‘and’, ‘or’, and ‘not’ of boolean expressions. The last <clause> may be an “else clause,” which has the form
  (else <expression1> <expression2> ...).

The first clause whose feature expression is satisfied is expanded. If no feature expression is satisfied and there is no else clause, an error is signaled.

SLIB cond-expand is an extension of SRFI-0, http://srfi.schemers.org/srfi-0/srfi-0.html.

• SRFI-2 Section 3.12 [Guarded LET* special form], page 34
Chapter 7: Other Packages

• SRFI-8 Section 3.11 [Binding to multiple values], page 34
• SRFI-9 Section 3.8 [Define-Record-Type], page 33
• SRFI-11 Section 3.11 [Binding to multiple values], page 34
• SRFI-23 (define error sib:error)
• SRFI-28 Section 4.2 [Format], page 46
• SRFI-39 Section 3.10 [Parameter Objects], page 34
• SRFI-47 Section 7.1.1 [Arrays], page 196
• SRFI-59 Section 2.1 [Vicinity], page 11
• SRFI-60 Section 5.1 [Bit-Twiddling], page 103
• SRFI-61 Section 3.13 [Guarded COND Clause], page 34
• SRFI-63 Section 7.1.1 [Arrays], page 196
• SRFI-94 Section 5.3 [Irrational Integer Functions], page 108 and Section 5.4 [Irrational Real Functions], page 109
• SRFI-95 Section 7.2.4 [Sorting], page 233
• SRFI-96 Chapter 2 [Universal SLIB Procedures], page 11

7.4.13.1 SRFI-1

(require 'srfi-1)

Implements the SRFI-1 list-processing library as described at http://srfi.schemers.org/srfi-1/srfi-1.html

Constructors

xcons d a
   (define (xcons d a) (cons a d)).

list-tabulate len proc
   Returns a list of length len. Element i is (proc i) for 0 < i < len.

cons* obj1 obj2

list-copy flist

iota count start step
iota count start
iota count
   Returns a list of count numbers: (start, start+step, . . . , start+(count-1)*step).

circular-list obj1 obj2 . . .
   Returns a circular list of obj1, obj2, . . .

Predicates

proper-list? obj

circular-list? x
dotted-list? obj
null-list? obj
not-pair? obj
list= =pred list ...

Selectors
first pair
second pair
third pair
fourth pair
fifth pair
sixth pair
seventh pair
eighth pair
ninth pair
tenth pair
car+cdr pair
drop lst k
take lst k
take! lst k
take-right lst k
drop-right lst k
drop-right! lst k
split-at lst k
split-at! lst k
last lst k ...

Miscellaneous
length+ clist
concatenate lists
concatenate! lists
reverse! lst
append-reverse rev-head tail
append-reverse! rev-head tail
zip list1 list2 ...
unzip1 lst
unzip2 lst
unzip3 lst
unzip4 lst
unzip5 lst
count pred list1 list2 ...
Fold and Unfold

fold  \texttt{kons knil clist1 clist2 \ldots} \texttt{[Function]}
fold-right  \texttt{kons knil clist1 clist2 \ldots} \texttt{[Function]}
pair-fold  \texttt{kons knil clist1 clist2 \ldots} \texttt{[Function]}
pair-fold-right  \texttt{kons knil clist1 clist2 \ldots} \texttt{[Function]}
reduce  \texttt{arg \ldots} \texttt{[Function]}
map!  \texttt{f clist1 clist2 \ldots} \texttt{[Procedure]}
pair-for-each  \texttt{f clist1 clist2 \ldots} \texttt{[Function]}

Filtering and Partitioning

filter  \texttt{pred list} \texttt{[Function]}
filter!  \texttt{pred list} \texttt{[Procedure]}
partition  \texttt{pred list} \texttt{[Function]}
remove  \texttt{pred list} \texttt{[Function]}
partition!  \texttt{pred list} \texttt{[Procedure]}
remove!  \texttt{pred list} \texttt{[Procedure]}

Searching

find  \texttt{pred clist} \texttt{[Function]}
find-tail  \texttt{pred clist} \texttt{[Function]}
span  \texttt{pred list} \texttt{[Function]}
span!  \texttt{pred list} \texttt{[Procedure]}
break  \texttt{pred list} \texttt{[Function]}
break!  \texttt{pred list} \texttt{[Procedure]}
any  \texttt{pred clist1 clist2 \ldots} \texttt{[Function]}
list-index  \texttt{pred clist1 clist2 \ldots} \texttt{[Function]}
member  \texttt{obj list =} \texttt{[Function]}
member  \texttt{obj list} \texttt{[Function]}

Deleting

delete-duplicates  \texttt{x list =} \texttt{[Function]}
delete-duplicates  \texttt{x list} \texttt{[Function]}
delete-duplicates!  \texttt{x list =} \texttt{[Procedure]}
delete-duplicates!  \texttt{x list} \texttt{[Procedure]}

Chapter 7: Other Packages
Association lists

assoc  obj  alist  pred  [Function]
assoc  obj  alist  [Function]
alist-cons  key  datum  alist  [Function]
alist-copy  alist  [Function]
alist-delete  key  alist =  [Function]
alist-delete  key  alist  [Function]
alist-delete!  key  alist =  [Procedure]
alist-delete!  key  alist  [Procedure]

Set operations

lset<= = list1 . . .  [Function]
   Determine if a transitive subset relation exists between the lists list1 . . ., using = to
determine equality of list members.
lset= = list1 list2 . . .  [Function]
lset-adjoin  list  elt1 . . .  [Function]
lset-union = list1 . . .  [Function]
lset-intersection = list1 list2 . . .  [Function]
lset-difference = list1 list2 . . .  [Function]
lset-xor = list1 . . .  [Function]
lset-diff+intersection = list1 list2 . . .  [Function]
   These are linear-update variants. They are allowed, but not required, to use the cons cells
   in their first list parameter to construct their answer. lset-union! is permitted to recycle
   cons cells from any of its list arguments.
lset-intersection! = list1 list2 . . .  [Procedure]
lset-difference! = list1 list2 . . .  [Procedure]
lset-union! = list1 list2 . . .  [Procedure]
lset-xor! = list1 . . .  [Procedure]
lset-diff+intersection! = list1 list2 . . .  [Procedure]

7.5 Session Support

If (provided? 'abort):

abort  [Function]
   Resumes the top level Read-Eval-Print loop. If provided, abort is used by the break
and debug packages.
7.5.1 Repl

(require 'repl)

Here is a read-eval-print-loop which, given an eval, evaluates forms.

repl:top-level repl:eval [Procedure]
reads, repl:evals and writes expressions from (current-input-port) to (current-output-port) until an end-of-file is encountered. load, slib:eval, slib:error, and repl:quit dynamically bound during repl:top-level.

repl:quit [Procedure]
Exits from the invocation of repl:top-level.

The repl: procedures establish, as much as is possible to do portably, a top level environment supporting macros. repl:top-level uses dynamic-wind to catch error conditions and interrupts. If your implementation supports this you are all set.

Otherwise, if there is some way your implementation can catch error conditions and interrupts, then have them call slib: error. It will display its arguments and reenter repl:top-level. slib:error dynamically bound by repl:top-level.

To have your top level loop always use macros, add any interrupt catching lines and the following lines to your Scheme init file:

(require 'macro)
(require 'repl)
(repl:top-level macro:eval)

7.5.2 Quick Print

(require 'qp)

When displaying error messages and warnings, it is paramount that the output generated for circular lists and large data structures be limited. This section supplies a procedure to do this. It could be much improved.

Notice that the necessity for truncating output eliminates Common-Lisp’s Section 4.2 [Format], page 46 from consideration; even when variables *print-level* and *print-level* are set, huge strings and bit-vectors are not limited.

qp arg1 ... [Procedure]
qpn arg1 ... [Procedure]
qpr arg1 ... [Procedure]
qp writes its arguments, separated by spaces, to (current-output-port). qp compresses printing by substituting ‘...’ for substructure it does not have sufficient room to print. qp is like qp but outputs a newline before returning. qpr is like qp except that it returns its last argument.

*qp-width* [Variable]
*qp-width* is the largest number of characters that qp should use. If *qp-width* is #f, then all items will be written. If *qp-width* is 0, then all items except procedures will be written; procedures will be indicated by ‘#[proc]’.
7.5.3 Debug

(require 'debug)

Requiring debug automatically requires trace and break.

An application with its own datatypes may want to substitute its own printer for qp. This example shows how to do this:

```
(define qpn (lambda args) ...)
(provide 'qp)
(require 'debug)
```

trace-all file ...  [Procedure]
Traces (see Section 7.5.5 [Trace], page 259) all procedures defined at top-level in 'file' ....

track-all file ...  [Procedure]
Tracks (see Section 7.5.5 [Trace], page 259) all procedures defined at top-level in 'file' ....

stack-all file ...  [Procedure]
Stacks (see Section 7.5.5 [Trace], page 259) all procedures defined at top-level in 'file' ....

break-all file ...  [Procedure]
Breakpoints (see Section 7.5.4 [Breakpoints], page 258) all procedures defined at top-level in 'file' ....

7.5.4 Breakpoints

(require 'break)

init-debug  [Function]
If your Scheme implementation does not support break or abort, a message will appear when you (require 'break) or (require 'debug) telling you to type (init-debug). This is in order to establish a top-level continuation. Typing (init-debug) at top level sets up a continuation for break.

breakpoint arg1 ...  [Function]
Returns from the top level continuation and pushes the continuation from which it was called on a continuation stack.

continue  [Function]
Pops the topmost continuation off of the continuation stack and returns an unspecified value to it.

continue arg1 ...  [Function]
Pops the topmost continuation off of the continuation stack and returns arg1 ... to it.
Chapter 7: Other Packages

break proc1 . . . [Macro]
Redefines the top-level named procedures given as arguments so that
breakpoint is called before calling proc1 . . . break
With no arguments, makes sure that all the currently broken identifiers are broken
(even if those identifiers have been redefined) and returns a list of the broken identifiers.

unbreak proc1 . . . [Macro]
Turns breakpoints off for its arguments. unbreak
With no arguments, unbreaks all currently broken identifiers and returns a list of
these formerly broken identifiers.

These are procedures for breaking. If defmacros are not natively supported by your
implementation, these might be more convenient to use.

breakf proc [Function]
breakf proc name [Function]
To break, type
(set! symbol (breakf symbol))
or
(set! symbol (breakf symbol 'symbol))
or
(define symbol (breakf function))
or
(define symbol (breakf function 'symbol))

unbreakf proc [Function]
To unbreak, type
(set! symbol (unbreakf symbol))

7.5.5 Tracing
(require 'trace)
This feature provides three ways to monitor procedure invocations:

stack Pushes the procedure-name when the procedure is called; pops when it returns.

track Pushes the procedure-name and arguments when the procedure is called; pops
when it returns.

trace Pushes the procedure-name and prints ‘CALL procedure-name arg1 . . .’ when
the procedure is called; pops and prints ‘RETN procedure-name value’ when the
procedure returns.

depend:max-count [Variable]
If a traced procedure calls itself or untraced procedures which call it, stack, track,
and trace will limit the number of stack pushes to depend:max-count.
print-call-stack [Function]
print-call-stack port [Function]
  Prints the call-stack to port or the current-error-port.

trace proc1 ... [Macro]
Traces the top-level named procedures given as arguments. trace
  With no arguments, makes sure that all the currently traced identifiers are traced
  (even if those identifiers have been redefined) and returns a list of the traced identifiers.

track proc1 ... [Macro]
Traces the top-level named procedures given as arguments. track
  With no arguments, makes sure that all the currently tracked identifiers are tracked
  (even if those identifiers have been redefined) and returns a list of the tracked identifiers.

stack proc1 ... [Macro]
Traces the top-level named procedures given as arguments. stack
  With no arguments, makes sure that all the currently stacked identifiers are stacked
  (even if those identifiers have been redefined) and returns a list of the stacked identifiers.

untrace proc1 ... [Macro]
Turns tracing, tracking, and off for its arguments. untrace
  With no arguments, untraces all currently traced identifiers and returns a list of these
  formerly traced identifiers.

untrack proc1 ... [Macro]
Turns tracing, tracking, and off for its arguments. untrack
  With no arguments, untracks all currently tracked identifiers and returns a list of these
  formerly tracked identifiers.

unstack proc1 ... [Macro]
Turns tracing, stacking, and off for its arguments. unstack
  With no arguments, unstacks all currently stacked identifiers and returns a list of these
  formerly stacked identifiers.

These are procedures for tracing. If defmacros are not natively supported by your
implementation, these might be more convenient to use.

tracef proc [Function]
tracef proc name [Function]
trackf proc [Function]
trackf proc name [Function]
stackf proc [Function]
stackf proc name [Function]
  To trace, type

  (set! symbol (tracef symbol))

  or
(set! symbol (tracef symbol 'symbol))

or

(define symbol (tracef function))

or

(define symbol (tracef function 'symbol))

untracef proc  
Removes tracing, tracking, or stacking for proc. To untrace, type

(set! symbol (untracef symbol))

7.6 System Interface

If (provided? 'getenv):

getenv name  
Looks up name, a string, in the program environment. If name is found a string of its value is returned. Otherwise, #f is returned.

If (provided? 'system):

system command-string  
Executes the command-string on the computer and returns the integer status code. This behaves the same as the POSIX system call.

If (provided? 'program-arguments):

program-arguments  
Returns a list of strings, the first of which is the program name followed by the command-line arguments.

7.6.1 Directories

(require 'directory)

current-directory  
current-directory returns a string containing the absolute file name representing the current working directory. If this string cannot be obtained, #f is returned.

If current-directory cannot be supported by the platform, then #f is returned.

make-directory name  
Creates a sub-directory name of the current-directory. If successful, make-directory returns #t; otherwise #f.

directory-for-each proc directory  
proc must be a procedure taking one argument. 'Directory-For-Each' applies proc to the (string) name of each file in directory. The dynamic order in which proc is applied to the filenames is unspecified. The value returned by 'directory-for-each' is unspecified.
**directory-for-each proc directory pred**  
Applies proc only to those filenames for which the procedure pred returns a non-false value.

**directory-for-each proc directory match**  
Applies proc only to those filenames for which (filename:match?? match) would return a non-false value (see Section “Filenames” in SLIB).

```
(require 'directory)
(directory-for-each print "." "[A-Z]*.scm")
⊣
"Bev2slib.scm"
"Template.scm"
```

**directory*-for-each proc path-glob**  
*path-glob* is a pathname whose last component is a (wildcard) pattern (see Section “Filenames” in SLIB). proc must be a procedure taking one argument. ‘directory*-for-each’ applies proc to the (string) name of each file in the current directory. The dynamic order in which proc is applied to the filenames is unspecified. The value returned by ‘directory*-for-each’ is unspecified.

### 7.6.2 Transactions

If system is provided by the Scheme implementation, the transact package provides functions for file-locking and file-replacement transactions.

```
(require 'transact)
```

**File Locking**

Unix file-locking is focussed on write permissions for segments of a existing file. While this might be employed for (binary) database access, it is not used for everyday contention (between users) for text files.

Microsoft has several file-locking protocols. Their model denies write access to a file if any reader has it open. This is too restrictive. Write access is denied even when the reader has reached end-of-file. And tracking read access (which is much more common than write access) causes havoc when remote hosts crash or disconnect.

It is bizarre that the concept of multi-user contention for modifying files has not been adequately addressed by either of the large operating system development efforts. There is further irony that both camps support contention detection and resolution only through weak conventions of some their document editing programs.

The file-lock procedures implement a transaction method for file replacement compatible with the methods used by the GNU emacs text editor on Unix systems and the Microsoft Word editor.

Both protocols employ what I term a certificate containing the user, hostname, time, and (on Unix) process-id. Intent to replace file is indicated by adding to file’s directory a certificate object whose name is derived from file.

The Microsoft Word certificate is contained in a 162 byte file named for the visited file with a ‘~$’ prefix. Emacs/Unix creates a symbolic link to a certificate named for the visited file...
prefixed with ‘.#’. Because Unix systems can import Microsoft file systems, these routines maintain and check both Emacs and Word certificates.

**file-lock-owner**  
*path*  
[Function]  
Returns the string ‘user@hostname’ associated with the lock owner of file *path* if locked; and #f otherwise.

**file-lock!**  
*path email*  
[Procedure]  
*path* must be a string naming the file to be locked. If supplied, *email* must be a string formatted as ‘user@hostname’. If absent, *email* defaults to the value returned by *user-email-address*.

If *path* is already locked, then **file-lock!** returns ‘#f’. If *path* is unlocked, then **file-lock!** returns the certificate string associated with the new lock for file *path*.

**file-unlock!**  
*path certificate*  
[Procedure]  
*path* must be a string naming the file to be unlocked. *certificate* must be the string returned by **file-lock!** for *path*.

If *path* is locked with *certificate*, then **file-unlock!** removes the locks and returns ‘#t’. Otherwise, **file-unlock!** leaves the file system unaltered and returns ‘#f’.

**describe-file-lock**  
*path prefix*  
[Function]  
*path* must be a string naming a file. Optional argument *prefix* is a string printed before each line of the message. **describe-file-lock** prints to (current-error-port) that *path* is locked for writing and lists its lock-files.

```scheme
(describe-file-lock "my.txt" ">> ")
-|  
  >> "my.txt" is locked for writing by 'luser@no.com.4829:1200536423'
  >> (lock files are "~$my.txt" and ".#my.txt")
```

### File Transactions

**emacs:backup-name**  
*path backup-style*  
[Function]  
*path* must be a string. *backup-style* must be a symbol. Depending on *backup-style*, **emacs:backup-name** returns:

- none  
  #f
- simple  
  the string "path~"
- numbered  
  the string "path."n~", where *n* is one greater than the highest number appearing in a filename matching "path."#~". *n* defaults to 1 when no filename matches.
- existing  
  the string "path."n~" if a numbered backup already exists in this directory; otherwise. "path~"
- orig  
  the string "path.orig"
- bak  
  the string "path.bak"
transact-file-replacement proc path backup-style certificate [Function]
transact-file-replacement proc path backup-style [Function]
transact-file-replacement proc path [Function]

path must be a string naming an existing file. backup-style is one of the symbols none, simple, numbered, existing, orig, bak or #f; with meanings described above; or a string naming the location of a backup file. backup-style defaults to #f. If supplied, certificate is the certificate with which path is locked.

proc must be a procedure taking two string arguments:

- path, the original filename (to be read); and
- a temporary file-name.

If path is locked by other than certificate, or if certificate is supplied and path is not locked, then transact-file-replacement returns #f. If certificate is not supplied, then transact-file-replacement creates temporary (Emacs and Word) locks for path during the transaction. The lock status of path will be restored before transact-file-replacement returns.

transact-file-replacement calls proc with path (which should not be modified) and a temporary file path to be written. If proc returns any value other than #t, then the file named by path is not altered and transact-file-replacement returns #f. Otherwise, emacs:backup-name is called with path and backup-style. If it returns a string, then path is renamed to it.

Finally, the temporary file is renamed path. transact-file-replacement returns #t if path was successfully replaced; and #f otherwise.

Identification

user-email-address [Function]
user-email-address returns a string of the form ‘username@hostname’. If this email address cannot be obtained, #f is returned.

7.6.3 CVS

(require 'cvs)

cvs-files directory/ [Function]
Returns a list of the local pathnames (with prefix directory/) of all CVS controlled files in directory/ and in directory/’s subdirectories.

cvs-directories directory/ [Function]
Returns a list of all of directory/ and all directory/’s CVS controlled subdirectories.

cvs-root path/ [Function]
Returns the (string) contents of path/CVS/Root; or (getenv "CVSROOT") if Root doesn’t exist.

cvs-repository directory/ [Function]
Returns the (string) contents of directory/CVS/Root appended with directory/CVS/Repository; or #f if directory/CVS/Repository doesn’t exist.
Chapter 7: Other Packages

7.7 Extra-SLIB Packages

Several Scheme packages have been written using SLIB. There are several reasons why a package might not be included in the SLIB distribution:

- Because it requires special hardware or software which is not universal.
- Because it is large and of limited interest to most Scheme users.
- Because it has copying terms different enough from the other SLIB packages that its inclusion would cause confusion.
- Because it is an application program, rather than a library module.
- Because I have been too busy to integrate it.

Once an optional package is installed (and an entry added to \texttt{\texttt{目录}}), the \texttt{require} mechanism allows it to be called up and used as easily as any other SLIB package. Some optional packages (for which \texttt{\texttt{目录}} already has entries) available from SLIB sites are:

\textbf{SLIB-PSD} is a portable debugger for Scheme (requires emacs editor).

\begin{verbatim}
http://groups.csail.mit.edu/mac/ftpdir/scm/slib-psd1-3.tar.gz
\end{verbatim}

With PSD, you can run a Scheme program in an Emacs buffer, set breakpoints, single step evaluation and access and modify the program’s variables. It works by instrumenting the original source code, so it should run with any R4RS compliant Scheme. It has been tested with SCM, Elk 1.5, and the sci interpreter in the Scheme->C system, but should work with other Schemes with a minimal amount of porting, if at all. Includes documentation and user’s manual. Written by Pertti Kellomäki, the Lisp Pointers article describing PSD (Lisp Pointers VI(1):15-23, January-March 1993) is available at

\begin{verbatim}
\end{verbatim}

\textbf{SCHELOG} is an embedding of Prolog in Scheme.

\begin{verbatim}
http://www.ccs.neu.edu/~dorai/schelog/schelog.html
\end{verbatim}

\textbf{JFILTER} is a Scheme program which converts text among the JIS, EUC, and Shift-JIS Japanese character sets.

\begin{verbatim}
http://www.math.u-toyama.ac.jp/~iwao/Scheme/Jfilter
\end{verbatim}
8 About SLIB

More people than I can name have contributed to SLIB. Thanks to all of you!

SLIB 3b4, released May 2013.
Aubrey Jaffer <agj@alum.mit.edu>

Current information about SLIB can be found on SLIB’s WWW home page:
http://people.csail.mit.edu/jaffer/SLIB

SLIB is part of the GNU project.

8.1 Installation

There are five parts to installation:

- Unpack the SLIB distribution.
- Install documentation and slib script.
- Configure the Scheme implementation(s) to locate the SLIB directory and implementation directories.
- Arrange for each Scheme implementation to load its SLIB initialization file.
- Build the SLIB catalog for each Scheme implementation.

8.1.1 Unpacking the SLIB Distribution

If the SLIB distribution is a GNU/Linux RPM, it will create the SLIB directory ‘/usr/share/slib’.

If the SLIB distribution is a ZIP file, unzip the distribution to create the SLIB directory. Locate this ‘slib’ directory either in your home directory (if only you will use this SLIB installation); or put it in a location where libraries reside on your system. On unix systems this might be ‘/usr/share/slib’, ‘/usr/local/lib/slib’, or ‘/usr/lib/slib’. If you know where SLIB should go on other platforms, please inform agj@alum.mit.edu.

8.1.2 Install documentation and slib script

```make
make infoz
make install
```

8.1.3 Configure Scheme Implementation to Locate SLIB

If the Scheme implementation supports getenv, then the value of the shell environment variable $SCHEME_LIBRARY_PATH$ will be used for (library-vicinity) if it is defined. Currently, Bigloo, Chez, Elk, Gambit, Gauche, Guile, Jscheme, Larceny, MITScheme, MzScheme, RScheme, S7, STk, VSCM, and SCM support getenv. Scheme48 supports getenv but does not use it for determining library-vicinity. (That is done from the Makefile.)

The (library-vicinity) can also be set from the SLIB initialization file or by implementation-specific means.
8.1.4 Configure Scheme Implementation to Locate and Implementation Directory

Support for locating an implementation’s auxiliary directory is uneven among implementations. Also, the person installing SLIB may not have write permission to some of these directories (necessary for writing slibcat). Therefore, those implementations supporting `getenv` (except SCM and Scheme48) provide a means for specifying the implementation-vicinity through environment variables. Define the indicated environment variable to the pathname (with trailing slash or backslash) of the desired directory. Do not use ‘slib/’ as an implementation-vicinity!

Bigloo BIGLOO_IMPLEMENTATION_PATH
Chez CHEZ_IMPLEMENTATION_PATH
ELK ELK_IMPLEMENTATION_PATH
Gambit GAMBIT_IMPLEMENTATION_PATH
Guile GUILE_IMPLEMENTATION_PATH
Jscheme JSCHEME_IMPLEMENTATION_PATH
MIT-Scheme MITSCHME_IMPLEMENTATION_PATH
MzScheme MZSCHEME_IMPLEMENTATION_PATH
RScheme RSCHEME_IMPLEMENTATION_PATH
S7 S7_IMPLEMENTATION_PATH
STk STK_IMPLEMENTATION_PATH
Vscm VSCM_IMPLEMENTATION_PATH

8.1.5 Loading SLIB Initialization File

If you use the `slib` script to start your SLIB session, then this step is unnecessary.

Check the manifest in ‘README’ to find a configuration file for your Scheme implementation. Initialization files for most IEEE P1178 compliant Scheme Implementations are included with this distribution.

You should check the definitions of `software-type`, `scheme-implementation-version`, `implementation-vicinity`, and `library-vicinity` in the initialization file. There are comments in the file for how to configure it.

Once this is done, modify the startup file for your Scheme implementation to load this initialization file.

8.1.6 Build New SLIB Catalog for the Implementation

When SLIB is first used from an implementation, a file named ‘slibcat’ is written to the implementation-vicinity for that implementation. Because users may lack permission to write in implementation-vicinity, it is good practice to build the new catalog when installing SLIB.

To build (or rebuild) the catalog, start the Scheme implementation (with SLIB), then:

```
(require 'new-catalog)
```

The catalog also supports color-name dictionaries. With an SLIB-installed scheme implementation, type:

```
(require 'color-names)
(make-slib-color-name-db)
```
Chapter 8: About SLIB

8.1.7 Implementation-specific Instructions

Multiple implementations of Scheme can all use the same SLIB directory. Simply configure each implementation’s initialization file as outlined above.

SCM
The SCM implementation does not require any initialization file as SLIB support is already built into SCM. See the documentation with SCM for installation instructions.

Larceny
Starting with version 0.96, Larceny contains its own SLIB initialization file, loaded by (require 'srfi-96). If SCHEME_LIBRARY_PATH is not set, then Larceny looks for an ‘slib’ subdirectory of a directory in the list returned by (current-require-path)

```
larceny -- -e "(require 'srfi-96)"
```

Gauche-0.9
Gauche also supports SLIB. It finds SLIB at installation time; (use slib) to enable.

```
gosh -u slib
```

ELK

```
elk -i -l ${SCHEME_LIBRARY_PATH}elk.init
```

PLT Scheme

DrScheme

MzScheme
The ‘init.ss’ file in the _slibinit_ collection is an SLIB initialization file. To run SLIB in MzScheme:

```
mzscheme -f ${SCHEME_LIBRARY_PATH}mzscheme.init
```

MIT Scheme

```
scheme -load ${SCHEME_LIBRARY_PATH}mitscheme.init
```

Gambit-C 3.0

```
gsi -s ${SCHEME_LIBRARY_PATH}gambit.init -
```

SISC

```
sisc -e "(load "${SCHEME_LIBRARY_PATH}sisc.init")" --
```

Kawa

```
kawa -f ${SCHEME_LIBRARY_PATH}kawa.init --
```

Guile
Guile versions 1.6 and earlier link to an archaic SLIB version. In RedHat or Fedora installations:
rm /usr/share/guile/slib
ln -s ${SCHEME_LIBRARY_PATH} /usr/share/guile/slib

In Debian installations:
rm /usr/share/guile/1.6/slib
ln -s ${SCHEME_LIBRARY_PATH} /usr/share/guile/1.6/slib

${SCHEME_LIBRARY_PATH} is where SLIB gets installed.

Guile before version 1.8 with SLIB can then be started thus:
guile -l ${SCHEME_LIBRARY_PATH}guile.init

Guile version 1.8 and after with SLIB can then be started thus:
guile -l ${SCHEME_LIBRARY_PATH}guile.init \
   -l ${SCHEME_LIBRARY_PATH}guile.use

The Guile manual has a different way of installing SLIB:


Scheme48

To make a Scheme48 image for an installation under <prefix>,

1. cd to the SLIB directory
2. type make prefix=<prefix> slib48.
3. To install the image, type make prefix=<prefix> install48. This will also create a shell script with the name slib48 which will invoke the saved image.

VSCM

From: Matthias Blume <blume @ cs.Princeton.EDU>
Date: Tue, 1 Mar 1994 11:42:31 -0500

Disclaimer: The code below is only a quick hack. If I find some time to spare I might get around to make some more things work.

You have to provide `vscm.init` as an explicit command line argument. Since this is not very nice I would recommend the following installation procedure:

1. run scheme
2. (load "vscm.init")
3. (slib:dump "dumpfile")
4. mv dumpfile place-where-vscm-standard-bootfile-resides. For example:
mv dumpfile /usr/local/vscm/lib/scheme-boot

In this case vscm should have been compiled with flag:
-DDEFAULT_BOOTFILE="/usr/local/vscm/lib/scheme-boot"

See Makefile (definition of DDP) for details.
8.2 The SLIB script

SLIB comes with shell script for Unix platforms.

```
slib [ scheme | scm | gsi | mzscheme | guile |
      | scheme48 | scmlit | elk | sisc | kawa ]
```

Starts an interactive Scheme-with-SLIB session.

The optional argument to the `slib` script is the Scheme implementation to run. Absent the argument, it searches for implementations in the above order.

8.3 Porting

If there is no initialization file for your Scheme implementation, you will have to create one. Your Scheme implementation must be largely compliant with

IEEE Std 1178-1990,
Revised^3 Report on the Algorithmic Language Scheme, or
Revised^5 Report on the Algorithmic Language Scheme

in order to support SLIB.\(^1\)

`http://cvs.savannah.gnu.org/viewcvs/*checkout*/scm/scm/r4rstest.scm` is a file which checks compliance with much of R4RS.

`Template.scm` is an example configuration file. The comments inside will direct you on how to customize it to reflect your system. Give your new initialization file the implementation’s name with `.init` appended. For instance, if you were porting `foo-scheme` then the initialization file might be called `foo.init`.

Your customized version should then be loaded as part of your scheme implementation’s initialization. It will load `require.scm` from the library; this will allow the use of `provide`, `provided?`, and `require` along with the `vicinity` functions (these functions are documented in the sections Section 1.1 [Feature], page 1 and Section 1.2 [Require], page 2). The rest of the library will then be accessible in a system independent fashion.

Please mail new working configuration files to `agj@alum.mit.edu` so that they can be included in the SLIB distribution.

8.4 Compiled and Implementation-Specific Features

Often an implementation can implement an SLIB feature more efficiently than the R4RS-compliant source code in SLIB. Alternatively, implementations with compilers can compile SLIB source code into binary files which run faster than loading source code.

---

\(^1\) If you are porting a Revised^3 Report on the Algorithmic Language Scheme implementation, then you will need to finish writing `sc4sc3.scm` and load it from your initialization file.
Additionally, the SLIB catalog can be augmented with extra-SLIB features which can be loaded by the implementation. The catalog format is described in See Section 1.3 [Library Catalogs], page 3.

These implementation-specific modifications are made when a new catalog is created (see Section 1.4 [Catalog Creation], page 3). If ‘mkimpcat.scm’ in implementation-invicinity exists, it is loaded. That should produce the file ‘implcat’ in implementation-invicinity, whose associations will override those of SLIB. ‘implcat’ is copied into ‘slibcat’ in implementation-vicinity as part of the catalog creation process; modifications to ‘implcat’ after that will have no effect.

8.5 Coding Guidelines

All library packages are written in IEEE P1178 Scheme and assume that a configuration file and ‘require.scm’ package have already been loaded. Other versions of Scheme can be supported in library packages as well by using, for example, (provided? ’r3rs) or (require ’r3rs) (see Section 1.2 [Require], page 2).

If a procedure defined in a module is called by other procedures in that module, then those procedures should instead call an alias defined in that module:

(define module-name:foo foo)

The module name and ‘:’ should prefix that symbol for the internal name. Do not export internal aliases.

A procedure is exported from a module by putting Schmooz-style comments (see Section 4.15 [Schmooz], page 100) or ‘@’ at the beginning of the line immediately preceding the definition (define, define-syntax, or defmacro). Modules, exports and other relevant issues are discussed in Section 1.6 [Compiling Scheme], page 6.

Code submitted for inclusion in SLIB should not duplicate (more than one) routines already in SLIB files. Use require to force those library routines to be used by your package.

Documentation should be provided in Emacs Texinfo format if possible, but documentation must be provided.

Your package will be released sooner with SLIB if you send me a file which tests your code. Please run this test before you send me the code!

8.5.1 Modifications

Please document your changes. A line or two for ‘ChangeLog’ is sufficient for simple fixes or extensions. Look at the format of ‘ChangeLog’ to see what information is desired. Please send me diff files from the latest SLIB distribution (remember to send diffs of ‘slib.texi’ and ‘ChangeLog’). This makes for less email traffic and makes it easier for me to integrate when more than one person is changing a file (this happens a lot with ‘slib.texi’ and ‘*.init’ files).

If someone else wrote a package you want to significantly modify, please try to contact the author, who may be working on a new version. This will insure against wasting effort on obsolete versions.
Please do not reformat the source code with your favorite beautifier, make 10 fixes, and send me the resulting source code. I do not have the time to fish through 10000 diffs to find your 10 real fixes.

8.6 Copyrights

This section has instructions for SLIB authors regarding copyrights.

Each package in SLIB must either be in the public domain, or come with a statement of terms permitting users to copy, redistribute and modify it. The comments at the beginning of ‘require.scm’ and ‘macwork.scm’ illustrate copyright and appropriate terms.

If your code or changes amount to less than about 10 lines, you do not need to add your copyright or send a disclaimer.

8.6.1 Putting code into the Public Domain

In order to put code in the public domain you should sign a copyright disclaimer and send it to the SLIB maintainer. Contact agj@alum.mit.edu for the address to mail the disclaimer to.

I, <my-name>, hereby affirm that I have placed the software package <name> in the public domain.

I affirm that I am the sole author and sole copyright holder for the software package, that I have the right to place this software package in the public domain, and that I will do nothing to undermine this status in the future.

signature and date

This wording assumes that you are the sole author. If you are not the sole author, the wording needs to be different. If you don’t want to be bothered with sending a letter every time you release or modify a module, make your letter say that it also applies to your future revisions of that module.

Make sure no employer has any claim to the copyright on the work you are submitting. If there is any doubt, create a copyright disclaimer and have your employer sign it. Mail the signed disclaimer to the SLIB maintainer. Contact agj@alum.mit.edu for the address to mail the disclaimer to. An example disclaimer follows.

8.6.2 Explicit copying terms

If you submit more than about 10 lines of code which you are not placing into the Public Domain (by sending me a disclaimer) you need to:

• Arrange that your name appears in a copyright line for the appropriate year. Multiple copyright lines are acceptable.

• With your copyright line, specify any terms you require to be different from those already in the file.

• Make sure no employer has any claim to the copyright on the work you are submitting. If there is any doubt, create a copyright disclaimer and have your employer sign it. Mail the signed disclaimer to the SLIB maintainer. Contact agj@alum.mit.edu for the address to mail the disclaimer to.
8.6.3 Example: Company Copyright Disclaimer

This disclaimer should be signed by a vice president or general manager of the company. If you can’t get at them, anyone else authorized to license out software produced there will do. Here is a sample wording:

<employer> Corporation hereby disclaims all copyright interest in the program <program> written by <name>.

<employer> Corporation affirms that it has no other intellectual property interest that would undermine this release, and will do nothing to undermine it in the future.

<signature and date>,
<name>, <title>, <employer> Corporation

8.7 About this manual

- Entries that are labeled as Functions are called for their return values. Entries that are labeled as Procedures are called primarily for their side effects.
- Examples in this text were produced using the scm Scheme implementation.
- At the beginning of each section, there is a line that looks like:

  (require 'feature)

  Include this line in your code prior to using the package.

8.7.1 GNU Free Documentation License

Version 1.3, 3 November 2008


http://fsf.org/

Everyone is permitted to copy and distribute verbatim copies of this license document, but changing it is not allowed.

0. PREAMBLE

The purpose of this License is to make a manual, textbook, or other functional and useful document free in the sense of freedom: to assure everyone the effective freedom to copy and redistribute it, with or without modifying it, either commercially or non-commercially. Secondarily, this License preserves for the author and publisher a way to get credit for their work, while not being considered responsible for modifications made by others.

This License is a kind of “copyleft”, which means that derivative works of the document must themselves be free in the same sense. It complements the GNU General Public License, which is a copyleft license designed for free software.

We have designed this License in order to use it for manuals for free software, because free software needs free documentation: a free program should come with manuals providing the same freedoms that the software does. But this License is not limited to software manuals; it can be used for any textual work, regardless of subject matter or whether it is published as a printed book. We recommend this License principally for works whose purpose is instruction or reference.
1. APPLICABILITY AND DEFINITIONS

This License applies to any manual or other work, in any medium, that contains a notice placed by the copyright holder saying it can be distributed under the terms of this License. Such a notice grants a world-wide, royalty-free license, unlimited in duration, to use that work under the conditions stated herein. The “Document”, below, refers to any such manual or work. Any member of the public is a licensee, and is addressed as “you”. You accept the license if you copy, modify or distribute the work in a way requiring permission under copyright law.

A “Modified Version” of the Document means any work containing the Document or a portion of it, either copied verbatim, or with modifications and/or translated into another language.

A “Secondary Section” is a named appendix or a front-matter section of the Document that deals exclusively with the relationship of the publishers or authors of the Document to the Document’s overall subject (or to related matters) and contains nothing that could fall directly within that overall subject. (Thus, if the Document is in part a textbook of mathematics, a Secondary Section may not explain any mathematics.) The relationship could be a matter of historical connection with the subject or with related matters, or of legal, commercial, philosophical, ethical or political position regarding them.

The “Invariant Sections” are certain Secondary Sections whose titles are designated, as being those of Invariant Sections, in the notice that says that the Document is released under this License. If a section does not fit the above definition of Secondary then it is not allowed to be designated as Invariant. The Document may contain zero Invariant Sections. If the Document does not identify any Invariant Sections then there are none.

The “Cover Texts” are certain short passages of text that are listed, as Front-Cover Texts or Back-Cover Texts, in the notice that says that the Document is released under this License. A Front-Cover Text may be at most 5 words, and a Back-Cover Text may be at most 25 words.

A “Transparent” copy of the Document means a machine-readable copy, represented in a format whose specification is available to the general public, that is suitable for revising the document straightforwardly with generic text editors or (for images composed of pixels) generic paint programs or (for drawings) some widely available drawing editor, and that is suitable for input to text formatters or for automatic translation to a variety of formats suitable for input to text formatters. A copy made in an otherwise Transparent file format whose markup, or absence of markup, has been arranged to thwart or discourage subsequent modification by readers is not Transparent. An image format is not Transparent if used for any substantial amount of text. A copy that is not “Transparent” is called “Opaque”.

Examples of suitable formats for Transparent copies include plain ASCII without markup, Texinfo input format, LaTeX input format, SGML or XML using a publicly available DTD, and standard-conforming simple HTML, PostScript or PDF designed for human modification. Examples of transparent image formats include PNG, XCF and JPG. Opaque formats include proprietary formats that can be read and edited only by proprietary word processors, SGML or XML for which the DTD and/or
processing tools are not generally available, and the machine-generated HTML, PostScript or PDF produced by some word processors for output purposes only.

The “Title Page” means, for a printed book, the title page itself, plus such following pages as are needed to hold, legibly, the material this License requires to appear in the title page. For works in formats which do not have any title page as such, “Title Page” means the text near the most prominent appearance of the work’s title, preceding the beginning of the body of the text.

The “publisher” means any person or entity that distributes copies of the Document to the public.

A section “Entitled XYZ” means a named subunit of the Document whose title either is precisely XYZ or contains XYZ in parentheses following text that translates XYZ in another language. (Here XYZ stands for a specific section name mentioned below, such as “Acknowledgements”, “Dedications”, “Endorsements”, or “History”.) To “Preserve the Title” of such a section when you modify the Document means that it remains a section “Entitled XYZ” according to this definition.

The Document may include Warranty Disclaimers next to the notice which states that this License applies to the Document. These Warranty Disclaimers are considered to be included by reference in this License, but only as regards disclaiming warranties: any other implication that these Warranty Disclaimers may have is void and has no effect on the meaning of this License.

2. VERBATIM COPYING

You may copy and distribute the Document in any medium, either commercially or noncommercially, provided that this License, the copyright notices, and the license notice saying this License applies to the Document are reproduced in all copies, and that you add no other conditions whatsoever to those of this License. You may not use technical measures to obstruct or control the reading or further copying of the copies you make or distribute. However, you may accept compensation in exchange for copies. If you distribute a large enough number of copies you must also follow the conditions in section 3.

You may also lend copies, under the same conditions stated above, and you may publicly display copies.

3. COPYING IN QUANTITY

If you publish printed copies (or copies in media that commonly have printed covers) of the Document, numbering more than 100, and the Document’s license notice requires Cover Texts, you must enclose the copies in covers that carry, clearly and legibly, all these Cover Texts: Front-Cover Texts on the front cover, and Back-Cover Texts on the back cover. Both covers must also clearly and legibly identify you as the publisher of these copies. The front cover must present the full title with all words of the title equally prominent and visible. You may add other material on the covers in addition. Copying with changes limited to the covers, as long as they preserve the title of the Document and satisfy these conditions, can be treated as verbatim copying in other respects.

If the required texts for either cover are too voluminous to fit legibly, you should put the first ones listed (as many as fit reasonably) on the actual cover, and continue the rest onto adjacent pages.
If you publish or distribute Opaque copies of the Document numbering more than 100, you must either include a machine-readable Transparent copy along with each Opaque copy, or state in or with each Opaque copy a computer-network location from which the general network-using public has access to download using public-standard network protocols a complete Transparent copy of the Document, free of added material. If you use the latter option, you must take reasonably prudent steps, when you begin distribution of Opaque copies in quantity, to ensure that this Transparent copy will remain thus accessible at the stated location until at least one year after the last time you distribute an Opaque copy (directly or through your agents or retailers) of that edition to the public.

It is requested, but not required, that you contact the authors of the Document well before redistributing any large number of copies, to give them a chance to provide you with an updated version of the Document.

4. MODIFICATIONS

You may copy and distribute a Modified Version of the Document under the conditions of sections 2 and 3 above, provided that you release the Modified Version under precisely this License, with the Modified Version filling the role of the Document, thus licensing distribution and modification of the Modified Version to whoever possesses a copy of it. In addition, you must do these things in the Modified Version:

A. Use in the Title Page (and on the covers, if any) a title distinct from that of the Document, and from those of previous versions (which should, if there were any, be listed in the History section of the Document). You may use the same title as a previous version if the original publisher of that version gives permission.

B. List on the Title Page, as authors, one or more persons or entities responsible for authorship of the modifications in the Modified Version, together with at least five of the principal authors of the Document (all of its principal authors, if it has fewer than five), unless they release you from this requirement.

C. State on the Title page the name of the publisher of the Modified Version, as the publisher.

D. Preserve all the copyright notices of the Document.

E. Add an appropriate copyright notice for your modifications adjacent to the other copyright notices.

F. Include, immediately after the copyright notices, a license notice giving the public permission to use the Modified Version under the terms of this License, in the form shown in the Addendum below.

G. Preserve in that license notice the full lists of Invariant Sections and required Cover Texts given in the Document’s license notice.

H. Include an unaltered copy of this License.

I. Preserve the section Entitled “History”, Preserve its Title, and add to it an item stating at least the title, year, new authors, and publisher of the Modified Version as given on the Title Page. If there is no section Entitled “History” in the Document, create one stating the title, year, authors, and publisher of the Document as given on its Title Page, then add an item describing the Modified Version as stated in the previous sentence.
Chapter 8: About SLIB

J. Preserve the network location, if any, given in the Document for public access to a Transparent copy of the Document, and likewise the network locations given in the Document for previous versions it was based on. These may be placed in the "History" section. You may omit a network location for a work that was published at least four years before the Document itself, or if the original publisher of the version it refers to gives permission.

K. For any section Entitled "Acknowledgements" or "Dedications", Preserve the Title of the section, and preserve in the section all the substance and tone of each of the contributor acknowledgements and/or dedications given therein.

L. Preserve all the Invariant Sections of the Document, unaltered in their text and in their titles. Section numbers or the equivalent are not considered part of the section titles.

M. Delete any section Entitled "Endorsements". Such a section may not be included in the Modified Version.

N. Do not retile any existing section to be Entitled "Endorsements" or to conflict in title with any Invariant Section.

O. Preserve any Warranty Disclaimers.

If the Modified Version includes new front-matter sections or appendices that qualify as Secondary Sections and contain no material copied from the Document, you may at your option designate some or all of these sections as invariant. To do this, add their titles to the list of Invariant Sections in the Modified Version’s license notice. These titles must be distinct from any other section titles.

You may add a section Entitled "Endorsements", provided it contains nothing but endorsements of your Modified Version by various parties—for example, statements of peer review or that the text has been approved by an organization as the authoritative definition of a standard.

You may add a passage of up to five words as a Front-Cover Text, and a passage of up to 25 words as a Back-Cover Text, to the end of the list of Cover Texts in the Modified Version. Only one passage of Front-Cover Text and one of Back-Cover Text may be added by (or through arrangements made by) any one entity. If the Document already includes a cover text for the same cover, previously added by you or by arrangement made by the same entity you are acting on behalf of, you may not add another; but you may replace the old one, on explicit permission from the previous publisher that added the old one.

The author(s) and publisher(s) of the Document do not by this License give permission to use their names for publicity for or to assert or imply endorsement of any Modified Version.

5. COMBINING DOCUMENTS

You may combine the Document with other documents released under this License, under the terms defined in section 4 above for modified versions, provided that you include in the combination all of the Invariant Sections of all of the original documents, unmodified, and list them all as Invariant Sections of your combined work in its license notice, and that you preserve all their Warranty Disclaimers.

The combined work need only contain one copy of this License, and multiple identical Invariant Sections may be replaced with a single copy. If there are multiple Invariant
Sections with the same name but different contents, make the title of each such section
unique by adding at the end of it, in parentheses, the name of the original author or
publisher of that section if known, or else a unique number. Make the same adjustment
to the section titles in the list of Invariant Sections in the license notice of the combined
work.

In the combination, you must combine any sections Entitled “History” in the vari-
ous original documents, forming one section Entitled “History”; likewise combine any
sections Entitled “Acknowledgements”, and any sections Entitled “Dedications”. You
must delete all sections Entitled “Endorsements.”

6. COLLECTIONS OF DOCUMENTS

You may make a collection consisting of the Document and other documents released
under this License, and replace the individual copies of this License in the various
documents with a single copy that is included in the collection, provided that you
follow the rules of this License for verbatim copying of each of the documents in all
other respects.

You may extract a single document from such a collection, and distribute it individu-
ally under this License, provided you insert a copy of this License into the extracted
document, and follow this License in all other respects regarding verbatim copying of
that document.

7. AGGREGATION WITH INDEPENDENT WORKS

A compilation of the Document or its derivatives with other separate and independent
documents or works, in or on a volume of a storage or distribution medium, is called
an “aggregate” if the copyright resulting from the compilation is not used to limit the
legal rights of the compilation’s users beyond what the individual works permit. When
the Document is included in an aggregate, this License does not apply to the other
works in the aggregate which are not themselves derivative works of the Document.

If the Cover Text requirement of section 3 is applicable to these copies of the Document,
then if the Document is less than one half of the entire aggregate, the Document’s Cover
Texts may be placed on covers that bracket the Document within the aggregate, or the
electronic equivalent of covers if the Document is in electronic form. Otherwise they
must appear on printed covers that bracket the whole aggregate.

8. TRANSLATION

Translation is considered a kind of modification, so you may distribute translations
of the Document under the terms of section 4. Replacing Invariant Sections with
translations requires special permission from their copyright holders, but you may
include translations of some or all Invariant Sections in addition to the original versions
of these Invariant Sections. You may include a translation of this License, and all the
license notices in the Document, and any Warranty Disclaimers, provided that you
also include the original English version of this License and the original versions of
those notices and disclaimers. In case of a disagreement between the translation and
the original version of this License or a notice or disclaimer, the original version will
prevail.

If a section in the Document is Entitled “Acknowledgements”, “Dedications”, or “His-
tory”, the requirement (section 4) to Preserve its Title (section 1) will typically require
changing the actual title.
9. TERMINATION

You may not copy, modify, sublicense, or distribute the Document except as expressly provided under this License. Any attempt otherwise to copy, modify, sublicense, or distribute it is void, and will automatically terminate your rights under this License. However, if you cease all violation of this License, then your license from a particular copyright holder is reinstated (a) provisionally, unless and until the copyright holder explicitly and finally terminates your license, and (b) permanently, if the copyright holder fails to notify you of the violation by some reasonable means prior to 60 days after the cessation.

Moreover, your license from a particular copyright holder is reinstated permanently if the copyright holder notifies you of the violation by some reasonable means, this is the first time you have received notice of violation of this License (for any work) from that copyright holder, and you cure the violation prior to 30 days after your receipt of the notice.

Termination of your rights under this section does not terminate the licenses of parties who have received copies or rights from you under this License. If your rights have been terminated and not permanently reinstated, receipt of a copy of some or all of the same material does not give you any rights to use it.

10. FUTURE REVISIONS OF THIS LICENSE

The Free Software Foundation may publish new, revised versions of the GNU Free Documentation License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns. See http://www.gnu.org/copyleft/.

Each version of the License is given a distinguishing version number. If the Document specifies that a particular numbered version of this License “or any later version” applies to it, you have the option of following the terms and conditions either of that specified version or of any later version that has been published (not as a draft) by the Free Software Foundation. If the Document does not specify a version number of this License, you may choose any version ever published (not as a draft) by the Free Software Foundation. If the Document specifies that a proxy can decide which future versions of this License can be used, that proxy’s public statement of acceptance of a version permanently authorizes you to choose that version for the Document.

11. RELICENSING

“Massive Multiauthor Collaboration Site” (or “MMC Site”) means any World Wide Web server that publishes copyrightable works and also provides prominent facilities for anybody to edit those works. A public wiki that anybody can edit is an example of such a server. A “Massive Multiauthor Collaboration” (or “MMC”) contained in the site means any set of copyrightable works thus published on the MMC site.

“CC-BY-SA” means the Creative Commons Attribution-Share Alike 3.0 license published by Creative Commons Corporation, a not-for-profit corporation with a principal place of business in San Francisco, California, as well as future copyleft versions of that license published by that same organization.

“Incorporate” means to publish or republish a Document, in whole or in part, as part of another Document.
An MMC is “eligible for relicensing” if it is licensed under this License, and if all works that were first published under this License somewhere other than this MMC, and subsequently incorporated in whole or in part into the MMC, (1) had no cover texts or invariant sections, and (2) were thus incorporated prior to November 1, 2008.

The operator of an MMC Site may republish an MMC contained in the site under CC-BY-SA on the same site at any time before August 1, 2009, provided the MMC is eligible for relicensing.
ADDENDUM: How to use this License for your documents

To use this License in a document you have written, include a copy of the License in the
document and put the following copyright and license notices just after the title page:

Copyright (C) year your name.
Permission is granted to copy, distribute and/or modify this document
under the terms of the GNU Free Documentation License, Version 1.3
or any later version published by the Free Software Foundation;
with no Invariant Sections, no Front-Cover Texts, and no Back-Cover
Texts. A copy of the license is included in the section entitled ‘‘GNU
Free Documentation License’’.

If you have Invariant Sections, Front-Cover Texts and Back-Cover Texts, replace the
“with...Texts.” line with this:

with the Invariant Sections being list their titles, with
the Front-Cover Texts being list, and with the Back-Cover Texts
being list.

If you have Invariant Sections without Cover Texts, or some other combination of the
three, merge those two alternatives to suit the situation.

If your document contains nontrivial examples of program code, we recommend releas-
ing these examples in parallel under your choice of free software license, such as the GNU
General Public License, to permit their use in free software.
Procedure and Macro Index

- ................................. 240
-1+ .................................. 248
/
/.................................... 240
<
<=? .................................. 248
<? .................................. 248
=
=? .................................. 236
>=? .................................. 248
>=? .................................. 248
>
> .................................. 248
1
1+ .................................. 248
add-domain ............................................. 164
add-domain on relational-database ................. 171
add-macro-support ..................................... 177
add-process! ........................................... 243
add-setter .............................................. 37
adjoin .................................................. 223
adjoin-parameters! ..................................... 62
alist->wt-tree .......................................... 190
alist-associator ....................................... 203
alist-cons .............................................. 256
alist-copy .............................................. 256
alist-delete ............................................ 256
alist-delete! ........................................... 256
alist-for-each .......................................... 204
alist-inquirer .......................................... 203
alist-map ............................................... 203
alist-remover .......................................... 203
alist-table ............................................. 180
and-let ................................................. 34
and ..................................................... 231
any ................................................. 255
any-bits-set? ........................................... 104
any? ................................................. 212
append! ............................................... 248
append-reverse ......................................... 254
append-reverse! ........................................ 254
apply ................................................. 249
arithmetic-shift ....................................... 106
array->list ............................................. 197
array->vector ......................................... 198
array-dimensions ...................................... 196
array-for-each ......................................... 201
array-in-bounds? ....................................... 198
array-index-for-each .................................. 202
array-index-map! ....................................... 202
array-indexes .......................................... 202
array-map ............................................. 201
array-map! ............................................ 201
array-map ............................................. 201
array-rank ............................................ 196
array-ref ............................................. 198
array-set! ............................................. 198
array-trim ............................................. 201
array:copy! ........................................... 202
array? ............................................... 196
asctime .............................................. 99
ash ................................................ 106
assoc ................................................. 256
atan ................................................... 109
atom? .................................................. 231
attlist-add ............................................ 87
attlist-remove-top .................................... 87
B
batch:call-with-output-script ..................... 66
batch:command ............................................. 67
cgi:serve-query ........................................... 76
batch:comment ............................................. 67
chap:next-string .......................................... 233
batch:delete-file .......................................... 67
chap:string<=? ............................................ 233
batch:initialize! .......................................... 66
chap:string?>? ............................................ 233
batch:lines->file .......................................... 67
chap:string?>? ............................................ 232
batch:rename-file .......................................... 67
chap:string>=? ............................................ 233
batch:run-script ........................................... 67
chap:string>=? ............................................ 233
batch:try-chopped-command ............................... 67
chap:string<? ............................................. 233
batch:try-command .......................................... 67
check-parameters ........................................ 62
bit-field .................................................. 106
color ...................................................... 151
bit-set ..................................................... 105
clear-sky-color-xyy .................................... 151
crude-bool ............................................... 105
circular-list ............................................. 253
circular-list? ............................................ 253
bytes ...................................................... 204
cksum ..................................................... 117
bytes->boolean ........................................... 204
codons<->cdna ............................................. 100
circular-list ............................................. 253
color ...................................................... 151
color->rgb709 ............................................ 146
color:rgb709 ............................................. 146
color:e-srgb ............................................. 146
Coercion .................................................. 241
color->l*u*v* ............................................. 146
color:l*u*v* ............................................. 146
color:l*c*h ............................................. 146
color:l*c*h ............................................. 146
color->e-srgb ............................................. 146
color->l*u*v* ............................................. 146
color->l*u*v* ............................................. 146
color->xyy ............................................... 146
color->xyy ............................................... 146
color:xyy ................................................ 146
color:xyy ................................................ 146
color:xyy ................................................ 146
CIE:DE* .................................................. 145
CIE:DE*94 ................................................ 145
CIE:DE* .................................................. 145
ciexyz->color ........................................... 137
code-integer ............................................. 204
code->integer ............................................ 204
code->integer ............................................ 205
CIE:DE*94 ................................................ 145
CMC-DE ................................................... 145
CMC:DE* .................................................. 145
coerce-<cdna .............................................. 241
collection ............................................... 211
 coerc-<cdna .............................................. 241
color ..................................................... 137
color->ciexyz ........................................... 137
color->e-srgb ............................................ 141
color->l*a*b* ............................................ 138
color->l*+*c+*h ........................................ 139
color->l*u*v* ............................................ 138
color->rgb709 ............................................ 137
color->r-srgb ............................................. 140
color->string ............................................ 136
color->xrgb ............................................. 140
color->xrgb ............................................. 140
color-dictionaries->lookup ................................ 147
color-dictionary ......................................... 147
color-name->color ...................................... 147
color-name:canonicalize ................................ 147
color-precision ......................................... 135
color-space ............................................. 135
color-white-point ...................................... 135
color:ciexyz ............................................. 137
color:e-srgb ............................................ 141
color:l*a*b* ............................................ 138
color:l*c*h ............................................. 139
color:l*u*v* ............................................ 138
color:linear-transform .................................. 146
color:rgb709 ............................................. 137
color:r-srgb ............................................. 140
color? .................................................... 134

C

call-with-dynamic-binding ............................... 214
call-with-input-string .................................. 242
call-with-open-ports ................................... 14
call-with-output-string ................................ 242
call-with-tempnam ...................................... 65,66
call-with-values ........................................ 252
capture-syntactic-environment ......................... 27
car+cdr .................................................. 254
cart-prod-tables on relational-database .............. 187
catalog->html ............................................ 73
catalog-id on base-table ................................ 181
catalog:read ............................................ 5
cdna:base-table .......................................... 100
cdna:report-base-count ................................ 100
<table>
<thead>
<tr>
<th>Domain</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>column-domains on relational-table</td>
<td>169</td>
</tr>
<tr>
<td>column-foreigns on relational-table</td>
<td>169</td>
</tr>
<tr>
<td>column-names on relational-table</td>
<td>169</td>
</tr>
<tr>
<td>column-range</td>
<td>121</td>
</tr>
<tr>
<td>column-types on relational-table</td>
<td>169</td>
</tr>
<tr>
<td>combine-ranges</td>
<td>121</td>
</tr>
<tr>
<td>combined-rulesets</td>
<td>157</td>
</tr>
<tr>
<td>command-&gt;x-specs</td>
<td>72</td>
</tr>
<tr>
<td>command:make-editable-table</td>
<td>74</td>
</tr>
<tr>
<td>command:modify-table</td>
<td>74</td>
</tr>
<tr>
<td>concatenate</td>
<td>254</td>
</tr>
<tr>
<td>concatenate!</td>
<td>254</td>
</tr>
<tr>
<td>cond</td>
<td>34</td>
</tr>
<tr>
<td>cond-expand</td>
<td>252</td>
</tr>
<tr>
<td>cons*</td>
<td>253</td>
</tr>
<tr>
<td>continue</td>
<td>258</td>
</tr>
<tr>
<td>convert-color</td>
<td>135</td>
</tr>
<tr>
<td>copy-bit</td>
<td>105</td>
</tr>
<tr>
<td>copy-bit-field</td>
<td>106</td>
</tr>
<tr>
<td>copy-list</td>
<td>222</td>
</tr>
<tr>
<td>copy-random-state</td>
<td>112</td>
</tr>
<tr>
<td>copy-tree</td>
<td>232</td>
</tr>
<tr>
<td>count</td>
<td>254</td>
</tr>
<tr>
<td>count-newlines</td>
<td>240</td>
</tr>
<tr>
<td>crc:make-table</td>
<td>116</td>
</tr>
<tr>
<td>crc16</td>
<td>117</td>
</tr>
<tr>
<td>crc5</td>
<td>117</td>
</tr>
<tr>
<td>create-array</td>
<td>197</td>
</tr>
<tr>
<td>create-database</td>
<td>162</td>
</tr>
<tr>
<td>create-database on relational-system</td>
<td>186</td>
</tr>
<tr>
<td>create-postscript-graph</td>
<td>120</td>
</tr>
<tr>
<td>create-table on relational-database</td>
<td>187</td>
</tr>
<tr>
<td>create-view on relational-database</td>
<td>187</td>
</tr>
<tr>
<td>cring:define-rule</td>
<td>157</td>
</tr>
<tr>
<td>ctime</td>
<td>99</td>
</tr>
<tr>
<td>current-directory</td>
<td>261</td>
</tr>
<tr>
<td>current-error-port</td>
<td>14</td>
</tr>
<tr>
<td>current-input-port</td>
<td>42</td>
</tr>
<tr>
<td>current-output-port</td>
<td>205</td>
</tr>
<tr>
<td>current-time</td>
<td>96</td>
</tr>
<tr>
<td>cvs-directories</td>
<td>264</td>
</tr>
<tr>
<td>cvs-files</td>
<td>264</td>
</tr>
<tr>
<td>cvs-repository</td>
<td>264</td>
</tr>
<tr>
<td>cvs-root</td>
<td>264</td>
</tr>
<tr>
<td>cvs-set-root</td>
<td>265</td>
</tr>
<tr>
<td>cvs-set-roots!</td>
<td>265</td>
</tr>
<tr>
<td>cvs-vet</td>
<td>265</td>
</tr>
<tr>
<td>define-macro</td>
<td>177</td>
</tr>
<tr>
<td>define-operation</td>
<td>36</td>
</tr>
<tr>
<td>define-predicate</td>
<td>36</td>
</tr>
<tr>
<td>define-record-type</td>
<td>33</td>
</tr>
<tr>
<td>define-structure</td>
<td>32</td>
</tr>
<tr>
<td>define-syntax</td>
<td>19</td>
</tr>
<tr>
<td>db-&gt;netscape</td>
<td>177</td>
</tr>
<tr>
<td>db-&gt;html-directory</td>
<td>75</td>
</tr>
<tr>
<td>db-&gt;html-files</td>
<td>74</td>
</tr>
<tr>
<td>db-&gt;netscape</td>
<td>75</td>
</tr>
<tr>
<td>decode-universal-time</td>
<td>99</td>
</tr>
<tr>
<td>define-<em>commands</em></td>
<td>170</td>
</tr>
<tr>
<td>define-access-operation</td>
<td>37</td>
</tr>
<tr>
<td>define-command</td>
<td>177</td>
</tr>
<tr>
<td>define-domains</td>
<td>164</td>
</tr>
<tr>
<td>db-&gt;html-directory</td>
<td>75</td>
</tr>
<tr>
<td>db-&gt;html-files</td>
<td>74</td>
</tr>
<tr>
<td>db-&gt;netscape</td>
<td>75</td>
</tr>
<tr>
<td>decode-universal-time</td>
<td>99</td>
</tr>
<tr>
<td>define-<em>commands</em></td>
<td>170</td>
</tr>
<tr>
<td>define-access-operation</td>
<td>37</td>
</tr>
<tr>
<td>define-command</td>
<td>177</td>
</tr>
<tr>
<td>define-domains</td>
<td>164</td>
</tr>
<tr>
<td>define-macro</td>
<td>177</td>
</tr>
<tr>
<td>define-operation</td>
<td>36</td>
</tr>
<tr>
<td>define-predicate</td>
<td>36</td>
</tr>
<tr>
<td>define-record-type</td>
<td>33</td>
</tr>
<tr>
<td>define-structure</td>
<td>32</td>
</tr>
<tr>
<td>define-syntax</td>
<td>19</td>
</tr>
<tr>
<td>db-&gt;netscape</td>
<td>177</td>
</tr>
<tr>
<td>db-&gt;html-directory</td>
<td>75</td>
</tr>
<tr>
<td>db-&gt;html-files</td>
<td>74</td>
</tr>
<tr>
<td>db-&gt;netscape</td>
<td>75</td>
</tr>
<tr>
<td>decode-universal-time</td>
<td>99</td>
</tr>
<tr>
<td>define-<em>commands</em></td>
<td>170</td>
</tr>
<tr>
<td>define-access-operation</td>
<td>37</td>
</tr>
<tr>
<td>define-command</td>
<td>177</td>
</tr>
<tr>
<td>define-domains</td>
<td>164</td>
</tr>
<tr>
<td>define-macro</td>
<td>177</td>
</tr>
<tr>
<td>define-operation</td>
<td>36</td>
</tr>
<tr>
<td>define-predicate</td>
<td>36</td>
</tr>
<tr>
<td>define-record-type</td>
<td>33</td>
</tr>
<tr>
<td>define-structure</td>
<td>32</td>
</tr>
<tr>
<td>define-syntax</td>
<td>19</td>
</tr>
<tr>
<td>e-sRGB-&gt;CIEXYZ</td>
<td>146</td>
</tr>
<tr>
<td>e-srgb-&gt;color</td>
<td>140</td>
</tr>
<tr>
<td>e-sRGB-&gt;e-sRGB</td>
<td>147</td>
</tr>
<tr>
<td>e-sRGB-&gt;sRGB</td>
<td>146</td>
</tr>
<tr>
<td>eighth</td>
<td>254</td>
</tr>
<tr>
<td>emacs:backup-name</td>
<td>263</td>
</tr>
<tr>
<td>empty?</td>
<td>212</td>
</tr>
</tbody>
</table>
Procedure and Macro Index

encode-universal-time ........................................... 99
equeue! .............................................................. 220
equal? ............................................................... 196, 204
eval ................................................................. 250
every ............................................................... 224
every? ............................................................... 212
exports<info-index .................................................. 10
extended-euclid ..................................................... 107

F
factor ............................................................... 111
feature->export-alist ............................................. 8
feature->exports .................................................. 8
feature->requires ................................................ 7
feature->requires* ................................................ 7
feature-eval ........................................................ 1
fft ................................................................. 114
fft-1 ............................................................... 114
fifth ............................................................... 254
file->color-dictionary ............................................ 148
file->definitions .................................................. 8
file->exports ...................................................... 8
file->loads ........................................................ 7
file->requires .................................................... 6
file->requires* ................................................... 7
file-exists? ....................................................... 13
file-lock! .......................................................... 263
file-lock-owner ................................................... 263
file-position ..................................................... 15
file-unlock! ....................................................... 263
filename:match-ci?? ............................................... 64
filename:match?? ................................................ 64
filename:substitute-ci?? ......................................... 65
filename:substitute?? ........................................... 65
fill-empty-parameters ............................................. 62
fill-rect ........................................................... 124
filter .............................................................. 255
filter! ............................................................. 255
find ............................................................... 255
find-if ............................................................ 225
find-ratio ........................................................ 249
find-ratio-between ................................................. 250
find-string-from-port? ............................................ 239
find-tail .......................................................... 255
first ............................................................... 254
first-set-bit ...................................................... 105
fluid-let .......................................................... 33
fold ............................................................... 255
fold-right ......................................................... 255
for-each-elt ...................................................... 211
for-each-key ...................................................... 211
for-each-key on base-table ....................................... 184
for-each-row on relational-table .................................. 167
for-each-row-in-order on relational-table ......................... 168
force ............................................................... 250
force-output ...................................................... 14
form:delimited .................................................... 72
form:element ..................................................... 71
form:image ........................................................ 71
form:reset ........................................................ 71
form:submit ...................................................... 71
format ............................................................. 47
fourth .............................................................. 254
fprintf ............................................................. 53
fscanf ............................................................. 56

G
gen-elts ........................................................... 211
gen-keys ........................................................... 211
generic-write ..................................................... 93
gentemp ............................................................ 18
get on relational-table .......................................... 165
get* on relational-table .......................................... 166
decoded-time ..................................................... 99
get-foreign-choices .............................................. 72
get-method ........................................................ 216
get-universal-time ............................................... 99
genv ............................................................... 261
gocept ............................................................. 58
gocept-- ........................................................... 60
gocept->arglist .................................................... 63
getopt->parameter-list .......................................... 63
glob-pattern? ..................................................... 79
gmtime ............................................................. 98
gmktime ........................................................... 98
golden-section-search ............................................ 153
gray-code->integer ............................................... 236
gray-code=? ...................................................... 237
gray-code<=? ..................................................... 236
gray-code>=? ..................................................... 237
grey ............................................................... 147
grid-horizontals .................................................. 125
grid-verticals ..................................................... 125
gtime ............................................................. 99

H
has-duplicates? .................................................... 226
hash ............................................................... 235
hash-associator .................................................. 214
hash-for-each .................................................... 215
hash-inquirer .................................................... 214
hash-map ........................................................ 215
hash-rehasher .................................................... 215
hash-remover ..................................................... 214
hashq ............................................................. 235
hashv ............................................................. 235
heap-extract-max! ................................................ 219
heap-insert! ...................................................... 219
heap-length ...................................................... 219
hilbert-coordinates->integer .................................... 236
histograph ........................................................ 119
list-tail.............................................. 249
list=#................................................ 254
ln................................................... 110
load->path............................................ 7
load-ciexyz.......................................... 141
load-color-dictionary.............................. 148
localtime........................................... 98
log2-binary-factors.................................. 105
logand............................................... 103
logbit?............................................... 105
logcount............................................ 104
logior............................................... 103
lognot.............................................. 103
logtest............................................. 104
logxor............................................... 103
lset-adjoin.......................................... 256
lset-diff+intersection.............................. 256
lset-diff+intersection!.............................. 256
lset-difference..................................... 256
lset-difference!.................................... 256
lset-intersection.................................. 256
lset-intersection!.................................. 256
lset-union.......................................... 256
lset-union!......................................... 256
lset-xor............................................ 256
lset-xor!............................................ 256
lset=................................................. 256
make-list-keyifier on base-table.................. 182
make-list........................................... 221
make-list-keyifier on base-table.................. 182
make-method!........................................ 216
make-nexter on base-table.......................... 184
make-object........................................ 215
make-parameter-list............................... 62
make-polar......................................... 110
make-predicate..................................... 216
make-prever on base-table.......................... 184
make-promise........................................ 250
make-putter on base-table.......................... 183
make-query-alist-command-server.................. 76
make-queue.......................................... 220
make-random-state.................................. 112
make-record-type................................... 220
make-rectangular................................... 110
make-relational-system............................ 186
make-ruleset........................................ 156
make-shared-array.................................. 197
make-sierpinski-indexer............................ 237
make-slib-color-name-db............................ 148
make-syntactic-closure............................. 26
make-table on base-table........................... 181
make-url............................................ 78
make-vicinity....................................... 11
make-wt-tree........................................ 189
make-wt-tree-type.................................. 189
map!.................................................. 255
map-eltz............................................. 211
map-key on base-table.............................. 184
map-keys............................................. 211
matfile:load........................................ 209
matfile:read......................................... 209
matrix->array....................................... 160
matrix->lists....................................... 160
matrix:difference.................................. 161
matrix:inverse..................................... 161
matrix:product..................................... 161
matrix:sum......................................... 161
mdbm:report........................................ 163
member............................................... 255
member-if.......................................... 223
merge............................................... 234
merge!............................................... 234
mtime................................................. 98, 99
mod.................................................. 110
modular:*.......................................... 108
modular:+.......................................... 108
modular:-.......................................... 108
modular:characteristic............................ 107
modular:expt....................................... 108
modular:invert..................................... 108
modular:invertable?................................. 108
modular:invertable................................. 108
modular:normalize.................................. 107
modulo.............................................. 109
mrna<->cdna......................................... 100
must-be-first....................................... 68
must-be-last........................................ 68
N
natural->peano-coordinates .................................. 237
ncbi:read-dna-sequence ..................................... 100
ncbi:read-file ................................................. 100
nconc ............................................................ 229
newton:find-integer-root ..................................... 151
newton:find-root ................................................. 152
ninth .............................................................. 254
not-pair? ......................................................... 254
notany .............................................................. 224
nreverse .......................................................... 230
nthcdr ............................................................. 229
null-directory? ................................................... 79
null-environment ................................................ 251
null-list? ........................................................ 254

O
object .............................................................. 36
object->limited-string ......................................... 93
object->string .................................................... 93
object-with-ancestors ......................................... 36
object? ............................................................ 216
offset-time ................................................................ 96
open-base on base-table ........................................ 181
open-command-database .......................................... 170, 171
open-command-database! ......................................... 170, 171
open-database ...................................................... 163
open-database on relational-system ............................. 186
open-database! ........................................................ 163
open-file ............................................................. 14, 205
open-table ................................................................ 163
open-table on base-table ......................................... 181
open-table on relational-database ............................... 187
open-table! ................................................................ 164
operate-as ............................................................ 36
or? ........................................................................ 231
ordered-for-each-key on base-table .............................. 184
os->batch-dialect ................................................... 68
outline-rect .......................................................... 124
output-port-height .................................................. 15
output-port-width ................................................... 15
overcast-sky-color-xyy .......................................... 151
partition-page ...................................................... 123
path->uri ............................................................. 79
pathname->vicinity ............................................... 11
peano-coordinates->integer ..................................... 237
peano-coordinates->natural ..................................... 237
plot ................................................................. 118, 125
plot-column .......................................................... 121
plot-text-column ................................................... 122
pmn:array-write .................................................... 210
pmn:image-file->array ........................................... 210
pmn:type-dimensions .............................................. 210
port? .................................................................... 14
position ................................................................ 226
pprint-file ............................................................ 95
pprint-filter-file .................................................... 95
prec:commentfix .................................................... 45
prec:define-grammar ............................................ 42
prec:delim ............................................................ 44
prec:infix .............................................................. 45
prec:inmatchfix ..................................................... 46
prec:make-led ....................................................... 44
prec:make-nud ....................................................... 44
prec:matchfix ........................................................ 46
prec:nary ............................................................. 45
prec:nofix ............................................................. 44
prec:parse ............................................................ 42
prec:postfix .......................................................... 45
prec:prefix ............................................................ 44
prec:prestfix ........................................................ 45
predicate->asso ..................................................... 203
predicate->hash ..................................................... 214
predicate->hash-asso ............................................. 214
present? on base-table ............................................ 183
pretty-print .......................................................... 94
pretty-print->string ............................................... 94
primary-limit on relational-table ............................... 169
prime? ................................................................. 111
primes< ............................................................... 111
primes> ............................................................... 111
print ................................................................. 36
print-call-stack ..................................................... 260
printf ............................................................... 53
process:action ...................................................... 243
program-arguments .............................................. 261
program-vicinity .................................................. 11
project-table on relational-database ........................... 187
proper-list? .......................................................... 253
protein<cdna ....................................................... 100
provide .............................................................. 2
provided? ........................................................... 1

P
p<cdna ............................................................... 100
pad-range ........................................................... 121
pair-fold ............................................................ 255
pair-fold-right ...................................................... 255
pair-for-each ....................................................... 255
parameter-list->arglist .......................................... 63
parameter-list-expand ............................................ 62
parameter-list-ref ............................................... 62
parse-ftp-address ................................................ 80
partition ............................................................ 255
partition! ............................................................ 255
partition-page ...................................................... 123
path->uri ............................................................. 79
pathname->vicinity ............................................... 11
peano-coordinates->integer ..................................... 237
peano-coordinates->natural ..................................... 237
plot ................................................................. 118, 125
plot-column .......................................................... 121
plot-text-column ................................................... 122
pmn:array-write .................................................... 210
pmn:image-file->array ........................................... 210
pmn:type-dimensions .............................................. 210
port? .................................................................... 14
position ................................................................ 226
pprint-file ............................................................ 95
pprint-filter-file .................................................... 95
prec:commentfix .................................................... 45
prec:define-grammar ............................................ 42
prec:delim ............................................................ 44
prec:infix .............................................................. 45
prec:inmatchfix ..................................................... 46
prec:make-led ....................................................... 44
prec:make-nud ....................................................... 44
prec:matchfix ........................................................ 46
prec:nary ............................................................. 45
prec:nofix ............................................................. 44
prec:parse ............................................................ 42
prec:postfix .......................................................... 45
prec:prefix ............................................................ 44
prec:prestfix ........................................................ 45
predicate->asso ..................................................... 203
predicate->hash ..................................................... 214
predicate->hash-asso ............................................. 214
present? on base-table ............................................ 183
pretty-print .......................................................... 94
pretty-print->string ............................................... 94
primary-limit on relational-table ............................... 169
prime? ................................................................. 111
primes< ............................................................... 111
primes> ............................................................... 111
print ................................................................. 36
print-call-stack ..................................................... 260
printf ............................................................... 53
process:action ...................................................... 243
program-arguments .............................................. 261
program-vicinity .................................................. 11
project-table on relational-database ........................... 187
proper-list? .......................................................... 253
protein<cdna ....................................................... 100
provide .............................................................. 2
provided? ........................................................... 1

Q
qp ................................................................. 257
qpn ................................................................. 257
qpr ................................................................. 257
queue-empty? ..................................................... 220
queue-front ....................................................... 220
queue-pop! ........................................... 220
queue-push! .......................................... 220
queue-rew! .......................................... 220
queue? ................................................ 220
quo ..................................................... 110
quotient .............................................. 109

R
random ................................................. 112
random:exp ......................................... 113
random:hollow-sphere! ........................... 113
random:normal ...................................... 113
random:normal-vector! ........................... 113
random:solid-sphere! .............................. 113
random:uniform .................................... 113
rationalize .......................................... 249
read-byte ........................................... 205
read-bytes .......................................... 205
read-cie-illuminant .............................. 141
read-command ....................................... 61
read-line ........................................... 243
read-line! ........................................... 243
read-normalized-illuminant .................... 142
read-options-file .................................. 61
real:acos ........................................... 109
real:asin ........................................... 109
real:atan ........................................... 109
real:cos ............................................ 109
real:exp ............................................ 109
real:expt ........................................... 109
real:ln ............................................... 109
real:log ............................................ 109
real:sin ............................................. 109
real:sqrt ........................................... 109
real:tan ............................................. 109
receive .............................................. 34
record-accessor .................................... 221
record-constructor ............................... 220
record-modifier ................................... 221
record-predicate ................................... 221
reduce .............................................. 211, 227, 255
reduce-init ........................................ 227
rem ..................................................... 110
remainder .......................................... 109
remove .............................................. 225, 255
remove! ............................................. 255
remove-duplicates ................................ 226
remove-if .......................................... 225
remove-if-not ..................................... 226
remove-parameter ................................. 62
remove-setter-for ................................. 37
repl:quit .......................................... 257
repl:top-level ...................................... 257
replace-suffix ..................................... 65
require .............................................. 2, 4
require-if .......................................... 2
resample-array! .................................... 203
resene .............................................. 289
restrict-table on relational-database .......... 187
reverse! ............................................ 254
reverse-bit-field .................................. 107
rgb709->CIEXYZ .................................... 146
rgb709->color ...................................... 137
rotate-bit-field ................................... 106
round-quotient .................................... 109
row:delete on relational-table ................. 166
row:delete* on relational-table ............... 167
row:insert on relational-table ............... 165
row:insert* on relational-table ............... 167
row:remove on relational-table ............... 166
row:remove* on relational-table ............... 167
row:retrieve on relational-table ............... 165
row:retrieve* on relational-table ............... 167
row:update on relational-table ............... 165
row:update* on relational-table ............... 167
rule-horizontal .................................... 124
rule-vertical ....................................... 124
S
saturate .............................................. 148
scanf ................................................. 56
scanf-read-list .................................... 56
scene:overcast ..................................... 128
scene:panorama .................................... 127
scene:sky-and-dirt ................................ 128
scene:sky-and-grass ................................ 128
scene:space ........................................ 127
scene:sky ........................................... 128
scene:viewpoint ................................... 128
scene:viewpoints .................................. 128
scheme-report-environment ...................... 251
schmooz ............................................. 101
secant:find-bracketed-root ..................... 152
secant:find-root .................................. 152
second .............................................. 254
seed->random-state ................................ 112
set ..................................................... 37
set-color ........................................... 122
set-difference ..................................... 223
set-font ........................................... 122
set-glyphsize ...................................... 123
set-linedash ....................................... 123
set-linewidth ...................................... 123
set-margin-templates ................................ 124
setter ................................................ 37
Setter .............................................. 212
setup-plot .......................................... 121
seventh ............................................. 254
sft ..................................................... 113
sft-1 ................................................ 114
si:conversion-factor ................................ 246
singleton-wt-tree ................................ 190
sixth ............................................... 254
size ................................................... 36, 212
<table>
<thead>
<tr>
<th>Procedure and Macro Index</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>sky-color-xyy</td>
<td>151</td>
</tr>
<tr>
<td>slib: error</td>
<td>16</td>
</tr>
<tr>
<td>slib: eval</td>
<td>15</td>
</tr>
<tr>
<td>slib: eval-load</td>
<td>15</td>
</tr>
<tr>
<td>slib: exit</td>
<td>16</td>
</tr>
<tr>
<td>slib: in-catalog?</td>
<td>3</td>
</tr>
<tr>
<td>slib: load</td>
<td>15</td>
</tr>
<tr>
<td>slib: load-compiled</td>
<td>15</td>
</tr>
<tr>
<td>slib: load-source</td>
<td>15</td>
</tr>
<tr>
<td>slib: report</td>
<td>13</td>
</tr>
<tr>
<td>slib: report-version</td>
<td>13</td>
</tr>
<tr>
<td>slib: warn</td>
<td>16</td>
</tr>
<tr>
<td>snap-range</td>
<td>121</td>
</tr>
<tr>
<td>software-type</td>
<td>12</td>
</tr>
<tr>
<td>solar-declination</td>
<td>150</td>
</tr>
<tr>
<td>solar-hour</td>
<td>150</td>
</tr>
<tr>
<td>solar-polar</td>
<td>150</td>
</tr>
<tr>
<td>solid: arrow</td>
<td>134</td>
</tr>
<tr>
<td>solid: barrelief</td>
<td>132</td>
</tr>
<tr>
<td>solid: box</td>
<td>130</td>
</tr>
<tr>
<td>solid: center-array-of</td>
<td>134</td>
</tr>
<tr>
<td>solid: center-pile-of</td>
<td>134</td>
</tr>
<tr>
<td>solid: center-row-of</td>
<td>134</td>
</tr>
<tr>
<td>solid: color</td>
<td>132, 133</td>
</tr>
<tr>
<td>solid: cone</td>
<td>131</td>
</tr>
<tr>
<td>solid: cylinder</td>
<td>131</td>
</tr>
<tr>
<td>solid: disk</td>
<td>131</td>
</tr>
<tr>
<td>solid: ellipsoid</td>
<td>131</td>
</tr>
<tr>
<td>solid: font</td>
<td>133</td>
</tr>
<tr>
<td>solid: lumber</td>
<td>131</td>
</tr>
<tr>
<td>solid: polyline</td>
<td>131</td>
</tr>
<tr>
<td>solid: prism</td>
<td>132</td>
</tr>
<tr>
<td>solid: pyramid</td>
<td>131</td>
</tr>
<tr>
<td>solid: rotation</td>
<td>134</td>
</tr>
<tr>
<td>solid: scale</td>
<td>134</td>
</tr>
<tr>
<td>solid: sphere</td>
<td>131</td>
</tr>
<tr>
<td>solid: text</td>
<td>132</td>
</tr>
<tr>
<td>solid: texture</td>
<td>133</td>
</tr>
<tr>
<td>solid: translation</td>
<td>134</td>
</tr>
<tr>
<td>solidify: database</td>
<td>163</td>
</tr>
<tr>
<td>solidify: database on relational-database</td>
<td>187</td>
</tr>
<tr>
<td>some</td>
<td>224</td>
</tr>
<tr>
<td>sort</td>
<td>234</td>
</tr>
<tr>
<td>sort!</td>
<td>234</td>
</tr>
<tr>
<td>sorted?</td>
<td>234</td>
</tr>
<tr>
<td>soundex</td>
<td>238</td>
</tr>
<tr>
<td>span</td>
<td>255</td>
</tr>
<tr>
<td>span!</td>
<td>255</td>
</tr>
<tr>
<td>spectrum-&gt;chromaticity</td>
<td>143</td>
</tr>
<tr>
<td>spectrum-&gt;XYZ</td>
<td>142</td>
</tr>
<tr>
<td>split-at</td>
<td>254</td>
</tr>
<tr>
<td>split-at!</td>
<td>254</td>
</tr>
<tr>
<td>sprintf</td>
<td>53</td>
</tr>
<tr>
<td>sRGB-&gt;CIEXYZ</td>
<td>146</td>
</tr>
<tr>
<td>srgb-&gt;color</td>
<td>140</td>
</tr>
<tr>
<td>sRGB-&gt;e-sRGB</td>
<td>146</td>
</tr>
<tr>
<td>sRGB-&gt;xRGB</td>
<td>146</td>
</tr>
<tr>
<td>ssax: assert-current-char</td>
<td>81</td>
</tr>
<tr>
<td>ssax: assert-token</td>
<td>90</td>
</tr>
<tr>
<td>ssax: complete-start-tag</td>
<td>88</td>
</tr>
<tr>
<td>ssax: handle-parsed-entity</td>
<td>87</td>
</tr>
<tr>
<td>ssax: init-buffer</td>
<td>81</td>
</tr>
<tr>
<td>ssax: make-elem-parser</td>
<td>90</td>
</tr>
<tr>
<td>ssax: make-parser</td>
<td>91</td>
</tr>
<tr>
<td>ssax: make-pi-parser</td>
<td>90</td>
</tr>
<tr>
<td>ssax: next-token</td>
<td>81</td>
</tr>
<tr>
<td>ssax: next-token-of</td>
<td>81</td>
</tr>
<tr>
<td>ssax: read-attributes</td>
<td>87</td>
</tr>
<tr>
<td>ssax: read-cdata-body</td>
<td>86</td>
</tr>
<tr>
<td>ssax: read-char-data</td>
<td>89</td>
</tr>
<tr>
<td>ssax: read-char-ref</td>
<td>86</td>
</tr>
<tr>
<td>ssax: read-external-id</td>
<td>89</td>
</tr>
<tr>
<td>ssax: read-markup-token</td>
<td>85</td>
</tr>
<tr>
<td>ssax: read-name</td>
<td>85</td>
</tr>
<tr>
<td>ssax: read-pi-body-as-string</td>
<td>86</td>
</tr>
<tr>
<td>ssax: read-string</td>
<td>82</td>
</tr>
<tr>
<td>ssax: resolve-name</td>
<td>88</td>
</tr>
<tr>
<td>ssax: reverse-collect-str</td>
<td>80</td>
</tr>
<tr>
<td>ssax: reverse-collect-str-drop-ws</td>
<td>80</td>
</tr>
<tr>
<td>ssax: scan-misc</td>
<td>89</td>
</tr>
<tr>
<td>ssax: skip-internal-std</td>
<td>86</td>
</tr>
<tr>
<td>ssax: skip-pi</td>
<td>86</td>
</tr>
<tr>
<td>ssax: skip-s</td>
<td>84</td>
</tr>
<tr>
<td>ssax: skip-while</td>
<td>81</td>
</tr>
<tr>
<td>ssax: xml-&gt;xmi</td>
<td>93</td>
</tr>
<tr>
<td>stack</td>
<td>260</td>
</tr>
<tr>
<td>stack-all</td>
<td>258</td>
</tr>
<tr>
<td>stackf</td>
<td>260</td>
</tr>
<tr>
<td>string-&gt;bytes</td>
<td>204</td>
</tr>
<tr>
<td>string-&gt;color</td>
<td>136</td>
</tr>
<tr>
<td>string-&gt;capitalized</td>
<td>241</td>
</tr>
<tr>
<td>string-&gt;capitalized!</td>
<td>242</td>
</tr>
<tr>
<td>string-ci-&gt;symbol</td>
<td>242</td>
</tr>
<tr>
<td>string-copy</td>
<td>249</td>
</tr>
<tr>
<td>string-downcase</td>
<td>241</td>
</tr>
<tr>
<td>string-downcase!</td>
<td>242</td>
</tr>
<tr>
<td>string-fill</td>
<td>249</td>
</tr>
<tr>
<td>string-index</td>
<td>239</td>
</tr>
<tr>
<td>string-index-ci</td>
<td>239</td>
</tr>
<tr>
<td>string-join</td>
<td>68</td>
</tr>
<tr>
<td>string-null</td>
<td>248</td>
</tr>
<tr>
<td>string-reverse-index</td>
<td>239</td>
</tr>
<tr>
<td>string-reverse-index-ci</td>
<td>239</td>
</tr>
<tr>
<td>string-subst</td>
<td>240</td>
</tr>
<tr>
<td>string-upcase</td>
<td>241</td>
</tr>
<tr>
<td>string-upcase!</td>
<td>242</td>
</tr>
<tr>
<td>StudyCapsExpand</td>
<td>242</td>
</tr>
<tr>
<td>sub-vicinity</td>
<td>12</td>
</tr>
<tr>
<td>subarray</td>
<td>200</td>
</tr>
<tr>
<td>subbytes</td>
<td>205</td>
</tr>
<tr>
<td>subbytes-read!</td>
<td>206</td>
</tr>
<tr>
<td>subbytes-write</td>
<td>206</td>
</tr>
<tr>
<td>subset?</td>
<td>223</td>
</tr>
</tbody>
</table>
## Variable Index

<table>
<thead>
<tr>
<th>Variable</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td><em>argv</em></td>
<td>58</td>
</tr>
<tr>
<td><em>base-table-implementations</em></td>
<td>180</td>
</tr>
<tr>
<td><em>catalog</em></td>
<td>2</td>
</tr>
<tr>
<td><em>http:byline</em></td>
<td>75</td>
</tr>
<tr>
<td><em>operating-system</em></td>
<td>66</td>
</tr>
<tr>
<td><em>optarg</em></td>
<td>58</td>
</tr>
<tr>
<td><em>optind</em></td>
<td>58</td>
</tr>
<tr>
<td><em>qwidth</em></td>
<td>257</td>
</tr>
<tr>
<td><em>random-state</em></td>
<td>112</td>
</tr>
<tr>
<td><em>ruleset</em></td>
<td>156</td>
</tr>
<tr>
<td><em>syn-defs</em></td>
<td>41</td>
</tr>
<tr>
<td><em>syn-ignore-whitespace</em></td>
<td>41</td>
</tr>
<tr>
<td><em>timezone</em></td>
<td></td>
</tr>
<tr>
<td><em>syn-ignore-whitespace</em></td>
<td></td>
</tr>
<tr>
<td><em>syn-defs</em></td>
<td></td>
</tr>
<tr>
<td><em>timezone</em></td>
<td></td>
</tr>
<tr>
<td>atm-hec-polynomial</td>
<td>116</td>
</tr>
<tr>
<td>bottomedge</td>
<td>124</td>
</tr>
<tr>
<td>charplot:dimensions</td>
<td>12</td>
</tr>
<tr>
<td>charplot:dimensions</td>
<td>177</td>
</tr>
<tr>
<td>CIEXYZ:A</td>
<td>146</td>
</tr>
<tr>
<td>CIEXYZ:B</td>
<td>146</td>
</tr>
<tr>
<td>CIEXYZ:C</td>
<td>146</td>
</tr>
<tr>
<td>CIEXYZ:D65</td>
<td>146</td>
</tr>
<tr>
<td>CIEXYZ:D65</td>
<td>146</td>
</tr>
<tr>
<td>CIEXYZ:E</td>
<td>146</td>
</tr>
<tr>
<td>crc-08-polynomial</td>
<td>116</td>
</tr>
<tr>
<td>crc-10-polynomial</td>
<td>116</td>
</tr>
<tr>
<td>crc-12-polynomial</td>
<td>116</td>
</tr>
<tr>
<td>crc-16-polynomial</td>
<td>115</td>
</tr>
<tr>
<td>crc-32-polynomial</td>
<td>115</td>
</tr>
<tr>
<td>crc-ccitt-polynomial</td>
<td>115</td>
</tr>
<tr>
<td>D50</td>
<td>136</td>
</tr>
<tr>
<td>D65</td>
<td>136</td>
</tr>
<tr>
<td>daylight?</td>
<td>98</td>
</tr>
<tr>
<td>debug:max-count</td>
<td>259</td>
</tr>
<tr>
<td>distribute*</td>
<td>157</td>
</tr>
<tr>
<td>distribute/</td>
<td>157</td>
</tr>
<tr>
<td>dowcrc-polynomial</td>
<td>116</td>
</tr>
<tr>
<td>G</td>
<td></td>
</tr>
<tr>
<td>graph:dimensions</td>
<td>125</td>
</tr>
<tr>
<td>graphrect</td>
<td>123</td>
</tr>
<tr>
<td>L</td>
<td></td>
</tr>
<tr>
<td>leftedge</td>
<td>124</td>
</tr>
<tr>
<td>M</td>
<td></td>
</tr>
<tr>
<td>most-positive-fixnum</td>
<td>12</td>
</tr>
<tr>
<td>N</td>
<td></td>
</tr>
<tr>
<td>nil</td>
<td>17</td>
</tr>
<tr>
<td>number-wt-type</td>
<td>189</td>
</tr>
<tr>
<td>P</td>
<td></td>
</tr>
<tr>
<td>plotrect</td>
<td>123</td>
</tr>
<tr>
<td>prime:prngs</td>
<td>111</td>
</tr>
<tr>
<td>prime:trials</td>
<td>111</td>
</tr>
<tr>
<td>R</td>
<td></td>
</tr>
<tr>
<td>rightedge</td>
<td>124</td>
</tr>
<tr>
<td>S</td>
<td></td>
</tr>
<tr>
<td>slib:form-feed</td>
<td>12</td>
</tr>
<tr>
<td>slib:tab</td>
<td>12</td>
</tr>
<tr>
<td>stderr</td>
<td>53</td>
</tr>
<tr>
<td>stdin</td>
<td>53</td>
</tr>
<tr>
<td>stdout</td>
<td>53</td>
</tr>
<tr>
<td>string-wt-type</td>
<td>189</td>
</tr>
<tr>
<td>T</td>
<td></td>
</tr>
<tr>
<td>tok:decimal-digits</td>
<td>17</td>
</tr>
<tr>
<td>tok:lower-case</td>
<td>43</td>
</tr>
<tr>
<td>tok:upper-case</td>
<td>43</td>
</tr>
<tr>
<td>tok:whitespaces</td>
<td>43</td>
</tr>
<tr>
<td>topeedge</td>
<td>124</td>
</tr>
<tr>
<td>tzname</td>
<td>98</td>
</tr>
<tr>
<td>U</td>
<td></td>
</tr>
<tr>
<td>usb-token-polynomial</td>
<td>116</td>
</tr>
</tbody>
</table>
Concept and Feature Index

A
aggregate .................................................. 3, 9
alarm ......................................................... 243
alarm-interrupt ........................................... 243
alist ........................................................... 203
alist-table .................................................. 179, 180, 186
and-let* ...................................................... 34
ange-ftp ...................................................... 80
appearance ................................................... 133
array ........................................................... 196
array-for-each .............................................. 201
association function ...................................... 203
Attribute ..................................................... 87
attribute-value ............................................ 70
AttValue ...................................................... 87
Auto-sharing ................................................ 162

B
balanced binary trees ..................................... 188
base ............................................................ 78
base-table ..................................................... 179
batch ........................................................... 66, 68
bignum .......................................................... 1
binary .......................................................... 205
binary trees .................................................... 188
binary trees, as discrete maps ............................ 188
binary trees, as sets .......................................... 188
binding power ................................................ 40
break ............................................................. 258
byte ............................................................. 204
byte-number ................................................... 206

calendar-time ............................................... 96, 98
Calendar-Time .............................................. 98
caltime ........................................................ 98
canonical ....................................................... 147
careful .......................................................... 155
catalog .......................................................... 2
Catalog File .................................................... 3
certificate ...................................................... 262
cgi ............................................................... 75
chapter-order ................................................ 232
charplot ...................................................... 117, 118
Chroma .......................................................... 139
cie1931 .......................................................... 141
cie1964 .......................................................... 141
ciexyz ........................................................... 141

CIEXYZ ........................................................ 137
class-string .................................................... 117
coerce ......................................................... 241
collect ........................................................... 210
color-database .............................................. 148
color-names ................................................... 147
command line ................................................ 61
commentfix ................................................... 41
common-list-functions .................................. 211, 221
commutative-ring .......................................... 155
compiled ........................................................ 3
compiling ....................................................... 6
complex ........................................................ 1
Coordinated Universal Time .............................. 98
copyright ....................................................... 272
crc ............................................................... 114, 117
cvs .............................................................. 264

debug ........................................................... 258
defmacro ....................................................... 3

defmacroexpand ............................................ 18, 95
delim ........................................................... 41
dequeues ....................................................... 220
determinant ................................................... 160
dft, Fourier-transform .................................... 113
diff ............................................................... 240
directory ....................................................... 261
Discrete Fourier Transform .............................. 113
discrete maps, using binary trees ....................... 188
DrScheme .................................................... 268
dynamic ........................................................ 213
dynamic-wind ................................................ 250

database-commands ....................................... 174
databases ..................................................... 68, 162, 171, 174
daylight ....................................................... 149
db->html ........................................................ 72
debug ........................................................... 258
defrecord-type ............................................... 33
defmacro ....................................................... 3
defmacroexpand ............................................ 18, 95
delims .......................................................... 41
dequeues ....................................................... 220
determinant ................................................... 160
dft, Fourier-transform .................................... 113
diff ............................................................... 240
directory ....................................................... 261
Discrete Fourier Transform .............................. 113
discrete maps, using binary trees ....................... 188
DrScheme .................................................... 268
dynamic ........................................................ 213
dynamic-wind ................................................ 250

e-eRGB ........................................................ 140
ELK ............................................................. 268
emacs .......................................................... 268
Encapsulated-PostScript ................................ 120
escaped ........................................................ 79
EUC ............................................................. 265
Euclidean Domain ......................................... 156
eval ............................................................. 250
exchanger ..................................................... 16
<table>
<thead>
<tr>
<th>Concept and Feature Index</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>F</strong></td>
</tr>
<tr>
<td>factor</td>
</tr>
<tr>
<td>feature</td>
</tr>
<tr>
<td>feature</td>
</tr>
<tr>
<td>File Transfer Protocol</td>
</tr>
<tr>
<td>file-lock</td>
</tr>
<tr>
<td>filename</td>
</tr>
<tr>
<td>fluid-let</td>
</tr>
<tr>
<td>fold</td>
</tr>
<tr>
<td>form</td>
</tr>
<tr>
<td>format</td>
</tr>
<tr>
<td><strong>G</strong></td>
</tr>
<tr>
<td>Gambit-C 3.0</td>
</tr>
<tr>
<td>gnunet</td>
</tr>
<tr>
<td>Gauche-0.9</td>
</tr>
<tr>
<td>generic-write</td>
</tr>
<tr>
<td>getenv</td>
</tr>
<tr>
<td>getit</td>
</tr>
<tr>
<td>getopt</td>
</tr>
<tr>
<td>getopt-parameters</td>
</tr>
<tr>
<td>glob</td>
</tr>
<tr>
<td>Gray code</td>
</tr>
<tr>
<td>guarded-cond-clause</td>
</tr>
<tr>
<td>Guile</td>
</tr>
<tr>
<td><strong>H</strong></td>
</tr>
<tr>
<td>hash</td>
</tr>
<tr>
<td>hash-table</td>
</tr>
<tr>
<td>Hilbert</td>
</tr>
<tr>
<td>Hilbert Space-Filling Curve</td>
</tr>
<tr>
<td>hilbert-fill</td>
</tr>
<tr>
<td>HOME</td>
</tr>
<tr>
<td>homecat</td>
</tr>
<tr>
<td>html-for-each</td>
</tr>
<tr>
<td>html-form</td>
</tr>
<tr>
<td>http</td>
</tr>
<tr>
<td>Hue</td>
</tr>
<tr>
<td><strong>I</strong></td>
</tr>
<tr>
<td>ICC Profile</td>
</tr>
<tr>
<td>implicat</td>
</tr>
<tr>
<td>indexed-sequential-access-method</td>
</tr>
<tr>
<td>inexact</td>
</tr>
<tr>
<td>infix</td>
</tr>
<tr>
<td>Info</td>
</tr>
<tr>
<td>inmatchfix</td>
</tr>
<tr>
<td>install</td>
</tr>
<tr>
<td>installation</td>
</tr>
<tr>
<td>intrinsic feature</td>
</tr>
<tr>
<td>ISAM</td>
</tr>
<tr>
<td><strong>J</strong></td>
</tr>
<tr>
<td>Japanese</td>
</tr>
<tr>
<td>JFILTER</td>
</tr>
<tr>
<td>JIS</td>
</tr>
<tr>
<td><strong>K</strong></td>
</tr>
<tr>
<td>Kawa</td>
</tr>
<tr>
<td><strong>L</strong></td>
</tr>
<tr>
<td>L<em>a</em>b*</td>
</tr>
<tr>
<td>L<em>C</em>h</td>
</tr>
<tr>
<td>L<em>u</em>v*</td>
</tr>
<tr>
<td>lamination</td>
</tr>
<tr>
<td>Larceny</td>
</tr>
<tr>
<td>Left Denotation, led</td>
</tr>
<tr>
<td>let-values</td>
</tr>
<tr>
<td>Lightness</td>
</tr>
<tr>
<td>line-i</td>
</tr>
<tr>
<td>list-processing library</td>
</tr>
<tr>
<td>load-option</td>
</tr>
<tr>
<td>logical</td>
</tr>
<tr>
<td><strong>M</strong></td>
</tr>
<tr>
<td>macro</td>
</tr>
<tr>
<td>macro</td>
</tr>
<tr>
<td>macro-by-example</td>
</tr>
<tr>
<td>macro-by-example</td>
</tr>
<tr>
<td>macros-that-work</td>
</tr>
<tr>
<td>macros-that-work</td>
</tr>
<tr>
<td>manifest</td>
</tr>
<tr>
<td>match</td>
</tr>
<tr>
<td>match-keys</td>
</tr>
<tr>
<td>matchfix</td>
</tr>
<tr>
<td>matfile</td>
</tr>
<tr>
<td>math-integer</td>
</tr>
<tr>
<td>math-real</td>
</tr>
<tr>
<td>matlab</td>
</tr>
<tr>
<td>metric-units</td>
</tr>
<tr>
<td>minimize</td>
</tr>
<tr>
<td>minimum field width (printf)</td>
</tr>
<tr>
<td>MIT Scheme</td>
</tr>
<tr>
<td>mkimpcat.scm</td>
</tr>
<tr>
<td>mklibcat.scm</td>
</tr>
<tr>
<td>modular</td>
</tr>
<tr>
<td>multiarg</td>
</tr>
<tr>
<td>multiarg-apply</td>
</tr>
<tr>
<td>MzScheme</td>
</tr>
<tr>
<td><strong>N</strong></td>
</tr>
<tr>
<td>nary</td>
</tr>
<tr>
<td>ncbi-dma</td>
</tr>
<tr>
<td>new-catalog</td>
</tr>
<tr>
<td>nofix</td>
</tr>
<tr>
<td>null</td>
</tr>
<tr>
<td>Null Denotation, nud</td>
</tr>
<tr>
<td>O</td>
</tr>
<tr>
<td>P</td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td>Q</td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td>R</td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td>S</td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td>Concept</td>
</tr>
<tr>
<td>---------</td>
</tr>
<tr>
<td>srfi-11</td>
</tr>
<tr>
<td>srfi-2</td>
</tr>
<tr>
<td>srfi-23</td>
</tr>
<tr>
<td>srfi-28</td>
</tr>
<tr>
<td>srfi-39</td>
</tr>
<tr>
<td>srfi-47</td>
</tr>
<tr>
<td>srfi-59</td>
</tr>
<tr>
<td>srfi-60</td>
</tr>
<tr>
<td>srfi-61</td>
</tr>
<tr>
<td>srfi-63</td>
</tr>
<tr>
<td>srfi-8</td>
</tr>
<tr>
<td>srfi-9</td>
</tr>
<tr>
<td>srfi-94</td>
</tr>
<tr>
<td>srfi-95</td>
</tr>
<tr>
<td>srfi-96</td>
</tr>
<tr>
<td>sRGB</td>
</tr>
<tr>
<td>stdio</td>
</tr>
<tr>
<td>string-case</td>
</tr>
<tr>
<td>string-port</td>
</tr>
<tr>
<td>string-search</td>
</tr>
<tr>
<td>subarray</td>
</tr>
<tr>
<td>sun</td>
</tr>
<tr>
<td>sunlight</td>
</tr>
<tr>
<td>symmetric</td>
</tr>
<tr>
<td>syntactic-closures</td>
</tr>
<tr>
<td>syntactic-closures</td>
</tr>
<tr>
<td>syntax tree</td>
</tr>
<tr>
<td>syntax-case</td>
</tr>
<tr>
<td>syntax-case</td>
</tr>
<tr>
<td>system</td>
</tr>
</tbody>
</table>

**T**

<table>
<thead>
<tr>
<th>Concept</th>
<th>Page Numbers</th>
</tr>
</thead>
<tbody>
<tr>
<td>time</td>
<td>96</td>
</tr>
<tr>
<td>time-zone</td>
<td>97</td>
</tr>
<tr>
<td>top-level variable references</td>
<td>9</td>
</tr>
<tr>
<td>top-refs</td>
<td>9</td>
</tr>
<tr>
<td>topological-sort</td>
<td>234</td>
</tr>
<tr>
<td>trace</td>
<td>259</td>
</tr>
<tr>
<td>transact</td>
<td>262</td>
</tr>
<tr>
<td>transcript</td>
<td>248</td>
</tr>
<tr>
<td>tree</td>
<td>232</td>
</tr>
<tr>
<td>trees, balanced binary</td>
<td>188</td>
</tr>
<tr>
<td>tristimulus</td>
<td>136</td>
</tr>
<tr>
<td>tsort</td>
<td>234, 235</td>
</tr>
<tr>
<td>turbidity</td>
<td>150</td>
</tr>
<tr>
<td>TZ-string</td>
<td>96</td>
</tr>
</tbody>
</table>

**U**

<table>
<thead>
<tr>
<th>Concept</th>
<th>Page Numbers</th>
</tr>
</thead>
<tbody>
<tr>
<td>Uniform Resource Identifiers</td>
<td>78</td>
</tr>
<tr>
<td>Uniform Resource Locator</td>
<td>80</td>
</tr>
<tr>
<td>Unique Factorization</td>
<td>156</td>
</tr>
<tr>
<td>unsafe</td>
<td>79</td>
</tr>
<tr>
<td>uri</td>
<td>78</td>
</tr>
<tr>
<td>URI</td>
<td>75, 76, 79</td>
</tr>
<tr>
<td>usercat</td>
<td>5</td>
</tr>
<tr>
<td>UTC</td>
<td>98</td>
</tr>
</tbody>
</table>

**V**

<table>
<thead>
<tr>
<th>Concept</th>
<th>Page Numbers</th>
</tr>
</thead>
<tbody>
<tr>
<td>values</td>
<td>252</td>
</tr>
<tr>
<td>variable references</td>
<td>9</td>
</tr>
<tr>
<td>vet</td>
<td>10</td>
</tr>
<tr>
<td>VSM</td>
<td>269</td>
</tr>
</tbody>
</table>

**W**

<table>
<thead>
<tr>
<th>Concept</th>
<th>Page Numbers</th>
</tr>
</thead>
<tbody>
<tr>
<td>WB</td>
<td>180</td>
</tr>
<tr>
<td>wb-table</td>
<td>180</td>
</tr>
<tr>
<td>weight-balanced binary trees</td>
<td>188</td>
</tr>
<tr>
<td>wget</td>
<td>148</td>
</tr>
<tr>
<td>white point</td>
<td>136</td>
</tr>
<tr>
<td>wild-card</td>
<td>183</td>
</tr>
<tr>
<td>with-file</td>
<td>247</td>
</tr>
<tr>
<td>Word</td>
<td>262</td>
</tr>
<tr>
<td>wt-tree</td>
<td>188</td>
</tr>
</tbody>
</table>

**X**

<table>
<thead>
<tr>
<th>Concept</th>
<th>Page Numbers</th>
</tr>
</thead>
<tbody>
<tr>
<td>xRGB</td>
<td>140</td>
</tr>
<tr>
<td>xyY</td>
<td>144</td>
</tr>
</tbody>
</table>

**Y**

<table>
<thead>
<tr>
<th>Concept</th>
<th>Page Numbers</th>
</tr>
</thead>
<tbody>
<tr>
<td>yasos</td>
<td>35</td>
</tr>
</tbody>
</table>