Standard ML of New Jersey

—

Release Notes

(Version 0.93)

February 15, 1993

License and Disclaimer

Standard ML of New Jersey

Version 0.93

Andrew Appel David MacQueen
Lal George John Reppy

1 Introduction

This document describes the latest release of the **Standard ML of New Jersey** (SML/NJ) system.

We are eager to receive your bug reports, comments, and constructive criticism. Any error message beginning with "`Compiler bug`" definitely indicates a bug in the compiler and should be reported. Please use an appropriate variation on the bug reporting form in the file `doc/bugs/bug.form` and send comments and bug reports to `macqueen@research.att.com`.

1.1 Acknowledgements

William Aitken rewrote the pattern-match compiler.

Bruce F. Duba helped improve the pattern-match compiler, the CPS constant-folding phase, the in-line expansion phase, the spill phase, and numerous other parts of the compiler. He also helped to design the *call with current continuation* mechanism.

Adam T. Dingle implemented the Emacs mode for the debugger.

Doug Currie and **Soren Christiansen** ported the system to Macintosh.

Pierre Cregut wrote the fifth-generation module system (introducing higher-order functors) and the sixth-generation module system.

Damien Doligez began the implementation of the fourth-generation module system.

Scott Draves did most of the HP Precision port.

Emden Gansner co-wrote the eXene window system (with John Reppy), worked on a separate compilation system, and contributed to and helped organized the software library.

Georges Gonthier devised new algorithms for the fourth-generation module system.

Yngvi Guttesen wrote the code generator and runtime system for the Intel 386 under MS-Windows.

Ivan Hajadi wrote the original Macintosh port.

Trevor Jim helped design the CPS representation, and implemented the match compiler and the closure-conversion phase, the original library of floating-point functions, and the original assembly-language implementation of external primitive functions.

Mark Leone rewrote Guttesen's i386 code generator and ported it to Unix for the PC.

Mark Lillibridge rewrote the type checker.

James S. Mattson implemented the first version of the lexical-analyzer generator used in constructing the compiler.

Greg Morrisett did the first multiprocessor implementation of SML/NJ, and wrote a key part

of the fourth-generation module system.

James W. O'Toole implemented the NS32032 code generator.

Norman Ramsey implemented the first version of the MIPS code generator.

Gene Rollins developed the SourceGroup compilation system.

Nick Rothwell helped implement the separate compilation mechanism.

Zhong Shao implemented common-subexpression elimination, as well as the callee-save convention that uses multiple-register continuations for faster procedure calls. He has also fixed many bugs.

Konrad Slind helped design and implemented the quote-antiquote facility and the pretty-printer.

David R. Tarditi improved the lexical-analyzer generator and implemented the parser generator used to build parts of the front end; he helped in implementing the type-reconstruction algorithm used by the debugger; he implemented the ML-to-C translator with **Anurag Acharya** and **Peter Lee**, and wrote most of the fourth-generation module system.

Mads Tofte helped implement the separate compilation mechanism and helped develop higher order modules.

Andrew P. Tolmach implemented the SML/NJ debugger. He also rewrote static environments (symbol tables) in a more functional style.

Peter Weinberger implemented the first version of the copying garbage collector.

2 What's New in 0.93

Much has changed since version 0.75, though most old programs should still be compatible. This is a very brief summary of important changes; for more details see the file `doc/bugs/NEWS` in the distribution.

New machines supported

SML/NJ now runs on the Intel 386 architecture (with Unix), the IBM PowerSystem (RS 6000), and the Hewlett-Packard Precision Architecture. As before, it runs on the Sparc, MIPS (DECstation, SGI, etc.), and Motorola 680x0 (Sun3, NeXT, Macintosh, etc.). See section 3.3.1 for more details.

SML/NJ 0.93 does not run under MS-DOS or Microsoft Windows, but we plan to port it to Windows NT.

Separate compilation

The **import** keyword is gone. Separate compilation is now supported by the SourceGroup system, which is located in `tools/sourcegroup`.

Higher-order functors

Functors can now take other functors as arguments, and return functors as results. For details, see the User Guide (the first chapter of the SML/NJ Reference Manual).

Bug fixes

Many bugs have been fixed. Others have been introduced, and most of these have been fixed too. For details, see `doc/bugs/masterbugs` for a complete history and `doc/bugs/openbugs` for a summary of the bugs that remain to be fixed.

Base environment documentation

The SML/NJ reference manual contains descriptions of the structures and bindings built into the `sml` executable (the SML/NJ base (pervasive) environment). A summary of the base environment is listed in Table 1.

Documentation for the **System** structure, which provides the interface to the compiler and operating system, is also available in the reference manual, and is summarized in Table 2.

Program Library

A library documented with manual pages is also provided. A summary of the library is listed in Table 3 These modules are not pre-loaded into the `sml` executable, but must be loaded by `use` or through SourceGroup.

Array	updateable constant-time indexable sequences
Bits	bitwise logical operations on integers
Bool	Boolean type and operations upon it
ByteArray	compact arrays of bytes
callcc, throw	call with current continuation
General	general-purpose predeclared identifiers
Integer	fixed-precision integers and operations upon them
IO	Input/Output structure
List	operations on lists
Quote/Antiquote	object language embedding
Real	floating-point numbers and operations thereupon
RealArray	compact arrays of real numbers
Ref	operations on references
String	operations on character strings
Vector	immutable, constant-time indexable sequences

Table 1: The SML/NJ base environment

Ast	externalized abstract syntax trees used in the compiler
Symbol	support for symbols used in ASTs
Env	manipulation of environments as first class objects
Code	utilities over code strings
Compile	various phases of the compiler
PrettyPrint	prettyprinting utilities
Print	control of output produced by the compiler
Control	various optimization and runtime flags
Tags	tags used for runtime objects
Timer	interface to operating system timer
Stats	record of internal compiler statistics
Unsafe	various unsafe primitives
Signals	interface to UNIX signals
Directory	interface to UNIX files and directories
exn_name	function returning a string for an exeception
version	string containing the version number
architecture	name of the architecture
runtimeStamp	stamp associated with the runtime system
interactive	boolean that is true in an interactive session
system	to execute a command shell
argv	string list of arguments supplied to sml
environ	string list containing the user UNIX environment

Table 2: The SML/NJ **System** structure

ArrayQSort	in-place sorting of arrays
Array2	two-dimensional arrays.
ARRAY_SORT	signature for in-place sorting of arrays
BinaryDict	binary tree dictionary functor
BinarySet	binary tree set functor
CharSet	sets of characters
CType	character classification and conversion functions
DICT	signature for applicative dictionaries
DynamicArray	dynamic array functor
Fifo	applicative queue
Finalizer	object finalization functor
Format	create formatted strings
HashString	computing hash keys on strings
HashTable	hash table functor
IntMap	applicative integer map
IntSet	applicative integer set
Iterate	general purpose iteration
LibBase	base definitions for the SML/NJ library.
ListFormat	formatting and scanning of lists
ListMergeSort	applicative list sorting using merge sort
ListUtil	list utility functions
LIST_SORT	generic interface for list sorting modules.
Makestring	convert primitive types to their string representation.
Name	unique strings
ORD_KEY	signature for ordered keys
ORD_SET	signature for applicative sets on ordered types
Pathname	support for pathname decomposition and search path lists.
PolyHashTable	polymorphic hash tables
Queue	imperative queue
Random	simple random number generator
SplayDict	splay tree dictionary functor
SplaySet	splay tree set functor
SplayTree	splay tree data structure
StringCvt	convert strings into primitive values.
StringUtil	string operations
TimeLimit	limit the amount of time spent evaluating a function application.
UnixEnv	manage name-value environments
UnixPath	support for UNIX search path lists.

Table 3: The SML/NJ Library

Debugger

The SML debugger can be used with this release. See `/tool/debug/debug.ps` for information.

Vectors

Patterns and expressions have been extended to include vector patterns and expressions. `#[pat1,..patn]` represents a vector pattern assuming `pat1,..patn` are patterns, and `#[exp1,..expn]` represents an vector expression assuming `exp1,..expn` are expressions.

Memory usage

The compiler is now fully "safe for space" in the sense described in Chapter 12 of *Compiling with Continuations*. Thus, it is now possible, in principle, to predict memory usage patterns just by reading the source code.

Internal changes

There have been many internal changes since release 0.75. Summaries of these may be found in `doc/bugs/NEWS` and `doc/bugs/LOGSUMMARY`.

CML and eXene

The **Concurrent ML** system, allowing multithreaded concurrent programming in an extension of Standard ML with synchronous communication channels, is available in a new release compatible with SML/NJ version 0.93.

eXene, an elegant concurrent interface to the X Window System, is also available. The **eXene** system is built on top of Concurrent ML.

CML and eXene are available for anonymous ftp on the same servers as SML/NJ. Fetch the files `CML-0.9.8.tar.Z` and `eXene-0.4.tar.Z`.

3 Installation

3.1 Getting this release

The primary means of distributing the compiler is via anonymous ftp from hosts `princeton.edu` and `research.att.com`. For those who do not have internet access directly or indirectory, distribution by tape is possible as a last resort. The connect info and distribution directory is given by the following table:

Host	Net Address	Directory
princeton.edu	128.112.128.1	/pub/ml
research.att.com	192.20.225.2	/dist/ml

To obtain the compiler by internet ftp, connect to one of the two hosts, use the login id "anonymous" with your email address as password, and go to the distribution directory. There you will find the following files:

`README`	Summary of release information
`93.release-notes.ps`	The postscript for this document
`93.release-notes.txt`	This document in ascii form
`93.src.tar.Z`	The src directory containing source code
`93.doc.tar.Z`	The doc directory containing documentation
`93.tools.tar.Z`	The tools directory containing various tools
`smlnj-lib.0.1.tar.Z`	The Standard ML of New Jersey Library
`93.contrib.tar.Z`	unsupported contributed software
`93.mo.m68.tar.Z`	The MC680x0 object files
`93.mo.sparc.tar.Z`	The SPARC object files
`93.mo.mipsl.tar.Z`	MIPS little-endian object files (for DEC machines)
`93.mo.mipsb.tar.Z`	MIPS big-endian (for MIPS, SGI and Sony machines)
`93.mo.hppa.tar.Z`	HP Precision Architecture object files
`93.mo.i386.tar.Z`	Intel i386 object files
`93.mo.rs6000.tar.Z`	IBM RiscSystem 6000 (PowerSystem) object files
`93.mac.tar.Z`	MacOS files
`CML-0.9.8.tar.Z`	Concurrent ML
`eXene-0.4.tar.Z`	eXene - an interface to X11

You will need to transfer the files `93.src.tar.Z` and `93.mo.`*arch*`.tar.Z` in order to build a version for the architecture *arch*. You will probably also want to retrieve the documentation, tools and library files. Here is a sample `ftp` dialog:

```
% ftp princeton.edu
Name: anonymous
Password: login@machine
ftp> binary
ftp> cd pub/ml
ftp> get README
ftp> get 93.src.tar.Z
ftp> get 93.tools.tar.Z
ftp> get smlnj-lib.0.1.tar.Z
ftp> get 93.doc.tar.Z
ftp> get 93.mo.m68.tar.Z
ftp> get 93.mo.rs6000.tar.Z
ftp> close
ftp> quit
```

NOTE: ftp must be put into binary mode before transferring the files.

After the files are transferred they should be uncompressed and unbundled (you will probably want to do this in a new directory). For example:

```
zcat 93.src.tar.Z | tar -xf -
```

unpacks the `src` directory tree, which occupies about 2.5 megabytes of disk space. The minimum required `src` tree can be extracted using the following command:

```
zcat 93.src.tar.Z | tar -xf - src/boot src/runtime src/makeml
```

3.2 Structure of the release

Each file `93.`*file*`.tar.Z` in the distribution produces a directory tree with the root directory named *file*. The `mo.`*arch* directories contain the precompiled ML object files for building the system for the architecture *arch*.

The `src` directory tree contains the source code of the compiler and run-time system, as well as the `makeml` script, which is used for building the system (see Section 3.3).

The `tools` directory contains several software tools for SML, which are also written in SML. At the moment, this consists of the following:

`tools/info` rudimentary tool for querying the sml environment.

`tools/lexgen` lexical analizer generator.

`tools/mlyacc` LALR(1) parser generator.

`tools/mltwig` code generator generator based on dynamic tree pattern matching.

`tools/sourcegroup` system for supporting incremental recompilation of SML programs.

`tools/debug` time travel based debugger.

`tools/prof` display profile data - similar to UNIX prof.

The `smlnj-lib-0.1` directory contains the *Standard ML of New Jersey Library;* see `smlnj-lib-0.1/README`, and also Table 3 of this document. The library contains source code for several ML structures and functions that may be loaded into the system, and their documentation.

The `doc` directory contains various pieces of documentation in subdirectories:

`doc/batch` description of the batch compiler.

`doc/bugs` contains the form for reporting bugs and a history of bug reports and fixes.

`doc/man` contains some UNIX manual pages.

`doc/manual` contains the parts of the Standard ML of New Jersey Reference Manual.

`doc/release-notes` contains this document

`doc/papers` contains copies of several papers and technical reports relating to SML/NJ.

`doc/examples` contains a number of small example programs.

Documentation for the tools and libraries can be found in those directories.

3.3 Installing the system

The compiler can be configured to generate native code for the following processors: MIPS (little and big endian), Motorola 68000, SPARC, HP Precision, IBM PowerSystem, Intel 386/486 (alas, only with Unix). The runtime system can also be configured for a variety of Unix-like operating systems including SunOS, 4.3BSD, MACH, ULTRIX, RISC/os, MORE, HPUX, AIX, and AUX.

3.3.1 Makeml

The file `src/makeml` is a tool for building the system from source and ML-object ('.mo') files. At the very least, the arguments to `makeml` must specify the hardware architecture and operating system. For example:

```
makeml -mips riscos
```

when executed in the `src` directory, would build the interactive compiler for a MIPS processor running RISC/os. `Makeml` assumes that the `mo` files are in a directory that is at the same level as the `src` directory.

The compiler for the HP Precision Architecture is in beta-release, but is likely to be adequate for classroom use.

Table 4 gives some useful computer/opsys combinations for arguments to `makeml`. More detailed options can be found in the makeml manpage (also included in the appendix).

Note,

VAX: While the VAX code generator has been maintained internally, due to the lack of access to a machine, we have been unable to build or test it. If anyone is interested in getting this to run, we would be glad to provide assistance and answer questions. Note: If there is not much interest in this, it is very likely that the VAX code generator will be dropped in future releases.

Processor	Vendor	Options	Comments
Vax	DEC	`-vax bsd`	(`bsd` includes Ultrix)
		`-vax mach`	
680x0	Sun	`-m68 sunos`	
		`-m68 mach`	
680x0	HP	`-m68 hpux`	
		`-m68 hpux8`	HPUX 8.0 or newer
		`-m68 more`	
680x0	Apple	`-m68 aux`	AUX 3.0
		see Section 3.3.2	MacOS 7.0
680x0	NeXT	`-next 2.0`	NeXTStep 2.0
		`-next 3.0`	NeXTStep 3.0
RS 6000	IBM	`-rs6000 aix`	AIX 3.2 or newer
MIPS	DEC	`-decstation ultrix`	little-endian R3000
		`-decstation mach`	little-endian R3000
MIPS	MIPS	`-mips riscos`	
		`-mips mach`	
MIPS	SGI	`-sgi irix`	IRIX 4.x
		`-sgi irix3`	old IRIX (3.x)
MIPS	Sony	`-mips news`	
SPARC	Sun	`-sun4 sunos`	
		`-sun4 mach`	
HPPA	HP	`-hppa hpux8`	beta release
386/486	Intel	`-i386 mach`	
		`-i386 bsd`	BSD386
386	Sequent	`-sequent dynix3`	

Table 4: Makeml machine and operating system options

MIPS R4000: There is a bug in the MIPS R4000 processor - namely a branch at the end of a page causes the system to crash. At present we cannot guarantee against this condition, though the next release should address this problem. Unfortunately, this condition occurs quite frequently.

HPPA: The Hewlett Packard HPPA port is in a beta release mode, but should be adequate for classroom use. We would appreciate any help in finding small examples that expose bugs in this implementation. The default mode on this machine is `-noshare`. It is impossible to build SML/NJ in `share` or `-pervshare` mode on this machine (see Section 3.3.3).

3.3.2 Machintosh OS

This section describes how to build SML/NJ on the Apple Machintosh.

You must be running MacOS System 7, use 32 bit addressing, and have at least 16Mbytes of memory in your Mac to build SML/NJ. The Mac must also have an FPU and the processor must be a M68020, M68030, or M68040.

You should have the SML/NJ distribution with:

- all the files from 93.src.tar.Z installed in directory src: ;
- 93.mo.m68.tar.Z files should be located in the directory src:mo:

There are some Mac specific files in the `src:` and `src:runtime:` directories; all of the files in the `src:runtime:mac:` directory are Mac specific.

The Mac specific files in `src:` are:

- `src:SMLeNJ.p`
- `src:SMLeNJ.p.rsrc`

Then to build SML/NJ for the Mac:

1. launch ThinkC (5.0.4 or later) with the SMLeNJ.p project. Choose **Build Application** from the Project menu. Save the application in the src: directory with an appropriate name. Quit ThinkC.
2. Verify that the application's preferred size is at least 12M bytes, and that at least 12M bytes are free in the Mac. (You can find the application's preferred size using Get Info in the File menu of the Finder. You can determine how much memory is available using About This Macintosh. in the Apple menu.)
3. Launch the newly built application; you will be presented with a dialog box expecting a command line, type: "-r 3 IntM68" and ¡return¿. The mo files will be loaded.
4. You may now save the SML image as a separate file to be imported later, or you may save it into the newly built application. To save it into the application, in this example named `SMLeNJ93`, type:

   ```
   if exportML("SMLeNJ93") then print(``Whoopee!\n'')
   else print(``\n'');
   ```

   To save the image into its own file, use a different filename.

If you have already built a Mac image into a separate file, step 3 above may be replaced by:

- Launch the newly built application; you will be presented with a dialog box expecting a command line, type: "-i $< filename >$" and $< return >$. The image file will be loaded.

Relaunching the newly built application will automatically load the image exported into the application.

You may also force such an application to accept a command line by holding down the ¡option¿ key while the application is being launched. At this point you may continue with step 3 above.

Developers: It is also possible to run with the ThinkC Debugger. In step 1 above, select Use Debugger, and use Run instead of Build Application. You will be presented with a command line dialog box. Proceed as above (step 3) or use the other command line options.

For further documentation on the machintosh see the directory `mac`.

3.3.3 Sharable Text Segment

By default, **sml** is built with a sharable, read-only text segment containing the runtime system (compiled from C code) and the compiler (compiled from ML code). However, to allow `exportFn` to create executable files not containing the compiler, it is necessary to make a special version of **sml** that contains the compiler in the data segment. This can be done with the `makeml` options

```
-noshare -o sml-noshare
```

where the `-o` option affects only the naming of the newly-built executable. In addition, using `-pervshare` it is possible to build an SML executable with just the base environment loaded into the text segment.

3.3.4 Managing Memory Use

ML provides automatic storage management through a garbage-collected heap. Since the heap is used intensively, choice of heap size can have a significant impact on performance. The compiler determines an efficient heap size automatically on startup, resizes the heap up or down as the amount of live data changes, and complains if it runs out of memory (the interactive system can be booted in approximately 8 megabytes).

The `-m` k option to `makeml` sets the target heap size ("softmax") to k kilobytes. The default is 12288k (12 Mbytes). On a 16-megabyte machine, `-m 8192` might be appropriate; on a 32-megabyte single-user workstation, `-m 20000` might do. In general, set it to the amount of physical memory you think should be available to the ML program.

The softmax is very loose; SML will tend to be generous with memory for programs until they hit the softmax; after that it will be more parsimonious. The point is that the larger the heap size, the smaller the garbage collection time overhead. But large-memory programs can still use much more than the softmax (indeed, they may use as much as the operating system is willing to give them).

The softmax can also be set dynamically at runtime by assigning to the variable **System.Control.Runtime.softmax** (which is counted in bytes, not kilobytes).

3.3.5 Batch Compiler

The SML/NJ batch compiler provides some (unsafe) cross compilation and bootstrapping capabilities. The Batch Compiler manual, `doc/batchcomp.ps`, describes these; but they are not recommended for casual users. Furthermore, this batch compiler will become obsolete immediately: it will not be used in any version after 0.93.

3.3.6 Other options

Options also exist for building a batch compiler, a version of the compiler with the debugger loaded, cross compilers, etc. The manpage for `makeml` should be consulted for further details of options that are available.

If you have trouble installing the system, please send us a request for help, including the version of the compiler (check the definition of the `version` variable in `src/boot/perv.sml` if in doubt), hardware configuration (machine type and memory size), operating system, and an input/output script showing the problem. Use the file `doc/bugs/bug.form` as a format for your message.

The following pages contain a man page for the makeml command.

NAME

makeml — build the Standard ML of New Jersey system

SYNOPSIS

makeml *options*

DESCRIPTION

Makeml is a tool for building the Standard ML of New Jersey system (SML/NJ) from source and ML-object ('.mo') files. SML/NJruns on a number of machine architectures (MC680x0, Mips, SPARC, RS/6000, HPPA, and i386/i486) and under a number of different operating systems (SunOS, 4.3bsd, Mach, IRIX, Ultrix, AIX, ...). There are also several different configurations of the system that can be built. Makeml provides a reasonable interface to these various options.

OPTIONS

The following options are used to specify the machine and operating system configuration. These are the only ones necessary for the basic installation.

-sun3 *(sunos | mach)*
: Build the system for the Sun 3.

-sun4 *(sunos | mach)*
: Build the system for sparc machines, including the Sparcstation 10.

(-rs6000 | -rs6k) *aix*
: Build the version for the IBM RS/6000 workstations. **Note**: this requires AIX version 3.2.

-decstation *(bsd | ultrix | mach)*
: Build the version for the DEC mips processor boxes. These are little-endian machines.

-mips *(riscos | mach)*
: Build the version for the MipsCo machines (R3000, R6280). This is a big-endian machine.

-sgi *(irix | irix3)*
: Build the version for the Silicon Graphics machines; the **irix3** option specifies Irix 3.x, otherwise Irix 4.x is assumed. These are big-endian mips processors.

-hppa *hpux8*
: Build the hppa version running under HPUX 8.0 (earlier versions of HPUX have not been tested). By default makeml builds a noshare version (see **-noshare** option), and the **-pervshare** option is ignored.

-m68 *(aux | sunos | mach | hpux | hpux8 | more)*
: Build a version for a M680x0 family machine. The **hpux8** option is for version 8.0 of the HPUX operating system; use **hpux** for earlier versions.

-next *(2 | 3)*
Build the version for the NeXT machine (either NeXTstep 2.x or NeXTstep 3.x). The NeXT machine uses a non-standard version of MACH as its operating system, which isn't BSD compatible.

-i386 *(mach | bsd | bsd386)*
Build the system for i386/i486 machines. The bsd386 version has patches to fix problems with signals in BSD/386.

-sequent *dynix3*
Build the system for the Sequent (i386).

-vax *(bsd | mach)*
Build the vax version. This version is "out of service" for version 0.93 of SML/NJ. Use version 0.75 on the vax.

The following options are used to specify the kind of system to build.

-debug Build an image (with default name 'smld') with the debugger loaded.

-i Make the 'sml' image start out using the interpreter for faster compilation and slower execution (for interactive system only; can switch back to native code once in 'sml' by 'System.Control.interp := false').

-ionly Build an image (with default name 'smli') that has only the interpreter. This gives fast compilation and saves space by eliminating the code generator from the executable, but results in slower execution.

-batch Build the batch compiler (with default name 'smlc') instead of an interactive system.

-target *machine*
Build a batch cross compiler for *machine*. Valid machine names are: **m68**, **sparc**, **mipsl**, **mipsb**, **vax**, and **i386**. Note that for the Mips architecture you must specify the endianess. This option implies the **-batch** option.

-o *imag*
Use image as the name of the system image. The default image name is 'sml' for interactive systems, 'smld' for the debugger version, 'smli' for the interpreter only system and 'smlc' for the batch compiler.

-noshare
Do not link the '.mo' files into an 'a.out' format object file and include it in the runtime executable.

-pervshare
Link only a minimal set of '.mo' files into the object. This is not applicable to the HPPA.

-gcc Use the GNU C compiler to compile the run-time system. This will improve the garbage collector performance on some machines (e.g., Sun3). **Note:** this only works with GCC 1.xx.

The following options may be used to tune garbage collection and paging performance.

-h *heapsize*
Set the initial heap size to *heapsize* kilo-bytes.

-m *softlimit*
Set the soft limit on the heap size to *softlimit* kilo-bytes.

-r *ratio* Set the ratio of the heap size to live data to *ratio*, which must be an integer value no less than 3.

The following options are for building and testing new versions of the system; they are not necessary for normal installation.

-run Build the run-time kernel ('runtime/run'), but don't build a system.

-noclean
Don't remove the existing '.o' files in the runtime directory.

-norun
Don't re-compile the runtime kernel. This implies the **-noclean** option.

-mo *path*
Use *path* as the directory containing the '.mo' files.

-runtime *path*
Use *path*] as the source directory for the runtime code.

-g Compile the runtime with the **-g** command line option.

-D *def* When compiling the runtime code add "**-D** *def*" as a command line option.

-debug0
Build a version with the debugger internals, but not the user-level code.

USAGE

For the standard configuration, the only options required are the machine type and operating system. For example

```
makeml -sun4 sunos
```

builds the SPARC version of the interactive system to run on SunOS systems. Another example is

makeml -sun4 sunos -target mipsl

which builds a **sparc** to **mipsl** cross compiler.

ENVIRONMENT

GCC

Specifies the path of **gcc**. Set this if your path doesn't contain **gcc** and you are using the '**-gcc**' option.

FILES

src/makeml	the makeml shell script.

SEE ALSO

makeml(1), linkdata(1), sml(1)