

The Standard ML Basis Manual

This book provides a description of the Standard ML (SML) Basis Library, the standard library for the SML language. For programmers using SML, it provides a complete description of the modules, types, and functions comprising the library, which is supported by all conforming implementations of the language. The book serves as a programmer's reference, providing manual pages with concise descriptions. In addition, it presents the principles and rationales used in designing the library and relates these to idioms and examples for using the library. A particular emphasis of the library is to encourage the use of SML in serious system programming. Major features of the library include I/O, a large collection of primitive types, support for internationalization, and a portable operating system interface.

This manual will be an indispensable reference for students, professional programmers, and language designers.

Emden R. Gansner is a Principal Technical Staff Member at AT&T Laboratories. Having taught at several prestigious universities, he is currently an adjunct Professor of Computer Science at Stevens Institute of Technology. He has published articles in numerous journals, such as the *Journal of Combinatorial Theory*, *Discrete Mathematics*, and *SIAM Journal of Algorithms and Discrete Methods*. He also jointly received a patent on a technique for drawing directed graphs.

John H. Reppy is an Associate Professor of Computer Science at the University of Chicago. He recently served as Associate Editor of *ACM TOPLAS* and is the author of *Concurrent Programming in ML*, also published by Cambridge University Press.

Cambridge University Press
0521791421 - The Standard ML Basis Manual
Edited by Emden R. Gansner and John H. Reppy
Frontmatter
[More information](#)

The Standard ML Basis Manual

Edited by

Emden R. Gansner
AT&T Laboratories

John H. Reppy
University of Chicago



Cambridge University Press
0521791421 - The Standard ML Basis Manual
Edited by Emden R. Gansner and John H. Reppy
Frontmatter
[More information](#)

PUBLISHED BY THE PRESS SYNDICATE OF THE UNIVERSITY OF CAMBRIDGE
The Pitt Building, Trumpington Street, Cambridge, United Kingdom

CAMBRIDGE UNIVERSITY PRESS
The Edinburgh Building, Cambridge CB2 2RU, UK
40 West 20th Street, New York, NY 10011-4211, USA
477 Williamstown Road, Port Melbourne, VIC 3207, Australia
Ruiz de Alarcón 13, 28014 Madrid, Spain
Dock House, The Waterfront, Cape Town 8001, South Africa
<http://www.cambridge.org>

© AT&T Corp and Lucent Technologies Inc. 2004

This book is in copyright. Subject to statutory exception and
to the provisions of relevant collective licensing agreements,
no reproduction of any part may take place without
the written permission of Cambridge University Press.

First published 2004

Printed in the United States of America

Typeface Times 10/13 pt. *System* L^AT_EX 2_ε [TB]

A catalog record for this book is available from the British Library.

Library of Congress Cataloging in Publication Data

ISBN 0 521 79142 1 hardback

ISBN 0 521 79478 1 paperback

Contents

Foreword	<i>page xi</i>
Preface	xiii
Overview of the book	xiv
Contributors	xv
Acknowledgments	xv
1 Introduction	1
1.1 Design rules and conventions	2
1.1.1 Orthographic conventions	3
1.1.2 Naming	3
1.1.3 Comparisons	4
1.1.4 Conversions	4
1.1.5 Characters and strings	5
1.1.6 Miscellany	5
1.2 Documentation conventions	6
1.2.1 Organization of manual pages	6
1.2.2 Terminology and notation	7
2 Library Modules	9
2.1 Required modules	9
2.2 Optional modules	10
3 Top-Level Environment	15
3.1 Modules in the top-level environment	15
3.2 Top-level type, exception, and value identifiers	15
3.3 Overloaded identifiers	17
3.4 Infix identifiers	18
3.5 The operating environment	19

4	General Usages	21
4.1	Ordering	22
4.2	Option	23
4.3	Exception handling	25
4.4	Miscellaneous functions	26
5	Text	29
5.1	Characters	29
5.2	Strings and substrings	30
5.3	Conversions to and from text	30
5.3.1	Converting to text	30
5.3.2	Converting from text	31
5.3.3	The <code>StringCvt</code> structure	31
5.3.4	The <code>Byte</code> structure	32
5.4	Taking strings apart	33
5.4.1	Tokenizing	33
5.4.2	Readers	34
6	Numerics	37
6.1	Numerical conversions	37
6.1.1	Integer to integer conversions	37
6.1.2	Word to word conversions	37
6.1.3	Word to integer conversions	38
6.1.4	Integer to word conversions	38
6.1.5	Defaults	39
6.2	Floating-point numbers	39
6.2.1	Floating-point conversions	40
6.3	Packed data	41
7	Sequential Data	45
7.1	Common patterns	45
7.1.1	Indexed iterations	49
7.2	Lists	50
7.3	Array modification	51
7.4	Subsequences and slices	51
7.5	Operating on pairs of lists	52
7.6	Two-dimensional arrays	53
8	Input/Output	55
8.1	The I/O model	55
8.1.1	Imperative I/O	56
8.1.2	Stream I/O and state	57
8.1.3	End-of-stream	58
8.1.4	Translation	59

<i>CONTENTS</i>	vii
8.2 Using the I/O subsystem	59
8.2.1 Opening files	60
8.2.2 Imperative stream input (<code>IMPERATIVE_IO</code>)	61
8.2.3 Functional stream input (<code>STREAM_IO</code>)	62
8.2.4 Stream output (<code>IMPERATIVE_IO</code> , <code>STREAM_IO</code>)	66
8.2.5 Readers and writers (<code>PRIM_IO</code>)	67
9 Systems Programming	73
9.1 Portable systems programming	73
9.1.1 File system pathnames	73
9.1.2 File system operations	74
9.1.3 Processes	77
9.1.4 I/O descriptors	77
9.1.5 Time and dates	77
9.2 Operating system-specific programming	80
9.2.1 The <code>Unix</code> structure	80
9.2.2 System flags	84
9.2.3 POSIX programming	84
9.2.4 The <code>Windows</code> structure	85
10 Sockets	89
10.1 Socket basics	89
10.2 Overview	89
10.2.1 Socket types	90
10.2.2 Sockets and addresses creation	90
10.2.3 Socket control	92
10.2.4 Socket I/O	92
10.3 Examples	92
10.3.1 Setting up stream-based sockets	92
10.3.2 Socket I/O	94
10.3.3 Polling sockets	95
11 Manual Pages	99
11.1 The <code>Array</code> structure	100
11.2 The <code>Array2</code> structure	104
11.3 The <code>ArraySlice</code> structure	109
11.4 The <code>BinIO</code> structure	114
11.5 The <code>BIT_FLAGS</code> signature	116
11.6 The <code>Bool</code> structure	118
11.7 The <code>Byte</code> structure	120
11.8 The <code>CHAR</code> signature	122
11.9 The <code>CommandLine</code> structure	129

11.10	The Date structure	130
11.11	The General structure	135
11.12	The GenericSock structure	139
11.13	The IEEEReal structure	141
11.14	The IMPERATIVE_IO signature	144
11.15	The ImperativeIO functor	150
11.16	The INetSock structure	151
11.17	The INTEGER signature	154
11.18	The IntInf structure	159
11.19	The IO structure	162
11.20	The List structure	165
11.21	The ListPair structure	170
11.22	The MATH signature	173
11.23	The MONO_ARRAY signature	177
11.24	The MONO_ARRAY2 signature	182
11.25	The MONO_ARRAY_SLICE signature	188
11.26	The MONO_VECTOR signature	194
11.27	The MONO_VECTOR_SLICE signature	198
11.28	The NetHostDB structure	203
11.29	The NetProtDB structure	206
11.30	The NetServDB structure	208
11.31	The Option structure	210
11.32	The OS structure	212
11.33	The OS.FileSys structure	214
11.34	The OS.IO structure	219
11.35	The OS.Path structure	223
11.36	The OS.Process structure	232
11.37	The PACK_REAL signature	235
11.38	The PACK_WORD signature	237
11.39	The Posix structure	239
11.40	The Posix.Error structure	241
11.41	The Posix.FileSys structure	245
11.42	The Posix.IO structure	258
11.43	The Posix.ProcEnv structure	266
11.44	The Posix.Process structure	271
11.45	The Posix.Signal structure	276
11.46	The Posix.SysDB structure	278
11.47	The Posix.TTY structure	280
11.48	The PRIM_IO signature	290
11.49	The PrimIO functor	298
11.50	The REAL signature	299

<i>CONTENTS</i>	ix
11.51 The <code>Socket</code> structure	310
11.52 The <code>STREAM_IO</code> signature	324
11.53 The <code>StreamIO</code> functor	335
11.54 The <code>STRING</code> signature	337
11.55 The <code>StringCvt</code> structure	342
11.56 The <code>SUBSTRING</code> signature	346
11.57 The <code>TEXT</code> signature	354
11.58 The <code>TEXT_IO</code> signature	355
11.59 The <code>TEXT_STREAM_IO</code> signature	359
11.60 The <code>Time</code> structure	360
11.61 The <code>Timer</code> structure	364
11.62 The <code>Unix</code> structure	366
11.63 The <code>UnixSock</code> structure	370
11.64 The <code>Vector</code> structure	373
11.65 The <code>VectorSlice</code> structure	377
11.66 The <code>Windows</code> structure	381
11.67 The <code>WORD</code> signature	391
Appendix A SML'97 Changes	397
General Index	405
SML Identifier Index	407
Raised Exception Index	429

Foreword

Of all modern programming languages, Standard ML has ascribed perhaps the highest priority to rigorous semantic definition. It is therefore the preferred language for many applications where rigor is important; this is notably true of tools for formal program analysis. It has also gained users who value its high degree of portability, a direct consequence of the unambiguity of its definition.

Now Emden Gansner and John Reppy have equipped SML with another essential ingredient: a library of signatures, structures, and functors which will greatly ease the programmer's task. The SML Basis Library has been long in gestation, but this has ensured that it contains the right things. Only by close cooperation with users, over a considerable period of time, can one be sure of consistency and balance in defining a library. We can therefore be confident that the Basis Library will bring SML into still wider use, and we owe warm thanks to its creators for undertaking an arduous task with skill, care, and dedication.

Robin Milner
Cambridge, July 2003

Preface

One essential for the success of a general-purpose language is an accompanying standard library that is rich enough and efficient enough to support the basic, day-to-day tasks common to all programming. Libraries provide the vocabulary with which a language can be used to say something about something. Without a broad common vocabulary, a language community cannot prosper as it might.

This document presents a standard basis library for SML. It is a basis library in the sense that it concerns itself with the fundamentals: primitive types such as integers and floating-point numbers, operations requiring runtime system or compiler support, such as I/O and arrays; and ubiquitous utility types such as booleans and lists. The SML Basis Library purposefully does not cover higher-level types, such as collection types, or application-oriented APIs, such as regular expression matching. The primary reason for limiting the scope in this way is that the design space for these interfaces is large (e.g. choosing between functors and polymorphism as a parameterization mechanism) and, unlike the case with lists and arrays, we do not have many years of common practice to guide the design. It is also the case that the SML Basis Library specification is a substantial document and expanding its scope would make it unwieldy.

The primary purpose of this book is to serve as a reference manual for the Basis Library, describing as clearly and completely as possible the types, values, and modules making up the Library. This specification is designed to serve both implementors of the SML Basis Library and users. While the specification is not formal, we have tried to make it precise and complete enough to guarantee a high degree of portability between implementations.

It is sometimes difficult to program from a reference manual; all the pieces are there but it is not clear how they fit together. For the working programmer who wants to use the Library, the book also discusses how the functions were meant to be used alone and together. Although not a tutorial, the book should assist the programmer in understanding and using the Library, clarifying when and how various structures should be used, and making the apparent arcana accessible.

There are certain roles the book does not attempt. As we've already noted, it is not a textbook, for either the Library or SML. There are already many fine books and papers teaching the joys of writing in SML, some of which address this Library as well. When dealing with the Library's interface to external software such as Unix or Windows, it assumes the reader already knows how to use them or has access to sources providing that information.

The Library is certainly not complete; there are some glaring omissions, such as a module for handling regular expressions or guidelines for internalization. It is assumed that, as needs are identified and consensus is reached on the design of a structure, new modules will be added to the Library or be standardized as a separate library. The evolution of the Library will be reflected in the online version of this document, the latest version of which can be found at

<http://standardml.org/Basis>

Overview of the book

The book is organized in three main parts: an overview of the Library, its structure and conventions; a tour of the main areas covered by the Library, providing programming tips, idioms, and examples aimed at Library users; and a set of manual pages defining the signatures and structures composing the Library.

The first three chapters form the first part. Chapter 1 presents the philosophy, principles, and rules concerning the design of the Library. It also notes the conventions used in documenting the Library. The second chapter lists all of the signatures, structures, and functions in the Library, noting their connections and whether they are optional. Chapter 3 considers those parts of the Library that are available at the top level, outside of any structure.

The following chapters describe some of the component areas, such as I/O and text handling, in more depth. These chapters discuss the common themes connecting the modules of a component, and note related assumptions and restrictions. The Library includes some elegant solutions to certain programming tasks, but these are not necessarily obvious from a bare presentation of the signatures. Thus, many of these chapters include short tutorial sections that discuss how various types and functions were intended to be used, including examples of idiomatic use.

Chapter 11 is the meat of the book, containing manual pages describing the signatures, structures, and functors specified by the Library, and their semantics. The modules are presented in alphabetical order. Generic modules, those with multiple possible implementations, are gathered under their defining signature. Thus, the `Char` structure is discussed in the `CHAR` section. Each non-generic module, those with a unique implementation, such as `Timer`, heads its own section shared with its signature. Significant substructures, for example, `Posix.IO` also rate their own sections.

During the design of the Library, the authors of the SML language have revised its definition [MTHM97], partly in response to the needs of the Library. The appendix describes some of the changes that have taken place in the SML language, especially in relation to the Library, and also notes where the Library differs from the initial basis described in the original SML definition. The back matter also provides an index of the exceptions defined in the Library.

Contributors

The main architects of the SML Basis Library are Andrew Appel, Dave Berry, Emden Gansner, John Reppy, and Peter Sestoft. In addition, the following people contributed to the design discussions and writing: Nick Barnes, Lal George, Lorenz Huelsbergen, David MacQueen, Dave Matthews, Carsten Müller, Larry Paulson, Riccardo Pucella, and Jon Thackray. This document is edited and maintained by Emden Gansner and John Reppy.

Acknowledgments

As usual in a work like this, many people have been involved in the process. We had helpful comments on the Library or this document from Peter Michael Bertelsen, Matthias Blume, Jeremy Dawson, Matthew Fluet, Elsa Gunter, Ken Larsen, Peter Lee, Neophytos Michael, Kevin Mitchell, Brian Monahan, Stefan Monnier, Chris Okasaki, Andreas Rossberg, Jon Thackray, Mads Tofte, Dan Wang, and Stephen Weeks. David Gay and Serban Jora helped with insights and pointers concerning IEEE floating-point numbers and the Windows operating system, respectively. We would especially like to thank our editor at Cambridge University Press, Lauren Cowles, whose patience has been unbounded.

This document was written using the *ML-Doc* toolkit, which is an SGML-based system for documenting SML interfaces. More information about ML-Doc can be found at

<http://people.cs.uchicago.edu/~jhr/tools/ml-doc.html>