This manual is for SCM (version 5f1, May 2013), an implementation of the algorithmic language Scheme.

Copyright © 1990-2007 Free Software Foundation, Inc.

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 Overview .................................................. 1
  1.1 Features .................................................. 1
  1.2 Authors .................................................. 1
  1.3 Copyright .................................................. 2
    1.3.1 The SCM License ....................................... 2
    1.3.2 SIOD copyright ......................................... 2
    1.3.3 GNU Free Documentation License ....................... 3
  1.4 Bibliography .............................................. 10

2 Installing SCM ............................................. 12
  2.1 Distributions .............................................. 12
  2.2 GNU configure and make .................................... 12
    2.2.1 Making scmlit ......................................... 13
    2.2.2 Makefile targets ....................................... 13
  2.3 Building SCM .............................................. 14
    2.3.1 Invoking Build ......................................... 15
    2.3.2 Build Options ......................................... 17
    2.3.3 Compiling and Linking Custom Files ................... 22
  2.4 Saving Executable Images .................................. 23
  2.5 Installation ............................................... 24
  2.6 Troubleshooting and Testing ................................ 24
    2.6.1 Problems Compiling .................................... 24
    2.6.2 Problems Linking ....................................... 25
    2.6.3 Testing ................................................ 25
    2.6.4 Problems Starting ...................................... 26
    2.6.5 Problems Running ...................................... 26
    2.6.6 Reporting Problems .................................... 27

3 Operational Features ................................. 28
  3.1 Invoking SCM .............................................. 28
  3.2 Options .................................................. 28
  3.3 Invocation Examples ....................................... 30
  3.4 Environment Variables .................................... 31
  3.5 Scheme Variables ......................................... 31
  3.6 SCM Session ............................................... 31
  3.7 Editing Scheme Code ..................................... 32
  3.8 Debugging Scheme Code ................................... 33
  3.9 Debugging Continuations ................................ 35
  3.10 Errors .................................................... 36
  3.11 Memoized Expressions .................................. 38
  3.12 Internal State ............................................ 39
    3.12.1 Executable path ...................................... 40
4 The Language ................................................. 44
  4.1 Standards Compliance .................................. 44
  4.2 Storage ................................................. 46
  4.3 Time .................................................. 46
  4.4 Interrupts .............................................. 47
  4.5 Process Synchronization ................................. 48
  4.6 Files and Ports ......................................... 49
    4.6.1 Opening and Closing ............................... 49
    4.6.2 Port Properties .................................. 50
    4.6.3 Port Redirection .................................. 51
    4.6.4 Soft Ports ........................................ 52
  4.7 Eval and Load ........................................... 52
    4.7.1 Line Numbers ...................................... 53
  4.8 Lexical Conventions .................................... 54
    4.8.1 Common-Lisp Read Syntax ........................... 54
    4.8.2 Load Syntax ....................................... 55
    4.8.3 Documentation and Comments ....................... 55
    4.8.4 Modifying Read Syntax ............................. 55
  4.9 Syntax ................................................ 56
    4.9.1 Define and Set .................................... 56
    4.9.2 Defmacro .......................................... 58
    4.9.3 Syntax-Rules ...................................... 58
    4.9.4 Macro Primitives .................................. 59
    4.9.5 Environment Frames ................................. 60
    4.9.6 Syntactic Hooks for Hygienic Macros ............... 61
    4.9.7 Use of Synthetic Identifiers ....................... 61
  4.10 Regular Expression Pattern Matching ................... 62
  4.11 Line Editing ........................................... 62
  4.12 Dynamic Linking ....................................... 62

5 Packages .................................................. 64
  5.1 Dynamic Linking ....................................... 64
  5.2 Dump .................................................. 65
  5.3 Numeric ............................................... 67
  5.4 Arrays ............................................... 68
    5.4.1 Conventional Arrays ............................... 69
    5.4.2 Uniform Array .................................... 70
    5.4.3 Bit Vectors ...................................... 71
    5.4.4 Array Mapping .................................... 72
  5.5 Records ............................................... 73
  5.6 I/O-Extensions ........................................ 73
  5.7 Posix Extensions ....................................... 77
  5.8 Unix Extensions ....................................... 81
  5.9 Sequence Comparison ................................... 82
  5.10 Regular Expression Pattern Matching ................... 82
  5.11 Line Editing ........................................... 84
5.12 Curses ....................................................... 84
  5.12.1 Output Options Setting ................................. 84
  5.12.2 Terminal Mode Setting ................................. 85
  5.12.3 Window Manipulation ................................. 86
  5.12.4 Output ............................................... 87
  5.12.5 Input ................................................ 89
  5.12.6 Curses Miscellany .................................. 89
5.13 Sockets .................................................... 90
  5.13.1 Host and Other Inquiries .............................. 90
  5.13.2 Internet Addresses and Socket Names ............... 91
  5.13.3 Socket ............................................. 92
5.14 SCMDB ..................................................... 95
5.15 Xlibscm ..................................................... 95
5.16 Hobbit ....................................................... 95

6 The Implementation ......................................... 96

6.1 Data Types .................................................. 96
  6.1.1 Immediates ............................................ 96
  6.1.2 Cells ................................................. 98
  6.1.3 Header Cells ........................................ 99
  6.1.4 Subr Cells .......................................... 101
  6.1.5 Defining Subrs ..................................... 102
  6.1.6 Ptob Cells .......................................... 103
  6.1.7 Defining Ptos ..................................... 104
  6.1.8 Smob Cells .......................................... 105
  6.1.9 Defining Smobs .................................... 106
  6.1.10 Data Type Representations .......................... 107
6.2 Operations ................................................... 109
  6.2.1 Garbage Collection .................................. 109
    6.2.1.1 Marking Cells ................................ 109
    6.2.1.2 Sweeping the Heap ............................ 110
  6.2.2 Memory Management for Environments ............... 110
  6.2.3 Dynamic Linking Support ............................ 111
  6.2.4 Configure Module Catalog ............................ 112
  6.2.5 Automatic C Preprocessor Definitions ............. 113
  6.2.6 Signals ............................................. 115
  6.2.7 C Macros ............................................ 115
  6.2.8 Changing Scm ...................................... 116
  6.2.9 Allocating memory .................................. 118
  6.2.10 Embedding SCM .................................... 119
  6.2.11 Callbacks .......................................... 122
  6.2.12 Type Conversions ................................ 123
  6.2.13 Continuations ..................................... 124
  6.2.14 Evaluation ......................................... 126
6.3 Program Self-Knowledge .................................. 127
  6.3.1 File-System Habitat ................................ 127
  6.3.2 Executable Pathname ................................ 128
  6.3.3 Script Support ...................................... 129
6.4 Improvements To Make ........................................ 129
6.4.1 VMS Dynamic Linking..................................... 130

Procedure and Macro Index ....................... 133

Variable Index........................................... 139

Type Index............................................... 140

Concept Index.......................................... 142
1 Overview

SCM is a portable Scheme implementation written in C. SCM provides a machine independent platform for [JACAL], a symbolic algebra system. SCM supports and requires the SLIB Scheme library. SCM, SLIB, and JACAL are GNU projects.

The most recent information about SCM can be found on SCM’s WWW home page:

http://people.csail.mit.edu/jaffer/SCM

1.1 Features

- Support for [SICP], [R2RS], [R3RS], and [R5RS] scheme code.
- Runs under Amiga, Atari-ST, MacOS, MS-DOS, OS/2, NOS/VE, Unicos, VMS, Unix and similar systems. Supports ASCII and EBCDIC character sets.
- Is fully documented in TeXinfo form, allowing documentation to be generated in info, TeX, html, nroff, and troff formats.
- Supports inexact real and complex numbers, 30 bit immediate integers and large precision integers.
- Char-code-limit, most-positive-fixnum, most-negative-fixnum, and internal-time-units-per-second constants. slib:features and *load-pathname* variables.
- Arrays and bit-vectors. String ports and software emulation ports. I/O extensions providing ANSI C and POSIX.1 facilities.
- Interfaces to standard libraries including REGEX string regular expression matching and the CURSES screen management package.
- Available add-on packages including an interactive debugger, database, X-window graphics, BGI graphics, Motif, and Open-Windows packages.
- The Hobbit compiler and dynamic linking of compiled modules.
- User definable responses to interrupts and errors, Process-synchronization primitives. Setable levels of monitoring and timing information printed interactively (the verbose function). Restart, quit, and exec.

1.2 Authors

Aubrey Jaffer (agj@alum.mit.edu)
Most of SCM.

Radey Shouman
Arrays, gsubrs, compiled closures, records, Ecache, syntax-rules macros, and safeports.
Chapter 1: Overview

Jerry D. Hedden
Real and Complex functions. Fast mixed type arithmetics.

Hugh Secker-Walker
Syntax checking and memoization of special forms by evaluator. Storage allocation strategy and parameters.

George Carrette
Siod, written by George Carrette, was the starting point for SCM. The major innovations taken from Siod are the evaluator’s use of the C-stack and being able to garbage collect off the C-stack (see Section 6.2.1 [Garbage Collection], page 109).

There are many other contributors to SCM. They are acknowledged in the file ‘ChangeLog’, a log of changes that have been made to scm.

1.3 Copyright

Authors have assigned their SCM copyrights to:

Free Software Foundation, Inc.
59 Temple Place, Suite 330, Boston, MA 02111, USA

1.3.1 The SCM License

This program is free software: you can redistribute it and/or modify it under the terms of the GNU Lesser General Public License as published by the Free Software Foundation, either version 3 of the License, or (at your option) any later version.

This program is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU Lesser General Public License for more details.

You should have received a copy of the GNU Lesser General Public License along with this program. If not, see http://www.gnu.org/licenses/.

1.3.2 SIOD copyright

COPYRIGHT © 1989 BY
PARADIGM ASSOCIATES INCORPORATED, CAMBRIDGE, MASSACHUSETTS.
ALL RIGHTS RESERVED

Permission to use, copy, modify, distribute and sell this software and its documentation for any purpose and without fee is hereby granted, provided that the above copyright notice appear in all copies and that both that copyright notice and this permission notice appear in supporting documentation, and that the name of Paradigm Associates Inc not be used in advertising or publicity pertaining to distribution of the software without specific, written prior permission.

PARADIGM DISCLAIMS ALL WARRANTIES WITH REGARD TO THIS SOFTWARE, INCLUDING ALL IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS, IN NO EVENT SHALL PARADIGM BE LIABLE FOR ANY SPECIAL, INDIRECT OR CONSEQUENTIAL DAMAGES OR ANY DAMAGES WHATSOEVER RESULTING
Chapter 1: Overview

FROM LOSS OF USE, DATA OR PROFITS, WHETHER IN AN ACTION OF CONTRACT, NEGLIGENCE OR OTHER TORTIOUS ACTION, ARISING OUT OF OR IN CONNECTION WITH THE USE OR PERFORMANCE OF THIS SOFTWARE.

gjc@paradigm.com

Paradigm Associates Inc
29 Putnam Ave, Suite 6
Cambridge, MA 02138

1.3.3 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.

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 retitle 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 various 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 individually 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 “History”, 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 releasing 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.

1.4 Bibliography


[SchemePrimer] 9B8$;Bgœ(B(Dai Inukai) ae$BF~Lgœ(BScheme 1999œ$BG/or(B12œ$B7n=iHGœ(B ISBN4-87966-954-7


'scm.texi'
'scm.info'
Documentation of scm extensions (beyond Scheme standards). Documentation on the internal representation and how to extend or include scm in other programs.

'Xlibscm.texi'
'Xlibscm.info'
Documentation of the Xlib - SCM Language X Interface.
2 Installing SCM

SCM runs on a wide variety of platforms. “Distributions” is the starting point for all platforms. The process described in “GNU configure and make” will work on most Unix and GNU/Linux platforms. If it works for you, then you may skip the later sections of “Installing SCM”.

2.1 Distributions

The SCM homepage contains links to precompiled binaries and source distributions. Downloads and instructions for installing the precompiled binaries are at http://people.csail.mit.edu/jaffer/SCM#QuickStart.

If there is no precompiled binary for your platform, you may be able to build from the source distribution. The rest of these instructions deal with building and installing SCM and SLIB from sources.

Download (both SCM and SLIB of) either the last release or current development snapshot from http://people.csail.mit.edu/jaffer/SCM#BuildFromSource.

Unzip both the SCM and SLIB zips. For example, if you are working in ‘/usr/local/src/’, this will create directories ‘/usr/local/src/scm/’ and ‘/usr/local/src/slib/’.

2.2 GNU configure and make

‘scm/configure’ and ‘slib/configure’ are Shell scripts which create the files ‘scm/config.status’ and ‘slib/config.status’ on Unix and MinGW systems.

The ‘config.status’ files are used (included) by the Makefile to control where the packages will be installed by make install. With GNU shell (bash) and utilities, the following commands should build and install SCM and SLIB:

```
bash$ (cd slib; ./configure --prefix=/usr/local/)
bash$ (cd scm
  > ./configure --prefix=/usr/local/
  > make scmlit
  > sudo make all
  > sudo make install)
bash$ (cd slib; sudo make install)
```

If the install commands worked, skip to Section 2.6.3 [Testing], page 25.

If ‘configure’ doesn’t work on your system, make ‘scm/config.status’ and ‘slib/config.status’ be empty files.

For additional help on using the ‘configure’ script, run ‘./configure --help’.

‘make all’ will attempt to create a dumped executable (see Section 2.4 [Saving Executable Images], page 23), which has very small startup latency. If that fails, it will try to compile an ordinary ‘scm’ executable.

Note that the compilation output may contain error messages; be concerned only if the ‘make install’ transcripts contain errors.
‘sudo’ runs the command after it as user root. On recent GNU/Linux systems, dumping
requires that ‘make all’ be run as user root; hence the use of ‘sudo’.
‘make install’ requires root privileges if you are installing to standard Unix locations as
specified to (or defaulted by) ‘./configure’. Note that this is independent of whether you
did ‘sudo make all’ or ‘make all’.

2.2.1 Making scmlit
The SCM distribution ‘Makefile’ contains rules for making scmlit, a “bare-bones” version
of SCM sufficient for running ‘build’. ‘build’ is a Scheme program used to compile (or
create scripts to compile) full featured versions of SCM (see Section 2.3 [Building SCM],
page 14). To create scmlit, run ‘make scmlit’ in the ‘scm/’ directory.

Makefiles are not portable to the majority of platforms. If you need to compile SCM without
‘scmlit’, there are several ways to proceed:

- Use the build web page to create custom batch scripts for compiling SCM.
- Use SCM on a different platform to run ‘build’ to create a script to build SCM;
- Use another implementation of Scheme to run ‘build’ to create a script to build SCM;
- Create your own script or ‘Makefile’.

Finding SLIB
If you didn’t create scmlit using ‘make scmlit’, then you must create a file named
‘scm/require.scm’. For most installations, ‘scm/require.scm’ can just be copied from
‘scm/requires.scm’, which is part of the SCM distribution.

If, when executing ‘scmlit’ or ‘scm’, you get a message like:

ERROR: "LOAD couldn’t find file " "/usr/local/src/scm/require"

then create a file ‘require.scm’ in the SCM implementation-vicinity (this is the same
directory as where the file ‘Init5f1.scm’ is). ‘require.scm’ should have the contents:

(define (library-vicinity) "/usr/local/lib/slib/")

where the pathname string ‘/usr/local/lib/slib/’ is to be replaced by the pathname
into which you unzipped (or installed) SLIB.

Alternatively, you can set the (shell) environment variable SCHEME_LIBRARY_PATH to the
pathname of the SLIB directory (see Section 3.4 [Environment Variables], page 31). If set,
this environment variable overrides ‘scm/require.scm’.

Absolute pathnames are recommended here; if you use a relative pathname, SLIB can get
confused when the working directory is changed (see Section 5.6 [I/O-Extensions], page 73).
The way to specify a relative pathname is to append it to the implementation-vicinity, which
is absolute:

(define library-vicinity
 (let ((lv (string-append (implementation-vicinity) ".../slib/")))
 (lambda () lv)))

2.2.2 Makefile targets
Each of the following four ‘make’ targets creates an executable named ‘scm’. Each target
takes its build options from a file with an ‘.opt’ suffix. If that options file doesn’t exist,
making that target will create the file with the ‘-F’ features: cautious, bignums, arrays, inexact, engineering-notation, and dynamic-linking. Once that ‘.opt’ file exists, you can edit it to your taste and it will be preserved.

**make scm4**  Produces a R4RS executable named ‘scm’ lacking hygienic macros (but with defmacro). The build options are taken from ‘scm4.opt’. If build or the executable fails, try removing ‘dynamic-linking’ from ‘scm4.opt’.

**make scm5**  R5RS; like ‘make scm4’ but with ‘-F macro’. The build options are taken from ‘scm5.opt’. If build or the executable fails, try removing ‘dynamic-linking’ from ‘scm5.opt’.

**make dscm4**  Produces a R4RS executable named ‘udscm4’, which it starts and dumps to a low startup latency executable named ‘scm’. The build options are taken from ‘udscm4.opt’.

If the build fails, then ‘build scm4’ instead. If the dumped executable fails to run, then send me a bug report (and use ‘build scm4’ until the problem with dump is corrected).

**make dscm5**  Like ‘make dscm4’ but with ‘-F macro’. The build options are taken from ‘udscm5.opt’.

If the build fails, then ‘build scm5’ instead. If the dumped executable fails to run, then send me a bug report (and use ‘build scm5’ until the problem with dump is corrected).

If the above builds fail because of ‘-F dynamic-linking’, then (because they can’t be dynamically linked) you will likely want to add some other features to the build’s ‘.opt’ file. See the ‘-F’ build option in Section 2.3.2 [Build Options], page 17.

If dynamic-linking is working, then you will likely want to compile most of the modules as DLLs. The build options for compiling DLLs are in ‘dlls.opt’.

**make x.so**  The Xlib module; Section “SCM Language X Interface ” in Xlibscm.

**make myturtle**  Creates a DLL named ‘turtlegr.so’ which is a simple graphics API.

**make wbscm.so**  The wb module; Section “B-tree database implementation ” in wb. Compiling this requires that wb source be in a peer directory to scm.

**make dlls**  Compiles all the distributed library modules, but not ‘wbscm.so’. Many of the module compiles are recursively invoked in such a way that failure of one (which could be due to a system library not being installed) doesn’t cause the top-level ‘make dlls’ to fail. If ‘make dlls’ fails as a whole, it is time to submit a bug report (see Section 2.6.6 [Reporting Problems], page 27).

### 2.3 Building SCM

The file build loads the file build.scm, which constructs a relational database of how to compile and link SCM executables. ‘build.scm’ has information for the platforms which
2.3.1 Invoking Build

This section teaches how to use ‘build’, a Scheme program for creating compilation scripts to produce SCM executables and library modules. The options accepted by ‘build’ are documented in Section 2.3.2 [Build Options], page 17.

Use the any method if you encounter problems with the other two methods (MS-DOS, Unix).

MS-DOS  From the SCM source directory, type ‘build’ followed by up to 9 command line arguments.

Unix  From the SCM source directory, type ‘./build’ followed by command line arguments.

any  From the SCM source directory, start ‘scm’ or ‘scmlit’ and type (load "build"). Alternatively, start ‘scm’ or ‘scmlit’ with the command line argument ‘-ilbuild’. This method will also work for MS-DOS and Unix.

After loading various SLIB modules, the program will print:

```
(type (b "build <command-line>") to build
(type (b*) to enter build command loop
```

The ‘b*’ procedure enters into a build shell where you can enter commands (with or without the ‘build’). Blank lines are ignored. To create a build script with all defaults type ‘build’.

If the build-shell encounters an error, you can reenter the build-shell by typing ‘(b*)’. To exit scm type ‘(quit)’.

Here is a transcript of an interactive (b*) build-shell.

```
bash$ scmlit
SCM version 5e7, Copyright (C) 1990-2006 Free Software Foundation.
SCM comes with ABSOLUTELY NO WARRANTY; for details type ‘(terms)’.
This is free software, and you are welcome to redistribute it under certain conditions; type ‘(terms)’ for details.
> (load "build")
;loading build
; loading /home/jaffer/slib/getparam
; loading /home/jaffer/slib/coerce
...;
done loading build.scm
(type (b "build <command-line>") to build
(type (b*) to enter build command loop
;done loading build
#<unspecified>
> (b*)
;loading /home/jaffer/slib/comparse
;done loading /home/jaffer/slib/comparse.scm
```
build> -t exe
#! /bin/sh
# unix (linux) script created by SLIB/batch Wed Oct 26 17:14:23 2011
# [-p linux]
# ================ Write file with C defines
rm -f scmflags.h
echo '#define IMPLINIT "Init5e7.scm"'
>scmflags.h
echo '#define BIGNUMS'
>scmflags.h
echo '#define FLOATS'
>scmflags.h
echo '#define ARRAYS'
>scmflags.h
# ================ Compile C source files
gcc -c continue.c scm.c scmmain.c findexec.c script.c time.c repl.c scl.c eval.c sys.c
# ================ Link C object files
gcc -rdynamic -o scm continue.o scm.o scmmain.o findexec.o script.o time.o repl.o scl.o
"scm"
bash$ build> -t exe -w myscript.sh
"scm"
bash$ (quit)

No compilation was done. The ‘-t exe’ command shows the compile script. The ‘-t exe -w
myscript.sh’ line creates a file ‘myscript.sh’ containing the compile script. To actually
compile and link it, type ‘./myscript.sh’.

Invoking build without the ‘-F’ option will build or create a shell script with the arrays,
inexact, and bignums options as defaults. Invoking ‘build’ with ‘-F lit -o scmlit’ will
make a script for compiling ‘scmlit’.

bash$ ./build
-1
#! /bin/sh
# unix (linux) script created by SLIB/batch
# ================ Write file with C defines
rm -f scmflags.h
echo '#define IMPLINIT "Init5f1.scm"'
>scmflags.h
echo '#define BIGNUMS'
>scmflags.h
echo '#define FLOATS'
>scmflags.h
echo '#define ARRAYS'
>scmflags.h
# ================ Compile C source files
gcc -O2 -c continue.c scm.c scmmain.c findexec.c script.c time.c repl.c scl.c eval.c sys.c
# ================ Link C object files
gcc -rdynamic -o scm continue.o scm.o scmmain.o findexec.o script.o time.o repl.o scl.o
"scm"
bash$ ./build -o scmlit -p darwin -F lit
-1
#! /bin/sh
# unix (darwin) script created by SLIB/batch
# ================ Write file with C defines

To cross compile for another platform, invoke build with the ‘-p’ or ‘--platform’ option.
This will create a script for the platform named in the ‘-p’ or ‘--platform’ option.

bash$ ./build -o scmlit -p darwin -F lit
-1
#! /bin/sh
# unix (darwin) script created by SLIB/batch
# ================ Write file with C defines
rm -f scmflags.h
echo '#define IMPLINIT "Init5f1.scm"'>>scmflags.h
# ================ Compile C source files
cc -O3 -c continue.c scm.c scmmain.c findexec.c script.c time.c repl.c scl.c eval.c sys.c subr.c debug.c unif.c rope.c
# ================ Link C object files
mv -f scmlit scmlit~
cc -o scmlit continue.o scm.o scmmain.o findexec.o script.o time.o repl.o scl.o eval.o

2.3.2 Build Options

The options to build specify what, where, and how to build a SCM program or dynamically linked module. These options are unrelated to the SCM command line options.

```
-p platform-name
---platform=platform-name
```

specifies that the compilation should be for a computer/operating-system combination called platform-name. Note The case of platform-name is distinguished. The current platform-names are all lower-case.

The platforms defined by table platform in ‘build.scm’ are:

<table>
<thead>
<tr>
<th>name</th>
<th>processor</th>
<th>operating-system</th>
<th>compiler</th>
</tr>
</thead>
<tbody>
<tr>
<td>#f</td>
<td>processor-family</td>
<td>operating-system</td>
<td>#f</td>
</tr>
<tr>
<td>symbol</td>
<td>processor-family</td>
<td>operating-system</td>
<td>symbol</td>
</tr>
<tr>
<td>symbol</td>
<td>symbol</td>
<td>symbol</td>
<td>symbol</td>
</tr>
</tbody>
</table>

| *unknown*  | *unknown*  | unix  | cc     |
| acorn-unixlib | acorn    | *unknown* | cc     |
| aix        | powerpc   | aix   | cc     |
| alpha-elf  | alpha     | unix  | cc     |
| alpha-linux| alpha     | linux | gcc    |
| amiga-aztec| m68000    | amiga | cc     |
| amiga-dice-c| m68000  | amiga | dcc    |
| amiga-gcc  | m68000    | amiga | gcc    |
| amiga-sas  | m68000    | amiga | lc     |
| atari-st-gcc| m68000  | atari-st| gcc    |
| atari-st-turbo-c| m68000| atari-st| tcc    |
| borland-c  | i8086     | ms-dos| bcc    |
| darwin     | powerpc   | unix  | cc     |
| djgpp      | i386      | ms-dos| gcc    |
| freebsd    | *unknown* | unix  | cc     |
| gcc        | *unknown* | unix  | gcc    |
| gnu-win32  | i386      | unix  | gcc    |
| highc      | i386      | ms-dos| hc386  |
| hp-ux      | hp-risc   | hp-ux | cc     |
| irix       | mips      | irix  | gcc    |
| linux      | *unknown* | linux | gcc    |
| linux-aout | i386      | linux | gcc    |
linux-ia64 ia64 linux gcc
microsoft-c i8086 ms-dos cl
microsoft-c-nt i386 ms-dos cl
microsoft-quick-c i8086 ms-dos qcl
ms-dos i8086 ms-dos cc
netbsd *unknown* unix gcc
openbsd *unknown* unix gcc
os/2-cset i386 os/2 icc
os/2-emx i386 os/2 gcc
osf1 alpha unix cc
plan9-8 i386 plan9 8c
sunos sparc sunos cc
svr4 *unknown* unix cc
svr4-gcc-sun-ld sparc sunos gcc
turbo-c i8086 ms-dos tcc
unicos cray unicos cc
unix *unknown* unix cc
vms sparc sunos gcc
vms-gcc sparc vms gcc
watcom-9.0 i386 ms-dos wcc386p

- f pathname [Build Option]
specifies that the build options contained in pathname be spliced into the argument list at this point. The use of option files can separate functional features from platform-specific ones.

The ‘Makefile’ calls out builds with the options in ‘.opt’ files:

‘dlls.opt’
Options for Makefile targets dlls, myturtle, and x.so.

‘gdb.opt’ Options for udgdbscm and gdbscm.

‘libscm.opt’
Options for libscm.a.

‘pg.opt’ Options for pgscm, which instruments C functions.

‘udscm4.opt’
Options for targets udscm4 and dscm4 (scm).

‘udscm5.opt’
Options for targets udscm5 and dscm5 (scm).

The Makefile creates options files it depends on only if they do not already exist.

- o filename [Build Option]
---outname=filename [Build Option]
specifies that the compilation should produce an executable or object name of filename. The default is ‘scm’. Executable suffixes will be added if necessary, e.g. ‘scm’ ⇒ ‘scm.exe’.
Chapter 2: Installing SCM

- `l libname` ... [Build Option]
- `---libraries=libname` [Build Option]
  specifies that the `libname` should be linked with the executable produced. If compile flags or include directories (`-I`) are needed, they are automatically supplied for compilations. The `c` library is always included. SCM features specify any libraries they need; so you shouldn’t need this option often.

- `D definition` ... [Build Option]
- `---defines=definition` [Build Option]
  specifies that the `definition` should be made in any C source compilations. If compile flags or include directories (`-I`) are needed, they are automatically supplied for compilations. SCM features specify any flags they need; so you shouldn’t need this option often.

- `---compiler-options=flag` [Build Option]
  specifies that that `flag` will be put on compiler command-lines.

- `---linker-options=flag` [Build Option]
  specifies that that `flag` will be put on linker command-lines.

- `s pathname` [Build Option]
- `---scheme-initial=pathname` [Build Option]
  specifies that `pathname` should be the default location of the SCM initialization file ‘Init5f1.scm’. SCM tries several likely locations before resorting to `pathname` (see Section 6.3.1 [File-System Habitat], page 127). If not specified, the current directory (where build is building) is used.

- `c pathname` ... [Build Option]
- `---c-source-files=pathname` [Build Option]
  specifies that the C source files `pathname` ... are to be compiled.

- `j pathname` ... [Build Option]
- `---object-files=pathname` [Build Option]
  specifies that the object files `pathname` ... are to be linked.

- `i call` ... [Build Option]
- `---initialization=call` [Build Option]
  specifies that the C functions `call` ... are to be invoked during initialization.

- `t build-what` [Build Option]
- `---type=build-what` [Build Option]
  specifies in general terms what sort of thing to build. The choices are:
  - ‘exe’ executable program.
  - ‘lib’ library module.
  - ‘dlls’ archived dynamically linked library object files.
  - ‘dll’ dynamically linked library object file.

The default is to build an executable.
Chapter 2: Installing SCM

-h batch-syntax
--batch-dialect=batch-syntax

specifies how to build. The default is to create a batch file for the host system. The
SLIB file ‘batch.scm’ knows how to create batch files for:

- unix
- dos
- vms
- amigaos (was amigados)
- system

This option executes the compilation and linking commands through the use of
the system procedure.

- *unknown*

This option outputs Scheme code.

-w batch-filename
--script-name=batch-filename

specifies where to write the build script. The default is to display it on (current-
output-port).

-F feature ...
---features=feature

specifies to build the given features into the executable. The defined features are:

array Alias for ARRAYS

array-for-each
array-map! and array-for-each (arrays must also be featured).

arrays Use if you want arrays, uniform-arrays and uniform-vectors.

bignums Large precision integers.

byte Treating strings as byte-vectors.

byte-number Byte/number conversions

careful-interrupt-masking

Define this for extra checking of interrupt masking and some simple checks
for proper use of malloc and free. This is for debugging C code in ‘sys.c’,
‘eval.c’, ‘repl.c’ and makes the interpreter several times slower than
usual.

cautious Normally, the number of arguments arguments to interpreted closures
(from LAMBDA) are checked if the function part of a form is not a
symbol or only the first time the form is executed if the function part
is a symbol. defining ‘reckless’ disables any checking. If you want to
have SCM always check the number of arguments to interpreted closures
define feature ‘cautious’.
**cheap-continuations**
If you only need straight stack continuations, executables compile with this feature will run faster and use less storage than not having it. Machines with unusual stacks need this. Also, if you incorporate new C code into scm which uses VMS system services or library routines (which need to unwind the stack in a orderly manner) you may need to use this feature.

**compiled-closure**
Use if you want to use compiled closures.

**curses**
For the curses screen management package.

**debug**
Turns on the features ‘cautious’ and ‘careful-interrupt-masking’; uses -g flags for debugging SCM source code.

**differ**
Sequence comparison

**dont-memoize-locals**
SCM normally converts references to local variables to ILOCs, which make programs run faster. If SCM is badly broken, try using this option to disable the MEMOIZE_LOCALS feature.

**dump**
Convert a running scheme program into an executable file.

**dynamic-linking**
Be able to load compiled files while running.

**edit-line**
interface to the editline or GNU readline library.

**engineering-notation**
Use if you want floats to display in engineering notation (exponents always multiples of 3) instead of scientific notation.

**generalized-c-arguments**
make_gsubr for arbitrary (< 11) arguments to C functions.

**i/o-extensions**
Commonly available I/O extensions: exec, line I/O, file positioning, file delete and rename, and directory functions.

**inexact**
Use if you want floating point numbers.

**lit**
Lightweight – no features

**macro**
C level support for hygienic and referentially transparent macros (syntax-rules macros).

**mysql**
Client connections to the mysql databases.

**no-heap-shrink**
Use if you want segments of unused heap to not be freed up after garbage collection. This may increase time in GC for *very* large working sets.

**none**
No features

**posix**
Posix functions available on all Unix-like systems. fork and process functions, user and group IDs, file permissions, and link.
If your scheme code runs without any errors you can disable almost all error checking by compiling all files with `reckless`.

The Record package provides a facility for user to define their own record data types. See SLIB for documentation.

String regular expression matching.

These procedures were specified in the Revised-2 Report on Scheme but not in R4RS.

Use if you want to run code from:

Differences from R5RS are:
- `(eq? '() '#f)
- `(define a 25) returns the symbol a.
- `(set! a 36) returns 36.

Use if you want all inexact real numbers to be single precision. This only has an effect if SINGLES is also defined (which is the default). This does not affect complex numbers.

BSD socket interface. Socket addr functions require inexact or bignums for 32-bit precision.

Use if you want the ticks and ticks-interrupt functions.

Turtle graphics calls for both Borland-C and X11 from sjm@ee.tut.fi.

Those unix features which have not made it into the Posix specs: nice, acct, lstat, readlink, symlink, mknod and sync.

WB database with relational wrapper.

no-comment

Microsoft Windows executable.

Alias for Xlib feature.

Interface to Xlib graphics routines.

### 2.3.3 Compiling and Linking Custom Files

A correspondent asks:

How can we link in our own c files to the SCM interpreter so that we can add our own functionality? (e.g. we have a bunch of tcp functions we want access to). Would this involve changing build.scm or the Makefile or both?
(see Section 6.2.8 [Changing Scm], page 116 has instructions describing the C code format). Suppose a C file foo.c has functions you wish to add to SCM. To compile and link your file at compile time, use the ‘-c’ and ‘-i’ options to build:

```
bash$ ./build -c foo.c -i init_foo
|
#! /bin/sh
rm -f scmflags.h
echo '#define IMPLINIT "/home/jaffer/scm/Init5f1.scm"'>>scmflags.h
echo '#define COMPILED_INITS init_foo();'>>scmflags.h
echo '#define BIGNUMS'>>scmflags.h
echo '#define FLOATS'>>scmflags.h
echo '#define ARRAYS'>>scmflags.h
gcc -O2 -c continue.c scm.c findexec.c script.c time.c repl.c scl.c \
   eval.c sys.c subr.c unif.c rope.c foo.c
gcc -rdynamic -o scm continue.o scm.o findexec.o script.o time.o \
   repl.o scl.o eval.o sys.o subr.o unif.o rope.o foo.o -lm -lc
```

To make a dynamically loadable object file use the ‘-t dll’ option:

```
bash$ ./build -t dll -c foo.c
|
#! /bin/sh
rm -f scmflags.h
echo '#define IMPLINIT "/home/jaffer/scm/Init5f1.scm"'>>scmflags.h
echo '#define BIGNUMS'>>scmflags.h
echo '#define FLOATS'>>scmflags.h
echo '#define ARRAYS'>>scmflags.h
echo '#define DLL'>>scmflags.h
gcc -O2 -fpic -c foo.c
gcc -shared -o foo.so foo.o -lm -lc
```

Once ‘foo.c’ compiles correctly (and your SCM build supports dynamic-loading), you can load the compiled file with the Scheme command (load "./foo.so”). See Section 6.2.4 [Configure Module Catalog], page 112 for how to add a compiled dll file to SLIB’s catalog.

### 2.4 Saving Executable Images

In SCM, the ability to save running program images is called dump (see Section 5.2 [Dump], page 65). In order to make dump available to SCM, build with feature ‘dump’. dumped executables are compatible with dynamic linking.

Most of the code for dump is taken from ‘emacs-19.34/src/unex*.c’. No modifications to the emacs source code were required to use ‘unexelf.c’. Dump has not been ported to all platforms. If ‘unexec.c’ or ‘unexelf.c’ don’t work for you, try using the appropriate ‘unex*.c’ file from emacs.

The ‘dscm4’ and ‘dscm5’ targets in the SCM ‘Makefile’ save images from ‘udscm4’ and ‘udscm5’ executables respectively.

Address space layout randomization interferes with dump. Here are the fixes for various operating-systems:
Fedora-Core-1
Remove the ‘#’ from the line ‘#SETARCH = setarch i386’ in the ‘Makefile’.

Fedora-Core-3
http://jamesthornton.com/writing/emacs-compile.html [For FC3] combr reloc has become the default for recent GNU ld, which breaks the unexec/undump on all versions of both Emacs and XEmacs...
Override by adding the following to ‘udscm5.opt’: ‘--linker-options="-z nocombreloc"’

Linux Kernels later than 2.6.11
http://www.opensubscriber.com/message/emacs-devel@gnu.org/1007118.html mentions the exec-shield feature. Kernels later than 2.6.11 must do (as root):

```
    echo 0 > /proc/sys/kernel/randomize_va_space
```
before dumping. ‘Makefile’ has this ‘randomize_va_space’ stuffing scripted for targets ‘dscm4’ and ‘dscm5’. You must either set ‘randomize_va_space’ to 0 or run as root to dump.

OS-X 10.6
The dynamic linker uses the following environment variables. They affect any program that uses the dynamic linker.

DYLD_NO_PIE
Causes dyld to not randomize the load addresses of images in a process where the main executable was built position independent. This can be helpful when trying to reproduce and debug a problem in a PIE.

2.5 Installation
Once scmlit, scm, and dlls have been built, these commands will install them to the locations specified when you ran ‘./configure’:

```
bash$ (cd scm; make install)
bash$ (cd slib; make install)
```
Note that installation to system directories (like ‘/usr/bin/’) will require that those commands be run as root:

```
bash$ (cd scm; sudo make install)
bash$ (cd slib; sudo make install)
```

2.6 Troubleshooting and Testing

2.6.1 Problems Compiling

<table>
<thead>
<tr>
<th>FILE</th>
<th>PROBLEM / MESSAGE</th>
<th>HOW TO FIX</th>
</tr>
</thead>
<tbody>
<tr>
<td>*.c</td>
<td>include file not found.</td>
<td>Correct the status of STDC_HEADERS in scmfig.h.</td>
</tr>
</tbody>
</table>
Chapter 2: Installing SCM

2.6.2 Problems Linking

PROBLEM HOW TO FIX
_sin etc. missing. Uncomment LIBS in makefile.

2.6.3 Testing

Loading ‘r4rstest.scm’ in the distribution will run an [R4RS] conformance test on scm.

> (load "r4rstest.scm")
.loading r4rstest.scm
SECTION(2 1)
SECTION(3 4)
#<primitive-procedure boolean?>
#<primitive-procedure char?>
#<primitive-procedure null?>
#<primitive-procedure number?>
...

Loading ‘pi.scm’ in the distribution will enable you to compute digits of pi.

> (load "pi.scm")
;loading pi.scm
#<unspecified>
> (pi 100 5)
00003 14159 26535 89793 23846 26433 83279 50288 41971 69399
37510 58209 74944 59230 78164 06286 20899 86280 34825 34211
Performance

Loading ‘bench.scm’ will compute and display performance statistics of SCM running ‘pi.scm’. ‘make bench’ or ‘make benchlit’ appends the performance report to the file ‘BenchLog’, facilitating tracking effects of changes to SCM on performance.

2.6.4 Problems Starting

<table>
<thead>
<tr>
<th>PROBLEM</th>
<th>HOW TO FIX</th>
</tr>
</thead>
<tbody>
<tr>
<td>/bin/bash: scm: program not found</td>
<td>Is ‘scm’ in a ‘$PATH’ directory?</td>
</tr>
<tr>
<td>/bin/bash: /usr/local/bin/scm: Permission denied</td>
<td>chmod +x /usr/local/bin/scm</td>
</tr>
<tr>
<td>Opening message and then machine crashes.</td>
<td>Change memory model option to C compiler (or makefile).</td>
</tr>
<tr>
<td></td>
<td>Make sure sizet definition is correct in scmfig.h.</td>
</tr>
<tr>
<td></td>
<td>Reduce the size of HEAP_SEG_SIZE in setjump.h.</td>
</tr>
<tr>
<td></td>
<td>#define NOSETBUF</td>
</tr>
<tr>
<td>Input hangs.</td>
<td>Increase initial heap allocation using -a&lt;kb&gt; or INIT_HEAP_SIZE.</td>
</tr>
<tr>
<td>ERROR: heap: need larger initial.</td>
<td>Check sizet definition.</td>
</tr>
<tr>
<td>ERROR: Could not allocate.</td>
<td>Use 32 bit compiler mode.</td>
</tr>
<tr>
<td></td>
<td>Don’t try to run as subproccess.</td>
</tr>
<tr>
<td></td>
<td>Do so and recompile files.</td>
</tr>
<tr>
<td></td>
<td>Assign correct IMPLINIT in makefile or scmfig.h.</td>
</tr>
<tr>
<td></td>
<td>Define environment variable SCM_INIT_PATH to be the full pathname of Init5f1.scm.</td>
</tr>
<tr>
<td></td>
<td>Define environment variable SCHEME_LIBRARY_PATH to be the full pathname of the scheme library [SLIB].</td>
</tr>
<tr>
<td></td>
<td>Change library-vicinity in Init5f1.scm to point to library or remove.</td>
</tr>
<tr>
<td></td>
<td>Make sure the value of (library-vicinity) has a trailing file separator (like / or \.</td>
</tr>
</tbody>
</table>

2.6.5 Problems Running

<table>
<thead>
<tr>
<th>PROBLEM</th>
<th>HOW TO FIX</th>
</tr>
</thead>
<tbody>
<tr>
<td>WARNING: require.scm not found.</td>
<td>Assign correct IMPLINIT in makefile or scmfig.h.</td>
</tr>
<tr>
<td></td>
<td>Define environment variable SCM_INIT_PATH to be the full pathname of Init5f1.scm.</td>
</tr>
<tr>
<td></td>
<td>Define environment variable SCHEME_LIBRARY_PATH to be the full pathname of the scheme library [SLIB].</td>
</tr>
<tr>
<td></td>
<td>Change library-vicinity in Init5f1.scm to point to library or remove.</td>
</tr>
<tr>
<td></td>
<td>Make sure the value of (library-vicinity) has a trailing file separator (like / or \.</td>
</tr>
</tbody>
</table>
Chapter 2: Installing SCM

2.6.6 Reporting Problems

Reported problems and solutions are grouped under Compiling, Linking, Running, and Testing. If you don’t find your problem listed there, you can send a bug report to agj@alum.mit.edu or scm-discuss@gnu.org. The bug report should include:

1. The version of SCM (printed when SCM is invoked with no arguments).
2. The type of computer you are using.
3. The name and version of your computer’s operating system.
4. The values of the environment variables SCM_INIT_PATH and SCHEME_LIBRARY_PATH.
5. The name and version of your C compiler.
6. If you are using an executable from a distribution, the name, vendor, and date of that distribution. In this case, corresponding with the vendor is recommended.
Chapter 3: Operational Features

3 Operational Features

3.1 Invoking SCM

```
scm [-a kbytes] [-muvbiq] [-version] [-help]
    [[-]no-init-file] [--no-symbol-case-fold]
    [-p int] [-r feature] [-h feature]
    [-d filename] [-f filename] [-l filename]
    [-c expression] [-e expression] [-o dumpname]
    [-- | - | -s] [filename] [arguments ...]
```

Upon startup `scm` loads the file specified by the environment variable `SCM_INIT_PATH`. If `SCM_INIT_PATH` is not defined or if the file it names is not present, `scm` tries to find the directory containing the executable file. If it is able to locate the executable, `scm` looks for the initialization file (usually ‘Init5f1.scm’) in platform-dependent directories relative to this directory. See Section 6.3.1 [File-System Habitat], page 127 for a blow-by-blow description.

As a last resort (if initialization file cannot be located), the C compile parameter `IMPLINIT` (defined in the makefile or ‘scmfig.h’) is tried.

Unless the option `-no-init-file` or `--no-init-file` occurs in the command line, or if `scm` is being invoked as a script, ‘Init5f1.scm’ checks to see if there is file ‘ScmInit.scm’ in the path specified by the environment variable `HOME` (or in the current directory if `HOME` is undefined). If it finds such a file, then it is loaded.

‘Init5f1.scm’ then looks for command input from one of three sources: From an option on the command line, from a file named on the command line, or from standard input.

This explanation applies to SCMLIT or other builds of SCM.

Scheme-code files can also invoke SCM and its variants. See Section 4.8 [Lexical Conventions], page 54.

3.2 Options

The options are processed in the order specified on the command line.

```
-a k  [Command Option]
    specifies that `scm` should allocate an initial heapsize of `k` kilobytes. This option, if present, must be the first on the command line. If not specified, the default is `INIT_HEAP_SIZE` in source file ‘setjump.h’ which the distribution sets at 25000*sizeof(cell).

-no-init-file  [Command Option]
    --no-init-file  [Command Option]
    Inhibits the loading of ‘ScmInit.scm’ as described above.

--no-symbol-case-fold  [Command Option]
    Symbol (and identifier) names will be case sensitive.
```
---**help**  
prints usage information and URI; then exit.

---**version**  
prints version information and exit.

**-r feature**  
requires *feature*. This will load a file from [SLIB] if that *feature* is not already provided. If *feature* is 2, 2rs, or r2rs; 3, 3rs, or r3rs; 4, 4rs, or r4rs; 5, 5rs, or r5rs; *scm* will require the features necessary to support [R2RS]; [R3RS]; [R4RS]; or [R5RS], respectively.

**-h feature**  
provides *feature*.

**-l filename**  
loads *filename*. *scm* will load the first (unoptioned) file named on the command line if no -c, -e, -f, -l, or -s option precedes it.

**-d filename**  
Loads SLIB databases *feature* and opens *filename* as a database.

**-e expression**  
**-c expression**  
specifies that the scheme expression *expression* is to be evaluated. These options are inspired by *perl* and *sh* respectively. On Amiga systems the entire option and argument need to be enclosed in quotes. For instance ‘"-e(newline)"’. 

**-o dumpname**  
saves the current SCM session as the executable program ‘*dumpname*’. This option works only in SCM builds supporting dump (see Section 5.2 [Dump], page 65).

If options appear on the command line after ‘-o dumpname’, then the saved session will continue with processing those options when it is invoked. Otherwise the (new) command line is processed as usual when the saved image is invoked.

**-p level**  
sets the prolixity (verboseness) to *level*. This is the same as the *scm* command (verobse *level*).

**-v**  
(verbos mode) specifies that *scm* will print prompts, evaluation times, notice of loading files, and garbage collection statistics. This is the same as -p3.

**-q**  
(quiet mode) specifies that *scm* will print no extra information. This is the same as -p0.
Chapter 3: Operational Features  

- **m** [Command Option] specifies that subsequent loads, evaluations, and user interactions will be with syntax-rules macro capability. To use a specific syntax-rules macro implementation from [SLIB] (instead of [SLIB]'s default) put `-r macropackage` before `-m` on the command line.

- **u** [Command Option] specifies that subsequent loads, evaluations, and user interactions will be without syntax-rules macro capability. Syntax-rules macro capability can be restored by a subsequent `-m` on the command line or from Scheme code.

- **i** [Command Option] specifies that `scm` should run interactively. That means that `scm` will not terminate until the `(quit)` or `(exit)` command is given, even if there are errors. It also sets the prolixity level to 2 if it is less than 2. This will print prompts, evaluation times, and notice of loading files. The prolixity level can be set by subsequent options. If `scm` is started from a tty, it will assume that it should be interactive unless given a subsequent `-b` option.

- **b** [Command Option] specifies that `scm` should run non-interactively. That means that `scm` will terminate after processing the command line or if there are errors.

- **s** [Command Option] specifies, by analogy with `sh`, that `scm` should run interactively and that further options are to be treated as program arguments.

- `-` [Command Option]  

- `---` [Command Option] specifies that further options are to be treated as program arguments.

### 3.3 Invocation Examples

```
% scm foo.scm
Loads and executes the contents of `foo.scm' and then enters interactive session.

% scm -f foo.scm arg1 arg2 arg3
Parameters arg1, arg2, and arg3 are stored in the global list *argv*; Loads and executes the contents of `foo.scm' and exits.

% scm -s foo.scm arg1 arg2
Sets *argv* to ("foo.scm" "arg1" "arg2") and enters interactive session.

% scm -e '(write (list-ref *argv* *optind*))' bar
Prints "bar".

% scm -r pretty-print -r format -i
Loads pretty-print and format and enters pretty-print and formats and enters pretty-print and formats session.
```

% scm -r5
Loads dynamic-wind, values, and syntax-rules macros and enters interactive (with macros) session.
% scm -r5 -r4
   Like above but rev4-optional-procedures are also loaded.

3.4 Environment Variables

SCM_INIT_PATH [Environment Variable]
   is the pathname where scm will look for its initialization code. The default is the file ‘Init5f1.scm’ in the source directory.

SCHEME_LIBRARY_PATH [Environment Variable]
   is the [SLIB] Scheme library directory.

HOME [Environment Variable]
   is the directory where ‘Init5f1.scm’ will look for the user initialization file ‘ScmInit.scm’.

EDITOR [Environment Variable]
   is the name of the program which ed will call. If EDITOR is not defined, the default is ‘ed’.

3.5 Scheme Variables

*argv* [Variable]
   contains the list of arguments to the program. *argv* can change during argument processing. This list is suitable for use as an argument to [SLIB] getopt.

*syntax-rules* [Variable]
   controls whether loading and interaction support syntax-rules macros. Define this in ‘ScmInit.scm’ or files specified on the command line. This can be overridden by subsequent -m and -u options.

*interactive* [Variable]
   controls interactivity as explained for the -i and -b options. Define this in ‘ScmInit.scm’ or files specified on the command line. This can be overridden by subsequent -i and -b options.

3.6 SCM Session

• Options, file loading and features can be specified from the command line. See Section “System interface” in SCM. See Section “Require” in SLIB.
• Typing the end-of-file character at the top level session (while SCM is not waiting for parenthesis closure) causes SCM to exit.
• Typing the interrupt character aborts evaluation of the current form and resumes the top level read-eval-print loop.

quit [Function]
quit n [Function]
exit [Function]
exit \( n \)  
[Function]
Aliases for exit (see Section “System” in SLIB). On many systems, SCM can also tail-call another program. See Section 5.6 [I/O-Extensions], page 73.

boot-tail dumped?  
[Callback procedure]
boot-tail is called by scm_top_level just before entering interactive top-level. If boot-tail calls quit, then interactive top-level is not entered.

program-arguments  
[Function]
Returns a list of strings of the arguments scm was called with.

getlogin  
[Function]
Returns the (login) name of the user logged in on the controlling terminal of the process, or #f if this information cannot be determined.

For documentation of the procedures getenv and system See Section “System Interface” in SLIB.

SCM extends getenv as suggested by draft SRFI-98:

getenv name  
[Function]
Looks up name, a string, in the program environment. If name is found a string of its value is returned. Otherwise, #f is returned.

getenv  
[Function]
Returns names and values of all the environment variables as an association-list.

\[
\begin{align*}
\text{getenv} & \Rightarrow \\
& (\text{"PATH"} . 
/\text{usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin"}) \\
& (\text{"USERNAME"} . 
\text{"taro"})
\end{align*}
\]

vms-debug  
[Function]
If SCM is compiled under VMS this vms-debug will invoke the VMS debugger.

3.7 Editing Scheme Code

ed arg1 . . .  
[Function]
The value of the environment variable EDITOR (or just ed if it isn’t defined) is invoked as a command with arguments arg1 . . . .

ed filename  
[Function]
If SCM is compiled under VMS ed will invoke the editor with a single the single argument filename.

Gnu Emacs:

Editing of Scheme code is supported by emacs. Buffers holding files ending in .scm are automatically put into scheme-mode.

If your Emacs can run a process in a buffer you can use the Emacs command ‘\texttt{M-x run-scheme}’ with SCM. Otherwise, use the emacs command ‘\texttt{M-x suspend-emacs}’; or see “other systems” below.
Epsilon (MS-DOS):

There is lisp (and scheme) mode available by use of the package ‘LISP.E’. It offers several different indentation formats. With this package, buffers holding files ending in ‘.L’, ‘.LSP’, ‘.S’, and ‘.SCM’ (my modification) are automatically put into lisp-mode.

It is possible to run a process in a buffer under Epsilon. With Epsilon 5.0 the command line options ‘-e512 -m0’ are necessary to manage RAM properly. It has been reported that when compiling SCM with Turbo C, you need to ‘#define NOSETBUF’ for proper operation in a process buffer with Epsilon 5.0.

One can also call out to an editor from SCM if RAM is at a premium; See “under other systems” below.

other systems:

Define the environment variable ‘EDITOR’ to be the name of the editing program you use. The SCM procedure (ed arg1 ...) will invoke your editor and return to SCM when you exit the editor. The following definition is convenient:

(define (e) (ed "work.scm") (load "work.scm"))

Typing ‘(e)’ will invoke the editor with the file of interest. After editing, the modified file will be loaded.

3.8 Debugging Scheme Code

The cautious option of build (see Section 2.3.2 [Build Options], page 17) supports debugging in Scheme.

CAUTIOUS

If SCM is built with the ‘CAUTIOUS’ flag, then when an error occurs, a stack trace of certain pending calls are printed as part of the default error response. A (memoized) expression and newline are printed for each partially evaluated combination whose procedure is not builtin. See Section 3.11 [Memoized Expressions], page 38 for how to read memoized expressions.

Also as the result of the ‘CAUTIOUS’ flag, both error and user-interrupt (invoked by C-C) to print stack traces and conclude by calling breakpoint (see Section “Breakpoints” in SLIB) instead of aborting to top level. Under either condition, program execution can be resumed by (continue).

In this configuration one can interrupt a running Scheme program with C-C, inspect or modify top-level values, trace or untrace procedures, and continue execution with (continue).

If verbose (see Section 3.12 [Internal State], page 39) is called with an argument greater than 2, then the interpreter will check stack size periodically. If the size of stack in use exceeds the C #define STACK_LIMIT (default is HEAP_SEG_SIZE), SCM generates a ‘stack’ segment violation.

There are several SLIB macros which so useful that SCM automatically loads the appropriate module from SLIB if they are invoked.
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.

untrace proc1 . . .  
[Macro]  
Turns tracing off for its arguments.  untrace
With no arguments, untraces all currently traced identifiers and returns a list of these  
formerly traced identifiers.

The routines I use most frequently for debugging are:

print arg1 . . .  
[Function]  
Print writes all its arguments, separated by spaces. Print outputs a newline at the  
end and returns the value of the last argument.

One can just insert `(print '<label>' and ')' around an expression in order to see  
its values as a program operates.

pprint arg1 . . .  
[Function]  
Pprint pretty-prints (see Section “Pretty-Print” in SLIB) all its arguments, separated  
by newlines. Pprint returns the value of the last argument.

One can just insert `(pprint '<label>' and ')’ around an expression in order to see  
its values as a program operates. Note pretty-print does not format procedures.

When typing at top level, pprint is not a good way to see nested structure because it will  
return the last object pretty-printed, which could be large. pp is a better choice.

pp arg1 . . .  
[Procedure]  
Pprint pretty-prints (see Section “Pretty-Print” in SLIB) all its arguments, separated  
by newlines. pp returns #<unspecified>.

print-args name  
[Syntax]  
print-args
Write name if supplied; then writes the names and values of the closest lexical  
bindings enclosing the call to Print-args.

(define (foo a b) (print-args foo) (+ a b))
(foo 3 6)
⇒ In foo: a = 3; b = 6;
⇒ 9

Sometimes more elaborate measures are needed to print values in a useful manner. When  
the values to be printed may have very large (or infinite) external representations, Section  
“Quick Print” in SLIB, can be used.

When trace is not sufficient to find program flow problems, SLIB-PSD, the Portable Scheme  
Debugger offers source code debugging from GNU Emacs. PSD runs slowly, so start by  
instrumenting only a few functions at a time.

http://groups.csail.mit.edu/mac/ftpdir/scm/slib-psd1-3.tar.gz
3.9 Debugging Continuations

These functions are defined in `debug.c`, all operate on captured continuations:

- **frame-trace** cont n
  - [Procedure]
  - Prints information about the code being executed and the environment scopes active for continuation frame n of continuation CONT. A "continuation frame" is an entry in the environment stack; a new frame is pushed when the environment is replaced or extended in a non-tail call context. Frame 0 is the top of the stack.

- **frame->environment** cont n
  - [Procedure]
  - Prints the environment for continuation frame n of continuation cont. This contains just the names, not the values, of the environment.

- **scope-trace** env
  - [Procedure]
  - will print information about active lexical scopes for environment env.

- **frame-eval** cont n expr
  - [Procedure]
  - Evaluates expr in the environment defined by continuation frame n of continuation CONT and returns the result. Values in the environment may be returned or SET!.

Section 3.10 [Errors], page 36 also now accepts an optional continuation argument. stack-trace differs from frame-trace in that it truncates long output using safeports and prints code from all available frames.

```scheme
(define k #f)
(define (foo x y)
  (set! k (call-with-current-continuation identity))
  #f)
(let ((a 3) (b 4))
  (foo a b)
  #f)
(stack-trace k)
⊣

(frame-trace k 0)
⊣
(call-with-current-continuation #identity)

(frame-trace k 1)
⊣
```

ftp.maths.tcd.ie:pub/bosullvn/jacal/slib-psd1-3.tar.gz
ftp.cs.indiana.edu:/pub/scheme-repository/utl/slib-psd1-3.tar.gz
3.10 Errors

A computer-language implementation designer faces choices of how reflexive to make the implementation in handling exceptions and errors; that is, how much of the error and exception routines should be written in the language itself. The design of a portable implementation is further constrained by the need to have (almost) all errors print meaningful messages, even when the implementation itself is not functioning correctly. Therefore, SCM implements much of its error response code in C.

The following common error and conditions are handled by C code. Those with callback names after them can also be handled by Scheme code (see Section 4.4 [Interrupts], page 47). If the callback identifier is not defined at top level, the default error handler (C code) is invoked. There are many other error messages which are not treated specially.

ARGn Wrong type in argument
ARG1 Wrong type in argument 1
ARG2 Wrong type in argument 2
ARG3 Wrong type in argument 3
ARG4 Wrong type in argument 4
ARG5 Wrong type in argument 5
WNA Wrong number of args

OVFLOW numerical overflow

OUTOF RANGE Argument out of range

NALLOC (out-of-storage)

THRASH GC is (thrashing)

EXIT (end-of-program)

HUP SIGNAL (hang-up)

INT SIGNAL (user-interrupt)

FPE SIGNAL (arithmetic-error)

BUS SIGNAL bus error

SEGV SIGNAL segment violation

ALRM SIGNAL (alarm-interrupt)

VTALRM SIGNAL (virtual-alarm-interrupt)

PROF SIGNAL (profile-alarm-interrupt)

errobj [Variable]

When SCM encounters a non-fatal error, it aborts evaluation of the current form, prints a message explaining the error, and resumes the top level read-eval-print loop. The value of errobj is the offending object if appropriate. The builtin procedure error does not set errobj.

errno and perror report ANSI C errors encountered during a call to a system or library function.

erro n [Function]

With no argument returns the current value of the system variable errno. When given an argument, errno sets the system variable errno to n and returns the previous value of errno. (errno 0) will clear outstanding errors. This is recommended after try-load returns #f since this occurs when the file could not be opened.
perror string

Prints on standard error output the argument string, a colon, followed by a space, the error message corresponding to the current value of errno and a newline. The value returned is unspecified.

warn and error provide a uniform way for Scheme code to signal warnings and errors.

warn arg1 arg2 arg3 . . .

Alias for Section “System” in SLIB. Outputs an error message containing the arguments. warn is defined in ‘Init5f1.scm’.

error arg1 arg2 arg3 . . .

Alias for Section “System” in SLIB. Outputs an error message containing the arguments, aborts evaluation of the current form and resumes the top level read-eval-print loop. Error is defined in ‘Init5f1.scm’.

If SCM is built with the ‘CAUTIOUS’ flag, then when an error occurs, a stack trace of certain pending calls are printed as part of the default error response. A (memoized) expression and newline are printed for each partially evaluated combination whose procedure is not builtin. See Section 3.11 [Memoized Expressions], page 38 for how to read memoized expressions.

Also as the result of the ‘CAUTIOUS’ flag, both error and user-interrupt (invoked by C-C) are defined to print stack traces and conclude by calling breakpoint (see Section “Breakpoints” in SLIB). This allows the user to interract with SCM as with Lisp systems.

stack-trace

Prints information describing the stack of partially evaluated expressions. stack-trace returns #t if any lines were printed and #f otherwise. See ‘Init5f1.scm’ for an example of the use of stack-trace.

3.11 Memoized Expressions

SCM memoizes the address of each occurrence of an identifier’s value when first encountering it in a source expression. Subsequent executions of that memoized expression is faster because the memoized reference encodes where in the top-level or local environment its value is.

When procedures are displayed, the memoized locations appear in a format different from references which have not yet been executed. I find this a convenient aid to locating bugs and untested expressions.

- The names of memoized lexically bound identifiers are replaced with #@$<m>-$<n>, where $<m>$ is the number of binding contours back and $<n>$ is the index of the value in that binding countour.
- The names of identifiers which are not lexiallly bound but defined at top-level have #@ prepended.

For instance, open-input-file is defined as follows in ‘Init5f1.scm’:

```
(define (open-input-file str)
  (or (open-file str open_read)
      (and (procedure? could-not-open) (could-not-open) #f))
```
If `open-input-file` has not yet been used, the displayed procedure is similar to the original definition (lines wrapped for readability):

```
open-input-file ⇒
#<CLOSURE (str) (or (open-file str open_read)
  (and (procedure? could-not-open) (could-not-open) #f)
  (error "OPEN-INPUT-FILE couldn’t open file " str))>
```

If we open a file using `open-input-file`, the sections of code used become memoized:

```
(open-input-file "r4rstest.scm") ⇒ #<input-port 3>
open-input-file ⇒
#<CLOSURE (str) (#@or (#@open-file #@0+0 #@open_read)
  (@#and (#@procedure? #@could-not-open) (could-not-open) #f)
  (@#error "OPEN-INPUT-FILE couldn’t open file " #@0+0))>
```

If we cause `open-input-file` to execute other sections of code, they too become memoized:

```
(open-input-file "foo.scm") ⇒
ERROR: No such file or directory
ERROR: OPEN-INPUT-FILE couldn’t open file "foo.scm"
```

```
open-input-file ⇒
#<CLOSURE (str) (#@or (#@open-file #@0+0 #@open_read)
  (#@and (#@procedure? #@could-not-open) (could-not-open) #f)
  (#@error "OPEN-INPUT-FILE couldn’t open file " #@0+0))>
```

### 3.12 Internal State

**`*interactive*`**  

The variable `*interactive*` determines whether the SCM session is interactive, or should quit after the command line is processed. `*interactive*` is controlled directly by the command-line options `-b`, `-i`, and `-s` (see Section 3.1 [Invoking SCM], page 28). If none of these options are specified, the rules to determine interactivity are more complicated; see `Init5f1.scm` for details.

**`abort`**  
Resumes the top level Read-Eval-Print loop.

**`restart`**  
Restarts the SCM program with the same arguments as it was originally invoked. All `-l` loaded files are loaded again; If those files have changed, those changes will be reflected in the new session.

*Note* When running a saved executable (see Section 5.2 [Dump], page 65), `restart` is redefined to be `exec-self`.

**`exec-self`**  
Exits and immediately re-invokes the same executable with the same arguments. If the executable file has been changed or replaced since the beginning of the current
session, the new executable will be invoked. This differentiates exec-self from restart.

**verbose n**

Controls how much monitoring information is printed. If n is:

- 0: no prompt or information is printed.
- >= 1: a prompt is printed.
- >= 2: messages bracketing file loading are printed.
- >= 3: the CPU time is printed after each top level form evaluated; notifications of heap growth printed; the interpreter checks stack depth periodically.
- >= 4: a garbage collection summary is printed after each top level form evaluated;
- >= 5: a message for each GC (see Section 6.2.1 [Garbage Collection], page 109) is printed; warnings issued for top-level symbols redefined.

**gc**

Scans all of SCM objects and reclaims for further use those that are no longer accessible.

**room**

Prints out statistics about SCM’s current use of storage. (room #t) also gives the hexadecimal heap segment and stack bounds.

**scm-version**

Contains the version string (e.g. ‘5f1’) of SCM.

### 3.12.1 Executable path

In order to dump a saved executable or to dynamically-link using DLD, SCM must know where its executable file is. Sometimes SCM (see Section 6.3.2 [Executable Pathname], page 128) guesses incorrectly the location of the currently running executable. In that case, the correct path can be set by calling execpath with the pathname.

**execpath**

Returns the path (string) which SCM uses to find the executable file whose invocation the currently running session is, or #f if the path is not set.

**execpath #f**

**execpath newpath**

Sets the path to #f or newpath, respectively. The old path is returned.

For other configuration constants and procedures See Section “Configuration” in SLIB.
3.13 Scripting

3.13.1 Unix Scheme Scripts

In reading this section, keep in mind that the first line of a script file has (different) meanings to SCM and the operating system (execve).

```plaintext
#! interpreter \ . . .
```

On unix systems, a Shell-Script is a file (with execute permissions) whose first two characters are ‘#!’. The interpreter argument must be the pathname of the program to process the rest of the file. The directories named by environment variable PATH are not searched to find interpreter.

When executing a shell-script, the operating system invokes interpreter with a single argument encapsulating the rest of the first line’s contents (if not just whitespace), the pathname of the Scheme Script file, and then any arguments which the shell-script was invoked with.

Put one space character between ‘#!’ and the first character of interpreter (‘/’). The interpreter name is followed by ‘\’; SCM substitutes the second line of file for ‘\’ (and the rest of the line), then appends any arguments given on the command line invoking this Scheme-Script.

When SCM executes the script, the Scheme variable *script* will be set to the script pathname. The last argument before ‘!#’ on the second line should be ‘-’; SCM will load the script file, preserve the unprocessed arguments, and set *argv* to a list of the script pathname and the unprocessed arguments.

Note that the interpreter, not the operating system, provides the ‘\’ substitution; this will only take place if interpreter is a SCM or SCSH interpreter.

```plaintext
#! ignored !#
```

When the first two characters of the file being loaded are #! and a ‘\’ is present before a newline in the file, all characters up to ‘!#’ will be ignored by SCM read.

This combination of interpretations allows SCM source files to be used as POSIX shell-scripts if the first line is:

```plaintext
#! /usr/local/bin/scm \ 
```

The following Scheme-Script prints factorial of its argument:

```plaintext
#! /usr/local/bin/scm \ %0 %*
- !#
```

```plaintext
(define (fact.script args)
 (cond ((and (= 1 (length args))
             (string->number (car args)))
          => (lambda (n) (print (fact n)) #t))
     (else (fact.usage)))))

(define (fact.usage)
 (print *argv*)
)```
Usage: fact N
Returns the factorial of N.
```
(c

Usage: fact N
Returns the factorial of N.
```

3.13.2 MS-DOS Compatible Scripts

It turns out that we can create scheme-scripts which run both under unix and MS-DOS. To implement this, I have written the MS-DOS programs: #!.bat and !!.exe, which are available from: http://groups.csail.mit.edu/mac/ftpdir/scm/sharpbang.zip

With these two programs installed in a PATH directory, we have the following syntax for <program>.BAT files.

```bash
#!/<interpreter> %0 %*
```

The first two characters of the Scheme-Script are ‘#!’. The interpreter can be either a unix style program path (using ‘/’ between filename components) or a DOS program name or path. The rest of the first line of the Scheme-Script should be literally ‘\ %0 %*’, as shown.

If interpreter has ‘/’ in it, interpreter is converted to a DOS style filename (‘/’ ⇒ ‘\’).

In looking for an executable named interpreter, #! first checks this (converted) filename; if interpreter doesn’t exist, it then tries to find a program named like the string starting after the last ‘\’ (or ‘/’) in interpreter. When searching for executables, #! tries all directories named by environment variable PATH.

Once the interpreter executable path is found, arguments are processed in the manner of scheme-shell, with all the text after the ‘\’ taken as part of the meta-argument. More precisely, #! calls interpreter with any options on the second line of the Scheme-Script up to ‘!#’, the name of the Scheme-Script file, and then any of at most 8 arguments given on the command line invoking this Scheme-Script.

The previous example Scheme-Script works in both MS-DOS and unix systems.
3.13.3 Unix Shell Scripts

Scheme-scripts suffer from two drawbacks:

- Some Unixes limit the length of the ‘#!’ interpreter line to the size of an object file header, which can be as small as 32 bytes.
- A full, explicit pathname must be specified, perhaps requiring more than 32 bytes and making scripts vulnerable to breakage when programs are moved.

The following approach solves these problems at the expense of slower startup. Make ‘#! /bin/sh’ the first line and prepend every subsequent line to be executed by the shell with ‘;’. The last line to be executed by the shell should contain an exec command; exec tail-calls its argument.

/bin/sh is thus invoked with the name of the script file, which it executes as a *sh script. Usually the second line starts ‘;exec scm -f$0’, which executes scm, which in turn loads the script file. When SCM loads the script file, it ignores the first and second lines, and evaluates the rest of the file as Scheme source code.

The second line of the script file does not have the length restriction mentioned above. Also, /bin/sh searches the directories listed in the ‘PATH’ environment variable for ‘scm’, eliminating the need to use absolute locations in order to invoke a program.

The following example additionally sets *script* to the script argument, making it compatible with the scheme code of the previous example.

```sh
#!/bin/sh
:;exec scm -e"(set! *script* "$0")" -l$0 "$@
```

```
(define (fact.script args)
  (cond ((and (= 1 (length args))
              (string->number (car args)))
         => (lambda (n) (print (fact n)) #t))
       (else (fact.usage))))

(define (fact.usage)
  (print *argv*)
  (display "\nUsage: fact N\n" Returns the factorial of N."
  (current-error-port))
#f)

(define (fact n) (if (< n 2) 1 (* n (fact (+ -1 n)))))

(if *script* (exit (fact.script (list-tail *argv* *optind*)))))
./fact 6
⇒ 720
```
4 The Language

4.1 Standards Compliance

Scm conforms to the IEEE Standard 1178-1990. IEEE Standard for the Scheme Programming Language. (see Section 1.4 [Bibliography], page 10), and Revised(5) Report on the Algorithmic Language Scheme. All the required features of these specifications are supported. Many of the optional features are supported as well.

Optionals of [R5RS] Supported by SCM

- and / of more than 2 arguments
exp
log
sin
cos
tan
asin
acos.atan
sqrt
expt
make-rectangular
make-polar
real-part
imag-part
magnitude
angle
exact->inexact
inexact->exact

See Section “Numerical operations” in Revised(5) Scheme.

with-input-from-file
with-output-to-file

See Section “Ports” in Revised(5) Scheme.

load
transcript-on
transcript-off

See Section “System interface” in Revised(5) Scheme.

Optionals of [R5RS] not Supported by SCM

numerator
denominator
rationalize

See Section “Numerical operations” in Revised(5) Scheme.
[SLIB] Features of SCM and SCMLIT

delay
full-continuation
ieee-p1178
object-hash
rev4-report
source See SLIB file ‘Template.scm’.
current-time See Section “Time and Date” in SLIB.
defmacro See Section “Defmacro” in SLIB.
getenv
system See Section “System Interface” in SLIB.
hash See Section “Hashing” in SLIB.
logical See Section “Bit-Twiddling” in SLIB.
multiarg-apply See Section “Multi-argument Apply” in SLIB.
multiarg/and- See Section “Multi-argument / and -” in SLIB.
rev4-optional-procedures See Section “Rev4 Optional Procedures” in SLIB.
string-port See Section “String Ports” in SLIB.
tmpnam See Section “Input/Output” in SLIB.
transcript See Section “Transcripts” in SLIB.
vicinity See Section “Vicinity” in SLIB.
with-file See Section “With-File” in SLIB.

[SLIB] Features of SCM

array See Section “Arrays” in SLIB.
array-for-each See Section “Array Mapping” in SLIB.
bignum
complex
inexact
rational
real See Section “Require” in SLIB.
4.2 Storage

\texttt{vector-set-length! object length} \hspace{1em} [Function]

Change the length of string, vector, bit-vector, or uniform-array \texttt{object} to \texttt{length}. If this shortens \texttt{object} then the remaining contents are lost. If it enlarges \texttt{object} then the contents of the extended part are undefined but the original part is unchanged. It is an error to change the length of literal datums. The new object is returned.

\texttt{copy-tree obj} \hspace{1em} [Function]
\texttt{@copy-tree obj} \hspace{1em} [Function]

See Section “Tree Operations” in \textit{SLIB}. This extends the SLIB version by also copying vectors. Use \texttt{@copy-tree} if you depend on this feature; \texttt{copy-tree} could get redefined.

\texttt{acons obj1 obj2 obj3} \hspace{1em} [Function]

\begin{verbatim}
(set! a-list (acons key datum a-list))
\end{verbatim}

Adds a new association to \texttt{a-list}.

\texttt{gc-hook \ldots} \hspace{1em} [Callback procedure]

Allows a Scheme procedure to be run shortly after each garbage collection. This procedure will not be run recursively. If it runs long enough to cause a garbage collection before returning a warning will be printed.

To remove the \texttt{gc-hook}, (\texttt{set! gc-hook \#f}).

\texttt{add-finalizer object finalizer} \hspace{1em} [Function]

\texttt{object} may be any garbage collected object, that is, any object other than an immediate integer, character, or special token such as \texttt{#f} or \texttt{#t}. See Section 6.1.1 [Immediates], page 96. \texttt{finalizer} is a thunk, or procedure taking no arguments.

\texttt{finalizer} will be invoked asynchronously exactly once some time after \texttt{object} becomes eligible for garbage collection. A reference to \texttt{object} in the environment of \texttt{finalizer} will not prevent finalization, but will delay the reclamation of \texttt{object} at least until the next garbage collection. A reference to \texttt{object} in some other object’s finalizer will necessarily prevent finalization until both objects are eligible for garbage collection.

Finalizers are not run in any predictable order. All finalizers will be run by the time the program ends.


4.3 Time

\texttt{internal-time-units-per-second} \hspace{1em} [Constant]

Is the integer number of internal time units in a second.
get-internal-run-time

[Function]
Returns the integer run time in internal time units from an unspecified starting time. The difference of two calls to get-internal-run-time divided by internal-time-units-per-second will give elapsed run time in seconds.

get-internal-real-time

[Function]
Returns the integer time in internal time units from an unspecified starting time. The difference of two calls to get-internal-real-time divided by internal-time-units-per-second will give elapsed real time in seconds.

current-time

[Function]
Returns the time since 00:00:00 GMT, January 1, 1970, measured in seconds. See Section “Time and Date” in SLIB. current-time is used in Section “Time and Date” in SLIB.

4.4 Interrupts

ticks n

[Function]
Returns the number of ticks remaining till the next tick interrupt. Ticks are an arbitrary unit of evaluation. Ticks can vary greatly in the amount of time they represent.

If n is 0, any ticks request is canceled. Otherwise a ticks-interrupt will be signaled n from the current time. ticks is supported if SCM is compiled with the ticks flag defined.

ticks-interrupt ...

[Callback procedure]
Establishes a response for tick interrupts. Another tick interrupt will not occur unless ticks is called again. Program execution will resume if the handler returns. This procedure should (abort) or some other action which does not return if it does not want processing to continue.

alarm secs

[Function]
Returns the number of seconds remaining till the next alarm interrupt. If secs is 0, any alarm request is canceled. Otherwise an alarm-interrupt will be signaled secs from the current time. ALARM is not supported on all systems.

milli-alarm milliseecs interval

[Function]

virtual-alarm milliseecs interval

[Function]

profile-alarm milliseecs interval

[Function]
milli-alarm is similar to alarm, except that the first argument milliseecs, and the return value are measured in milliseconds rather than seconds. If the optional argument interval is supplied then alarm interrupts will be scheduled every interval milliseconds until turned off by a call to milli-alarm or alarm.

virtual-alarm and profile-alarm are similar. virtual-alarm decrements process execution time rather than real time, and causes SIGVTALRM to be signaled. profile-alarm decrements both process execution time and system execution time on behalf of the process, and causes SIGPROF to be signaled.
milli-alarm, virtual-alarm, and profile-alarm are supported only on systems providing the setitimer system call.

user-interrupt ... [Callback procedure]
alarm-interrupt ... [Callback procedure]
virtual-alarm-interrupt ... [Callback procedure]
profile-alarm-interrupt ...
Establishes a response for SIGINT (control-C interrupt) and SIGALRM, SIGVTALRM, and SIGPROF interrupts. Program execution will resume if the handler returns. This procedure should (abort) or some other action which does not return if it does not want processing to continue after it returns.

Interrupt handlers are disabled during execution system and ed procedures.

To unestablish a response for an interrupt set the handler symbol to #f. For instance, (set! user-interrupt #f).

out-of-storage ... [Callback procedure]
could-not-open ... [Callback procedure]
end-of-program ...
hang-up ...
arithmetic-error ...
Establishes a response for storage allocation error, file opening error, end of program, SIGHUP (hang up interrupt) and arithmetic errors respectively. This procedure should (abort) or some other action which does not return if it does not want the default error message to also be displayed. If no procedure is defined for hang-up then end-of-program (if defined) will be called.

To unestablish a response for an error set the handler symbol to #f. For instance, (set! could-not-open #f).

4.5 Process Synchronization

An exchanger is a procedure of one argument regulating mutually exclusive access to a resource. When a 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)))
          (let ((car lst)))
          (set! lst (cdr lst))
Chapter 4: The Language

make-arbiter name
Returns an object of type arbiter and name name. Its state is initially unlocked.

try-arbiter arbiter
Returns #t and locks arbiter if arbiter was unlocked. Otherwise, returns #f.

release-arbiter arbiter
Returns #t and unlocks arbiter if arbiter was locked. Otherwise, returns #f.

4.6 Files and Ports
These procedures generalize and extend the standard capabilities in Section “Ports” in Revised(5) Scheme.

4.6.1 Opening and Closing

open-file string modes
try-open-file string modes
Returns a port capable of receiving or delivering characters as specified by the modes string. If a file cannot be opened #f is returned.

Internal functions opening files callback to the SCM function open-file. You can extend open-file by redefining it. try-open-file is the primitive procedure; Do not redefine try-open-file!

open_read
open_write
open_both
Contain modes strings specifying that a file is to be opened for reading, writing, and both reading and writing respectively.

Both input and output functions can be used with io-ports. An end of file must be read or a two-argument file-position done on the port between a read operation and a write operation or vice-versa.

/ionbf modestr
Returns a version of modestr which when open-file is called with it as the second argument will return an unbuffered port. An input-port must be unbuffered in order for char-ready? and wait-for-input to work correctly on it. The initial value of (current-input-port) is unbuffered if the platform supports it.

/tracked modestr
Returns a version of modestr which when open-file is called with it as the second argument will return a tracked port. A tracked port maintains current line and column numbers, which may be queried with port-line and port-column.
Chapter 4: The Language

_function\_exclusive\_modestr

Returns a version of modestr which when open-file is called with it as the second argument will return a port only if the named file does not already exist. This functionality is provided by calling try-create-file See Section 5.6 [I/O-Extensions], page 73, which is not available for all platforms.

(open-ports)

Returns a list of all currently open ports, excluding string ports, see See Section “String Ports” in SLIB. This may be useful after a fork See Section 5.7 [Posix Extensions], page 77, or for debugging. Bear in mind that ports that would be closed by gc will be kept open by a reference to this list.

close-port port

Closes port. The same as close-input-port and close-output-port.

4.6.2 Port Properties

_port-closed? port

Returns #t if port is closed.

_port-type obj

If obj is not a port returns false, otherwise returns a symbol describing the port type, for example string or pipe.

_port-filename port

Returns the filename port was opened with. If port is not open to a file the result is unspecified.

_file-position port

_file-position port #f

Returns the current position of the character in port which will next be read or written. If port is open to a non-file then #f is returned.

_file-position port k

Sets the current position in port which will next be read or written. If successful, #f is returned. If port is open to a non-file, then file-position returns #f.

_port-line port

_port-column port

If port is a tracked port, return the current line (column) number, otherwise return #f. Line and column numbers begin with 1. The column number applies to the next character to be read; if that character is a newline, then the column number will be one more than the length of the line.

_freshline port

Outputs a newline to optional argument port unless the current output column number of port is known to be zero, ie output will start at the beginning of a new line. port defaults to current-output-port. If port is not a tracked port freshline is equivalent to newline.
isatty? port

Returns #t if port is input or output to a serial non-file device.

char-ready? port

Returns #t if a character is ready on the input port and returns #f otherwise. If char-ready? returns #t then the next read-char operation on the given port is guaranteed not to hang. If the port is at end of file then char-ready? returns #t. Port may be omitted, in which case it defaults to the value returned by current-input-port.

Rationale Char-ready? exists to make it possible for a program to accept characters from interactive ports without getting stuck waiting for input. Any input editors associated with such ports must ensure that characters whose existence has been asserted by char-ready? cannot be rubbed out. If char-ready? were to return #f at end of file, a port at end of file would be indistinguishable from an interactive port that has no ready characters.

wait-for-input x

wait-for-input x port1 ...

Returns a list those ports port1 ... which are char-ready?. If none of port1 ... become char-ready? within the time interval of x seconds, then #f is returned. The port1 ... arguments may be omitted, in which case they default to the list of the value returned by current-input-port.

4.6.3 Port Redirection

current-error-port

Returns the current port to which diagnostic output is directed.

with-error-to-file string thunk

thunk must be a procedure of no arguments, and string must be a string naming a file. The file is opened for output, an output port connected to it is made the default value returned by current-error-port, and the thunk is called with no arguments. When the thunk returns, the port is closed and the previous default is restored. With-error-to-file returns the value yielded by thunk.

with-input-from-port port thunk

with-output-to-port port thunk

with-error-to-port port thunk

These routines differ from with-input-from-file, with-output-to-file, and with-error-to-file in that the first argument is a port, rather than a string naming a file.

call-with-outputs thunk proc

Calls the thunk procedure while the current-output-port and current-error-port are directed to string-ports. If thunk returns, the proc procedure is called with the output-string, the error-string, and the value returned by thunk. If thunk does not return a value (perhaps because of error), proc is called with just the output-string and the error-string as arguments.
4.6.4 Soft Ports

A soft-port is a port based on a vector of procedures capable of accepting or delivering characters. It allows emulation of I/O ports.

**make-soft-port** vector modes  
Returns a port capable of receiving or delivering characters as specified by the modes string (see Section 4.6 [Files and Ports], page 49). vector must be a vector of length 5. Its components are as follows:

0. procedure accepting one character for output  
1. procedure accepting a string for output  
2. thunk for flushing output  
3. thunk for getting one character  
4. thunk for closing port (not by garbage collection)

For an output-only port only elements 0, 1, 2, and 4 need be procedures. For an input-only port only elements 3 and 4 need be procedures. Thunks 2 and 4 can instead be #f if there is no useful operation for them to perform.

If thunk 3 returns #f or an eof-object (see Section “Input” in Revised(5) Scheme) it indicates that the port has reached end-of-file. For example:

If it is necessary to explicitly close the port when it is garbage collected, (see Section 4.4 [Interrupts], page 47).

```scheme
(define stdout (current-output-port))
(define p (make-soft-port
  (vector
    (lambda (c) (write c stdout))
    (lambda (s) (display s stdout))
    (lambda () (display "." stdout))
    (lambda () (char-upcase (read-char)))
    (lambda () (display "@" stdout)))
"rw")

(write p p) ⇒ #<input-output-soft#\space45d10#>```

4.7 Eval and Load

**try-load** filename  
If the string filename names an existing file, the try-load procedure reads Scheme source code expressions and definitions from the file and evaluates them sequentially and returns #t. If not, try-load returns #f. The try-load procedure does not affect the values returned by current-input-port and current-output-port.

**load-pathname**  
Is set to the pathname given as argument to load, try-load, and dyn:link (see Section “Compiling And Linking” in Hobbit). *load-pathname* is used to compute the value of Section “Vicinity” in SLIB.
eval obj
   [Function]
   Alias for Section “System” in SLIB.

eval-string str
   [Function]
   Returns the result of reading an expression from str and evaluating it. eval-string does not
   change *load-pathname* or line-number.

load-string str
   [Function]
   Reads and evaluates all the expressions from str. As with load, the value returned is
   unspecified. load-string does not change *load-pathname* or line-number.

line-number
   [Function]
   Returns the current line number of the file currently being loaded.

4.7.1 Line Numbers

Scheme code defined by load may optionally contain line number information. Currently
this information is used only for reporting expansion time errors, but in the future run-time
error messages may also include line number information.

try-load pathname reader
   [Function]
   This is the primitive for loading, pathname is the name of a file containing Scheme
code, and optional argument reader is a function of one argument, a port. reader
should read and return Scheme code as list structure. The default value is read,
which is used if reader is not supplied or is false.

Line number objects are disjoint from integers or other Scheme types. When evaluated or
loaded as Scheme code, an s-expression containing a line-number in the car is equivalent
to the cdr of the s-expression. A pair consisting of a line-number in the car and a vector
in the cdr is equivalent to the vector. The meaning of s-expressions with line-numbers in
other positions is undefined.

read-numbered port
   [Function]
   Behaves like read, except that
        bullet Load (read) sytnaxes are enabled.
        bullet every s-expression read will be replaced with a cons of a line-number object and
        the sexp actually read. This replacement is done only if port is a tracked port
        See See Section 4.6 [Files and Ports], page 49.

integer->line-number int
   [Function]
   Returns a line-number object with value int. int should be an exact non-negative
   integer.

line-number->integer linum
   [Function]
   Returns the value of line-number object linum as an integer.

line-number? obj
   [Function]
   Returns true if and only if obj is a line-number object.
read-for-load port

Behaves like read, except that load syntaxes are enabled.

*load-reader*  [Variable]
*slib-load-reader*

The value of *load-reader* should be a value acceptable as the second argument to try-load (note that #f is acceptable). This value will be used to read code during calls to scm:load. The value of *slib-load-reader* will similarly be used during calls to slib:load and require.

In order to disable all line-numbering, it is sufficient to set! *load-reader* and *slib-load-reader* to #f.

4.8 Lexical Conventions

4.8.1 Common-Lisp Read Syntax

#

If token is a sequence of two or more digits, then this syntax is equivalent to 
#. (integer->char (string->number token 8)).

If token is C-, c-, or ^ followed by a character, then this syntax is read as a control character. If token is M- or m- followed by a character, then a meta character is read. c- and m- prefixes may be combined.

#+ feature form

If feature is provided? then form is read as a scheme expression. If not, then form is treated as whitespace.

Feature is a boolean expression composed of symbols and and, or, and not of boolean expressions.

For more information on provided?, See Section “Require” in SLIB.

#- feature form

is equivalent to #+ (not feature) expression.

#| any thing |#

Is a balanced comment. Everything up to the matching |# is ignored by the read. Nested #|...|# can occur inside any thing.

Load syntax is Read syntax enabled for read only when that read is part of loading a file or string. This distinction was made so that reading from a datafile would not be able to corrupt a scheme program using ‘#.’.

#. expression

Is read as the object resulting from the evaluation of expression. This substitution occurs even inside quoted structure.

In order to allow compiled code to work with #. it is good practice to define those symbols used inside of expression with #.(define ...). For example:
Chapter 4: The Language

#.(define foo 9) ⇒ #<unspecified>
'(#.foo #.(+ foo foo)) ⇒ (9 18)

#’ form is equivalent to form (for compatibility with common-lisp).

4.8.2 Load Syntax

#! is the unix mechanism for executing scripts. See Section 3.13.1 [Unix Scheme Scripts], page 41 for the full description of how this comment supports scripting.

#?line
#?column

Return integers for the current line and column being read during a load.

#?file

Returns the string naming the file currently being loaded. This path is the string passed to load, possibly with ‘.scm’ appended.

4.8.3 Documentation and Comments

procedure-documentation proc

Returns the documentation string of proc if it exists, or #f if not.

If the body of a lambda (or the definition of a procedure) has more than one expression, and the first expression (preceeding any internal definitions) is a string, then that string is the documentation string of that procedure.

(procedure-documentation (lambda (x) "Identity" x)) ⇒ "Identity"
(define (square x)
  "Return the square of X."
  (* x x))
⇒ #<unspecified>
(procedure-documentation square) ⇒ "Return the square of X."

comment string1 ...

Appends string1 ... to the strings given as arguments to previous calls comment.

comment

Returns the (appended) strings given as arguments to previous calls comment and empties the current string collection.

#;text-till-end-of-line

Behaves as (comment "text-till-end-of-line").

4.8.4 Modifying Read Syntax

read:sharp c port

If a # followed by a character (for a non-standard syntax) is encountered by read, read will call the value of the symbol read:sharp with arguments the character and the port being read from. The value returned by this function will be the value of read for this expression unless the function returns #<unspecified> in which case
the expression will be treated as whitespace. `<unspecified>` is the value returned by the expression `(if #f #f)`.

```scheme
load:sharp c port
```

[Callback procedure]
 Dispatches like `read:sharp`, but only during `loads`. The read-syntaxes handled by `load:sharp` are a superset of those handled by `read:sharp`. `load:sharp` calls `read:sharp` if none of its syntaxes match `c`.

```scheme
char:sharp token
```

[Callback procedure]
 If the sequence `\#\` followed by a non-standard character name is encountered by `read`, `read` will call the value of the symbol `char:sharp` with the token (a string of length at least two) as argument. If the value returned is a character, then that will be the value of `read` for this expression, otherwise an error will be signaled.

Note When adding new `#` syntaxes, have your code save the previous value of `load:sharp`, `read:sharp`, or `char:sharp` when defining it. Call this saved value if an invocation's syntax is not recognized. This will allow `#+`, `#-`, and Section 5.4.2 [Uniform Array], page 70s to still be supported (as they dispatch from `read:sharp`).

## 4.9 Syntax

SCM provides a native implementation of `defmacro`. See Section “Defmacro” in SLIB.

When built with `-F macro` build option (see Section 2.3.2 [Build Options], page 17) and `*syntax-rules*` is non-false, SCM also supports [R5RS] `syntax-rules` macros. See Section “Macros” in Revised(5) Scheme.

Other Scheme Syntax Extension Packages from SLIB can be employed through the use of `macro:eval` and `macro:load`; Or by using the SLIB read-eval-print-loop:

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

With the appropriate catalog entries (see Section “Library Catalogs” in SLIB), files using macro packages will automatically use the correct macro loader when `require'd.

### 4.9.1 Define and Set

```scheme
defined? symbol
```

[Special Form]
 Equivalent to `#t` if `symbol` is a syntactic keyword (such as `if`) or a symbol with a value in the top level environment (see Section “Variables and regions” in Revised(5) Scheme). Otherwise equivalent to `#f`.

```scheme
defvar identifier initial-value
```

[Special Form]
 If `identifier` is unbound in the top level environment, then `identifier` is defined to the result of evaluating the form `initial-value` as if the `defvar` form were instead the form `(define identifier initial-value)`. If `identifier` already has a value, then `initial-value` is not evaluated and `identifier`’s value is not changed. `defvar` is valid only when used at top-level.

```scheme
defconst identifier value
```

[Special Form]
 If `identifier` is unbound in the top level environment, then `identifier` is defined to the result of evaluating the form `value` as if the `defconst` form were instead the form
(define identifier value). If identifier already has a value, then value is not evaluated, identifier's value is not changed, and an error is signaled. defconst is valid only when used at top-level.

set! (variable1 variable2 ...) <expression>    [Special Form]
The identifiers variable1, variable2, ... must be bound either in some region enclosing the 'set!' expression or at top level.

<Expression> is evaluated, and the elements of the resulting list are stored in the locations to which each corresponding variable is bound. The result of the 'set!' expression is unspecified.

(def const x 2)
(define y 3)
(+ x y) => 5
(set! (x y) (list 4 5)) => unspecified
(+ x y) => 9

qase key clause1 clause2 ...

qase is an extension of standard Scheme case: Each clause of a qase statement must have as first element a list containing elements which are:

- literal datums, or
- a comma followed by the name of a symbolic constant, or
- a comma followed by an at-sign (@) followed by the name of a symbolic constant whose value is a list.

A qase statement is equivalent to a case statement in which these symbolic constants preceded by commas have been replaced by the values of the constants, and all symbolic constants preceded by comma-at-signs have been replaced by the elements of the list values of the constants. This use of comma, (or, equivalently, unquote) is similar to that of quasiquote except that the unquoted expressions must be symbolic constants.

Symbolic constants are defined using defconst, their values are substituted in the head of each qase clause during macro expansion. defconst constants should be defined before use. qase can be substituted for any correct use of case.

(defconst unit '1)
(defconst semivowels '(w y))
(qase (* 2 3)
  ((2 3 5 7) 'prime)
  (((unit 4 6 8 9) 'composite)) => composite
(qase (car '(c d))
  ((a) 'a)
  ((b) 'b)) => unspecified
(qase (car '(c d))
  ((a e i o u) 'vowel)
  (((@semivowels) 'semivowel)
  (else 'consonant)) => consonant
4.9.2 Defmacro

SCM supports the following constructs from Common Lisp: defmacro, macroexpand, macroexpand-1, and gentemp. See Section “Defmacro” in SLIB.

SCM defmacro is extended over that described for SLIB:

```scheme
(defmacro (macro-name . arguments) body)
```

is equivalent to

```scheme
(defmacro macro-name arguments body)
```

As in Common Lisp, an element of the formal argument list for defmacro may be a possibly nested list, in which case the corresponding actual argument must be a list with as many members as the formal argument. Rest arguments are indicated by improper lists, as in Scheme. It is an error if the actual argument list does not have the tree structure required by the formal argument list.

For example:

```scheme
(let1 ((x (foo))) (print x) x)
```

≡

```scheme
((lambda (x) (print x) x) (foo))
```

(let1 not legal syntax) [error] not "does not match" ((name value))

4.9.3 Syntax-Rules


The pattern language is extended by the syntax (... <obj>), which is identical to <obj> except that ellipses in <obj> are treated as ordinary identifiers in a template, or as literals in a pattern. In particular, (... ...) quotes the ellipsis token ... in a pattern or template.

For example:

```scheme
(define-syntax check-tree
  (syntax-rules ()
    ((_ (?pattern (... ...)) ?obj)
      (let loop ((obj ?obj))
        (or (null? obj)
            (and (pair? obj)
              (check-tree ?pattern (car obj))
              (loop (cdr obj))))))
    ((_ (?first . ?rest) ?obj)
      (let ((obj ?obj))
        (and (pair? obj)
          (check-tree ?first (car obj))
          (check-tree ?rest (cdr obj))))))
    ((_ ?atom ?obj) #t))
```
(check-tree ((a b) ...) '((1 2) (3 4) (5 6))) ⇒ #t

(check-tree ((a b) ...) '((1 2) (3 4) not-a-2list) ⇒ #f

Note that although the ellipsis is matched as a literal token in the defined macro it is not included in the literals list for syntax-rules.

The pattern language is also extended to support identifier macros. A reference to an identifier macro keyword that is not the first identifier in a form may expand into Scheme code, rather than raising a “keyword as variable” error. The pattern for expansion of such a bare macro keyword is a single identifier, as in other syntax rules the identifier is ignored.

For example:

(define-syntax eight
  (syntax-rules ()
    (_ 8)))

(+ 3 eight) ⇒ 11
(eight) ⇒ ERROR
(set! eight 9) ⇒ ERROR

4.9.4 Macro Primitives

procedure->syntax proc
[Function]
Returns a macro which, when a symbol defined to this value appears as the first symbol in an expression, returns the result of applying proc to the expression and the environment.

procedure->macro proc
[Function]
procedure->memoizing-macro proc
[Function]
procedure->identifier-macro
[Function]

Returns a macro which, when a symbol defined to this value appears as the first symbol in an expression, evaluates the result of applying proc to the expression and the environment. The value returned from proc which has been passed to procedure->macro replaces the form passed to proc. For example:

(defsyntax trace
  (procedure->macro
    (lambda (x env) '(set! ,(cadr x) (tracef ,(cadr x) ',(cadr x))),(cadr x)))))

(trace foo) ≡ (set! foo (tracef foo 'foo)).

PROCEDURE->IDENTIFIER-MACRO is similar to procedure->memoizing-macro except that proc is also called in case the symbol bound to the macro appears in an expression but not as the first symbol, that is, when it looks like a variable reference. In that case, the form passed to proc is a single identifier.

defsyntax name expr
[Special Form]
Defines name as a macro keyword bound to the result of evaluating expr, which should be a macro. Using define for this purpose may not result in name being interpreted as a macro keyword.
4.9.5 Environment Frames

An environment is a list of frames representing lexical bindings. Only the names and scope of the bindings are included in environments passed to macro expanders – run-time values are not included.

There are several types of environment frames:

- `((lambda (variable1 ... ...) value1 ...)`
- `(let ((variable1 value1) (variable2 value2) ...) ...)`
- `(letrec ((variable1 value1) ...) ...)`
  - result in a single environment frame:
    - `(variable1 variable2 ...)`
- `(let ((variable1 value1)) ...)`
- `(let* ((variable1 value1) ...) ...)`
  - result in an environment frame for each variable:
    - variable1 variable2 ...
- `(let-syntax ((key1 macro1) (key2 macro2)) ...)`
- `(letrec-syntax ((key1 value1) (key2 value2)) ...)`
  - Lexically bound macros result in environment frames consisting of a marker and an alist of keywords and macro objects:
    - `<env-syntax-marker> (key1 . value1) (key2 . value2)>
  - Currently `<env-syntax-marker>` is the integer 6.

line numbers

Line numbers (see Section 4.7.1 [Line Numbers], page 53) may be included in the environment as frame entries to indicate the line number on which a function is defined. They are ignored for variable lookup.

```
#<line 8>
```

miscellaneous

Debugging information is stored in environments in a plist format: Any exact integer stored as an environment frame may be followed by any value. The two frame entries are ignored when doing variable lookup. Load file names, procedure names, and closure documentation strings are stored in this format.

```
<env-filename-marker> "foo.scm" <env-procedure-name-marker> foo ...
```

Currently `<env-filename-marker>` is the integer 1 and `<env-procedure-name-marker>` the integer 2.
@apply procedure argument-list

Returns the result of applying procedure to argument-list. @apply differs from apply when the identifiers bound by the closure being applied are set!; setting affects argument-list.

\[
\begin{align*}
\text{(define lst (list 'a 'b 'c))} \\
\text{(@apply (lambda (v1 v2 v3) (set! v1 (cons v2 v3))) lst)} \\
\text{lst} & \Rightarrow ((b . c) b c)
\end{align*}
\]

Thus a mutable environment can be treated as both a list and local bindings.

### 4.9.6 Syntactic Hooks for Hygienic Macros

SCM provides a synthetic identifier type for efficient implementation of hygienic macros (for example, syntax-rules see Section “Macros” in Revised(5) Scheme) A synthetic identifier may be inserted in Scheme code by a macro expander in any context where a symbol would normally be used. Collectively, symbols and synthetic identifiers are identifiers.

**identifier?** obj

Returns #t if obj is a symbol or a synthetic identifier, and #f otherwise.

If it is necessary to distinguish between symbols and synthetic identifiers, use the predicate symbol?.

A synthetic identifier includes two data: a parent, which is an identifier, and an environment, which is either #f or a lexical environment which has been passed to a macro expander (a procedure passed as an argument to procedure->macro, procedure->memoizing-macro, or procedure->syntax).

**renamed-identifier** parent env

Returns a synthetic identifier. parent must be an identifier, and env must either be #f or a lexical environment passed to a macro expander. renamed-identifier returns a distinct object for each call, even if passed identical arguments.

There is no direct way to access all of the data internal to a synthetic identifier, those data are used during variable lookup. If a synthetic identifier is inserted as quoted data then during macro expansion it will be repeatedly replaced by its parent, until a symbol is obtained.

**identifier->symbol** id

Returns the symbol obtained by recursively extracting the parent of id, which must be an identifier.

### 4.9.7 Use of Synthetic Identifiers

renamed-identifier may be used as a replacement for gentemp:

\[
\begin{align*}
\text{(define gentemp} \\
\text{ (let ((name (string->symbol "An unlikely variable")))} \\
\text{ (lambda ()} \\
\text{ (renamed-identifier name #f)))))}
\end{align*}
\]

If an identifier returned by this version of gentemp is inserted in a binding position as the name of a variable then it is guaranteed that no other identifier (except one produced
by passing the first to `renamed-identifier`) may denote that variable. If an identifier
returned by `gentemp` is inserted free, then it will denote the top-level value bound to its
parent, the symbol named “An unlikely variable”. This behavior, of course, is meant to be
put to good use:

```lisp
(define top-level-foo
  (procedure->memoizing-macro
   (lambda (exp env)
     (renamed-identifier 'foo #f)))))
```

Defines a macro which may always be used to refer to the top-level binding of `foo`.

```lisp
(define foo 'top-level)
(let ((foo 'local))
  (top-level-foo)) ⇒ top-level
```

In other words, we can avoid capturing `foo`.

If a lexical environment is passed as the second argument to `renamed-identifier` then if
the identifier is inserted free its parent will be looked up in that environment, rather than
in the top-level environment. The use of such an identifier must be restricted to the lexical
scope of its environment.

There is another restriction imposed for implementation convenience: Macros passing their
lexical environments to `renamed-identifier` may be lexically bound only by the special
forms `let-syntax` or `letrec-syntax`. No error is signaled if this restriction is not met, but
synthetic identifier lookup will not work properly.

In order to maintain referential transparency it is necessary to determine whether two
identifiers have the same denotation. With synthetic identifiers it is not necessary that two
identifiers be `eq?` in order to denote the same binding.

`identifier-equal? id1 id2 env`  [Function]

Returns `#t` if identifiers `id1` and `id2` denote the same binding in lexical environment
`env`, and `#f` otherwise. `env` must either be a lexical environment passed to a macro
transformer during macro expansion or the empty list.

For example,

```lisp
(define top-level-foo?
  (procedure->memoizing-macro
   (let ((foo-name (renamed-identifier 'foo #f)))
     (lambda (exp env)
       (identifier-equal? (cadr exp) foo-name env)))))

(top-level-foo? foo) ⇒ #t

(let ((foo 'local))
  (top-level-foo? foo)) ⇒ #f
```

`@macroexpand1 expr env`  [Function]

If the `car` of `expr` denotes a macro in `env`, then if that macro is a primitive, `expr`
will be returned, if the macro was defined in Scheme, then a macro expansion will be
returned. If the `car` of `expr` does not denote a macro, the `#f` is returned.
**extended-environment** names values env  
Returns a new environment object, equivalent to env, which must either be an environment object or null, extended by one frame. names must be an identifier, or an improper list of identifiers, usable as a formals list in a lambda expression. values must be a list of objects long enough to provide a binding for each of the identifiers in names. If names is an identifier or an improper list then vals may be, respectively, any object or an improper list of objects.

**syntax-quote** obj  
Synthetic identifiers are converted to their parent symbols by quote and quasiquote so that literal data in macro definitions will be properly transcribed. syntax-quote behaves like quote, but preserves synthetic identifier intact.

**the-macro** mac  
the-macro is the simplest of all possible macro transformers: mac may be a syntactic keyword (macro name) or an expression evaluating to a macro, otherwise an error is signaled. mac is evaluated and returned once only, after which the same memoized value is returned.
the-macro may be used to protect local copies of macros against redefinition, for example:

```
(@let-syntax ((let (the-macro let)))
  ;; code that will continue to work even if LET is redefined.
  ...)
```

**renaming-transformer** proc  
A low-level “explicit renaming” macro facility very similar to that proposed by W. Clinger [Exrename] is supported. Syntax may be defined in define-syntax, let-syntax, and letrec-syntax using renaming-transformer instead of syntax-rules. proc should evaluate to a procedure accepting three arguments: expr, rename, and compare. expr is a representation of Scheme code to be expanded, as list structure. rename is a procedure accepting an identifier and returning an identifier renamed in the definition environment of the new syntax. compare accepts two identifiers and returns true if and only if both denote the same binding in the usage environment of the new syntax.
5 Packages

5.1 Dynamic Linking

If SCM has been compiled with `dynl.c` then the additional properties of load and ([SLIB]) require specified here are supported. The require form is preferred.

**require feature**

If the symbol feature has not already been given as an argument to require, then the object and library files associated with feature will be dynamically-linked, and an unspecified value returned. If feature is not found in *catalog*, then an error is signaled.

**usr:lib lib**

Returns the pathname of the C library named lib. For example: `(usr:lib "m")` returns "/usr/lib/libm.a", the path of the C math library.

**x:lib lib**

Returns the pathname of the X library named lib. For example: `(x:lib "X11")` returns "/usr/X11/lib/libX11.sa", the path of the X11 library.

**load filename lib1 ...**

In addition to the [R5RS] requirement of loading Scheme expressions if filename is a Scheme source file, load will also dynamically load/link object files (produced by compile-file, for instance). The object-suffix need not be given to load. For example,

(load (in-vicinity (implementation-vicinity) "sc2"))

or (load (in-vicinity (implementation-vicinity) "sc2.o"))

or (require 'rev2-procedures)

or (require 'rev3-procedures)

will load/link 'sc2.o' if it exists.

The lib1 ... pathnames specify additional libraries which may be needed for object files not produced by the Hobbit compiler. For instance, crs is linked on GNU/Linux by

(load (in-vicinity (implementation-vicinity) "crs.o")
     (usr:lib "ncurses") (usr:lib "c"))

or (require 'curses)

Turtlegr graphics library is linked by:

(load (in-vicinity (implementation-vicinity) "turtlegr")
     (usr:lib "X11") (usr:lib "c") (usr:lib "m"))

or (require 'turtle-graphics)

And the string regular expression (see Section 5.10 [Regular Expression Pattern Matching], page 82) package is linked by:

(load (in-vicinity (implementation-vicinity) "rgx") (usr:lib "c"))

or
The following functions comprise the low-level Scheme interface to dynamic linking. See the file ‘Link.scm’ in the SCM distribution for an example of their use.

**Function**

dyn:link filename

*filename* should be a string naming an object or archive file, the result of C-compiling. The *dyn:link* procedure links and loads *filename* into the current SCM session. If successful, *dyn:link* returns a link-token suitable for passing as the second argument to *dyn:call*. If not successful, #f is returned.

**Function**

dyn:call name link-token

*link-token* should be the value returned by a call to *dyn:link*. *name* should be the name of C function of no arguments defined in the file named *filename* which was successfully *dyn:linked* in the current SCM session. The *dyn:call* procedure calls the C function corresponding to *name*. If successful, *dyn:call* returns #t; If not successful, #f is returned.

dyn:call is used to call the init... function after loading SCM object files. The init... function then makes the identifiers defined in the file accessible as Scheme procedures.

**Function**

dyn:main-call name link-token arg1 ...

*link-token* should be the value returned by a call to *dyn:link*. *name* should be the name of C function of 2 arguments, (int argc, const char **argv), defined in the file named *filename* which was successfully *dyn:linked* in the current SCM session. The *dyn:main-call* procedure calls the C function corresponding to *name* with *argv* style arguments, such as are given to C main functions. If successful, *dyn:main-call* returns the integer returned from the call to *name*.

dyn:main-call can be used to call a main procedure from SCM. For example, I link in and dyn:main-call a large C program, the low level routines of which callback (see Section 6.2.11 [Callbacks], page 122) into SCM (which emulates PCI hardware).

**Function**

dyn:unlink link-token

*link-token* should be the value returned by a call to *dyn:link*. The *dyn:unlink* procedure removes the previously loaded file from the current SCM session. If successful, *dyn:unlink* returns #t; If not successful, #f is returned.

### 5.2 Dump

*Dump*, (also known as unexec), saves the continuation of an entire SCM session to an executable file, which can then be invoked as a program. Dumped executables start very quickly, since no Scheme code has to be loaded.

There are constraints on which sessions are savable using *dump*

- Saved continuations are invalid in subsequent invocations; they cause segmentation faults and other unpleasant side effects.
- Although DLD (see Section 5.1 [Dynamic Linking], page 64) can be used to load compiled modules both before and after dumping, ‘SUN_DL’ ELF systems can load compiled
modules only after dumping. This can be worked around by compiling in those features you wish to dump.

- Ports (other than `current-input-port`, `current-output-port`, `current-error-port`), X windows, etc. are invalid in subsequent invocations.

  This restriction could be removed; See Section 6.4 [Improvements To Make], page 129.

- **Dump** should only be called from a loading file when the call to dump is the last expression in that file.

- **Dump** can be called from the command line.

```
dump newpath                      [Function]
dump newpath #f                   [Function]
dump newpath #t                   [Function]
dump newpath thunk                [Function]
```

- Calls `gc`.
- Creates an executable program named `newpath` which continues the state of the current SCM session when invoked. The optional argument `thunk`, if provided, should be a procedure of no arguments; `boot-tail` will be set to this procedure, causing it to be called in the restored executable.

  If the optional argument is missing or a boolean, SCM's standard command line processing will be called in the restored executable.

  If the second argument to `dump` is `#t`, argument processing will continue from the command line passed to the dumping session. If the second argument is missing or `#f` then the command line arguments of the restoring invocation will be processed.

  - Resumes the top level Read-Eval-Print loop. This is done instead of continuing normally to avoid creating a saved continuation in the dumped executable.

`dump` may set the values of `boot-tail`, `*argv*`, `restart`, and `*interactive*`. `dump` returns an unspecified value.

When a dumped executable is invoked, the variable `*interactive*` (see Section 3.12 [Internal State], page 39) has the value it possessed when `dump` created it. Calling `dump` with a single argument sets `*interactive*` to `#f`, which is the state it has at the beginning of command line processing.

The procedure `program-arguments` returns the command line arguments for the current invocation. More specifically, `program-arguments` for the restored session are *not* saved from the dumping session. Command line processing is done on the value of the identifier `*argv*`.

The following example shows how to create `rscm`, which is like regular `scm`, but which loads faster and has the `random` package already provided.

```
bash$ scm -rrandom
> (dump "rscm")
#<unspecified>
> (quit)
bash$ ./rscm -lpi.scm -e"(pi (random 200) 5)"
```
This task can also be accomplished using the ‘-o’ command line option (see Section 3.2 [SCM Options], page 28).

bash$ scm -rrandom -o rscm
> (quit)
bash$ ./rscm -lpi.scm -e"(pi (random 200) 5)"
00003 14159 26535 89793 23846 26433 83279 50288 41971 69399
37510 58209 74944 59230 78164 06286 20899 86280 34825 34211
70679 82148 08651 32823 06647 09384 46095 50582 23172 53594
08128 48111 74502 84102 70193 85211 05559 64462 29489
bash$

5.3 Numeric

most-positive-fixnum
[Constant]
The immediate integer closest to positive infinity. See Section “Configuration” in SLIB.

most-negative-fixnum
[Constant]
The immediate integer closest to negative infinity.

$pi
[Constant]
pi
[Constant]
The ratio of the circumference to the diameter of a circle.

These procedures are in addition to those in See Section “Irrational Integer Functions” in SLIB.

exact-round x
[Function]
exact-floor x
[Function]
exact-ceiling x
[Function]
exact-truncate x
[Function]
    Return exact integers.

These procedures augment the standard capabilities in Section “Numerical operations” in Revised(5) Scheme. Many are from See Section “Irrational Real Functions” in SLIB.

pi* z
[Function]
    (* pi z)

pi/ z
[Function]
    (/ pi z)
sinh \( z \)  
\[ \text{Function} \]
\( \text{cosh} \ z \)  
\[ \text{Function} \]
\( \text{tanh} \ z \)  
\[ \text{Function} \]

Return the hyperbolic sine, cosine, and tangent of \( z \)

asinh \( z \)  
\[ \text{Function} \]
acosh \( z \)  
\[ \text{Function} \]
atanh \( z \)  
\[ \text{Function} \]

Return the inverse hyperbolic sine, cosine, and tangent of \( z \)

real-sqrt \( x \)  
\[ \text{Function} \]
real-exp \( x \)  
\[ \text{Function} \]
real-ln \( x \)  
\[ \text{Function} \]
real-sin \( x \)  
\[ \text{Function} \]
real-cos \( x \)  
\[ \text{Function} \]
real-tan \( x \)  
\[ \text{Function} \]
real-asin \( x \)  
\[ \text{Function} \]
real-acos \( x \)  
\[ \text{Function} \]
real-atan \( x \)  
\[ \text{Function} \]
atan \( y \ x \)  
\[ \text{Function} \]
real-sinh \( x \)  
\[ \text{Function} \]
real-cosh \( x \)  
\[ \text{Function} \]
real-tanh \( x \)  
\[ \text{Function} \]
real-asinh \( x \)  
\[ \text{Function} \]
real-acosh \( x \)  
\[ \text{Function} \]
real-atanh \( x \)  
\[ \text{Function} \]

Real-only versions of these popular functions. The argument \( x \) must be a real number. It is an error if the value which should be returned by a call to these procedures is not real.

real-log10 \( x \)  
\[ \text{Function} \]
Real-only base 10 logarithm.

$\text{atan2} \ y \ x$  
\[ \text{Function} \]
Computes \((\text{angle} \ (\text{make-rectangular} \ x \ y))\) for real numbers \( y \) and \( x \).

real-expt \( x1 \ x2 \)  
\[ \text{Function} \]
Returns real number \( x1 \) raised to the real power \( x2 \). It is an error if the value which should be returned by a call to real-expt is not real.

infinite? \( z \)  
\[ \text{Function} \]
finite? \( z \)  
\[ \text{Function} \]
All IEEE-754 numbers except positive and negative infinity and NaN (non-a-number) are finite.

5.4 Arrays
5.4.1 Conventional Arrays

The following syntax and procedures are SCM extensions to feature array in Section “Arrays” in SLIB.

Arrays read and write as a # followed by the rank (number of dimensions) followed by the character #\a or #\A and what appear as lists (of lists) of elements. The lists must be nested to the depth of the rank. For each depth, all lists must be the same length.

\begin{verbatim}
(make-array '#(ho) 4 3) ⇒
\end{verbatim}

Unshared, conventional (not uniform) 0-based arrays of rank 1 are equivalent to (and can’t be distinguished from) scheme vectors.

\begin{verbatim}
(make-array '#(ho) 3) ⇒ #(ho ho ho)
\end{verbatim}

\textbf{transpose-array} \hspace{1em} array \hspace{1em} dim0 \hspace{1em} dim1 \ldots 

[Function]

Returns an array sharing contents with array, but with dimensions arranged in a different order. There must be one \textit{dim} argument for each dimension of array. \textit{dim0}, \textit{dim1}, \ldots should be integers between 0 and the rank of the array to be returned. Each integer in that range must appear at least once in the argument list.

The values of \textit{dim0}, \textit{dim1}, \ldots correspond to dimensions in the array to be returned, their positions in the argument list to dimensions of array. Several \textit{dims} may have the same value, in which case the returned array will have smaller rank than \textit{array}.

\begin{verbatim}
examples:
(transpose-array '#2A((a b) (c d)) 1 0) ⇒ #2A((a c) (b d))
(transpose-array '#2A((a b) (c d)) 0 0) ⇒ #1A(a d)
(transpose-array '#3A(((a b c) (d e f)) ((1 2 3) (4 5 6))) 1 1 0) ⇒
#2A((a 4) (b 5) (c 6))
\end{verbatim}

\textbf{enclose-array} \hspace{1em} array \hspace{1em} dim0 \hspace{1em} dim1 \ldots 

[Function]

\textit{dim0}, \textit{dim1} \ldots should be nonnegative integers less than the rank of \textit{array}. \textbf{enclose-array} returns an array resembling an array of shared arrays. The dimensions of each shared array are the same as the \textit{dim}th dimensions of the original array, the dimensions of the outer array are the same as those of the original array that did not match a \textit{dim}.

An enclosed array is not a general Scheme array. Its elements may not be set using \textbf{array-set!}. Two references to the same element of an enclosed array will be \texttt{equal?} but will not in general be \texttt{eq?}. The value returned by \textbf{array-prototype} when given an enclosed array is unspecified.

\begin{verbatim}
examples:
(enclose-array #'3A(((a b c) (d e f)) ((1 2 3) (4 5 6))) 1) ⇒
#<enclosed-array (#1A(a d) #1A(b e) #1A(c f)) (#1A(1 4) #1A(2 5) #1A(3 6))>

(enclose-array #'3A(((a b c) (d e f)) ((1 2 3) (4 5 6))) 1 0) ⇒
#<enclosed-array #2A((a 1) (d 4)) #2A((b 2) (e 5)) #2A((c 3) (f 6))>
\end{verbatim}
array->list array

[Function]

Returns a list consisting of all the elements, in order, of array. In the case of a rank-0 array, returns the single element.

array-contents array

array-contents array strict

[Function]

If array may be unrolled into a one dimensional shared array without changing their order (last subscript changing fastest), then array-contents returns that shared array, otherwise it returns #f. All arrays made by make-array may be unrolled, some arrays made by make-shared-array may not be.

If the optional argument strict is provided, a shared array will be returned only if its elements are stored internally contiguous in memory.

5.4.2 Uniform Array

Uniform Arrays and vectors are arrays whose elements are all of the same type. Uniform vectors occupy less storage than conventional vectors. Uniform Array procedures also work on vectors, uniform-vectors, bit-vectors, and strings.

SLIB now supports uniform arrays. The primary array creation procedure is make-array, detailed in See Section “Arrays” in SLIB.

Unshared uniform character 0-based arrays of rank 1 (dimension) are equivalent to (and can’t be distinguished from) strings.

```
(make-array "" 3) ⇒ "$q2"
```

Unshared uniform boolean 0-based arrays of rank 1 (dimension) are equivalent to (and can’t be distinguished from) Section 5.4.3 [Bit Vectors], page 71.

```
(make-array '#1at() 3) ⇒ #*000
⇒ #1At(#f #f #f) ⇒ #*000
```

prototype arguments in the following procedures are interpreted according to the table:

<table>
<thead>
<tr>
<th>prototype</th>
<th>type</th>
<th>display prefix</th>
</tr>
</thead>
<tbody>
<tr>
<td>()</td>
<td>conventional vector</td>
<td>#A</td>
</tr>
<tr>
<td>+64i</td>
<td>complex (double precision)</td>
<td>#A:floC64b</td>
</tr>
<tr>
<td>64.0</td>
<td>double (double precision)</td>
<td>#A:floR64b</td>
</tr>
<tr>
<td>32.0</td>
<td>float (single precision)</td>
<td>#A:floR32b</td>
</tr>
<tr>
<td>32</td>
<td>unsigned integer (32-bit)</td>
<td>#A:fixN32b</td>
</tr>
<tr>
<td>-32</td>
<td>signed integer (32-bit)</td>
<td>#A:fixZ32b</td>
</tr>
<tr>
<td>-16</td>
<td>signed integer (16-bit)</td>
<td>#A:fixZ16b</td>
</tr>
<tr>
<td>\a</td>
<td>char (string)</td>
<td>#A:char</td>
</tr>
<tr>
<td>#t</td>
<td>boolean (bit-vector)</td>
<td>#A:bool</td>
</tr>
</tbody>
</table>

Other uniform vectors are written in a form similar to that of general arrays, except that one or more modifying characters are put between the \A character and the contents list. For example, '#1A:fixZ32b(3 5 9) returns a uniform vector of signed integers.

array? obj prototype

[Function]

Returns #t if the obj is an array of type corresponding to prototype, and #f if not.
array-prototype array  
Returns an object that would produce an array of the same type as array, if used as the prototype for list->uniform-array.

list->uniform-array rank prot lst  
Returns a uniform array of the type indicated by prototype prot with elements the same as those of lst. Elements must be of the appropriate type, no coercions are done.

In, for example, the case of a rank-2 array, lst must be a list of lists, all of the same length. The length of lst will be the first dimension of the result array, and the length of each element the second dimension.

If rank is zero, lst, which need not be a list, is the single element of the returned array.

uniform-array-read!  
Attempts to read all elements of ura, in lexicographic order, as binary objects from port. If an end of file is encountered during uniform-array-read! the objects up to that point only are put into ura (starting at the beginning) and the remainder of the array is unchanged.

uniform-array-read! returns the number of objects read. port may be omitted, in which case it defaults to the value returned by (current-input-port).

uniform-array-write  
Writes all elements of ura as binary objects to port. The number of of objects actually written is returned. port may be omitted, in which case it defaults to the value returned by (current-output-port).

logaref array index1 index2 . . .  
If an index is provided for each dimension of array returns the index1, index2, . . .'th element of array. If one more index is provided, then the last index specifies bit position of the twos-complement representation of the array element indexed by the other indexes returning #t if the bit is 1, and #f if 0. It is an error if this element is not an exact integer.

logaset! array val index1 index2 . . .  
If an index is provided for each dimension of array sets the index1, index2, . . .'th element of array to val. If one more index is provided, then the last index specifies bit position of the twos-complement representation of an exact integer array element, setting the bit to 1 if val is #t and to 0 if val is #f. In this case it is an error if the array element is not an exact integer or if val is not boolean.

5.4.3 Bit Vectors

Bit vectors can be written and read as a sequence of 0s and 1s prefixed by #*.
Some of these operations will eventually be generalized to other uniform-arrays.

bit-count bool bv  
Returns the number of occurrences of bool in bv.

bit-position bool bv k  
Returns the minimum index of an occurrence of bool in bv which is at least k. If no bool occurs within the specified range #f is returned.

bit-invert! bv  
Modifies bv by replacing each element with its negation.

bit-set*! bv uve bool  
If uve is a bit-vector, then bv and uve must be of the same length. If bool is #t, then uve is OR’ed into bv; If bool is #f, the inversion of uve is AND’ed into bv. 
If uve is a unsigned integer vector, then all the elements of uve must be between 0 and the LENGTH of bv. The bits of bv corresponding to the indexes in uve are set to bool.
The return value is unspecified.

bit-count* bv uve bool  
Returns
(bit-count (bit-set*! (if bool bv (bit-invert! bv)) uve #t) #t).
bv is not modified.

5.4.4 Array Mapping

SCM has some extra functions in feature array-for-each:

array-fill! array fill  
Stores fill in every element of array. The value returned is unspecified.

serial-array:copy! destination source  
Same as array:copy! but guaranteed to copy in row-major order.

array-equal? array0 array1 . . .  
Returns #t iff all arguments are arrays with the same shape, the same type, and have corresponding elements which are either equal? or array-equal?. This function differs from equal? in that a one dimensional shared array may be array-equal? but not equal? to a vector or uniform vector.

array-map! array0 proc array1 . . .  
If array1, . . . are arrays, they 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. If they are scalars, that is, not arrays, vectors, or strings, then they will be converted internally to arrays of the appropriate shape. proc is applied to each
tuple of elements of \textit{array1} \ldots and the result is stored as the corresponding element in \textit{array0}. The value returned is unspecified. The order of application is unspecified. Handling non-array arguments is a SCM extension of Section “Array Mapping” in \textsc{SLIB}.

\textbf{serial-array-map!} \ \textit{array0 \ proc \ array1} \ldots

Same as \textbf{array-map!}, but guaranteed to apply \textit{proc} in row-major order.

\textbf{array-map} \ \textit{prototype \ proc \ array1 \ array2} \ldots

\textit{array2}, \ldots must have the same number of dimensions as \textit{array1} and have a range for each index which includes the range for the corresponding index in \textit{array1}. \textit{proc} is applied to each tuple of elements of \textit{array1}, \textit{array2}, \ldots and the result is stored as the corresponding element in a new array of type \textit{prototype}. The new array is returned. The order of application is unspecified.

\textbf{scalar->array} \ \textit{scalar \ array \ prototype}

Returns a uniform array of the same shape as \textit{array}, having only one shared element, which is \texttt{eqv?} to \textit{scalar}. If the optional argument \textit{prototype} is supplied it will be used as the prototype for the returned array. Otherwise the returned array will be of the same type as \textit{array} if that is possible, and a conventional array if it is not. This function is used internally by \textbf{array-map!} and friends to handle scalar arguments.

\section{Records}

SCM provides user-definable datatypes with the same interface as SLIB, see See Section “Records” in \textsc{SLIB}, with the following extension.

\textbf{record-printer-set!} \ \textit{rtd \ printer}

Causes records of type \textit{rtd} to be printed in a user-specified format. \textit{rtd} must be a record type descriptor returned by \textbf{make-record-type}, \textit{printer} a procedure accepting three arguments: the record to be printed, the port to print to, and a boolean which is true if the record is being written on behalf of \texttt{write} and false if for \texttt{display}. If \textit{printer} returns \#f, the default record printer will be called.

A \textit{printer} value of \#f means use the default printer.

Only the default printer will be used when printing error messages.

\section{I/O-Extensions}

If \texttt{‘i/o-extensions} is provided (by linking in \texttt{‘ioext.o’}), Section “Line I/O” in \textsc{SLIB}, and the following functions are defined:

\textbf{stat} \ \texttt{<port-or-string>}

Returns a vector of integers describing the argument. The argument can be either a string or an open input port. If the argument is an open port then the returned vector describes the file to which the port is opened; If the argument is a string then the returned vector describes the file named by that string. If there exists no file with the name string, or if the file cannot be accessed \#f is returned. The elements of the returned vector are as follows:
0 st_dev  ID of device containing a directory entry for this file
1 st_ino  Inode number
2 st_mode File type, attributes, and access control summary
3 st_nlink Number of links
4 st_uid  User ID of file owner
5 st_gid  Group ID of file group
6 st_rdev  Device ID; this entry defined only for char or blk spec files
7 st_size  File size (bytes)
8 st_atime Time of last access
9 st_mtime Last modification time
10 st_ctime Last file status change time

g getpid
  [Function]
  Returns the process ID of the current process.

t try-create-file name modes perms
  [Function]
  If the file with name name already exists, return #f; otherwise try to create and
  open the file like try-open-file. See Section 4.6 [Files and Ports], page 49. If the
  optional integer argument perms is provided, it is used as the permissions of the new
  file (modified by the current umask).

r reopen-file filename modes port
  [Function]
  Closes port port and reopens it with filename and modes. reopen-file returns #t
  if successful, #f if not.

d duplicate-port port modes
  [Function]
  Creates and returns a duplicate port from port. Duplicate unbuffered ports share one
  file position. modes are as for Section 4.6 [Files and Ports], page 49.

r redirect-port! from-port to-port
  [Function]
  Closes to-port and makes to-port be a duplicate of from-port. redirect-port! re-
  turns to-port if successful, #f if not. If unsuccessful, to-port is not closed.

o opendir dirname
  [Function]
  Returns a directory object corresponding to the file system directory named dirname.
  If unsuccessful, returns #f.

r readdir dir
  [Function]
  Returns the string name of the next entry from the directory dir. If there are no more
  entries in the directory, readdir returns a #f.

r rewinddir dir
  [Function]
  Reinitializes dir so that the next call to readdir with dir will return the first entry
  in the directory again.
Chapter 5: Packages

closedir dir

Closes dir and returns #t. If dir is already closed., closedir returns a #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")

-|

"Init.scm"
"Iedline.scm"
"Link.scm"
"Macro.scm"
"Transcen.scm"
"Init5f1.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.

mkdir path mode

The mkdir function creates a new, empty directory whose name is path. The integer argument mode specifies the file permissions for the new directory. See Section “The Mode Bits for Access Permission” in Gnu C Library, for more information about this. mkdir returns if successful, #f if not.

rmdir path

The rmdir function deletes the directory path. The directory must be empty before it can be removed. rmdir returns if successful, #f if not.

chdir filename

Changes the current directory to filename. If filename does not exist or is not a directory, #f is returned. Otherwise, #t is returned.

getcwd

The function getcwd returns a string containing the absolute file name representing the current working directory. If this string cannot be obtained, #f is returned.
rename-file oldfilename newfilename
   Renames the file specified by oldfilename to newfilename. If the renaming is successful, #t is returned. Otherwise, #f is returned.

copy-file oldfilename newfilename
   Copies the file specified by oldfilename to newfilename. If the copying is successful, #t is returned. Otherwise, #f is returned.

chmod file mode
   The function chmod sets the access permission bits for the file named by file to mode. The file argument may be a string containing the filename or a port open to the file. chmod returns if successful, #f if not.

utime pathname acctime modtime
   Sets the file times associated with the file named pathname to have access time acctime and modification time modtime. utime returns if successful, #f if not.

umask mode
   The function umask sets the file creation mask of the current process to mask, and returns the previous value of the file creation mask.

fileno port
   Returns the integer file descriptor associated with the port port. If an error is detected, #f is returned.

access pathname how
   Returns #t if the file named by pathname can be accessed in the way specified by the how argument. The how argument can be the logior of the flags:
   0. File-exists?
   1. File-is-executable?
   2. File-is-writable?
   4. File-is-readable?

Or the how argument can be a string of 0 to 3 of the following characters in any order. The test performed is the and of the associated tests and file-exists?.

X File-is-executable?
W File-is-writable?
R File-is-readable?

execl command arg0 ...
execlp command arg0 ...
   Transfers control to program command called with arguments arg0 ... For execl, command must be an exact pathname of an executable file. execlp searches for command in the list of directories specified by the environment variable PATH. The convention is that arg0 is the same name as command.
If successful, this procedure does not return. Otherwise an error message is printed and the integer \texttt{errno} is returned.

\textbf{execv} \texttt{command arglist} \quad \textbf{[Function]}
\textbf{execvp} \texttt{command arglist} \quad \textbf{[Function]}

Like \texttt{execl} and \texttt{execlp} except that the set of arguments to \texttt{command} is \texttt{arglist}.

\textbf{putenv} \texttt{string} \quad \textbf{[Function]}

Adds or removes definitions from the \textit{environment}. If the \texttt{string} is of the form \texttt{`NAME=VALUE'}, the definition is added to the environment. Otherwise, the \texttt{string} is interpreted as the name of an environment variable, and any definition for this variable in the environment is removed.

Names of environment variables are case-sensitive and must not contain the character '='. System-defined environment variables are invariably uppercase.

\texttt{Putenv} is used to set up the environment before calls to \texttt{execl}, \texttt{execlp}, \texttt{execv}, \texttt{execvp}, \texttt{system}, or \texttt{open-pipe} (see \textbf{Section 5.7 [Posix Extensions]}, page 77).

To access environment variables, use \texttt{getenv} (see \textbf{Section “System Interface” in SLIB}).

\section*{5.7 Posix Extensions}

If \texttt{`posix} is provided (by linking in \texttt{`posix.o'}), the following functions are defined:

\textbf{open-pipe} \texttt{string modes} \quad \textbf{[Function]}

If the \texttt{string} \texttt{modes} contains an \texttt{R}, returns an input port capable of delivering characters from the standard output of the system command \texttt{string}. Otherwise, returns an output port capable of receiving characters which become the standard input of the system command \texttt{string}. If a pipe cannot be created \texttt{#f} is returned.

\textbf{open-input-pipe} \texttt{string} \quad \textbf{[Function]}

Returns an input port capable of delivering characters from the standard output of the system command \texttt{string}. If a pipe cannot be created \texttt{#f} is returned.

\textbf{open-output-pipe} \texttt{string} \quad \textbf{[Function]}

Returns an output port capable of receiving characters which become the standard input of the system command \texttt{string}. If a pipe cannot be created \texttt{#f} is returned.

\textbf{broken-pipe} \texttt{port} \quad \textbf{[Function]}

If this function is defined at top level, it will be called when an output pipe is closed from the other side (this is the condition under which a SIGPIPE is sent). The already closed \texttt{port} will be passed so that any necessary cleanup may be done. An error is not signaled when output to a pipe fails in this way, but any further output to the closed pipe will cause an error to be signaled.

\textbf{close-port} \texttt{pipe} \quad \textbf{[Function]}

Closes the \texttt{pipe}, rendering it incapable of delivering or accepting characters. This routine has no effect if the pipe has already been closed. The value returned is unspecified.
pipe  
Returns (\texttt{cons \texttt{rd \texttt{wd}}}) where \texttt{rd} and \texttt{wd} are the read and write (port) ends of a \texttt{pipe} respectively.

fork  
Creates a copy of the process calling \texttt{fork}. Both processes return from \texttt{fork}, but the calling (parent) process’s \texttt{fork} returns the child process’s ID whereas the child process’s \texttt{fork} returns 0.

For a discussion of IDs See Section “Process Persona” in \textit{libc}.

getppid  
Returns the process ID of the parent of the current process. For a process’s own ID See Section 5.6 [I/O-Extensions], page 73.

getuid  
Returns the real user ID of this process.

getgid  
Returns the real group ID of this process.

getegid  
Returns the effective group ID of this process.

geteuid  
Returns the effective user ID of this process.

setuid \textit{id}  
Sets the real user ID of this process to \textit{id}. Returns \texttt{#t} if successful, \texttt{#f} if not.

setgid \textit{id}  
Sets the real group ID of this process to \textit{id}. Returns \texttt{#t} if successful, \texttt{#f} if not.

setegid \textit{id}  
Sets the effective group ID of this process to \textit{id}. Returns \texttt{#t} if successful, \texttt{#f} if not.

seteuid \textit{id}  
Sets the effective user ID of this process to \textit{id}. Returns \texttt{#t} if successful, \texttt{#f} if not.

kill \textit{pid \textit{sig}}  
The \texttt{kill} function sends the signal \textit{signum} to the process or process group specified by \textit{pid}. Besides the signals listed in Section “Standard Signals” in \textit{GNU C Library}, \textit{signum} can also have a value of zero to check the validity of the \textit{pid}.

The \textit{pid} specifies the process or process group to receive the signal:

\begin{itemize}
  \item \texttt{> 0} The process whose identifier is \textit{pid}.
  \item \texttt{0} All processes in the same process group as the sender. The sender itself does not receive the signal.
\end{itemize}
If the process is privileged, send the signal to all processes except for some special system processes. Otherwise, send the signal to all processes with the same effective user ID.

The process group whose identifier is \((\text{abs } \text{pid})\).

A process can send a signal to itself with \((\text{kill (getpid) signum})\). If \text{kill} is used by a process to send a signal to itself, and the signal is not blocked, then \text{kill} delivers at least one signal (which might be some other pending unblocked signal instead of the signal \text{signum}) to that process before it returns.

The return value from \text{kill} is zero if the signal can be sent successfully. Otherwise, no signal is sent, and a value of \(-1\) is returned. If \text{pid} specifies sending a signal to several processes, \text{kill} succeeds if it can send the signal to at least one of them. There’s no way you can tell which of the processes got the signal or whether all of them did.

\text{waitpid \ pid options} [\text{Function}]

The \text{waitpid} function suspends execution of the current process until a child as specified by the \text{pid} argument has exited, or until a signal is delivered whose action is to terminate the current process or to call a signal handling function. If a child as requested by \text{pid} has already exited by the time of the call (a so-called \text{zombie} process), the function returns immediately. Any system resources used by the child are freed.

The value of \text{pid} can be:

\(-1\) which means to wait for any child process whose process group ID is equal to the absolute value of \text{pid}.

\(-1\) which means to wait for any child process; this is the same behaviour which \text{wait} exhibits.

\(0\) which means to wait for any child process whose process group ID is equal to that of the calling process.

\(>0\) which means to wait for the child whose process ID is equal to the value of \text{pid}.

The value of \text{options} is one of the following:

0. Nothing special.
1. (\text{WNOHANG}) which means to return immediately if no child is there to be waited for.
2. (\text{WUNTRACED}) which means to also return for children which are stopped, and whose status has not been reported.
3. Which means both of the above.

The return value normally is the exit status of the child process, including the exit value along with flags indicating whether a coredump was generated or the child terminated as a result of a signal. If the \text{WNOHANG} option was specified and no child process is waiting to be noticed, the value is zero. A value of \#f is returned in case of error and \text{errno} is set. For information about the \text{errno} codes see Section “Process Completion” in \text{libc}.
### uname

You can use the `uname` procedure to find out some information about the type of computer your program is running on.

Returns a vector of strings. These strings are:

0. The name of the operating system in use.
1. The network name of this particular computer.
2. The current release level of the operating system implementation.
3. The current version level within the release of the operating system.
4. Description of the type of hardware that is in use.

Some examples are "i386-ANYTHING", "m68k-hp", "sparc-sun", "m68k-sun", "m68k-sony" and "mips-dec".

### getpw

#### getpw name

Returns a vector of information for the entry for `NAME`, `UID`, or the next entry if no argument is given. The information is:

0. The user's login name.
1. The encrypted password string.
2. The user ID number.
3. The user's default group ID number.
4. A string typically containing the user's real name, and possibly other information such as a phone number.
5. The user's home directory, initial working directory, or `#f`, in which case the interpretation is system-dependent.
6. The user's default shell, the initial program run when the user logs in, or `#f`, indicating that the system default should be used.

#### getpw uid

#### getpw

### setpwent

#### setpwent #t

Rewinds the pw entry table back to the begining.

#### setpwent #f

#### setpwent

Closes the pw table.

### getgr

#### getgr name

#### getgr uid

#### getgr

Returns a vector of information for the entry for `NAME`, `UID`, or the next entry if no argument is given. The information is:

0. The name of the group.
1. The encrypted password string.
2. The group ID number.
3. A list of (string) names of users in the group.
**setgrent #t**  
Rewinds the group entry table back to the begining.

**setgrent #f**

**setgrent**  
Closes the group table.

**getgroups**  
Returns a vector of all the supplementary group IDs of the process.

**link oldname newname**  
The `link` function makes a new link to the existing file named by `oldname`, under the new name `newname`.  
`link` returns a value of `#t` if it is successful and `#f` on failure.

**chown filename owner group**  
The `chown` function changes the owner of the file `filename` to `owner`, and its group owner to `group`.  
`chown` returns a value of `#t` if it is successful and `#f` on failure.

**ttyname port**  
If `port` is associated with a terminal device, returns a string containing the filename of terminal device; otherwise `#f`.

### 5.8 Unix Extensions

If ‘unix’ is provided (by linking in ‘unix.o’), the following functions are defined:

These privileged and symbolic link functions are not in Posix:

**symlink oldname newname**  
The `symlink` function makes a symbolic link to `oldname` named `newname`.  
`symlink` returns a value of `#t` if it is successful and `#f` on failure.

**readlink filename**  
Returns the value of the symbolic link `filename` or `#f` for failure.

**lstat filename**  
The `lstat` function is like `stat`, except that it does not follow symbolic links. If `filename` is the name of a symbolic link, `lstat` returns information about the link itself; otherwise, `lstat` works like `stat`. See Section 5.6 [I/O-Extensions], page 73.

**nice increment**  
Increment the priority of the current process by `increment`. `chown` returns a value of `#t` if it is successful and `#f` on failure.

**acct filename**  
When called with the name of an existing file as argument, accounting is turned on, records for each terminating process are appended to `filename` as it terminates. An argument of `#f` causes accounting to be turned off.  
`acct` returns a value of `#t` if it is successful and `#f` on failure.
mknod filename mode dev

The mknod function makes a special file with name filename and modes mode for
device number dev.

mknod returns a value of #t if it is successful and #f on failure.

sync

sync first commits inodes to buffers, and then buffers to disk. sync() only schedules
the writes, so it may return before the actual writing is done. The value returned is
unspecified.

5.9 Sequence Comparison

(require 'diff)

A blazing fast implementation of the sequence-comparison module in SLIB, see See Section
"Sequence Comparison" in SLIB.

5.10 Regular Expression Pattern Matching

These functions are defined in 'rgx.c' using a POSIX or GNU regex library. If your computer does not support regex, a package is available via ftp from
'ftp.gnu.org:/pub/gnu/regex-0.12.tar.gz'. For a description of regular expressions, See Section "syntax" in "regex" regular expression matching library.

regcomp pattern [flags]

Compile a regular expression. Return a compiled regular expression, or an integer
error code suitable as an argument to regerror.

flags in regcomp is a string of option letters used to control the compilation of the
regular expression. The letters may consist of:

‘n’ newlines won’t be matched by . or hat lists; ( [^...] )
‘i’ ignore case.
only when compiled with _GNU_SOURCE:

‘0’ allows dot to match a null character.
‘f’ enable GNU fastmaps.

regerror errno

Returns a string describing the integer errno returned when regcomp fails.

regexec re string

Returns #f or a vector of integers. These integers are in doublets. The first of each
doublet is the index of string of the start of the matching expression or sub-expression
(delimited by parentheses in the pattern). The last of each doublet is index of string
of the end of that expression. #f is returned if the string does not match.

regmatch? re string

Returns #t if the pattern such that regexp = (regcomp pattern) matches string as a
POSIX extended regular expressions. Returns #f otherwise.
Regsearch searches for the pattern within the string.

Regmatch anchors the pattern and begins matching it against string.

Regsearch returns the character position where re starts, or #f if not found.

Regmatch returns the number of characters matched, #f if not matched.

Regsearchv and regmatchv return the match vector is returned if re is found, #f otherwise.

re may be either:
1. a compiled regular expression returned by regcomp;
2. a string representing a regular expression;
3. a list of a string and a set of option letters.

string The string to be operated upon.

start The character position at which to begin the search or match. If absent, the default is zero.

Compiled GNU_SOURCE and using GNU libregex only

When searching, if start is negative, the absolute value of start will be used as the start location and reverse searching will be performed.

len The search is allowed to examine only the first len characters of string. If absent, the entire string may be examined.

string-split re string [Function]
string-splitv re string [Function]

String-split splits a string into substrings that are separated by re, returning a vector of substrings.

String-splitv returns a vector of string positions that indicate where the substrings are located.

string-edit re edit-spec string [count] [Function]

Returns the edited string.

edit-spec Is a string used to replace occurrences of re. Backquoted integers in the range of 1-9 may be used to insert subexpressions in re, as in sed.

count The number of substitutions for string-edit to perform. If #t, all occurrences of re will be replaced. The default is to perform one substitution.


5.11 Line Editing

(require 'edit-line)

These procedures provide input line editing and recall. These functions are defined in 'edline.c' and 'Iedline.scm' using the editline or GNU readline (see Section “Overview ” in GNU Readline Library) libraries available from:

- ftp.sys.toronto.edu:/pub/rc/editline.shar
- ftp.gnu.org:/pub/gnu/readline-2.0.tar.gz

When edit-line package is initialized, if the current input port is the default input port and the environment variable EMACS is not defined, line-editing mode will be entered.

default-input-port [Function]
Returns the initial current-input-port SCM was invoked with (stdin).

default-output-port [Function]
Returns the initial current-output-port SCM was invoked with (stdout).

make-edited-line-port [Function]
Returns an input/output port that allows command line editing and retrieval of history.

line-editing [Function]
Returns the current edited line port or #f.

line-editing bool [Function]
If bool is false, exits line-editing mode and returns the previous value of (line-editing). If bool is true, sets the current input and output ports to an edited line port and returns the previous value of (line-editing).

5.12 Curses

These functions are defined in ‘crs.c’ using the curses library. Unless otherwise noted these routines return #t for successful completion and #f for failure.

initscr [Function]
Returns a port for a full screen window. This routine must be called to initialize curses.

endwin [Function]
A program should call endwin before exiting or escaping from curses mode temporarily, to do a system call, for example. This routine will restore termio modes, move the cursor to the lower left corner of the screen and reset the terminal into the proper non-visual mode. To resume after a temporary escape, call Section 5.12.3 [Window Manipulation], page 86.

5.12.1 Output Options Setting

These routines set options within curses that deal with output. All options are initially #f, unless otherwise stated. It is not necessary to turn these options off before calling endwin.
clearok win bf

If enabled (bf is #t), the next call to force-output or refresh with win will clear the screen completely and redraw the entire screen from scratch. This is useful when the contents of the screen are uncertain, or in some cases for a more pleasing visual effect.

idlok win bf

If enabled (bf is #t), curses will consider using the hardware “insert/delete-line” feature of terminals so equipped. If disabled (bf is #f), curses will very seldom use this feature. The “insert/delete-character” feature is always considered. This option should be enabled only if your application needs “insert/delete-line”, for example, for a screen editor. It is disabled by default because “insert/delete-line” tends to be visually annoying when used in applications where it is not really needed. If “insert/delete-line” cannot be used, curses will redraw the changed portions of all lines.

leaveok win bf

Normally, the hardware cursor is left at the location of the window cursor being refreshed. This option allows the cursor to be left wherever the update happens to leave it. It is useful for applications where the cursor is not used, since it reduces the need for cursor motions. If possible, the cursor is made invisible when this option is enabled.

scrolllok win bf

This option controls what happens when the cursor of window win is moved off the edge of the window or scrolling region, either from a newline on the bottom line, or typing the last character of the last line. If disabled (bf is #f), the cursor is left on the bottom line at the location where the offending character was entered. If enabled (bf is #t), force-output is called on the window win, and then the physical terminal and window win are scrolled up one line.

Note in order to get the physical scrolling effect on the terminal, it is also necessary to call idlok.

nodelay win bf

This option causes wgetch to be a non-blocking call. If no input is ready, wgetch will return an eof-object. If disabled, wgetch will hang until a key is pressed.

5.12.2 Terminal Mode Setting

These routines set options within curses that deal with input. The options involve using ioctl(2) and therefore interact with curses routines. It is not necessary to turn these options off before calling endwin. The routines in this section all return an unspecified value.

cbreak
nocbreak

These two routines put the terminal into and out of CBREAK mode, respectively. In CBREAK mode, characters typed by the user are immediately available to the program and erase/kill character processing is not performed. When in NOCBREAK mode, the
tty driver will buffer characters typed until a LFD or RET is typed. Interrupt and flowcontrol characters are unaffected by this mode. Initially the terminal may or may not be in CBREAK mode, as it is inherited, therefore, a program should call cbreak or nocbreak explicitly. Most interactive programs using curses will set CBREAK mode.

*Note* cbreak overrides raw. For a discussion of how these routines interact with echo and noecho See Section 5.12.5 [Input], page 89.

**raw** [Function]
**noraw** [Function]

The terminal is placed into or out of RAW mode. RAW mode is similar to CBREAK mode, in that characters typed are immediately passed through to the user program. The differences are that in RAW mode, the interrupt, quit, suspend, and flow control characters are passed through uninterpreted, instead of generating a signal. RAW mode also causes 8-bit input and output. The behavior of the BREAK key depends on other bits in the terminal driver that are not set by curses.

**echo** [Function]
**noecho** [Function]

These routines control whether characters typed by the user are echoed by read-char as they are typed. Echoing by the tty driver is always disabled, but initially read-char is in ECHO mode, so characters typed are echoed. Authors of most interactive programs prefer to do their own echoing in a controlled area of the screen, or not to echo at all, so they disable echoing by calling noecho. For a discussion of how these routines interact with echo and noecho See Section 5.12.5 [Input], page 89.

**nl** [Function]
**nonl** [Function]

These routines control whether LFD is translated into RET and LFD on output, and whether RET is translated into LFD on input. Initially, the translations do occur. By disabling these translations using nonl, curses is able to make better use of the linefeed capability, resulting in faster cursor motion.

**resettty** [Function]
**savetty** [Function]

These routines save and restore the state of the terminal modes. savetty saves the current state of the terminal in a buffer and resettty restores the state to what it was at the last call to savetty.

### 5.12.3 Window Manipulation

**newwin** nlines ncols begy begx

Create and return a new window with the given number of lines (or rows), nlines, and columns, ncols. The upper left corner of the window is at line begy, column begx. If either nlines or ncols is 0, they will be set to the value of LINES-begy and COLS-begx. A new full-screen window is created by calling newwin(0,0,0,0).

**subwin** orig nlines ncols begy begx

Create and return a pointer to a new window with the given number of lines (or rows), nlines, and columns, ncols. The window is at position (begy, begx) on the screen.
This position is relative to the screen, and not to the window orig. The window is made in the middle of the window orig, so that changes made to one window will affect both windows. When using this routine, often it will be necessary to call touchwin or touchline on orig before calling force-output.

**close-port win**

Delete the window win, freeing up all memory associated with it. In the case of sub-windows, they should be deleted before the main window win.

**refresh**

**force-output**

These routines are called to write output to the terminal, as most other routines merely manipulate data structures. force-output copies the window win to the physical terminal screen, taking into account what is already there in order to minimize the amount of information that’s sent to the terminal (called optimization). Unless leaveok has been enabled, the physical cursor of the terminal is left at the location of window win’s cursor. With refresh, the number of characters output to the terminal is returned.

**mvwin win y x**

Move the window win so that the upper left corner will be at position (y, x). If the move would cause the window win to be off the screen, it is an error and the window win is not moved.

**overlay srcwin dstwin**

**overwrite srcwin dstwin**

These routines overlay srcwin on top of dstwin; that is, all text in srcwin is copied into dstwin. srcwin and dstwin need not be the same size; only text where the two windows overlap is copied. The difference is that overlay is non-destructive (blanks are not copied), while overwrite is destructive.

**touchwin win**

**touchline win start count**

Throw away all optimization information about which parts of the window win have been touched, by pretending that the entire window win has been drawn on. This is sometimes necessary when using overlapping windows, since a change to one window will affect the other window, but the records of which lines have been changed in the other window will not reflect the change. touchline only pretends that count lines have been changed, beginning with line start.

**wmove win y x**

The cursor associated with the window win is moved to line (row) y, column x. This does not move the physical cursor of the terminal until refresh (or force-output) is called. The position specified is relative to the upper left corner of the window win, which is (0, 0).

### 5.12.4 Output

These routines are used to draw text on windows
display ch win
display str win
wadd win ch
wadd win str

The character ch or characters in str are put into the window win at the current
cursor position of the window and the position of win’s cursor is advanced. At the
right margin, an automatic newline is performed. At the bottom of the scrolling
region, if scrollok is enabled, the scrolling region will be scrolled up one line.

If ch is a TAB, LFD, or backspace, the cursor will be moved appropriately within
the window win. A LFD also does a wclrtobot before moving. TAB characters are
considered to be at every eighth column. If ch is another control character, it will be
drawn in the C-x notation. (Calling winch after adding a control character will not
return the control character, but instead will return the representation of the control
character.)

Video attributes can be combined with a character by or-ing them into the para-
meter. This will result in these attributes also being set. The intent here is that text,
including attributes, can be copied from one place to another using inch and display.
See standout, below.

Note For wadd ch can be an integer and will insert the character of the corresponding
value.

werase win
This routine copies blanks to every position in the window win.

wclear win
This routine is like werase, but it also calls Section 5.12.1 [Output Options Setting],
page 84, arranging that the screen will be cleared completely on the next call to
refresh or force-output for window win, and repainted from scratch.

wclrtobot win
All lines below the cursor in window win are erased. Also, the current line to the
right of the cursor, inclusive, is erased.

wclrtoeol win
The current line to the right of the cursor, inclusive, is erased.

wdelch win
The character under the cursor in the window win is deleted. All characters to the
right on the same line are moved to the left one position and the last character on
the line is filled with a blank. The cursor position does not change. This does not
imply use of the hardware “delete-character” feature.

wdeleteln win
The line under the cursor in the window win is deleted. All lines below the current
line are moved up one line. The bottom line win is cleared. The cursor position does
not change. This does not imply use of the hardware “deleteline” feature.
**winsch** win ch  
The character ch is inserted before the character under the cursor. All characters  
to the right are moved one SPC to the right, possibly losing the rightmost character  
of the line. The cursor position does not change. This does not imply use of the  
hardware “insertcharacter” feature.

**winsertln** win  
A blank line is inserted above the current line and the bottom line is lost. This does  
not imply use of the hardware “insert-line” feature.

**scroll** win  
The window win is scrolled up one line. This involves moving the lines in win’s data  
structure. As an optimization, if win is stdscr and the scrolling region is the entire  
window, the physical screen will be scrolled at the same time.

### 5.12.5 Input

**read-char** win  
A character is read from the terminal associated with the window win. Depending  
on the setting of cbreak, this will be after one character (CBREAK mode), or after the  
first newline (NOCBREAK mode). Unless noecho has been set, the character will also  
be echoed into win.

When using read-char, do not set both NOCBREAK mode (nocbreak) and ECHO mode  
(echo) at the same time. Depending on the state of the terminal driver when each  
character is typed, the program may produce undesirable results.

**winch** win  
The character, of type chtype, at the current position in window win is returned.  
If any attributes are set for that position, their values will be OR’ed into the value  
returned.

**getyx** win  
A list of the y and x coordinates of the cursor position of the window win is returned.

### 5.12.6 Curses Miscellany

**wstandout** win  
**wstandend** win  
These functions set the current attributes of the window win. The current attributes  
of win are applied to all characters that are written into it. Attributes are a property  
of the character, and move with the character through any scrolling and insert/delete  
line/character operations. To the extent possible on the particular terminal, they will  
be displayed as the graphic rendition of characters put on the screen.

wstandout sets the current attributes of the window win to be visibly different from  
other text. wstandend turns off the attributes.
box win vertch horch  
A box is drawn around the edge of the window win. vertch and horch are the characters the box is to be drawn with. If vertch and horch are 0, then appropriate default characters, ACS_VLINE and ACS_HLINE, will be used.

Note vertch and horch can be an integers and will insert the character (with attributes) of the corresponding values.

unctrl c  
This macro expands to a character string which is a printable representation of the character c. Control characters are displayed in the C-x notation. Printing characters are displayed as is.

5.13 Sockets

These procedures (defined in ’socket.c’) provide a Scheme interface to most of the C socket library. For more information on sockets, See Section “Sockets” in The GNU C Library Reference Manual.

5.13.1 Host and Other Inquiries

af_inet  
af_unix  
Integer family codes for Internet and Unix sockets, respectively.

gethost host-spec  
gethost  
Returns a vector of information for the entry for HOST-SPEC or the next entry if HOST-SPEC isn’t given. The information is:

0. host name string
1. list of host aliases strings
2. integer address type (AF_INET)
3. integer size of address entries (in bytes)
4. list of integer addresses

sethostent stay-open  
sethostent  
Rewinds the host entry table back to the begining if given an argument. If the argument stay-open is #f queries will be be done using UDP datagrams. Otherwise, a connected TCP socket will be used. When called without an argument, the host table is closed.

getnet name-or-number  
getnet  
Returns a vector of information for the entry for name-or-number or the next entry if an argument isn’t given. The information is:

0. official network name string
1. list of network aliases strings
2. integer network address type (AF_INET)
3. integer network number

**setnetent** stay-open

**setnetent**
Rewinds the network entry table back to the beginning if given an argument. If the argument **stay-open** is #f the table will be closed between calls to getnet. Otherwise, the table stays open. When called without an argument, the network table is closed.

**getproto** name-or-number

**getproto**
Returns a vector of information for the entry for name-or-number or the next entry if an argument isn’t given. The information is:
1. official protocol name string
2. list of protocol aliases strings
3. integer protocol number

**setprotoent** stay-open

**setprotoent**
Rewinds the protocol entry table back to the beginning if given an argument. If the argument **stay-open** is #f the table will be closed between calls to getproto. Otherwise, the table stays open. When called without an argument, the protocol table is closed.

**getserv** name-or-port-number protocol

**getserv**
Returns a vector of information for the entry for name-or-port-number and protocol or the next entry if arguments aren’t given. The information is:
0. official service name string
1. list of service aliases strings
2. integer port number
3. protocol

**setservent** stay-open

**setservent**
Rewinds the service entry table back to the beginning if given an argument. If the argument **stay-open** is #f the table will be closed between calls to getserv. Otherwise, the table stays open. When called without an argument, the service table is closed.

### 5.13.2 Internet Addresses and Socket Names

**inet:string->address** string
Returns the host address number (integer) for host string or #f if not found.

**inet:address->string** address
Converts an internet (integer) address to a string in numbers and dots notation.

**inet:network** address
Returns the network number (integer) specified from address or #f if not found.
inet:local-network-address address

Returns the integer for the address of address within its local network or #f if not found.

inet:make-address network local-address

Returns the Internet address of local-address in network.

The type socket-name is used for inquiries about open sockets in the following procedures:

getsockname socket

Returns the socket-name of socket. Returns #f if unsuccessful or socket is closed.

getpeername socket

Returns the socket-name of the socket connected to socket. Returns #f if unsuccessful or socket is closed.

socket-name:family socket-name

Returns the integer code for the family of socket-name.

socket-name:port-number socket-name

Returns the integer port number of socket-name.

socket-name:address socket-name

Returns the integer Internet address for socket-name.

5.13.3 Socket

When a port is returned from one of these calls it is unbuffered. This allows both reading and writing to the same port to work. If you want buffered ports you can (assuming sock-port is a socket i/o port):

```
(require 'i/o-extensions)
(define i-port (duplicate-port sock-port "r"))
(define o-port (duplicate-port sock-port "w"))
```

make-stream-socket family

Returns a SOCK_STREAM socket of type family using protocol. If family has the value AF_INET, SO_REUSEADDR will be set. The integer argument protocol corresponds to the integer protocol numbers returned (as vector elements) from (getproto). If the protocol argument is not supplied, the default (0) for the specified family is used. SCM sockets look like ports opened for neither reading nor writing.

make-stream-socketpair family

make-stream-socketpair family protocol

Returns a pair (cons) of connected SOCK_STREAM (socket) ports of type family using protocol. Many systems support only socketpairs of the af-unix family. The integer argument protocol corresponds to the integer protocol numbers returned (as vector elements) from (getproto). If the protocol argument is not supplied, the default (0) for the specified family is used.
socket:shutdown socket how
    Makes socket no longer respond to some or all operations depending on the integer argument how:
    0. Further input is disallowed.
    1. Further output is disallowed.
    2. Further input or output is disallowed.

Socket:shutdown returns socket if successful, #f if not.

socket:connect inet-socket host-number port-number
socket:connect unix-socket pathname
    Returns socket (changed to a read/write port) connected to the Internet socket on host
    host-number, port port-number or the Unix socket specified by pathname. Returns
    #f if not successful.

socket:bind inet-socket port-number
socket:bind unix-socket pathname
    Returns inet-socket bound to the integer port-number or the unix-socket bound to
    new socket in the file system at location pathname. Returns #f if not successful.
    Binding a unix-socket creates a socket in the file system that must be deleted by the
    caller when it is no longer needed (using delete-file).

socket:listen socket backlog
    The bound (see Section 5.13.3 [Socket], page 92) socket is readied to accept connec-
    tions. The positive integer backlog specifies how many pending connections will be
    allowed before further connection requests are refused. Returns socket (changed to a
    read-only port) if successful, #f if not.

char-ready? listen-socket
    The input port returned by a successful call to socket:listen can be polled for
    connections by char-ready? (see Section 4.6 [Files and Ports], page 49). This avoids
    blocking on connections by socket:accept.

socket:accept socket
    Accepts a connection on a bound, listening socket. Returns an input/output port for
    the connection.

The following example is not too complicated, yet shows the use of sockets for multiple
connections without input blocking.

;;; Scheme chat server

;;; This program implements a simple ‘chat’ server which accepts
;;; connections from multiple clients, and sends to all clients any
;;; characters received from any client.

;;; To connect to chat ‘telnet localhost 8001’

=require 'socket}
(require 'i/o-extensions)

(let ((listener-socket (socket:bind (make-stream-socket af_inet) 8001))
  (connections '()))
  (socket:listen listener-socket 5)
  (do () (#f)
    (let ((actives (or (apply wait-for-input 5 listener-socket connections)
                      ',()))))
      (cond ((null? actives))
        ((memq listener-socket actives)
         (set! actives (cdr (memq listener-socket actives)))
         (let ((con (socket:accept listener-socket)))
           (display "accepting connection from ")
           (display (getpeername con))
           (newline)
           (set! connections (cons con connections))
           (display "connected")
           (newline con))))
      (set! connections
        (let next ((con-list connections))
          (cond ((null? con-list) '())
            (else
              (let ((con (car con-list)))
                (cond ((memq con actives)
                  (let ((c (read-char con)))
                    (cond ((eof-object? c)
                      (display "closing connection from ")
                      (display (getpeername con))
                      (newline)
                      (close-port con)
                      (next (cdr con-list)))
                    (else
                      (for-each (lambda (con)
                          (file-position con 0)
                          (write-char c con)
                          (file-position con 0))
                        connections)
                      (cons con (next (cdr con-list))))))))
              (else (cons con (next (cdr con-list))))))))))

You can use ‘telnet localhost 8001’ to connect to the chat server, or you can use a client
written in scheme:

;;; Scheme chat client

;;; this program connects to socket 8001. It then sends all
;;; characters from current-input-port to the socket and sends all
;;; characters from the socket to current-output-port.
(require 'socket)
(require 'i/o-extensions)

(define con (make-stream-socket af_inet))
(set! con (socket:connect con (inet:string->address "localhost") 8001))

(define (go)
  (define actives (wait-for-input (* 30 60) con (current-input-port)))
  (let ((cs (and actives (memq con actives) (read-char con)))
      (ct (and actives (memq (current-input-port) actives) (read-char))))
    (cond ((or (eof-object? cs) (eof-object? ct)) (close-port con))
      (else (cond (cs (display cs))
                    (ct (file-position con 0)
                        (display ct con))
                    (go))))))

(cond (con (display "Connecting to ")
         (display (getpeername con))
         (newline)
         (go))
      (else (display "Server not listening on port 8001")
             (newline)))

5.14 SCMDB

(require 'mysql)

SCMDB is an add-on for SCM that ports the MySQL C-library to SCM.
It is available from: http://www.dedecker.net/jessie/scmdb/

5.15 Xlibscm

(require 'Xlib)

See Section “SCM Language X Interface ” in Xlibscm for the SCM interface to the X Window System.

5.16 Hobbit

(require 'hobbit)
(require 'compile)

See Section “SCM Compiler” in hobbit for a small optimizing scheme-to-C compiler for use with the SCM interpreter.
6 The Implementation

6.1 Data Types

In the descriptions below it is assumed that long int are 32 bits in length. Actually, SCM is written to work with any long int size larger than 31 bits. With some modification, SCM could work with word sizes as small as 24 bits.

All SCM objects are represented by type SCM. Type SCM come in 2 basic flavors, Immediates and Cells:

6.1.1 Immediates

An immediate is a data type contained in type SCM (long int). The type codes distinguishing immediate types from each other vary in length, but reside in the low order bits.

IMP x                     [Macro]
NIMP x                    [Macro]

Return non-zero if the SCM object x is an immediate or non-immediate type, respectively.

INUM x                     [Immediate]
immediate 30 bit signed integer. An INUM is flagged by a 1 in the second to low order bit position. The high order 30 bits are used for the integer’s value.

INUMP x                    [Macro]
NINUMP x                   [Macro]

Return non-zero if the SCM x is an immediate integer or not an immediate integer, respectively.

INUM x                     [Macro]
Returns the C long integer corresponding to SCM x.

MAKINUM x                  [Macro]
Returns the SCM inum corresponding to C long integer x.

INUMO                      [Immediate Constant]
is equivalent to MAKINUM(0).

Computations on INUMs are performed by converting the arguments to C integers (by a shift), operating on the integers, and converting the result to an inum. The result is checked for overflow by converting back to integer and checking the reverse operation.

The shifts used for conversion need to be signed shifts. If the C implementation does not support signed right shift this fact is detected in a #if statement in ‘scmfig.h’ and a signed right shift, SRS, is constructed in terms of unsigned right shift.

ichr                        [Immediate]
characters.
ICHRP \( x \)  
Return non-zero if the SCM object \( x \) is a character.

ICH\( \text{CHR} \) \( x \)  
Returns corresponding unsigned char.

MAKICH\( \text{CHR} \) \( x \)  
Given char \( x \), returns SCM character.

\textbf{iflags}  
These are frequently used immediate constants.

SCM BOOL\_T  
\#t  
[Immediate Constant]

SCM BOOL\_F  
\#f  
[Immediate Constant]

SCM EOL  
\;()\; . If SICP is \#defined, EOL is \#defined to be identical with BOOL\_F. In this case, both print as \#f.

SCM EOF\_VAL  
end of file token, \#<eof>.

SCM UNDEFINED  
\#<undefined> used for variables which have not been defined and absent optional arguments.

SCM UNSPECIFIED  
\#<unspecified> is returned for those procedures whose return values are not specified.

IFLAGP \( n \)  
Returns non-zero if \( n \) is an ispcsym, isym or iflag.

ISYMP \( n \)  
Returns non-zero if \( n \) is an ispcsym or isym.

ISYMNUM \( n \)  
Given ispcsym, isym, or iflag \( n \), returns its index in the C array \text{isymnames[]}.

ISYMCHARS \( n \)  
Given ispcsym, isym, or iflag \( n \), returns its char * representation (from \text{isymnames[]}).

MAKSPCSYM \( n \)  
Returns SCM ispcsym \( n \).

MAKISYM \( n \)  
Returns SCM isym \( n \).
MAKIFLAG \( n \)  
Returns SCM iflag \( n \).

**isymnames**  
An array of strings containing the external representations of all the ispcsym, isym, and iflag immediates. Defined in `repl.c`.

**NUM_ISPCSYM**  
**NUM_ISYMS**  
The number of ispcsyms and ispcsyms+isyms, respectively. Defined in `scm.h`.

**isym**  
and, begin, case, cond, define, do, if, lambda, let, let*, letrec, or, quote, set!, 
#f, #t, #<undefined>, #<eof>, (), and #<unspecified>.

**ispcsym**  
special symbols: syntax-checked versions of first 14 isyms

**iloc**  
indexes to a variable’s location in environment

**gloc**  
pointer to a symbol’s value cell

CELLPTR  
pointer to a cell (not really an immediate type, but here for completeness). Since 
cells are always 8 byte aligned, a pointer to a cell has the low order 3 bits 0. 
There is one exception to this rule, *CAR Immediates*, described next.

A *CAR Immediate* is an Immediate point which can only occur in the CARs of evaluated 
code (as a result of ceval’s memoization process).

### 6.1.2 Cells

*Cells* represent all SCM objects other than immediates. A cell has a *CAR* and a *CDR*. Low-
order bits in *CAR* identify the type of object. The rest of CAR and CDR hold object data. The 
number after \( tc \) specifies how many bits are in the type code. For instance, \( tc7 \) indicates 
that the type code is 7 bits.

**NEWCELL** \( x \)  
Allocates a new cell and stores a pointer to it in SCM local variable \( x \). 
Care needs to be taken that stores into the new cell pointed to by \( x \) do not create an 
inconsistent object. See Section 6.2.6 [Signals], page 115.

All of the C macros decribed in this section assume that their argument is of type SCM and 
points to a cell (CELLPTR).

**CAR** \( x \)  
**CDR** \( x \)  
Returns the *car* and *cdr* of cell \( x \), respectively.
TYP3 x [Macro]
TYP7 x [Macro]
TYP16 x [Macro]

Returns the 3, 7, and 16 bit type code of a cell.

tc3_cons [Cell]
scheme cons-cell returned by (cons arg1 arg2).

CONSP x [Macro]
NCONSP x [Macro]

Returns non-zero if x is a tc3_cons or isn’t, respectively.

tc3_closure [Cell]
applicable object returned by (lambda (args) ...). tc3_closures have a pointer to
the body of the procedure in the CAR and a pointer to the environment in the CDR.
Bits 1 and 2 (zero-based) in the CDR indicate a lower bound on the number of required
arguments to the closure, which is used to avoid allocating rest argument lists in the
environment cache. This encoding precludes an immediate value for the CDR: In the
case of an empty environment all bits above 2 in the CDR are zero.

CLOSEREP x [Macro]

Returns non-zero if x is a tc3_closure.

CODE x [Macro]
ENV x [Macro]

Returns the code body or environment of closure x, respectively.

ARGC x [Macro]

Returns the a lower bound on the number of required arguments to closure x,
it cannot exceed 3.

6.1.3 Header Cells

Headers are Cells whose CDRs point elsewhere in memory, such as to memory allocated by
malloc.

spare [Header]

spare tc7 type code

tc7_vector [Header]
scheme vector.

VECTORP x [Macro]
NVVECTORP x [Macro]

Returns non-zero if x is a tc7_vector or if not, respectively.

VELTS x [Macro]
LENGTH x [Macro]

Returns the C array of SCMs holding the elements of vector x or its length,
respectively.
tc7_sssymbol
static scheme symbol (part of initial system)

[Header]
tc7_mssymbol
malloced scheme symbol (can be GCed)

[Header]
SYMBOLP x
Returns non-zero if x is a tc7_sssymbol or tc7_mssymbol.

[Macro]
CHARS x
UCHARS x
LENGTH x
Returns the C array of chars or as unsigned chars holding the elements of
symbol x or its length, respectively.

[Macro]
tc7_string
scheme string

[Header]
STRINGP x
NSTRINGP x
Returns non-zero if x is a tc7_string or isn’t, respectively.

[Macro]
CHARS x
UCHARS x
LENGTH x
Returns the C array of chars or as unsigned chars holding the elements of
string x or its length, respectively.

[Macro]
tc7_Vbool
uniform vector of booleans (bit-vector)

[Header]
tc7_VfixZ32
uniform vector of integers

[Header]
tc7_VfixN32
uniform vector of non-negative integers

[Header]
tc7_VfixN16
uniform vector of non-negative short integers

[Header]
tc7_VfixZ16
uniform vector of short integers

[Header]
tc7_VfixN8
uniform vector of non-negative bytes

[Header]
tc7_VfixZ8
uniform vector of signed bytes

[Header]
tc7_VfloR32
uniform vector of short inexact real numbers
**Chapter 6: The Implementation**

**Header**

- **tc7_VfloR64**
  - uniform vector of double precision inexact real numbers

- **tc7_VfloC64**
  - uniform vector of double precision inexact complex numbers

- **tc7_contin**
  - applicable object produced by call-with-current-continuation

- **tc7_specfun**
  - subr that is treated specially within the evaluator

  *apply* and *call-with-current-continuation* are denoted by these objects. Their behavior as functions is built into the evaluator; they are not directly associated with C functions. This is necessary in order to make them properly tail recursive.

  tc16_cclo is a subtype of tc7_specfun, a cclo is similar to a vector (and is GCed like one), but can be applied as a function:
  1. the cclo itself is consed onto the head of the argument list
  2. the first element of the cclo is applied to that list. Cclo invocation is currently not tail recursive when given 2 or more arguments.

**Function**

- **makcclo**
  - *proc len*
  - makes a closure from the *subr proc* with *len*-1 extra locations for SCM data.
  - Elements of a cclo are referenced using VELTS(cclo)[n] just as for vectors.

**Macro**

- **CCLO_LENGTH** cclo
  - Expands to the length of cclo.

### 6.1.4 Subr Cells

A *Subr* is a header whose CDR points to a C code procedure. Scheme primitive procedures are subsrs. Except for the arithmetic *tc7_cxrs*, the C code procedures will be passed arguments (and return results) of type SCM.

**Subr**

- **tc7_asubr**
  - associative C function of 2 arguments. Examples are +, -, *, /, max, and min.

- **tc7_subr_0**
  - C function of no arguments.

- **tc7_subr_1**
  - C function of one argument.

- **tc7_cxrs**
  - These subsrs are handled specially. If inexact numbers are enabled, the CDR should be a function which takes and returns type double. Conversions are handled in the interpreter.

    floor, ceiling, truncate, round, real-sqrt, real-exp, real-ln, real-sin, real-cos, real-tan, real-asin, real-acos, real-atan, real-sinh, real-cosh, real-tanh, real-asinh, real-acosh, real-atanh, and exact->inexact are defined this way.
If the CDR is 0 (NULL), the name string of the procedure is used to control traversal of its list structure argument.

\texttt{car}, \texttt{cdr}, \texttt{caar}, \texttt{cadr}, \texttt{cddr}, \texttt{caaar}, \texttt{caadr}, \texttt{caaddr}, \texttt{caddr}, \texttt{cddar}, \texttt{cdddr}, \texttt{caaaar}, \texttt{caaaadr}, \texttt{caadaar}, \texttt{caaddr}, \texttt{caaddr}, \texttt{cddaaar}, \texttt{cddadar}, \texttt{cddaddr}, \texttt{cdddar}, \texttt{cdddddar}, \texttt{and cdddddar} are defined this way.

\begin{verbatim}
tc7_subr_3
  C function of 3 arguments.

  tc7_subr_2
  C function of 2 arguments.

  tc7_rpsubr
  transitive relational predicate C function of 2 arguments. The C function should return either \texttt{BOOL_T} or \texttt{BOOL_F}.

  tc7_subr_1o
  C function of one optional argument. If the optional argument is not present, \texttt{UNDEFINED} is passed in its place.

  tc7_subr_2o
  C function of 1 required and 1 optional argument. If the optional argument is not present, \texttt{UNDEFINED} is passed in its place.

  tc7_lsubr_2
  C function of 2 arguments and a list of (rest of) \texttt{SCM} arguments.

  tc7_lsubr
  C function of list of \texttt{SCM} arguments.
\end{verbatim}

\section*{6.1.5 Defining Subrs}

If \texttt{CCLO} is \texttt{defined} when compiling, the compiled closure feature will be enabled. It is automatically enabled if dynamic linking is enabled.

The SCM interpreter directly recognizes subrs taking small numbers of arguments. In order to create subrs taking larger numbers of arguments use:

\begin{verbatim}
make_gsubr name req opt rest fcn
  returns a cclo (compiled closure) object of name \texttt{char *} \texttt{name} which takes \texttt{int \ req} required arguments, \texttt{int \ opt} optional arguments, and a list of rest arguments if \texttt{int \ rest} is 1 (0 for not).

  SCM (*fcn)() is a pointer to a C function to do the work.

  The C function will always be called with \texttt{req + opt + rest} arguments, optional arguments not supplied will be passed \texttt{UNDEFINED}. An error will be signaled if the subr is called with too many or too few arguments. Currently a total of 10 arguments may be specified, but increasing this limit should not be difficult.

  /* A silly example, taking 2 required args,
  1 optional, and a list of rest args */
\end{verbatim}


```c
#include <scm.h>

SCM gsubr_21l(req1,req2,opt,rst)
    SCM req1,req2,opt,rst;
{
    lputs("gsubr-2-1-l: \n req1: ", cur_outp);
    display(req1,cur_outp);
    lputs("\n req2: ", cur_outp);
    display(req2,cur_outp);
    lputs("\n opt: ", cur_outp);
    display(opt,cur_outp);
    lputs("\n rest: ", cur_outp);
    display(rst,cur_outp);
    newline(cur_outp);
    return UNSPECIFIED;
}

void init_gsubr211()
{
    make_gsubr("gsubr-2-1-l", 2, 1, 1, gsubr_21l);
}
```

### 6.1.6 Ptob Cells

A *ptob* is a port object, capable of delivering or accepting characters. See Section “Ports” in *Revised(5) Report on the Algorithmic Language Scheme*. Unlike the types described so far, new varieties of ptobs can be defined dynamically (see Section 6.1.7 [Defining Ptobs], page 104). These are the initial ptobs:

- **tc16_inport**
  - input port.

- **tc16_outport**
  - output port.

- **tc16_ioport**
  - input-output port.

- **tc16_inpipe**
  - input pipe created by `popen()`.

- **tc16_outpipe**
  - output pipe created by `popen()`.

- **tc16_strport**
  - String port created by `cwos()` or `cwis()`.

- **tc16_sfport**
  - Software (virtual) port created by `mksfpt()` (see Section 4.6.4 [Soft Ports], page 52).
PORTP \( x \)

OPPORTP \( x \)

OPINPORTP \( x \)

OPOUTPORTP \( x \)

INPORTP \( x \)

OUTPORTP \( x \)

Returns non-zero if \( x \) is a port, open port, open input-port, open output-port, input-port, or output-port, respectively.

OPENP \( x \)

CLOSEDP \( x \)

Returns non-zero if port \( x \) is open or closed, respectively.

STREAM \( x \)

Returns the FILE * stream for port \( x \).

Ports which are particularly well behaved are called fports. Advanced operations like file-position and reopen-file only work for fports.

FPORTP \( x \)

OPFPORTP \( x \)

OPINFPORTP \( x \)

OPOUTFPORTP \( x \)

Returns non-zero if \( x \) is a port, open port, open input-port, or open output-port, respectively.

6.1.7 Defining Ptobs

ptobs are similar to smobs but define new types of port to which SCM procedures can read or write. The following functions are defined in the ptobfuns:

```c
typedef struct {
    SCM (*mark)(SCM ptr);
    int (*free)(FILE *p);
    int (*print)(SCM exp, SCM port, int writing);
    SCM (*equalp)(SCM, SCM);
    int (*fputc)(int c, FILE *p);
    int (*fputs)(char *s, FILE *p);
    sizet (*fwrite)(char *s, sizet siz, sizet num, FILE *p);
    int (*fflush)(FILE *stream);
    int (*fgetc)(FILE *p);
    int (*fclose)(FILE *p);
} ptobfuns;
```

The .free component to the structure takes a FILE * or other C construct as its argument, unlike .free in a smob, which takes the whole smob cell. Often, .free and .fclose can be the same function. See fptob and pipob in ‘sys.c’ for examples of how to define ptobs. Ptobs that must allocate blocks of memory should use, for example, must_malloc rather than malloc. See Section 6.2.9 [Allocating memory], page 118.
6.1.8 Smob Cells

A smob is a miscellaneous datatype. The type code and GCMARK bit occupy the lower order 16 bits of the CAR half of the cell. The rest of the CAR can be used for sub-type or other information. The CDR contains data of size long and is often a pointer to allocated memory.

Like ptobs, new varieties of smobs can be defined dynamically (see Section 6.1.9 [Defining Smobs], page 106). These are the initial smobs:

\texttt{tc\_free\_cell} \hspace{1cm} \texttt{[smob]}

unused cell on the freelist.

\texttt{tc16\_flo} \hspace{1cm} \texttt{[smob]}

double-precision float.

Inexact number data types are subtypes of type \texttt{tc16\_flo}. If the sub-type is:

0. a single precision float is contained in the CDR.
1. CDR is a pointer to a \texttt{malloc}ed double.
2. CDR is a pointer to a \texttt{malloc}ed pair of doubles.

\texttt{tc\_dbl1r} \hspace{1cm} \texttt{[smob]}

double-precision float.

\texttt{tc\_dbl1c} \hspace{1cm} \texttt{[smob]}

double-precision complex.

\texttt{tc16\_bigpos} \hspace{1cm} \texttt{[smob]}

positive and negative bignums, respectively.

Scm has large precision integers called bignums. They are stored in sign-magnitude form with the sign occurring in the type code of the SMOBs bigpos and bigneg. The magnitude is stored as a \texttt{malloc}ed array of type BIGDIG which must be an unsigned integral type with size smaller than \texttt{long}. BIGRAD is the radix associated with BIGDIG.

\texttt{NUMDIGS\_MAX} (defined in \texttt{scmfig.h}) limits the number of digits of a bignum to 1000. These digits are base BIGRAD, which is typically 65536, giving 4816 decimal digits.

Why only 4800 digits? The simple multiplication algorithm SCM uses is O(n^2); this means the number of processor instructions required to perform a multiplication is some multiple of the product of the number of digits of the two multiplicands.

\[
\text{digits} \times \text{digits} \implies \text{operations}
\]

\[
\begin{array}{c|c}
5 & x \\
50 & 100 \times x \\
500 & 10000 \times x \\
5000 & 100000 \times x \\
\end{array}
\]

To calculate numbers larger than this, FFT multiplication [O(n*\log(n))] and other specialized algorithms are required. You should obtain a package which specializes in number-theoretical calculations:

\texttt{ftp://megrez.math.u-bordeaux.fr/pub/pari/}
tc16_promise [smob]
made by DELAY. See Section “Control features” in Revised(5) Scheme.

tc16_arbiter [smob]
synchronization object. See Section 4.5 [Process Synchronization], page 48.

tc16_macro [smob]
macro expanding function. See Section 4.9.4 [Macro Primitives], page 59.

tc16_array [smob]
multi-dimensional array. See Section 5.4 [Arrays], page 68.

This type implements both conventional arrays (those with arbitrary data as elements see Section 5.4.1 [Conventional Arrays], page 69) and uniform arrays (those with elements of a uniform type see Section 5.4.2 [Uniform Array], page 70).

Conventional Arrays have a pointer to a vector for their CDR. Uniform Arrays have a pointer to a Uniform Vector type (string, Vbool, VfixZ32, VfixN32, VfloR32, VfloR64, or VfloC64) in their CDR.

6.1.9 Defining Smobs

Here is an example of how to add a new type named foo to SCM. The following lines need to be added to your code:

long tc16_foo;

The type code which will be used to identify the new type.

static smobfuns foosmob = {markfoo, freefoo, printfoo, equalpfoo};

smobfuns is a structure composed of 4 functions:

typedef struct {
    SCM (*mark)P((SCM));
    sizet (*free)P((CELLPTR));
    int (*print)P((SCM exp, SCM port, int writing));
    SCM (*equalp)P((SCM, SCM));
} smobfuns;

smob.mark

is a function of one argument of type SCM (the cell to mark) and returns type SCM which will then be marked. If no further objects need to be marked then return an immediate object such as BOOL_F. The smob cell itself will already have been marked. Note This is different from SCM versions prior to 5c5. Only additional data specific to a smob type need be marked by smob.mark.

2 functions are provided:

markcdr(ptr)
returns CDR(ptr).

mark0(ptr)
is a no-op used for smobs containing no additional SCM data. 0 may also be used in this case.
smob.free

is a function of one argument of type CELLPTR (the cell to collected) and returns type size_t which is the number of malloced bytes which were freed. Smob.free should free any malloced storage associated with this object. The function free(ptr) is provided which does not free any storage and returns 0.

smob.print

is 0 or a function of 3 arguments. The first, of type SCM, is the smob object. The second, of type SCM, is the stream on which to write the result. The third, of type int, is 1 if the object should be written, 0 if it should be displayed, and 2 if it should be written for an error report. This function should return non-zero if it printed, and zero otherwise (in which case a hexadecimal number will be printed).

smob.equalp

is 0 or a function of 2 SCM arguments. Both of these arguments will be of type tc16foo. This function should return BOOL_T if the smobs are equal, BOOL_F if they are not. If smob.equalp is 0, equal? will return BOOL_F if they are not eq?.

```
tc16foo = newsmob(&foosmob);
Allocate the new type with the functions from foosmob. This line goes in an init_routine.
```

Promises and macros in ‘eval.c’ and arbiters in ‘repl.c’ provide examples of SMOBs. There are a maximum of 256 SMOBs. Smobs that must allocate blocks of memory should use, for example, must_malloc rather than malloc See Section 6.2.9 [Allocating memory], page 118.

6.1.10 Data Type Representations

IMMEDIATE: B,D,E,F=data bit, C=flag code, P=pointer address bit

<p>| | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>inum</td>
<td>BBBBBBBB</td>
<td>BBBBBBB</td>
<td>BBBBBB</td>
<td>10</td>
</tr>
<tr>
<td>ichr</td>
<td>BBBBBBBB</td>
<td>BBBBBBB</td>
<td>BBBBBB</td>
<td>11110100</td>
</tr>
<tr>
<td>iflag</td>
<td>CCCCCCC</td>
<td>10111010</td>
<td></td>
<td></td>
</tr>
<tr>
<td>isym</td>
<td>CCCCCCC</td>
<td>00111010</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

IMCAR: only in car of evaluated code, cdr has cell’s GC bit

<p>| | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>ispcsym</td>
<td>00000000</td>
<td>00000000</td>
<td>100</td>
<td></td>
<td></td>
</tr>
<tr>
<td>iloc</td>
<td>0DDDDDDDDDDEFFFFFFFFFFFFF11111100</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>pointer</td>
<td>PPPPPPPPPPPPPPPPPPPPPPPPPPP0000</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>gloc</td>
<td>PPPPPPPPPPPPPPPPPPPPPPPPPPP0001</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

HEAP CELL: G=gc_mark; 1 during mark, 0 other times.

HEAP CELL: 1s and 0s here indicate type.  G missing means sys (not GC’d)

SIMPLE

```
cons ..........SCM car..........0 ..........SCM cdr..........G

   closure ..........SCM code.........011 ..........SCM env.........CCG

HEADERs:```
ssymbol       long length...G0000101       .char *chars............
msymbol       long length...G0000111       .char *chars............
string        long length...G0001011       .char *chars............
vector        long length...G0001111       .SCM **elts.............
VfixN8        long length...G0100101       .unsigned char *words....
VfixZ8        long length...G0101111       .char *words.............
VfixN16       long length...G0111111       .unsigned short *words....
VfixZ16       long length...G0111111       .short *words............
VfixN32       long length...G0100101       .unsigned medium *words...VfixN32
VfixZ32       long length...G0100101       .medium *words...........
VfixN64       long length...G0101101       .unsigned long *words....VfixN64
VfixZ64       long length...G0101111       .long *words.............VfixZ64
VfloR32       long length...G0110101       .float *words............VfloR32
VfloC32       long length...G0110101       .double *words...........VfloC32
VfloR64       long length...G0111101       .double *words...........
VfloC64       long length...G0111111       .double *words...........

Vbool         long length...G1000101       .long *words.............Vbool
contin        long length...G1001111       .*regs........................
specfun       .long length...G1000111       .SCM name...............
cclo          .short length...G1000111       .SCM **elts.............cclo

PTOBs
port int portnum.CwroxxxxxxxG1000111       .FILE *stream...........
socket int portnum.C001xxxxxxxG1000111       .FILE *stream...........
inport int portnum.011xxxxxxxG1000111       .FILE *stream...........
outport int portnum.0101xxxxxxxG1000111       .FILE *stream...........
ioport int portnum.C111xxxxxxxG1000111       .FILE *stream...........
fport int portnum.C 00000000G1000111       .FILE *stream...........
pipe int portnum.C 00000001G1000111       .FILE *stream...........
strport 00000000000.0 00000010G1000111       .FILE *stream...........
sfport int portnum.C 00000011G1000111       .FILE *stream...........

SUBRs
subr_0        .int hpoff.....01010101       .SCM (*f)()..............subr_0
subr_1        .int hpoff.....01010111       .SCM (*f)()..............subr_1
ccr           .int hpoff.....01011101       .double (*f)()...........
subr_3        .int hpoff.....01011111       .SCM (*f)()..............subr_3
subr_2        .int hpoff.....01100101       .SCM (*f)()..............subr_2
asubr         .int hpoff.....01100111       .SCM (*f)()..............asubr
subr_1o       .int hpoff.....01101101       .SCM (*f)()..............subr_1o
subr_2o       .int hpoff.....01101111       .SCM (*f)()..............subr_2o
lsubr_2       .int hpoff.....01110101       .SCM (*f)()..............lsubr_2
lsubr         .int hpoff.....01110111       .SCM (*f)()..............lsubr
rpsubr        .int hpoff.....01111101       .SCM (*f)()..............rpsubr

SMOBs
free_cell
000000000000000000000000G1111111       .*free_cell........000
flo
00000000000000000000000001111111       .float num.............
6.2 Operations

6.2.1 Garbage Collection

The garbage collector is in the latter half of ‘sys.c’. The primary goal of garbage collection (or GC) is to recycle those cells no longer in use. Immediates always appear as parts of other objects, so they are not subject to explicit garbage collection.

All cells reside in the heap (composed of heap segments). Note that this is different from what Computer Science usually defines as a heap.

6.2.1.1 Marking Cells

The first step in garbage collection is to mark all heap objects in use. Each heap cell has a bit reserved for this purpose. For pairs (cons cells) the lowest order bit (0) of the CDR is used. For other types, bit 8 of the CAR is used. The GC bits are never set except during garbage collection. Special C macros are defined in ‘scm.h’ to allow easy manipulation when GC bits are possibly set. CAR, TYP3, and TYP7 can be used on GC marked cells as they are.

GCCDR x
[Macro]
Returns the CDR of a cons cell, even if that cell has been GC marked.

GCTYP16 x
[Macro]
Returns the 16 bit type code of a cell.

We need to (recursively) mark only a few objects in order to assure that all accessible objects are marked. Those objects are sys_protects[] (for example, dynwinds), the current C-stack and the hash table for symbols, symhash.

void gc_mark (SCM obj)
[Function]
The function gc_mark() is used for marking SCM cells. If obj is marked, gc_mark() returns. If obj is unmarked, gc_mark sets the mark bit in obj, then calls gc_mark() on any SCM components of obj. The last call to gc_mark() is tail-called (looped).

void mark_locations (STACKITEM x[], sizet len)
[Function]
The function mark_locations is used for marking segments of C-stack or saved segments of C-stack (marked continuations). The argument len is the size of the stack in units of size (STACKITEM).
Each longword in the stack is tried to see if it is a valid cell pointer into the heap. If it is, the object itself and any objects it points to are marked using gc_mark. If the stack is word rather than longword aligned (#define WORD_ALIGN), both alignments are tried. This arrangement will occasionally mark an object which is no longer used. This has not been a problem in practice and the advantage of using the c-stack far outweighs it.

6.2.1.2 Sweeping the Heap

After all found objects have been marked, the heap is swept.

The storage for strings, vectors, continuations, doubles, complexes, and bignums is managed by malloc. There is only one pointer to each malloc object from its type-header cell in the heap. This allows malloc objects to be freed when the associated heap object is garbage collected.

```c
static void gc_sweep ()
```

The function gc_sweep scans through all heap segments. The mark bit is cleared from marked cells. Unmarked cells are spliced into freelist, where they can again be returned by invocations of NEWCELL.

If a type-header cell pointing to malloc space is unmarked, the malloc object is freed. If the type header of smob is collected, the smob's free procedure is called to free its storage.

6.2.2 Memory Management for Environments

- Ecache was designed and implemented by Radey Shouman.
- This documentation of ecache was written by Tom Lord.

The memory management component of SCM contains special features which optimize the allocation and garbage collection of environments.

The optimizations are based on certain facts and assumptions:

The SCM evaluator creates many environments with short lifetimes and these account of a large portion of the total number of objects allocated.

The general purpose allocator allocates objects from a freelist, and collects using a mark/sweep algorithm. Research into garbage collection suggests that such an allocator is sub-optimal for object populations containing a large portion of short-lived members and that allocation strategies involving a copying collector are more appropriate.

It is a property of SCM, reflected throughout the source code, that a simple copying collector can not be used as the general purpose memory manager: much code assumes that the run-time stack can be treated as a garbage collection root set using conservative garbage collection techniques, which are incompatible with objects that change location.

Nevertheless, it is possible to use a mostly-separate copying-collector, just for environments. Roughly speaking, cons pairs making up environments are initially allocated from a small heap that is collected by a precise copying collector. These objects must be handled specially for the collector to work. The (presumably) small number of these objects that survive one collection of the copying heap are copied to the general purpose heap, where they will later be collected by the mark/sweep collector. The remaining pairs are more rapidly collected.
than they would otherwise be and all of this collection is accomplished without having to mark or sweep any other segment of the heap.

Allocating cons pairs for environments from this special heap is a heuristic that approximates the (unachievable) goal:

allocate all short-lived objects from the copying-heap, at no extra cost in allocation time.

Implementation Details

A separate heap (ecache_v) is maintained for the copying collector. Pairs are allocated from this heap in a stack-like fashion. Objects in this heap may be protected from garbage collection by:

1. Pushing a reference to the object on a stack specially maintained for that purpose. This stack (scm_estk) is used in place of the C run-time stack by the SCM evaluator to hold local variables which refer to the copying heap.

2. Saving a reference to every object in the mark/sweep heap which directly references the copying heap in a root set that is specially maintained for that purpose (scm_egc_roots). If no object in the mark/sweep heap directly references an object from the copying heap, that object can be preserved by storing a direct reference to it in the copying-collector root set.

3. Keeping no other references to these objects, except references between the objects themselves, during copying collection.

When the copying heap or root-set becomes full, the copying collector is invoked. All protected objects are copied to the mark-sweep heap. All references to those objects are updated. The copying collector root-set and heap are emptied.

References to pairs allocated specifically for environments are inaccessible to the Scheme procedures evaluated by SCM. These pairs are manipulated by only a small number of code fragments in the interpreter. To support copying collection, those code fragments (mostly in 'eval.c') have been modified to protect environments from garbage collection using the three rules listed above.

During a mark-sweep collection, the copying collector heap is marked and swept almost like any ordinary segment of the general purpose heap. The only difference is that pairs from the copying heap that become free during a sweep phase are not added to the freelist.

The environment cache is disabled by adding #define NO_ENV_CACHE to 'eval.c'; all environment cells are then allocated from the regular heap.

Relation to Other Work

This work seems to build upon a considerable amount of previous work into garbage collection techniques about which a considerable amount of literature is available.

6.2.3 Dynamic Linking Support

Dynamic linking has not been ported to all platforms. Operating systems in the BSD family (a.out binary format) can usually be ported to DLD. The dl library (#define SUN_DL for SCM) was a proposed POSIX standard and may be available on other machines with
COFF binary format. For notes about porting to MS-Windows and finishing the port to VMS Section 6.4.1 [VMS Dynamic Linking], page 130.

DLD is a library package of C functions that performs dynamic link editing on GNU/Linux, VAX (Ultrix), Sun 3 (SunOS 3.4 and 4.0), SPARCstation (SunOS 4.0), Sequent Symmetry (Dynix), and Atari ST. It is available from:

• ftp.gnu.org:pub/gnu/ldd-3.3.tar.gz

These notes about using libdl on SunOS are from ‘gcc.info’:

On a Sun, linking using GNU CC fails to find a shared library and reports that the library doesn’t exist at all.

This happens if you are using the GNU linker, because it does only static linking and looks only for unshared libraries. If you have a shared library with no unshared counterpart, the GNU linker won’t find anything.

We hope to make a linker which supports Sun shared libraries, but please don’t ask when it will be finished—we don’t know.

Sun forgot to include a static version of ‘libdl.a’ with some versions of SunOS (mainly 4.1). This results in undefined symbols when linking static binaries (that is, if you use ‘-static’). If you see undefined symbols ‘_dlclose’, ‘_dlsym’ or ‘_dlopen’ when linking, compile and link against the file ‘mit/util/misc/dlsym.c’ from the MIT version of X windows.

6.2.4 Configure Module Catalog

The SLIB module catalog can be extended to define other require-able packages by adding calls to the Scheme source file ‘mkimpcat.scm’. Within ‘mkimpcat.scm’, the following procedures are defined.

(add-link feature object-file lib1 . . .) [Function]

feature should be a symbol. object-file should be a string naming a file containing compiled object-code. Each libn argument should be either a string naming a library file or #f.

If object-file exists, the add-link procedure registers symbol feature so that the first time require is called with the symbol feature as its argument, object-file and the lib1 . . . are dynamically linked into the executing SCM session.

If object-file exists, add-link returns #t, otherwise it returns #f.

For example, to install a compiled dll ‘foo’, add these lines to ‘mkimpcat.scm’:

    (add-link 'foo
        (in-vicinity (implementation-vicinity) "foo"
            link:able-suffix))

(add-alias alias feature) [Function]

alias and feature are symbols. The procedure add-alias registers alias as an alias for feature. An unspecified value is returned.

add-alias causes (require ‘alias) to behave like (require ‘feature).
add-source feature filename

(feature is a symbol. filename is a string naming a file containing Scheme source code. The procedure add-source registers feature so that the first time require is called with the symbol feature as its argument, the file filename will be loaded. An unspecified value is returned.

Remember to delete the file ‘slibcat’ after modifying the file ‘mkimpcat.scm’ in order to force SLIB to rebuild its cache.

6.2.5 Automatic C Preprocessor Definitions

These ‘#defines’ are automatically provided by preprocessors of various C compilers. SCM uses the presence or absence of these definitions to configure include file locations and aliases for library functions. If the definition(s) corresponding to your system type is missing as your system is configured, add -Dflag to the compilation command lines or add a #define flag line to ‘scmfig.h’ or the beginning of ‘scmfig.h’.

#define Platforms:
------ -------
ARM_ULIB Huw Rogers free unix library for acorn archimedes
AZTEC_C Aztec_C 5.2a
__CYGWIN__ Cygwin
__CYGWIN32__ Cygwin
_DCC Dice C on AMIGA
__GNUC__ GNU C (and DJGPP)
__EMX__ Gnu C port (gcc/emx 0.8e) to OS/2 2.0
__HIGHC__ MetaWare High C
__IBMC__ C-Set++ on OS/2 2.1
_MSC_VER MS VisualC++ 4.2
MWC Mark Williams C on COHERENT
__MWERKS__ Metrowerks Compiler; Macintosh and WIN32 (?)
__POSIX_SOURCE ??
_QC Microsoft QuickC
__STDC__ ANSI C compliant
__TURBOC__ Turbo C and Borland C
__USE_POSIX ??
__WATCOMC__ Watcom C on MS-DOS
__ZTC__ Zortech C

__AIX__ AIX operating system
__APPLE__ Apple Darwin
AMIGA SAS/C 5.10 or Dice C on AMIGA
__amigaos__ Gnu CC on AMIGA
atarist ATARI-ST under Gnu CC
__DragonflyBSD__ DragonflyBSD
__FreeBSD__ FreeBSD
GNUDOS DJGPP (obsolete in version 1.08)
__GO32__ DJGPP (future?)
hpux HP-UX
linux GNU/Linux
macintosh Macintosh (THINK_C and __MWERKS__ define)
MCH_AMIGA Aztec_c 5.2a on AMIGA
__MACH__ Apple Darwin
__MINGW32__ MinGW - Minimalist GNU for Windows
MSDOS Microsoft C 5.10 and 6.00A
__MSDOS__ Microsoft CLARM and CLTHUMB compilers.
__MSDOS__ Turbo C, Borland C, and DJGPP
__NetBSD__ NetBSD
nosve Control Data NOS/VE
__OpenBSD__ OpenBSD
SVR2 System V Revision 2.
sun SunOS
__SVR4__ SunOS
THINK_C development environment for the Macintosh
ultrix VAX with ULTRIX operating system.
unix most Unix and similar systems and DJGPP (!?)
__unix__ Gnu CC and DJGPP
__UNICOS__ Cray operating system
vaxc VAX C compiler
VAXC VAX C compiler
vaxc11c VAX C compiler
VAX11 VAX C compiler
Windows Borland C 3.1 compiling for Windows
_WIN32 MS VisualC++ 4.2 and Cygwin (Win32 API)
_WIN32_WCE MS Windows CE
vms (and VMS) VAX-11 C under VMS.

__alpha__ DEC Alpha processor
__alpha__ DEC Alpha processor
__hppa__ HP RISC processor
hp9000s800 HP RISC processor
__ia64__ GCC on IA64
__ia64__ GCC on IA64
__LONGLONG___ GCC on IA64
__i386__ DJGPP
i386 DJGPP
__M_ARM__ Microsoft CLARM compiler defines as 4 for ARM.
__M_ARMT__ Microsoft CLTHUMB compiler defines as 4 for Thumb.
MULTIMAX Encore computer
ppc PowerPC
__ppc__ PowerPC
pyr Pyramid 9810 processor
__sgi__ Silicon Graphics Inc.
sparc SPARC processor
sequent Sequent computer
tahoe CCI Tahoe processor
6.2.6 Signals

`init_signals` [Function]
(in `scm.c`) initializes handlers for SIGINT and SIGALRM if they are supported by the C implementation. All of the signal handlers immediately reestablish themselves by a call to `signal()`.

`int_signal sig` [Function]
`alrm_signal sig` [Function]

The low level handlers for SIGINT and SIGALRM.

If an interrupt handler is defined when the interrupt is received, the code is interpreted. If the code returns, execution resumes from where the interrupt happened. `Call-with-current-continuation` allows the stack to be saved and restored.

SCM does not use any signal masking system calls. These are not a portable feature. However, code can run uninterrupted by use of the C macros `DEFER_INTS` and `ALLOW_INTS`.

`DEFER_INTS` [Macro]
sets the global variable `ints_disabled` to 1. If an interrupt occurs during a time when `ints_disabled` is 1, then `deferred_proc` is set to non-zero, one of the global variables `SIGINT_deferred` or `SIGALRM_deferred` is set to 1, and the handler returns.

`ALLOW_INTS` [Macro]
Checks the deferred variables and if set the appropriate handler is called.

Calls to `DEFER_INTS` can not be nested. An `ALLOW_INTS` must happen before another `DEFER_INTS` can be done. In order to check that this constraint is satisfied `#define CAREFUL_INTS` in `scmfig.h`.

6.2.7 C Macros

`ASRTER cond arg pos subr` [Macro]
signals an error if the expression (`cond`) is 0. `arg` is the offending object, `subr` is the string naming the subr, and `pos` indicates the position or type of error. `pos` can be one of

- `ARGn` (5 or unknown ARG number)
- `ARG1`
- `ARG2`
- `ARG3`
- `ARG4`
- `ARG5`
- `WNA` (wrong number of args)
- `OVFLOW`
- `OUTOF RANGE`
• NALLOC
• EXIT
• HUP_SIGNAL
• INT_SIGNAL
• FPE_SIGNAL
• BUS_SIGNAL
• SEGV_SIGNAL
• ALRM_SIGNAL
• a C string (char *)

Error checking is not done by ASRTER if the flag RECKLESS is defined. An error condition can still be signaled in this case with a call to \text{wta(arg, pos, subr)}.

\texttt{ASRTGO cond label} \hspace{1cm} \text{[Macro]}
\begin{itemize}
  \item \texttt{goto label} if the expression (\texttt{cond}) is 0. Like ASRTER, ASRTGO does is not active if the flag RECKLESS is defined.
\end{itemize}

\subsection*{6.2.8 Changing Scm}
When writing C-code for SCM, a precaution is recommended. If your routine allocates a non-cons cell which will \textit{not} be incorporated into a SCM object which is returned, you need to make sure that a SCM variable in your routine points to that cell as long as part of it might be referenced by your code.

In order to make sure this SCM variable does not get optimized out you can put this assignment after its last possible use:

\begin{verbatim}
SCM_dummy1 = foo;
\end{verbatim}

or put this assignment somewhere in your routine:

\begin{verbatim}
SCM_dummy1 = (SCM) &foo;
\end{verbatim}

SCM\_dummy variables are not currently defined. Passing the address of the local SCM variable to \textit{any} procedure also protects it. The procedure \texttt{scm\_protect\_temp} is provided for this purpose.

\begin{verbatim}
void scm_protect_temp (SCM *ptr) \hspace{1cm} \text{[Function]}
  Forces the SCM object \texttt{ptr} to be saved on the C-stack, where it will be traced for GC.
\end{verbatim}

Also, if you maintain a static pointer to some (non-immediate) SCM object, you must either make your pointer be the value cell of a symbol (see \texttt{errobj} for an example) or (permanently) add your pointer to \texttt{sys\_protects} using:

\begin{verbatim}
SCM scm_gc_protect (SCM obj) \hspace{1cm} \text{[Function]}
  Permanently adds \texttt{obj} to a table of objects protected from garbage collection. \texttt{scm\_gc\_protect} returns \texttt{obj}.
\end{verbatim}

To add a C routine to scm:
1. choose the appropriate subr type from the type list.
2. write the code and put into ‘scm.c’.
3. add a make_subr or make_gsubr call to init_scm. Or put an entry into the appropriate iproc structure.

To add a package of new procedures to scm (see ‘crs.c’ for example):
1. create a new C file (‘foo.c’).
2. at the front of ‘foo.c’ put declarations for strings for your procedure names.
   
   ```c
   static char s_twiddle_bits[]="twiddle-bits!";
   static char s_bitsp[]="bits?";
   ```
3. choose the appropriate subr types from the type list in ‘code.doc’.
4. write the code for the procedures and put into ‘foo.c’
5. create one iproc structure for each subr type used in ‘foo.c’
   ```c
   static iproc subr3s[] = {
     {s_twiddle_bits,twiddle-bits},
     {s_bitsp,bitsp},
     {0,0} 
   };
   ```
6. create an init_<name of file> routine at the end of the file which calls init_iprocs with the correct type for each of the iprocs created in step 5.
   ```c
   void init_foo()
   {
     init_iprocs(subr1s, tc7_subr_1);
     init_iprocs(subr3s, tc7_subr_3);
   }
   ```
   If your package needs to have a finalization routine called to free up storage, close files, etc, then also have a line in init_foo like:

   ```c
   add_final(final_foo);
   ```
   final_foo should be a (void) procedure of no arguments. The finals will be called in opposite order from their definition.
   The line:

   ```c
   add_feature("foo");
   ```
   will append a symbol 'foo to the (list) value of slib:features.
7. put any scheme code which needs to be run as part of your package into ‘Ifoo.scm’.
8. put an if into ‘Init5f1.scm’ which loads ‘Ifoo.scm’ if your package is included:
   ```c
   (if (defined? twiddle-bits!)
     (load (in-vicinity (implementation-vicinity)
       "Ifoo"
       (scheme-file-suffix))))
   ```
   or use (provided? 'foo) instead of (defined? twiddle-bits!) if you have added the feature.
9. put documentation of the new procedures into ‘foo.doc’
10. add lines to your ‘Makefile’ to compile and link SCM with your object file. Add a init_foo\(\(\)\)\; to the INITS=... line at the beginning of the makefile.
These steps should allow your package to be linked into SCM with a minimum of difficulty.
Your package should also work with dynamic linking if your SCM has this capability.

Special forms (new syntax) can be added to scm.

1. define a new MAKISYM in ‘scm.h’ and increment NUM_ISYMS.
2. add a string with the new name in the corresponding place in isymnames in ‘repl.c’.
3. add case clause to ceval() near i_quasiquote (in ‘eval.c’).

New syntax can now be added without recompiling SCM by the use of the procedure-
>syntax, procedure->macro, procedure->memoizing-macro, and defmacro. For details, See Section 4.9 [Syntax], page 56.

### 6.2.9 Allocating memory

SCM maintains a count of bytes allocated using malloc, and calls the garbage collector
when that number exceeds a dynamically managed limit. In order for this to work properly,
malloc and free should not be called directly to manage memory freeable by garbage
collection. The following functions are provided for that purpose:

```c
SCM must_malloc_cell (long len, SCM c, char *what)  [Function]
char * must_malloc (long len, char *what)  [Function]
len is the number of bytes that should be allocated, what is a string to be used in
error or gc messages. must_malloc returns a pointer to newly allocated memory.
must_malloc_cell returns a newly allocated cell whose car is c and whose cdr is a
pointer to newly allocated memory.

void must_realloc_cell (SCM z, long olen, long len, char *what)  [Function]
char * must_realloc (char *where, long olen, long len, char *what)  [Function]
must_realloc_cell takes as argument z a cell whose cdr should be a pointer to a
block of memory of length olen allocated with must_malloc_cell and modifies the
cdr to point to a block of memory of length len. must_realloc takes as argument
where the address of a block of memory of length olen allocated by must_malloc and
returns the address of a block of length len.

The contents of the reallocated block will be unchanged up to the minimum of the
old and new sizes.

what is a pointer to a string used for error and gc messages.

must_malloc, must_malloc_cell, must_realloc, and must_realloc_cell must be called
with interrupts deferred See Section 6.2.6 [Signals], page 115. must_realloc and must_-
realloc_cell must not be called during initialization (non-zero errjmp_bad) – the initial
allocations must be large enough.

void must_free (char *ptr, sizet len)  [Function]
must_free is used to free a block of memory allocated by the above functions and
pointed to by ptr. len is the length of the block in bytes, but this value is used only
for debugging purposes. If it is difficult or expensive to calculate then zero may be
used instead.
6.2.10 Embedding SCM

The file ‘scmmain.c’ contains the definition of main(). When SCM is compiled as a library ‘scmmain.c’ is not included in the library; a copy of ‘scmmain.c’ can be modified to use SCM as an embedded library module.

```c
int main (int argc, char **argv) [Function]
   This is the top level C routine. The value of the argc argument is the number of
command line arguments. The argv argument is a vector of C strings; its elements
are the individual command line argument strings. A null pointer always follows the
last element: argv[argc] is this null pointer.
```

```c
char *execpath [Variable]
   This string is the pathname of the executable file being run. This variable can be
examined and set from Scheme (see Section 3.12 [Internal State], page 39). execpath
must be set to executable’s path in order to use DUMP (see Section 5.2 [Dump],
page 65) or DLD.
```

Rename main() and arrange your code to call it with an argv which sets up SCM as you
want it.

If you need more control than is possible through argv, here are descriptions of the functions
which main() calls.

```c
void init_sbrk (void) [Function]
   Call this before SCM calls malloc(). Value returned from sbrk() is used to gauge how
much storage SCM uses.
```

```c
char * scm_find_execpath (int argc, char **argv, char
   *script_arg) [Function]
   argc and argv are as described in main(). script_arg is the pathname of the SCSH-
style script (see Section 3.13 [Scripting], page 41) being invoked; 0 otherwise. scm_
find_execpath returns the pathname of the executable being run; if scm_find_
exepath cannot determine the pathname, then it returns 0.
```

scm_find_implpath is defined in ‘scmmain.c’. Preceding this are definitions
of GENERIC_NAME and INIT_GETENV. These, along with IMPLINIT and dirsep
control scm_find_implpath()’s operation.

If your application has an easier way to locate initialization code for SCM, then you can
replace scm_find_implpath.

```c
char * scm_find_implpath (char *execpath) [Function]
   Returns the full pathname of the Scheme initialization file or 0 if it cannot find it.
   The string value of the preprocessor variable INIT_GETENV names an environment
variable (default “SCM_INIT_PATH”). If this environment variable is defined, its value
will be returned from scm_find_implpath. Otherwise find_impl_file() is called with
the arguments execpath, GENERIC_NAME (default "scm"), INIT_FILE_NAME
(default "Init5f1_scm"), and the directory separator string dirsep. If find_impl_file()
returns 0 and IMPLINIT is defined, then a copy of the string IMPLINIT is returned.
```
**int init_buf0 (FILE *inport)**

Tries to determine whether inport (usually stdin) is an interactive input port which should be used in an unbuffered mode. If so, inport is set to unbuffered and non-zero is returned. Otherwise, 0 is returned.

init_buf0 should be called before any input is read from inport. Its value can be used as the last argument to scm_init_from_argv().

**void scm_init_from_argv (int argc, char **argv, char *script_arg,** iverbose, int buf0stdin)**

Initializes SCM storage and creates a list of the argument strings program-arguments from argv. argc and argv must already be processed to accomodate Scheme Scripts (if desired). The scheme variable *script* is set to the string script_arg, or #f if script_arg is 0. iverbose is the initial prolixity level. If buf0stdin is non-zero, stdin is treated as an unbuffered port.

Call init_signals and restore_signals only if you want SCM to handle interrupts and signals.

**void init_signals (void)**

Initializes handlers for SIGINT and SIGALRM if they are supported by the C implementation. All of the signal handlers immediately reestablish themselves by a call to signal().

**void restore_signals (void)**

Restores the handlers in effect when init_signals was called.

**SCM scm_top_level (char *initpath, SCM (*toplvl_fun))**

This is SCM’s top-level. Errors longjmp here. toplvl_fun is a callback function of zero arguments that is called by scm_top_level to do useful work – if zero, then repl, which implements a read-eval-print loop, is called.

If toplvl_fun returns, then scm_top_level will return as well. If the return value of toplvl_fun is an immediate integer then it will be used as the return value of scm_top_level. In the main function supplied with SCM, this return value is the exit status of the process.

If the first character of string initpath is ‘;’, ‘(‘ or whitespace, then scm_ldstr() is called with initpath to initialize SCM; otherwise initpath names a file of Scheme code to be loaded to initialize SCM.

When a Scheme error is signaled; control will pass into scm_top_level by longjmp, error messages will be printed to current-error-port, and then toplvl_fun will be called again. toplvl_fun must maintain enough state to prevent errors from being resignalled. If toplvl_fun can not recover from an error situation it may simply return.

**void final_scm (int freeall)**

Calls all finalization routines registered with add_final(). If freeall is non-zero, then all memory which SCM allocated with malloc() will be freed.
You can call individual Scheme procedures from C code in the `toplvl_fun` argument passed to `scm_top_level()`, or from module subrs (registered by an `init_` function, see Section 6.2.8 [Changing Scm], page 116).

Use `apply` to call Scheme procedures from your C code. For example:

```c
/* If this apply fails, SCM will catch the error */
apply(CDR(intern("srv:startup",sizeof("srv:startup")-1)),
mksproc(srvproc),
listofnull);

func = CDR(intern(rpcname,strlen(rpcname)));
retval = apply(func, cons(mksproc(srvproc), args), EOL);
```

Functions for loading Scheme files and evaluating Scheme code given as C strings are described in the next section, (see Section 6.2.11 [Callbacks], page 122).

Here is a minimal embedding program `libtest.c`:

```c
/* gcc -o libtest libtest.c libscm.a -ldl -lm -lc */
#include "scm.h"

/* include patchlvl.h for SCM's INIT_FILE_NAME. */
#include "patchlvl.h"

void libtest_init_user_scm()
{
    fputs("This is libtest_init_user_scm\n", stderr); fflush(stderr);
syrintern("*the-string*", makfrom0str("hello world\n"));
}

SCM user_main()
{
    static int done = 0;
    if (done++) return MAKINUM(EXIT_FAILURE);
    scm_LDstr("(display *the-string*)");
    return MAKINUM(EXIT_SUCCESS);
}

int main(argc, argv)
{
    int argc;
    const char **argv;
{
    SCM retval;
    char *implpath, *execpath;

    init_user_scm = libtest_init_user_scm;
    execpath = dld_find_executable(argv[0]);
    fprintf(stderr, "dld_find_executable(%s): %s\n", argv[0], execpath);
    implpath = find_impl_file(execpath, "scm", INIT_FILE_NAME, dirsep);
    fprintf(stderr, "implpath: %s\n", implpath);
```
scm_init_from_argv(argc, argv, 0L, 0, 0);

retval = scm_top_level(implpath, user_main);

final_scm(!0);
return (int)INUM(retval);
}

⊣
dld_find_executable('./libtest'): /home/jaffer/scm/libtest
implpath: /home/jaffer/scm/Init5f1.scm
This is libtest_init_user_scm
hello world

6.2.11 Callbacks
SCM now has routines to make calling back to Scheme procedures easier. The source code
for these routines are found in ‘rope.c’.

int scm_ldfile (char *file)  [Function]
Loads the Scheme source file file. Returns 0 if successful, non-0 if not. This function
is used to load SCM’s initialization file ‘Init5f1.scm’.

int scm_ldprog (char *file)  [Function]
Loads the Scheme source file (in-vicinity (program-vicinity) file). Returns 0
if successful, non-0 if not.
This function is useful for compiled code init_ functions to load non-compiled Scheme
(source) files. program-vicinity is the directory from which the calling code was
loaded (see Section “Vicinity” in SLIB).

SCM scm_evstr (char *str)  [Function]
Returns the result of reading an expression from str and evaluating it.

void scm_ldstr (char *str)  [Function]
Reads and evaluates all the expressions from str.

If you wish to catch errors during execution of Scheme code, then you can use a wrapper
like this for your Scheme procedures:

(define (srv:protect proc)
  (lambda args
    (define result #f) ; put default value here
    (call-with-current-continuation
      (lambda (cont)
        (dynamic-wind (lambda () #t)
          (lambda ()
            (set! result (apply proc args)))
          (set! cont #f)))
      (lambda ()
        (lambda args
          (define result #f) ; put default value here
          (calling proc args))
        (lambda ()
          (lambda args
            (define result #f) ; put default value here
            (calling proc args))))
      (lambda ()
        (lambda args
          (define result #f) ; put default value here
          (calling proc args))))
    (lambda args
      (define result #f) ; put default value here
      (calling proc args))))
Calls to procedures so wrapped will return even if an error occurs.

### 6.2.12 Type Conversions

These type conversion functions are very useful for connecting SCM and C code. Most are defined in ‘rope.c’.

- **SCM long2num (long n)**
- **SCM ulong2num (unsigned long n)**
  
  Return an object of type SCM corresponding to the long or unsigned long argument n. If n cannot be converted, BOOL_F is returned. Which numbers can be converted depends on whether SCM was compiled with the BIGDIG or FLOATS flags.

  To convert integer numbers of smaller types (short or char), use the macro MAKINUM(n).

- **long num2long (SCM num, char *pos, char *s_caller)**
- **unsigned long num2ulong (SCM num, char *pos, char *s_caller)**
- **short num2short (SCM num, char *pos, char *s_caller)**
- **unsigned short num2ushort (SCM num, char *pos, char *s_caller)**
- **unsigned char num2uchar (SCM num, char *pos, char *s_caller)**
- **double num2dbl (SCM num, char *pos, char *s_caller)**

  These functions are used to check and convert SCM arguments to the named C type. The first argument num is checked to see it it is within the range of the destination type. If so, the converted number is returned. If not, the ASRTER macro calls wta with num and strings pos and s_caller. For a listing of useful predefined pos macros, See Section 6.2.7 [C Macros], page 115.

  Note Inexact numbers are accepted only by num2dbl, num2long, and num2ulong (for when SCM is compiled without bignums). To convert inexact numbers to exact numbers, See Section “Numerical operations” in Revised(5) Scheme.

- **unsigned long scm_addr (SCM args, char *s_name)**
  
  Returns a pointer (cast to an unsigned long) to the storage corresponding to the location accessed by aref(CAR(args), CDR(args)). The string s_name is used in any messages from error calls by scm_addr.

  scm_addr is useful for performing C operations on strings or other uniform arrays (see Section 5.4.2 [Uniform Array], page 70).

- **unsigned long scm_base_addr(SCM ra, char *s_name)**
  
  Returns a pointer (cast to an unsigned long) to the beginning of storage of array ra. Note that if ra is a shared-array, the storage accessed this way may be much larger than ra.

  Note While you use a pointer returned from scm_addr or scm_base_addr you must keep a pointer to the associated SCM object in a stack allocated variable or GC-protected location in order to assure that SCM does not reuse that storage before you are done with it. See Section 6.2.8 [Changing Scm], page 116.
SCM makfrom0str (char *src)  [Function]
Return a newly allocated string SCM object copy of the null-terminated string src or
the string src of length len, respectively.

SCM makfromstr (char *src, size_t len)  [Function]

SCM makfromstrs (int argc, char **argv)  [Function]
Returns a newly allocated SCM list of strings corresponding to the argc length array
of null-terminated strings argv. If argv is less than 0, argv is assumed to be NULL
terminated. makfromstrs is used by scm_init_from_argv to convert the arguments
SCM was called with to a SCM list which is the value of SCM procedure calls to
program-arguments (see Section 3.6 [SCM Session], page 31).

char ** makargvfrmstrs (SCM args, char *s_name)  [Function]
Returns a NULL terminated list of null-terminated strings copied from the SCM
list of strings args. The string s_name is used in messages from error calls by
makargvfrmstrs.

makargvfrmstrs is useful for constructing argument lists suitable for passing to main
functions.

void must_free_argv (char **argv)  [Function]
Frees the storage allocated to create argv by a call to makargvfrmstrs.

6.2.13 Continuations
The source files ‘continue.h’ and ‘continue.c’ are designed to function as an independent
resource for programs wishing to use continuations, but without all the rest of the SCM
machinery. The concept of continuations is explained in Section “Control features” in
Revised(5) Scheme.
The C constructs jmp_buf, setjmp, and longjmp implement escape continuations. On VAX
and Cray platforms, the setjmp provided does not save all the registers. The source files
‘setjump.mar’, ‘setjump.s’, and ‘ugsetjump.s’ provide implementations which do meet
this criteria.
SCM uses the names jump_buf, setjump, and longjump in lieu of jmp_buf, setjmp, and
longjmp to prevent name and declaration conflicts.

CONTINUATION jmpbuf length stkbase other parent  [Data type]
is a typedefed structure holding all the information needed to represent a continuation. The other slot can be used to hold any data the user wishes to put there by
defining the macro CONTINUATION_OTHER.

SHORT_ALIGN  [Macro]
If SHORT_ALIGN is #defined (in ‘scmfig.h’), then the it is assumed that pointers in
the stack can be aligned on short int boundaries.

STACKITEM  [Data type]
is a pointer to objects of the size specified by SHORT_ALIGN being #defined or not.
CHEAP_CONTINUATIONS

If CHEAP_CONTINUATIONS is #defined (in ‘scmfig.h’) each CONTINUATION has size sizeof CONTINUATION. Otherwise, all but root CONTINUATIONS have additional storage (immediately following) to contain a copy of part of the stack.

Note On systems with nonlinear stack disciplines (multiple stacks or non-contiguous stack frames) copying the stack will not work properly. These systems need to #define CHEAP_CONTINUATIONS in ‘scmfig.h’.

STACK_GROWS_UP

Expresses which way the stack grows by its being #defined or not.

long thrown_value

Gets set to the value passed to throw_to_continuation.

long stack_size (STACKITEM *start)

Returns the number of units of size STACKITEM which fit between start and the current top of stack. No check is done in this routine to ensure that start is actually in the current stack segment.

CONTINUATION * make_root_continuation (STACKITEM *stack_base)

Allocates (malloc) storage for a CONTINUATION of the current extent of stack. This newly allocated CONTINUATION is returned if successful, 0 if not. After make_root_continuation returns, the calling routine still needs to setjump(new_continuation->jmpbuf) in order to complete the capture of this continuation.

CONTINUATION * make_continuation (CONTINUATION *parent_cont)

Allocates storage for the current CONTINUATION, copying (or encapsulating) the stack state from parent_cont->stkbse to the current top of stack. The newly allocated CONTINUATION is returned if successful, 0q if not. After make_continuation returns, the calling routine still needs to setjump(new_continuation->jmpbuf) in order to complete the capture of this continuation.

void free_continuation (CONTINUATION *cont)

Frees the storage pointed to by cont. Remember to free storage pointed to by cont->other.

void throw_to_continuation (CONTINUATION *cont, long value, CONTINUATION *root_cont)

Sets thrown_value to value and returns from the continuation cont.

If CHEAP_CONTINUATIONS is #defined, then throw_to_continuation does longjump(cont->jmpbuf, val).

If CHEAP_CONTINUATIONS is not #defined, the CONTINUATION cont contains a copy of a portion of the C stack (whose bound must be CONT(root_cont)->stkbse). Then:
Chapter 6: The Implementation

- the stack is grown larger than the saved stack, if necessary.
- the saved stack is copied back into its original position.
- `longjump(cont->jmpbuf, val);`

### 6.2.14 Evaluation

SCM uses its type representations to speed evaluation. All of the `subr` types (see Section 6.1.4 [Subr Cells], page 101) are `tc7` types. Since the `tc7` field is in the low order bit position of the `CAR` it can be retrieved and dispatched on quickly by dereferencing the SCM pointer pointing to it and masking the result.

All the SCM `Special Forms` get translated to immediate symbols (`isym`) the first time they are encountered by the interpreter (`ceval`). The representation of these immediate symbols is engineered to occupy the same bits as `tc7`. All the `isyms` occur only in the `CAR` of lists.

If the `CAR` of an expression to evaluate is not immediate, then it may be a symbol. If so, the first time it is encountered it will be converted to an immediate type `ILOC` or `GLOC` (see Section 6.1.1 [Immediates], page 96). The codes for `ILOC` and `GLOC` lower 7 bits distinguish them from all the other types we have discussed.

Once it has determined that the expression to evaluate is not immediate, `ceval` need only retrieve and dispatch on the low order 7 bits of the `CAR` of that cell, regardless of whether that cell is a closure, header, or `subr`, or a `cons` containing `ILOC`s or `GLOC`s.

In order to be able to convert a SCM symbol pointer to an immediate `ILOC` or `GLOC`, the evaluator must be holding the pointer to the list in which that symbol pointer occurs. Turning this requirement to an advantage, `ceval` does not recursively call itself to evaluate symbols in lists; it instead calls the macro `EVALCAR`. `EVALCAR` does symbol lookup and memoization for symbols, retrieval of values for `ILOC`s and `GLOC`s, returns other immediates, and otherwise recursively calls itself with the `CAR` of the list.

`ceval` inlines evaluation (using `EVALCAR`) of almost all procedure call arguments. When `ceval` needs to evaluate a list of more than length 3, the procedure `eval_args` is called. So `ceval` can be said to have one level lookahead. The avoidance of recursive invocations of `ceval` for the most common cases (special forms and procedure calls) results in faster execution. The speed of the interpreter is currently limited on most machines by interpreter size, probably having to do with its cache footprint. In order to keep the size down, certain `EVALCAR` calls which don’t need to be fast (because they rarely occur or because they are part of expensive operations) are instead calls to the C function `evalcar`.

#### symhash

[Variable]

Top level symbol values are stored in the `symhash` table. `symhash` is an array of lists of `ISYM`s and pairs of symbols and values.

#### ILOC

[Immediate]

Whenever a symbol’s value is found in the local environment the pointer to the symbol in the code is replaced with an immediate object (ILOC) which specifies how many environment frames down and how far in to go for the value. When this immediate object is subsequently encountered, the value can be retrieved quickly.

ILOCs work up to a maximum depth of 4096 frames or 4096 identifiers in a frame. Radey Shouman added `FARLOC` to handle cases exceeding these limits. A `FARLOC` consists of a
pair whose CAR is the immediate type IM_FARLOC_CAR or IM_FARLOC_CDR, and whose CDR is a pair of INUMs specifying the frame and distance with a larger range than ILOCs span. Adding \#define TEST_FARLOC to ‘eval.c’ causes FARLOCs to be generated for all local identifiers; this is useful only for testing memoization.

GLOC [Immediate]

Pointers to symbols not defined in local environments are changed to one plus the value cell address in symhash. This incremented pointer is called a GLOC. The low order bit is normally reserved for GCmark; But, since references to variables in the code always occur in the CAR position and the GCmark is in the CDR, there is no conflict.

If the compile FLAG CAUTIOUS is \#defined then the number of arguments is always checked for application of closures. If the compile FLAG RECKLESS is \#defined then they are not checked. Otherwise, number of argument checks for closures are made only when the function position (whose value is the closure) of a combination is not an ILOC or GLOC. When the function position of a combination is a symbol it will be checked only the first time it is evaluated because it will then be replaced with an ILOC or GLOC.

EVAL expression env  [Macro]
SIDEVAL expression env  [Macro]

EVAL Returns the result of evaluating expression in env. SIDEVAL evaluates expression in env when the value of the expression is not used.

Both of these macros alter the list structure of expression as it is memoized and hence should be used only when it is known that expression will not be referenced again. The C function eval is safe from this problem.

SCM eval (SCM expression)  [Function]

Returns the result of evaluating expression in the top-level environment. eval copies expression so that memoization does not modify expression.

6.3 Program Self-Knowledge

6.3.1 File-System Habitat

Where should software reside? Although individually a minor annoyance, cumulatively this question represents many thousands of frustrated user hours spent trying to find support files or guessing where packages need to be installed. Even simple programs require proper habitat; games need to find their score files.

Aren’t there standards for this? Some Operating Systems have devised regimes of software habitats – only to have them violated by large software packages and imports from other OS varieties.

In some programs, the expected locations of support files are fixed at time of compilation. This means that the program may not run on configurations unanticipated by the authors. Compiling locations into a program also can make it immovable – necessitating recompilation to install it.

Programs of the world unite! You have nothing to lose but loss itself.
Chapter 6: The Implementation

The function `find_impl_file` in `scm.c` is an attempt to create a utility (for inclusion in programs) which will hide the details of platform-dependent file habitat conventions. It takes as input the pathname of the executable file which is running. If there are systems for which this information is either not available or unrelated to the locations of support files, then a higher level interface will be needed.

```
char * find_impl_file (char *exec_path, char *generic_name, char *initname, char *sep)
```

Given the pathname of this executable (`exec_path`), test for the existence of `initname` in the implementation-vicinity of this program. Return a newly allocated string of the path if successful, 0 if not. The `sep` argument is a null-terminated string of the character used to separate directory components.

- One convention is to install the support files for an executable program in the same directory as the program. This possibility is tried first, which satisfies not only programs using this convention, but also uninstalled builds when testing new releases, etc.
- Another convention is to install the executables in a directory named `bin`, `BIN`, `exe`, or `EXE` and support files in a directory named `lib`, which is a peer the executable directory. This arrangement allows multiple executables can be stored in a single directory. For example, the executable might be in `'/usr/local/bin/'` and initialization file in `'/usr/local/lib/'`. If the executable directory name matches, the peer directory `lib` is tested for `initname`.
- Sometimes `lib` directories become too crowded. So we look in any subdirectories of `lib` or `src` having the name (sans type suffix such as `.EXE`) of the program we are running. For example, the executable might be `'/usr/local/bin/foo'` and initialization file in `'/usr/local/lib/foo/'`.
- But the executable name may not be the usual program name; So also look in any `generic_name` subdirectories of `lib` or `src` peers.
- Finally, if the name of the executable file being run has a (system dependent) suffix which is not needed to invoke the program, then look in a subdirectory (of the one containing the executable file) named for the executable (without the suffix); And look in a `generic_name` subdirectory. For example, the executable might be `C:\foo\bar.exe` and the initialization file in `C:\foo\bar\`.

### 6.3.2 Executable Pathname

For purposes of finding `Init5f1.scm`, dumping an executable, and dynamic linking, a SCM session needs the pathname of its executable image. When a program is executed by MS-DOS, the full pathname of that executable is available in `argv[0]`. This value can be passed directly to `find_impl_file` (see Section 6.3.1 [File-System Habitat], page 127).

In order to find the habitat for a unix program, we first need to know the full pathname for the associated executable file.

```
char * dld_find_executable (const char *command)
```

dld_find_executable returns the absolute path name of the file that would be executed if `command` were given as a command. It looks up the environment variable
PATH, searches in each of the directory listed for command, and returns the absolute path name for the first occurrence. Thus, it is advisable to invoke dld_init as:

```c
main (int argc, const char **argv)
{
    ...
    if (dld_init (dld_find_executable (argv[0]))) {
        ...
    }
    ...
}
```

Note: If the current process is executed using the execve call without passing the correct path name as argument 0, dld_find_executable (argv[0]) will also fail to locate the executable file.

dld_find_executable returns zero if command is not found in any of the directories listed in PATH.

### 6.3.3 Script Support

Source code for these C functions is in the file `script.c`. Section 3.13 [Scripting], page 41 for a description of script argument processing.

script_find_executable is only defined on unix systems.

```c
char * script_find_executable (const char *name) [Function]
script_find_executable returns the path name of the executable which is invoked by the script file name; name if it is a binary executable (not a script); or 0 if name does not exist or is not executable.
```

```c
char ** script_process_argv (int argc; char **argv) [Function]
Given an main style argument vector argv and the number of arguments, argc, script_process_argv returns a newly allocated argument vector in which the second line of the script being invoked is substituted for the corresponding meta-argument.
If the script does not have a meta-argument, or if the file named by the argument following a meta-argument cannot be opened for reading, then 0 is returned.
script_process_argv correctly processes argument vectors of nested script invocations.
```

```c
int script_count_argv (char **argv) [Function]
Returns the number of argument strings in argv.
```

### 6.4 Improvements To Make

- Allow users to set limits for malloc() storage.
- Prefix and make more uniform all C function, variable, and constant names. Provide a file full of #define’s to provide backward compatibility.
- lgcd() needs to generate at most one bignum, but currently generates more.
- divide() could use shifts instead of multiply and divide when scaling.
Chapter 6: The Implementation

- Currently, dumping an executable does not preserve ports. When loading a dumped executable, disk files could be reopened to the same file and position as they had when the executable was dumped.

- Copying all of the stack is wasteful of storage. Any time a call-with-current-continuation is called the stack could be re-rooted with a frame which calls the contin just created. This in combination with checking stack depth could also be used to allow stacks deeper than 64K on the IBM PC.

- In the quest for speed, there has been some discussion about a "Forth" style Scheme interpreter.

  Provided there is still type code space available in SCM, if we devote some of the IMCAR codes to "inlined" operations, we should get a significant performance boost. What is eliminated is the having to look up a GLOC or ILOC and then dispatch on the subr type. The IMCAR operation would be dispatched to directly. Another way to view this is that we make available special form versions of CAR, CDR, etc. Since the actual operation code is localized in the interpreter, it is much easier than uncompilation and then recompilation to handle (trace car); For instance a switch gets set which tells the interpreter to instead always look up the values of the associated symbols.

- Scott Schwartz <schwartz@galapagos.cse.psu.edu> suggests: One way to tidy up the dynamic loading stuff would be to grab the code from perl5.

6.4.1 VMS Dynamic Linking

George Carrette (gjc@mitech.com) outlines how to dynamically link on VMS. There is already some code in 'dynl.c' to do this, but someone with a VMS system needs to finish and debug it.

1. Say you have this 'main.c' program:

   ```c
   main()
   {init_lisp();
    lisp_repl();}
   ```

2. and you have your Lisp in files 'repl.c', 'gc.c', eval.c and there are some toplevel non-static variables in use called the_heap, the_environment, and some read-only toplevel structures, such as the_subr_table.

   ```
   $ LINK/SHARE=LISPRTL.EXE/DEBUG REPL.OBJ,GC.OBJ,EVAL.OBJ,LISPRTL.OPT/OPT
   ```

3. where 'LISPRTL.OPT' must contain at least this:

   ```
   SYS$LIBRARY:VAXCRTL/SHARE
   UNIVERSAL=init_lisp
   UNIVERSAL=lisp_repl
   PSECT_ATTR=the_subr_table,SHR,NOWRT,LCL
   PSECT_ATTR=the_heap,NOSHR,LCL
   PSECT_ATTR=the_environment,NOSHR,LCL
   ```

   Notice The psect (Program Section) attributes.

   LCL means to keep the name local to the shared library. You almost always want to do that for a good clean library.
SHR,NOWRT

means shared-read-only. Which is the default for code, and is also good for efficiency of some data structures.

NOSHR,LCL

is what you want for everything else.

Note: If you do not have a handy list of all these toplevel variables, do not dispair. Just do your link with the /MAP=LISPRTL.MAP/FULL and then search the map file, 

$SEARCH/OUT=LISPRTL. LOSERS LISPRTL.MAP ", SHR,NOEXE, RD, WRT"

And use an emacs keyboard macro to muck the result into the proper form. Of course only the programmer can tell if things can be made read-only. I have a DCL command procedure to do this if you want it.

4. Now MAIN.EXE would be linked thusly:

$ DEFINE LISPRTL USER$DISK:[JAFFER]LISPRTL.EXE

$LINK MAIN.OBJ,SYS$INPUT:/OPT
SYS$LIBRARY:VAXCRTL/SHARE
LISPRTL/SHARE

Note the definition of the LISPRTL logical name. Without such a definition you will need to copy `LISPRTL.EXE` over to `SYS$SHARE` (aka `SYS$LIBRARY`) in order to invoke the main program once it is linked.

5. Now say you have a file of optional subrs, `MYSUBRS.C`. And there is a routine INIT_MYSUBRS that must be called before using it.

$ CC MYSUBRS.C

$ LINK/SHARE=MYSUBRS.EXE MYSUBRS.OBJ,SYS$INPUT:/OPT
SYS$LIBRARY:VAXCRTL/SHARE
LISPRTL/SHARE
UNIVERSAL=INIT_MYSUBRS

Ok. Another hint is that you can avoid having to add the PSECT declaration of NOSHR,LCL by declaring variables status in the C language source. That works great for most things.

6. Then the dynamic loader would have to do this:

```c
{void (*init_fcn)();
 long retval;
 retval = lib$find_image_symbol("MYSUBRS","INIT_MYSUBRS",&init_fcn,
 "SYS$DISK:[].EXE");
if (retval != SS$_NORMAL) error(...);
(*init_fcn)();
}
```

But of course all string arguments must be (struct dsc$descriptor *) and the last argument is optional if MYSUBRS is defined as a logical name or if `MYSUBRS.EXE` has been copied over to `SYS$SHARE`. The other consideration is that you will want to turn off C-C or other interrupt handling while you are inside most lib$ calls.

As far as the generation of all the UNIVERSAL=... declarations. Well, you could do well to have that automatically generated from the public `LISPRTL.H` file, of course.
VMS has a good manual called the *Guide to Writing Modular Procedures* or something like that, which covers this whole area rather well, and also talks about advanced techniques, such as a way to declare a program section with a pointer to a procedure that will be automatically invoked whenever any shared image is dynamically activated. Also, how to set up a handler for normal or abnormal program exit so that you can clean up side effects (such as opening a database). But for use with LISPRTL you probably don’t need that hair.

One fancier option that is useful under VMS for ‘LISPLIB.EXE’ is to define all your exported procedures through an *call vector* instead of having them just be pointers into random places in the image, which is what you get by using UNIVERSAL.

If you set up the call vector thing correctly it will allow you to modify and relink ‘LISPLIB.EXE’ without having to relink programs that have been linked against it.
<table>
<thead>
<tr>
<th>Procedure and Macro Index</th>
</tr>
</thead>
<tbody>
<tr>
<td>#</td>
</tr>
<tr>
<td>#!</td>
</tr>
<tr>
<td>#+</td>
</tr>
<tr>
<td>#-</td>
</tr>
<tr>
<td>#:text-till-end-of-line</td>
</tr>
<tr>
<td>#?column</td>
</tr>
<tr>
<td>#?file</td>
</tr>
<tr>
<td>#?line</td>
</tr>
<tr>
<td>#?token</td>
</tr>
<tr>
<td>#</td>
</tr>
<tr>
<td>@</td>
</tr>
<tr>
<td>@apply</td>
</tr>
<tr>
<td>@copy-tree</td>
</tr>
<tr>
<td>@macroexpand1</td>
</tr>
<tr>
<td>-p</td>
</tr>
<tr>
<td>-q</td>
</tr>
<tr>
<td>-r</td>
</tr>
<tr>
<td>-s</td>
</tr>
<tr>
<td>-t</td>
</tr>
<tr>
<td>-u</td>
</tr>
<tr>
<td>-v</td>
</tr>
<tr>
<td>-w</td>
</tr>
<tr>
<td>@</td>
</tr>
<tr>
<td>_exclusive</td>
</tr>
<tr>
<td>_ionbf</td>
</tr>
<tr>
<td>_tracked</td>
</tr>
<tr>
<td>A</td>
</tr>
<tr>
<td>abort</td>
</tr>
<tr>
<td>access</td>
</tr>
<tr>
<td>acct</td>
</tr>
<tr>
<td>acons</td>
</tr>
<tr>
<td>add-alias</td>
</tr>
<tr>
<td>add-finalizer</td>
</tr>
<tr>
<td>add-link</td>
</tr>
<tr>
<td>add-source</td>
</tr>
<tr>
<td>alarm</td>
</tr>
<tr>
<td>alarm-interrupt</td>
</tr>
<tr>
<td>ALLOW_INTS</td>
</tr>
<tr>
<td>alrm_signal</td>
</tr>
<tr>
<td>ARGC</td>
</tr>
<tr>
<td>arithmetic-error</td>
</tr>
<tr>
<td>array-&gt;list</td>
</tr>
<tr>
<td>array-contents</td>
</tr>
<tr>
<td>array-equal?</td>
</tr>
<tr>
<td>array-fill!</td>
</tr>
<tr>
<td>array-map</td>
</tr>
<tr>
<td>array-map!</td>
</tr>
<tr>
<td>array-prototype</td>
</tr>
<tr>
<td>array?</td>
</tr>
<tr>
<td>asinh</td>
</tr>
<tr>
<td>ASRTER</td>
</tr>
<tr>
<td>ASRTGO</td>
</tr>
<tr>
<td>atan</td>
</tr>
<tr>
<td>atanh</td>
</tr>
<tr>
<td>B</td>
</tr>
<tr>
<td>bit-count</td>
</tr>
<tr>
<td>Procedure and Macro Index</td>
</tr>
<tr>
<td>---------------------------</td>
</tr>
<tr>
<td>bit-count*</td>
</tr>
<tr>
<td>bit-invert!</td>
</tr>
<tr>
<td>bit-position</td>
</tr>
<tr>
<td>bit-set*!</td>
</tr>
<tr>
<td>boot-tail</td>
</tr>
<tr>
<td>box</td>
</tr>
<tr>
<td>broken-pipe</td>
</tr>
<tr>
<td>call-with-outputs</td>
</tr>
<tr>
<td>CAR</td>
</tr>
<tr>
<td>cbreak</td>
</tr>
<tr>
<td>CCLO_LENGTH</td>
</tr>
<tr>
<td>CDR</td>
</tr>
<tr>
<td>char</td>
</tr>
<tr>
<td>char-ready</td>
</tr>
<tr>
<td>char-ready?</td>
</tr>
<tr>
<td>char:sharp</td>
</tr>
<tr>
<td>CHARS</td>
</tr>
<tr>
<td>chdir</td>
</tr>
<tr>
<td>CHEAP_CONTINUATIONS</td>
</tr>
<tr>
<td>chmod</td>
</tr>
<tr>
<td>chown</td>
</tr>
<tr>
<td>clearok</td>
</tr>
<tr>
<td>close-port</td>
</tr>
<tr>
<td>closedir</td>
</tr>
<tr>
<td>CLOSEDRP</td>
</tr>
<tr>
<td>CLOSUREP</td>
</tr>
<tr>
<td>CODE</td>
</tr>
<tr>
<td>comment</td>
</tr>
<tr>
<td>CONSP</td>
</tr>
<tr>
<td>copy-file</td>
</tr>
<tr>
<td>copy-tree</td>
</tr>
<tr>
<td>cosh</td>
</tr>
<tr>
<td>could-not-open</td>
</tr>
<tr>
<td>current-error-port</td>
</tr>
<tr>
<td>current-input-port</td>
</tr>
<tr>
<td>current-time</td>
</tr>
<tr>
<td>default-input-port</td>
</tr>
<tr>
<td>default-output-port</td>
</tr>
<tr>
<td>defconst</td>
</tr>
<tr>
<td>DEFER_INTS</td>
</tr>
<tr>
<td>defined?</td>
</tr>
<tr>
<td>defmacro</td>
</tr>
<tr>
<td>defsyntax</td>
</tr>
<tr>
<td>defvar</td>
</tr>
<tr>
<td>directory-&gt;for-each</td>
</tr>
<tr>
<td>directory-for-each</td>
</tr>
<tr>
<td>display</td>
</tr>
<tr>
<td>did_find_executable</td>
</tr>
<tr>
<td>dump</td>
</tr>
<tr>
<td>duplicate-port</td>
</tr>
<tr>
<td>dyn:call</td>
</tr>
<tr>
<td>dyn:link</td>
</tr>
<tr>
<td>dyn:main-call</td>
</tr>
<tr>
<td>dyn:unlink</td>
</tr>
</tbody>
</table>

**E**

| echo                     | 86  |
| ed                       | 32  |
| enclose-array            | 69  |
| end-of-program           | 48  |
| endwin                   | 84  |
| ENV                      | 99  |
| errno                    | 37  |
| error                    | 38  |
| eval                     | 53,127|
| EVAL                     | 127 |
| eval-string              | 53  |
| exact-ceiling            | 67  |
| exact-floor              | 67  |
| exact-round              | 67  |
| exact-truncate           | 67  |
| exec-self                | 39  |
| execl                    | 76  |
| execcl                   | 76  |
| execpath                 | 40  |
| execv                    | 77  |
| execvp                   | 77  |
| exit                     | 31  |
| extended-environment     | 63  |

**F**

| file-position            | 50  |
| fileno                   | 76  |
| final scm                | 120 |
| find_impl_file           | 128 |
| finite?                  | 68  |
| force-output             | 87  |
| fork                     | 78  |
| FPORTP                   | 104 |
| frame->environment      | 35  |
| frame-eval               | 35  |
| frame-trace              | 35  |
| free_continuation        | 125 |
| freshline                | 50  |

**G**

| gc                       | 40  |
| gc-hook                  | 46  |
| gc_mark                  | 109 |
| GCCDR                    | 109 |
| GCTYP16                  | 109 |
| gentemp                  | 58  |
| get-internal-real-time   | 47  |
| get-internal-run-time    | 47  |
| getcwd                   | 75  |
| getegid                  | 78  |
| getenv                   | 32  |
| Procedure and Macro Index | 135 |

<table>
<thead>
<tr>
<th>Procedure</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>geteuid</td>
<td>78</td>
</tr>
<tr>
<td>getgid</td>
<td>78</td>
</tr>
<tr>
<td>getgr</td>
<td>80</td>
</tr>
<tr>
<td>getgroups</td>
<td>81</td>
</tr>
<tr>
<td>gethost</td>
<td>90</td>
</tr>
<tr>
<td>getlogin</td>
<td>32</td>
</tr>
<tr>
<td>getnet</td>
<td>90</td>
</tr>
<tr>
<td>getpeername</td>
<td>92</td>
</tr>
<tr>
<td>getpid</td>
<td>74</td>
</tr>
<tr>
<td>getppid</td>
<td>78</td>
</tr>
<tr>
<td>getproto</td>
<td>91</td>
</tr>
<tr>
<td>getserv</td>
<td>91</td>
</tr>
<tr>
<td>getsockname</td>
<td>92</td>
</tr>
<tr>
<td>getuid</td>
<td>78</td>
</tr>
<tr>
<td>getyx</td>
<td>89</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Procedure</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>H</td>
<td>48</td>
</tr>
<tr>
<td>hang-up</td>
<td>48</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Procedure</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>I</td>
<td>97</td>
</tr>
<tr>
<td>ICHR</td>
<td>97</td>
</tr>
<tr>
<td>ICHR</td>
<td>97</td>
</tr>
<tr>
<td>identifier-&gt;symbol</td>
<td>61</td>
</tr>
<tr>
<td>identifier-equal?</td>
<td>62</td>
</tr>
<tr>
<td>identifier?</td>
<td>61</td>
</tr>
<tr>
<td>idlok</td>
<td>85</td>
</tr>
<tr>
<td>IFLAGP</td>
<td>97</td>
</tr>
<tr>
<td>IMP</td>
<td>96</td>
</tr>
<tr>
<td>inet:address-&gt;string</td>
<td>91</td>
</tr>
<tr>
<td>inet:local-network-address</td>
<td>92</td>
</tr>
<tr>
<td>inet:make-address</td>
<td>92</td>
</tr>
<tr>
<td>inet:network</td>
<td>91</td>
</tr>
<tr>
<td>inet:string-&gt;address</td>
<td>91</td>
</tr>
<tr>
<td>infinite?</td>
<td>68</td>
</tr>
<tr>
<td>init_buf0</td>
<td>120</td>
</tr>
<tr>
<td>init_sbrk</td>
<td>119</td>
</tr>
<tr>
<td>init_signals</td>
<td>115, 120</td>
</tr>
<tr>
<td>initscr</td>
<td>84</td>
</tr>
<tr>
<td>IMPORTP</td>
<td>104</td>
</tr>
<tr>
<td>int_signal</td>
<td>115</td>
</tr>
<tr>
<td>integer-&gt;line-number</td>
<td>53</td>
</tr>
<tr>
<td>INUM</td>
<td>96</td>
</tr>
<tr>
<td>INUMP</td>
<td>96</td>
</tr>
<tr>
<td>isatty?</td>
<td>51</td>
</tr>
<tr>
<td>ISYMCCHARS</td>
<td>97</td>
</tr>
<tr>
<td>ISYMNUM</td>
<td>97</td>
</tr>
<tr>
<td>ISYMP</td>
<td>97</td>
</tr>
<tr>
<td>LINE ...</td>
<td>99, 100</td>
</tr>
<tr>
<td>line-editing</td>
<td>84</td>
</tr>
<tr>
<td>line-number</td>
<td>53</td>
</tr>
<tr>
<td>line-number-&gt;integer</td>
<td>53</td>
</tr>
<tr>
<td>line-number?</td>
<td>53</td>
</tr>
<tr>
<td>link</td>
<td>81</td>
</tr>
<tr>
<td>list-&gt;uniform-array</td>
<td>71</td>
</tr>
<tr>
<td>load</td>
<td>64</td>
</tr>
<tr>
<td>load-string</td>
<td>53</td>
</tr>
<tr>
<td>load:sharp</td>
<td>56</td>
</tr>
<tr>
<td>logaref</td>
<td>71</td>
</tr>
<tr>
<td>logaset!</td>
<td>71</td>
</tr>
<tr>
<td>long</td>
<td>123</td>
</tr>
<tr>
<td>long2num</td>
<td>123</td>
</tr>
<tr>
<td>lstat</td>
<td>81</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Procedure</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>M</td>
<td>58</td>
</tr>
<tr>
<td>macroexpand</td>
<td>58</td>
</tr>
<tr>
<td>macroexpand-1</td>
<td>119</td>
</tr>
<tr>
<td>makargvfmrstrs</td>
<td>124</td>
</tr>
<tr>
<td>makclos</td>
<td>101</td>
</tr>
<tr>
<td>make-arbiter</td>
<td>49</td>
</tr>
<tr>
<td>make-edited-line-port</td>
<td>84</td>
</tr>
<tr>
<td>make-exchanger</td>
<td>48</td>
</tr>
<tr>
<td>make-soft-port</td>
<td>52</td>
</tr>
<tr>
<td>make-stream-socket</td>
<td>92</td>
</tr>
<tr>
<td>makfromOstr</td>
<td>124</td>
</tr>
<tr>
<td>makfromstr</td>
<td>124</td>
</tr>
<tr>
<td>makfromstrs</td>
<td>124</td>
</tr>
<tr>
<td>MAKICHR</td>
<td>97</td>
</tr>
<tr>
<td>MAKIFLAG</td>
<td>98</td>
</tr>
<tr>
<td>MAKINU</td>
<td>96</td>
</tr>
<tr>
<td>MAKISYM</td>
<td>97</td>
</tr>
<tr>
<td>MAKSPCSYM</td>
<td>97</td>
</tr>
<tr>
<td>mark_locations</td>
<td>109</td>
</tr>
<tr>
<td>milli-alarm</td>
<td>47</td>
</tr>
<tr>
<td>mkdir</td>
<td>75</td>
</tr>
<tr>
<td>mkmod</td>
<td>82</td>
</tr>
<tr>
<td>must-free</td>
<td>118</td>
</tr>
<tr>
<td>must-free-argv</td>
<td>124</td>
</tr>
<tr>
<td>must-malloc</td>
<td>118</td>
</tr>
<tr>
<td>must_malloc_cell</td>
<td>118</td>
</tr>
<tr>
<td>must_realloc</td>
<td>118</td>
</tr>
<tr>
<td>must_realloc_cell</td>
<td>118</td>
</tr>
<tr>
<td>mvwin</td>
<td>87</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Procedure</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>N</td>
<td>99</td>
</tr>
<tr>
<td>NCDNSP</td>
<td>99</td>
</tr>
<tr>
<td>NEWCELL</td>
<td>98</td>
</tr>
<tr>
<td>newwin</td>
<td>86</td>
</tr>
<tr>
<td>nice</td>
<td>81</td>
</tr>
</tbody>
</table>
Procedure and Macro Index

NIMP ..................................................... 96
NINUMP .................................................. 96
nl ......................................................... 86
nocbreak ............................................... 85
nodef ...................................................... 85
noecho ..................................................... 86
nonl ......................................................... 86
noraw ....................................................... 86
NSTRINGP ................................................ 100
num2dbl .................................................. 123
num2long .................................................. 123
num2short ............................................... 123
NVECTORP ............................................... 99

O
open-file ............................................... 49
open-input-pipe ........................................ 77
open-output-pipe ....................................... 77
open-pipe ............................................... 77
open-ports .............................................. 50
opendir .................................................. 74
OPENP .................................................... 104
OFFPORTP ................................................. 104
OPINFOPORTP .......................................... 104
OPINPORTP .............................................. 104
OPOUTFPORTP .......................................... 104
OPOUTPORTP ........................................... 104
OPPORTP ................................................ 104
out-of-storage ........................................ 48
OUTPORTP ............................................... 104
overlay .................................................. 87
overwrite ............................................... 87

P
perror .................................................... 38
pi ........................................................ 67
pi/ ....................................................... 67
pipe ....................................................... 78
port-closed? ........................................... 50
port-column ............................................. 50
port-filename .......................................... 50
port-line ................................................. 50
port-type ................................................ 50
PORTP ..................................................... 104
pp .......................................................... 34
pprint ..................................................... 34
print ....................................................... 34
print-args .............................................. 34
procedure->identifier-macro .......................... 59
procedure->macro ....................................... 59
procedure->memoizing-macro .......................... 59
procedure->syntax ....................................... 59
procedure-documentation ................................ 55
profile-alarm .......................................... 47
profile-alarm-interrupt ................................ 48
program-arguments .................................... 32
putenv ..................................................... 77

Q
qase ....................................................... 57
quit ....................................................... 31

R
rav ......................................................... 86
read-char ............................................... 51, 89
read-for-load .......................................... 54
read-numbered ......................................... 53
read:sharp .............................................. 55
readdir ................................................... 74
readlink .................................................. 81
real-acos ............................................... 68
real-acosh .............................................. 68
real-asin ............................................... 68
real-asinh ............................................. 68
real-atan ............................................... 68
real-atanh ............................................. 68
real-cos ............................................... 68
real-cosh .............................................. 68
real-exp ................................................. 68
real-expn ............................................... 68
real-ln ................................................... 68
real-log10 .............................................. 68
real-sin ............................................... 68
real-sinh ............................................... 68
real-sqrt ............................................... 68
real-tan ................................................ 68
real-tanh ............................................... 68
record-printer-set! .................................... 73
redirect-port! ......................................... 74
refresh ................................................... 87
regcomp .................................................. 82
regerror ................................................ 82
regexec .................................................. 82
regmatch ............................................... 83
regmatch? ............................................. 82
regmatchv ............................................. 83
regsearch ............................................. 83
regsearchv ............................................ 83
release-arbiter ....................................... 49
rename-file ............................................. 76
renamed-identifier ................................... 61
renaming-transformer .................................. 63
reopen-file ............................................ 74
require ................................................ 64
reset ..................................................... 86
restart .................................................. 39
restore_signals ........................................ 120
rewinddir .............................................. 74
rmdir .................................................... 75
room ...................................................... 40
S
savetty ........................................... 86
scalar->array .................................. 73
scm_evstr .................................... 122
scm_find_execpath .......................... 119
scm_find_implpath ............................. 119
scm_gc_protect ................................. 116
scm_init_from_argv ........................... 120
scm_ldfile ................................... 122
scm_idprog .................................. 122
scm_idstr .................................... 122
scm_protect_temp .............................. 116
scm_top_level ................................ 120
scope-trace .................................... 35
script_count_argv ............................. 129
script_find_executable ....................... 129
script_find_executable ....................... 129
script_process_argv ........................... 129
scroll ......................................... 89
scrollok ...................................... 85
serial-array-map! ............................... 73
serial-array:copy! ............................. 72
set! ............................................ 57
setegid ....................................... 78
seteuid ........................................ 78
setgid .......................................... 78
setgrent ...................................... 81
sethostent .................................... 90
setnetent ..................................... 91
setprotoent ................................... 91
setpwent ...................................... 80
setservent .................................... 91
setuid ........................................ 78
short .......................................... 123
SHORT_ALIGN ................................ 124
SIDEVAL ....................................... 127
sinh ........................................... 68
socket-name:address .......................... 92
socket-name:family ............................ 92
socket-name:port-number ..................... 92
socket:accept ................................ 93
socket:bind ................................... 93
socket:connect ................................ 93
socket:listen ................................ 93
socket:shutdown ................................ 93
stack-trace .................................... 38
STACK_GROWS_UP ................................ 125
stack_size .................................... 125
stat ............................................ 73
STREAM ......................................... 104
string-edit .................................... 83
string-split ................................... 83
string-splitv ................................ 83
STRINGP ........................................ 100
subwin ......................................... 86
SYMBOLP ........................................ 100
symlink ....................................... 81
sync ............................................ 82
syntax-quote .................................. 63
syntax-rules ................................. 58
T
tanh ............................................ 68
the-macro ...................................... 63
throw_to_continuation ....................... 125
ticks .......................................... 47
ticks-interrupt ................................ 47
touchline ...................................... 87
touchwin ....................................... 87
trace ........................................... 34
transpose-array ................................ 69
try-arbiter .................................... 49
try-create-file ................................. 74
try-load ....................................... 52, 53
try-open-file .................................. 49
ttyname ........................................ 81
TYP16 ........................................... 99
TYP3 ............................................ 99
TYP7 ............................................ 99
U
UCHARS ......................................... 100
ulong2num ..................................... 123
umask .......................................... 76
uname .......................................... 80
unctrl .......................................... 90
uniform-array-read! ........................... 71
uniform-array-write ........................... 71
untrace ........................................ 34
user-interrupt ................................ 48
usr:lib ......................................... 64
utime .......................................... 76
V
vector-set-length! ............................ 46
VECTORP ...................................... 99
VELTS .......................................... 99
verbose ........................................ 40
virtual-alarm .................................. 47
virtual-alarm-interrupt ....................... 48
vms-debug ..................................... 32
void ............................................ 110
W
wadd ............................................ 88
wait-for-input ................................ 51
waitpid ........................................ 79
warn ............................................ 38
wcclear ....................................... 88
wclrtobt ...................................... 88
wclrtobs ...................................... 88
wdelch ......................................... 88
wdeleteln ..................................... 88
werase ........................................ 88
winch ....................................... 89
winsch ...................................... 89
winscln ..................................... 89
with-error-to-file ......................... 51
with-error-to-port ......................... 51
with-input-from-port ..................... 51
with-output-to-port ...................... 51

X

x:lib ........................................ 64
Variable Index

$................................................. 67
$pi.............................................. 67

* .................................................. 31
argv*........................................... 31
execpath........................................ 119
*interactive*................................. 31, 39
*load-pathname*............................... 52
*load-reader*................................. 54
*scm-version*................................. 40
*slib-load-reader*......................... 54
*syntax-rules*............................... 31

A.................................................. 90
af_inet....................................... 90
af_unix....................................... 90

B.................................................. 97
BOOL_F....................................... 97
BOOL_T....................................... 97

E.................................................. 97
EDITOR....................................... 31
EOF_VAL..................................... 97
EOF........................................... 97
erobj........................................... 37

H.................................................. 31
HOME.......................................... 31

I.................................................. 46
internal-time-units-per-second.......... 46

INUM0......................................... 96
isymnames................................... 98

M.................................................. 67
most-negative-fixnum...................... 67
most-positive-fixnum...................... 67

N.................................................. 98
NUM_ISPCSYM................................ 98
NUM_ISYMS.................................. 98

O.................................................. 49
open_both.................................... 49
open_read.................................... 49
open_write................................... 49

P.................................................. 67
pi.............................................. 67

S.................................................. 31
SCHEME_LIBRARY_PATH...................... 31
SCM_INIT_PATH............................... 31
symhash...................................... 126

T.................................................. 125
thrown_value................................ 125

U.................................................. 97
UNDEFINED................................... 97
UNSPECIFIED................................. 97
## Type Index

### #
- #! ........................................... 41, 42

### A
- array-for-each .................................. 72

### C
- CELLPTR ........................................ 98
- CONTINUATION .................................... 124
- curses ........................................... 64

### D
- dump ............................................. 65

### F
- FARLOC ......................................... 126

### G
- gloc ............................................ 98
- GLOC ............................................ 127

### I
- i/o-extensions ................................... 92, 94, 95
- ichr ............................................. 96
- iflags .......................................... 97
- iloc ............................................ 98
- ILOC ............................................ 126
- inum ........................................... 96
- ispccsym ....................................... 98
- isym ............................................ 98

### M
- meta-argument .................................... 41, 129

### P
- ptob ........................................... 103

### R
- regex ............................................ 65, 82
- rev2-procedures .................................. 64
- rev3-procedures .................................. 64

### S
- Scheme Script .................................... 41, 42
- Scheme-Script .................................... 41, 42
- smob ............................................ 105

## T
- tc_dblc ........................................... 105
- tc_dblr ........................................... 105
- tc_free_cell ..................................... 105
- tc16_arbiter ..................................... 106
- tc16_array ....................................... 106
- tc16_bigneg ...................................... 106
- tc16_bigpos ...................................... 105
- tc16_flo ......................................... 106
- tc16_inpipe ....................................... 103
- tc16_inport ....................................... 103
- tc16_iport ....................................... 103
- tc16_outpipe ...................................... 103
- tc16_outport ...................................... 103
- tc16_promise ...................................... 106
- tc16_sicport ....................................... 103
- tc16_strport ....................................... 103
- tc3_closure ....................................... 99
- tc3_cons ......................................... 99
- tc7_assubr ....................................... 101
- tc7_contin ........................................ 101
- tc7_cxr .......................................... 101
- tc7_lsubr ....................................... 102
- tc7_lsubr ....................................... 102
- tc7_msymbol ..................................... 100
- tc7_rsubbr ....................................... 102
- tc7_specfun ...................................... 101
- tc7_ssymbol ..................................... 100
- tc7_string ........................................ 100
- tc7_subr_0 ....................................... 101
- tc7_subr_2 ....................................... 102
- tc7_subr_2o ...................................... 102
- tc7_subr_3 ....................................... 102
- tc7_Vbool ........................................ 100
- tc7_vector ....................................... 99
- tc7_VfixN16 ..................................... 100
- tc7_VfixN32 ..................................... 100
- tc7_VfixN8 ....................................... 100
- tc7_VfixZ16 ..................................... 100
- tc7_VfixZ32 ..................................... 100
- tc7_VfixZ8 ....................................... 100
- tc7_Vfl0C64 ..................................... 101
- tc7_Vfl0R32 ..................................... 100
- tc7_Vfl0R64 ..................................... 101
- turtle-graphics .................................. 64

### U
- unexec .......................................... 65
Concept Index

A
array ........................................... 20, 69
array-for-each ................................ 20
arrays .......................................... 20

B
bignums ........................................ 20
build ............................................ 14
build.scm .................................... 14
byte ............................................ 20
byte-number .................................. 20

calls ........................................... 122
careful-interrupt-masking .................. 20
cautious ...................................... 20
cheap-continuations ....................... 21
compiled-closure ..............................
continuations ................................ 124
curses ......................................... 21

dbg ............................................. 21
differ .......................................... 21
documentation string ....................... 55
dont-memoize-locals ....................... 21
dump ........................................... 21
dynamic-linking ..............................

E
ecache .......................................... 110
documentation string ........................
edit-line ...................................... 21
Embedding SCM ................................ 119
engineering-notation ..................... 21
environments .................................. 110
exchanger .................................... 48
Extrename .................................... 10
Extending Scm ................................ 23

F
foo.c .......................................... 23

G
generalized-c-arguments ................. 21
graphics ...................................... 64

H
hobbit ......................................... 64

I
i/o-extensions ............................... 21
IEEE .......................................... 10
inexact ....................................... 21

J
JACAL .......................................... 11

L
lit ............................................... 21

M
macro .......................................... 21
memory management ....................... 110
mysql ......................................... 21

N
no-heap-shrink .............................. 21
NO_ENV_CACHE .............................. 111
none .......................................... 21

P
posix .......................................... 21, 77
Posix .......................................... 77

R
R4RS .......................................... 10
R5RS .......................................... 10
redless ....................................... 22
record ........................................ 22
regex .......................................... 22
rev2-procedures ............................ 22
rope .......................................... 122, 123
<table>
<thead>
<tr>
<th>Concept</th>
<th>Page Numbers</th>
</tr>
</thead>
<tbody>
<tr>
<td>S</td>
<td></td>
</tr>
<tr>
<td>SchemePrimer</td>
<td>10</td>
</tr>
<tr>
<td>sicp</td>
<td>22</td>
</tr>
<tr>
<td>SICP</td>
<td>10, 22</td>
</tr>
<tr>
<td>signals</td>
<td>115</td>
</tr>
<tr>
<td>Simply</td>
<td>10</td>
</tr>
<tr>
<td>single-precision-only</td>
<td>22</td>
</tr>
<tr>
<td>SLIB</td>
<td>10</td>
</tr>
<tr>
<td>socket</td>
<td>22</td>
</tr>
<tr>
<td>T</td>
<td></td>
</tr>
<tr>
<td>tick-interrupts</td>
<td>22</td>
</tr>
<tr>
<td>turtlegr</td>
<td>22</td>
</tr>
<tr>
<td>U</td>
<td></td>
</tr>
<tr>
<td>unix</td>
<td>22, 81</td>
</tr>
<tr>
<td>Unix</td>
<td>81</td>
</tr>
<tr>
<td>W</td>
<td></td>
</tr>
<tr>
<td>wb</td>
<td>22</td>
</tr>
<tr>
<td>wb-no-threads</td>
<td>22</td>
</tr>
<tr>
<td>windows</td>
<td>22</td>
</tr>
<tr>
<td>X</td>
<td></td>
</tr>
<tr>
<td>x</td>
<td>22, 64</td>
</tr>
<tr>
<td>xlib</td>
<td>22, 64</td>
</tr>
<tr>
<td>Xlib</td>
<td>64</td>
</tr>
<tr>
<td>xlibscm</td>
<td>64</td>
</tr>
<tr>
<td>Xlibscm</td>
<td>64</td>
</tr>
</tbody>
</table>