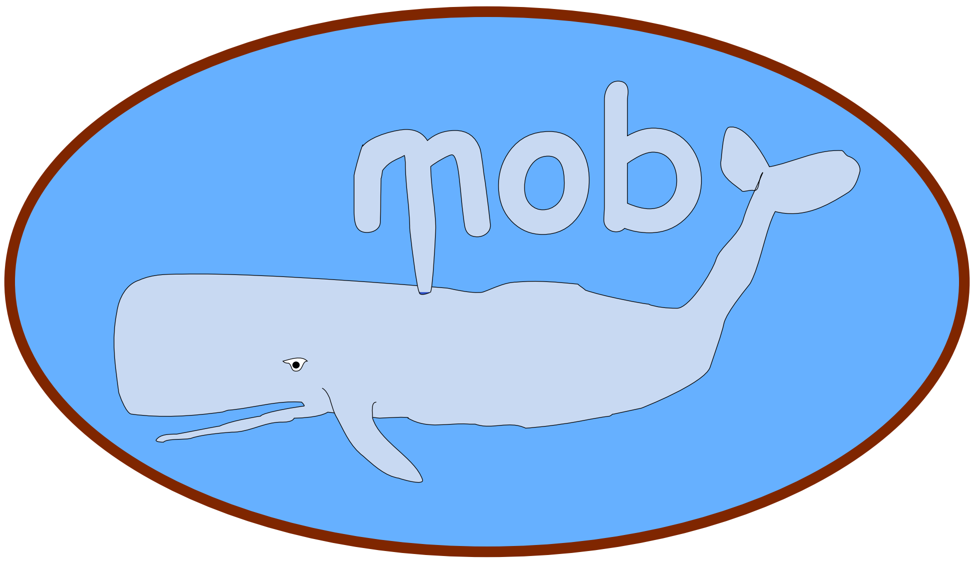
Moby User's Guide
(Version 0.9.18)

The Moby Project
`moby.cs.uchicago.edu`

Last revised: November 7, 2003



Contents

1 Introduction

This document is guide to installing and using the MOBY system.

The MOBY system is distributed with source code under the *Moby Source code license*, which is included in the distribution.

1.1 Acquiring the software

The MOBY system is distributed in source form as a gzipped tar file. You can download it from the MOBY web page.

`http://moby.cs.uchicago.edu`

Installation instructions can be found in the file `INSTALL` in the distribution or in Section 3 of this document.

2 Release notes

This release of the MOBY system is an *alpha* release. As such, there are many aspects of the system that are either not implemented yet or not tested. The more significant limitations are:

Separate compilation The current release is limited to single-file MOBY programs. It is possible to have multiple modules in a single file and nested modules.

Concurrency The multi-threaded runtime and concurrency features of MOBY are not supported in this release.

Parameterized modules This release supports a first-cut at parameterized modules (thanks to Mike Rainey). There are some outstanding issues related to the interaction of parameterized modules and classes, and type revelations on signatures are not yet supported.

In addition, there are other features that have not been implemented yet:

- The `Long` (64-bit integer) type has not been implemented, but `Int` (32-bit) and `Integer` (arbitrary precision) are implemented.

The MOBY system is currently supported on 32-bit IA32 (a.k.a. x86) machines running the Linux operating system. We anticipate ports to the AMD64 (a.k.a. x86-64) architecture running Linux and the Power PC running MacOS X. The runtime system is written in ANSI C using some GCC extensions and should port to other UNIX-like systems on the x86 without too much effort.

3 Installation

Installing the MOBY system is a matter of a few simple steps. For purposes of this discussion, assume that "*PATH*" is the parent directory of where the MOBY distribution will be located.

1. Download the MOBY system and any prerequesites you need (see Section 3.2).
2. Choose a location for the MOBY system's source tree and unbundle the tar file. This step will create a directory tree rooted at "*PATH*`/Moby`."
3. Configure the system (see Section 3.3). This step generates makefiles, *etc.*, that are specialized to your installation.
4. Build the system (see Section 3.4).
5. Once the system is built, it is ready to be used, but you can optionally install the executables and library files in another location (see Section 3.5).

3.1 Supported machines

We currently only support the Intel IA32 (a.k.a., x86) running Linux (2.2 or 2.4 kernels).

3.2 Prerequisites

Before you can configure or build the MOBY system, you should make sure that you have the necessary prerequesites:

SML/NJ

The MOBY compiler is primarily implemented in SML using the SML/NJ system and will compile with recent working versions of the SML/NJ system (110.52 or later). See the SML/NJ homepage at

```
http://www.smlnj.org
```

for information on how to get and install the SML/NJ system. You should ensure that your SML/NJ installation includes the following components:

ml-lex
ml-yacc
smlnj-lib
mlrisc
ckit

The first four of these are part of the default installation; the CKit library is not installed by default, so you must edit the `config/targets` file to include them in the SML/NJ installation.

GNU tools

You will need GNU **make**, **gcc** (the C compiler), and assembler for your machine. The C compiler is used to compile the run-time libraries, while the assembler is used in the MOBY compiler toolchain. We are currently using **make** version 3.79.1, **gcc** version 3.2, and **as** version 2.13.90. You will also need to have the gmp library installed.

3.3 Configuration

The **configure** command is run in the root directory of the MOBY tree.

```
% cd /usr/local/src/moby
% ./configure options
```

where *options* may include one or more of the following:

`--prefix=`*path*

This option specifies the installation path prefix (default `/usr/local`). Library files will be installed in *path*`/lib` and executables will be installed in *path*`/bin`.

`--with-mlrisc=`*path*

This option is used to override the default location of the MLRISC Library. The *path* should specify the *absolute* path to the root of the MLRISC source tree that you want to use.

`--with-ckit=`*path*

This option is used to override the default location of the CKit Library. The *path* should specify the *absolute* path to the root of the CKit source tree that you want to use.

`--with-smlnj-lib=`*path*

This option is used to override the default location of the SML/NJ Library. The *path* should specify the *absolute* path to the root of the SML/NJ Library source tree that you want to use. For some versions of SML/NJ, you may need to use a version of the library that is more recent than the one included in the SML/NJ release (see Section 3.2).

`--with-asdlGen=`*path*

This option is used to specify the location of the **asdlGen** command. The *path* should specify the *absolute* path to the **asdlGen** command. Unless you plan to change the internal representations of the MOBY compiler (see Section 3.6), you will not need this option.

`--enable-heap-frames`

enable heap-allocated activation frames.

> Heap-allocated activation frames are an experimental feature that is still under development.

`--enable-threads`

This option enables compiling versions of the runtime system and MOBY libraries that support threads.

Multithreading is not supported in this release.

`--enable-mt-one-to-one`
enable threads with one-to-one thread/task mapping

Multithreading is not supported in this release.

`--enable-mt-many-to-many`
enable threads with many-to-many thread/task mapping

Multithreading is not supported in this release.

`--help`
This option causes the **configure** command to print out an annotated list of its options.

The **configure** command normally picks up the path of the **sml** command from your path. To override the default version of SML/NJ to use, set the environment variable `SMLNJ_CMD` to the path of the **sml** command that you want to use. For example:

```
% SMLNJ_CMD=/usr/local/smlnj-110.49/bin/sml ./configure
```

3.4 Building the system

Once the system has been configured, you can build the system by running the command

```
% make build
```

in the root directory of the MOBY tree. Assuming that there are no problems, these commands will install the Moby compiler and other tools in the `bin` subdirectory, and it will install the MOBY libraries in the `lib` subdirectory.

3.5 Installing the system

If the make is successful, you can install the compiler, tools, and libraries using the command

```
% make install
```

The default behaviour is to install the moby system in `/usr/local`, but you can override this behaviour by using the "`--prefix`" option to **configure** (see Section 3.3).

The intermediate files produced by the build process can be removed by the command

```
% make clean
```

and the files produced by configuration can be removed by the command

```
% make distclean
```

3.6 Additional information for compiler hackers

3.6.1 CGG

Primitive operations in the MOBY compiler's optimization and code generation phases are supported largely via code generated from a single specification file (`src/CGG/primops.cgg`). The format of this file is described in the *Implementation Notes*.

3.6.2 ASDL

If you want to modify the MOBY compiler, you may need additional tools. The format of MBI files is specified using ASDL (*Abstract Syntax Description Language*), which requires the **asdlGen** tool (`http://asdl.sourceforge.net`). A version of **asdlGen** that is compatible with SML/NJ 110.43 and later can be downloaded from

```
http://moby.cs.uchicago.edu/downloads
```

3.6.3 Running the program generators

The top-level make target `regen` causes the files produced by the **cgg** and **asdlGen** tools to be regenerated. After regenerating these files, the compiler and libraries should be recompiled. We recommend the following three-step process after making a change that requires regeneration:

```
% make regen
% make clean
% make build
```

3.6.4 Configuration management

We use the GNU **autoconf** tool to manage configuration of the system. If the file `configure.ac` changes, the `configuration` file can be regenerated as follows:

```
% autoheader -B config
% autoconf -B config
```

3.6.5 Runtime-system libraries

The configuration system allows one to build runtime systems that support a range of different configuration operations. For each combination, a build directory is created under the

```
src/runtime/build
```

directory with its own, customized, makefile.

4 How to compile Moby programs

The Moby compiler is a *batch* compiler. You should write your program using your favorite text editor and then save it in a file with a "`.mby`" suffix.

4.1 The compilation environment

In order to compile a Moby source file the compiler needs to be able to determine the signatures of imported modules.

> This release does not support separate compilation of Moby modules, so the only MBI files are either hand written or generated by tools like **charon** and **moby-idl**.

4.2 The filemap

When the compiler encounters a free module identifier it must find the MBI file that specifies the module's interface. This task requires a mapping from modules to their signatures, a mapping from modules and signatures to source files, and a list of the imported libraries. These mappings are usually specified using a *filemap* file for each library and application program.

Filemaps have the following simple syntax:

FileMap
    ::= *Entry**

Entry
    ::= **`include`** *String* **`;`**
      | **`library`** *LibraryName* **`;`**
      | **`signature`** *UnitId* **`;`**
      | **`module`** *UnitId* (**`:`** *UnitId*)^{opt} **`;`**
      | **`module`** *UnitId* **`(`** (*UnitId* (**`,`** *UnitId*)*)^{opt} **`)`** (**`:`** *UnitId*)^{opt} **`;`**

LibraryName
    ::= *LibraryId*
      | *String*

UnitId
    ::= *Id* (**`@`** *String*)^{opt}

The lexical class of identifiers are upper-case Moby identifiers; Moby-style comments are also supported.

4.3 An example

As an example of how MOBY programs are compiled, consider the example of an application that consists of two modules: the main program (`Main`) in `main.mby` and an utility module (`Util`) in `util.mby`. Furthermore, assume that the `Util` module has the signature `UTIL` in the file `util-sig.mby`. Lastly, assume that the application uses the `ConsoleIO` module from the `console-io` library. The `FILEMAP` file for this application is given in Figure 1. The first line

```
library console-io;
module Util @ "util.mby : UTIL @ "util-sig.mby";
module Main @ "main.mby";
```

Figure 1: A sample `FILEMAP`

specifies that the `console-io` library is being used, the next line specifies the source code location for the `UTIL` signature, the third line specifies the signature and source-cde location for the `Util` module, and the last line specifies the source-code location for the `Main` module (which we assume has an anonymous or implicit signature).

Since **mobyc** is a batch compiler, we can use UNIX **make** to build our program. Figure 2 gives the source of a possible Makefile.[1]

```
SOURCES = main.mby util.mby util-sig.mby
OBJECTS = main.o   util.o

prog : $(OBJECTS)
	mobyc -o prog $(OBJECTS)

main.o : main.mby util.mbi util-sig.mbi FILEMAP
	mobyc -c main.mby

util.mbi : util.o

util.o : util.mby util-sig.mbi FILEMAP
	mobyc -c util.mby

util-sig.mbi : util-sig.mby FILEMAP
	mobyc -c util-sig.mby
```

Figure 2: A sample `Makefile`

[1]To keep the presentation simple we have not used advanced features of **make**.

NAME

intro — Introduction to MOBY command-line tools.

DESCRIPTION

This chapter describes user commands provided by the MOBY system.

SEEALSO

NAME

charon — an interoperability tool that allows MOBY programs to access C data and call C functions.

SYNOPSIS

charon file . . .

DESCRIPTION

The *charon* command is used to create glue code that enables MOBY programs to access C data structures and invoke C functions. The tool takes a C header file and produces a MOBY MBX file, which is a textual representation of the MOBY compiler's internal representation. The *gen-mbi* tool converts MBX files into MBI files, which are the binary representation of MOBY library code.

In more detail, if *charon* is invoked on C header file c.h, it will produce a file c.mbx. *gen-mbi* can then be invoked to produce c.mbi. A MOBY program m.mby may access the data and use the functions described by c.h by passing the associated C object file c.o to *mobyc* and by including c.mbi in the filemap given to *mobyc*.

OPTIONS

`--cpp`

Run the C pre-processor on the argument files before generating MBX code.

SEEALSO

mobyc, *gen-mbi*

NAME

gen-mbi — tool to generate MBI files from MBX files.

SYNOPSIS

gen-mbi file . . .

DESCRIPTION

The *gen-mbi* command is used to generate MBI files, which have a binary representation, from MBX files, which are ASCII. This tool is used to create low-level library modules and foreign interfaces (*e.g.,* as generated by *charon*).

OPTIONS

`--cpp`
Run the MBX file through the C preprocessor.

`-I` *dir*
This option is passed to the C preprocessor; it adds *dir* to the search path used to find include files.

`-D` *symbol*, **`-D`** *symbol=def*
This option is passed to the C preprocessor; it defines the preprocessor symbol *symb*.

SEEALSO

charon(1)

NAME

mbi-dump —

SYNOPSIS

mbi-dump

DESCRIPTION

SEEALSO

NAME

moby-idl —

SYNOPSIS

moby-idl

DESCRIPTION

SEEALSO

NAME

mobyc — a compiler for the MOBY programming language.

SYNOPSIS

mobyc file . . .

DESCRIPTION

The *mobyc* command is used to compile and link MOBY programs.

OPTIONS

`-?`, **`--help`**
Display a list of command-line options and then exit. If this option used in conjunction with the **`-v`** flag, a more detailed list of options is displayed (including internal debugging flags).

`-t`, **`--threads`**
Enable the concurrency features of MOBY and link with the multithreaded versions of the MOBY libraries and runtime system.

`-T`, **`--checkonly`**

`-S`
Creates an assembly file for each named source file, but does not produce object files or executables. The assembly-file name corresponds to the name of the source file, with a "`.s`" suffix substituted for the suffix of the source file.

`-c`
Creates an object file for each named source file, but does not link the object files into an executable. The object-file name corresponds to the name of the source file, with a "`.o`" suffix substituted for the suffix of the source file.

`-o` *outfile*
Create an executable named *outfile*. When specified with the **`-S`** option, the **`-o`** option is ignored. If neither **`-o`** and **`-c`** are not specified, a file named "`a.out`" is produced.

`--filemap=`*file*
Use the *file* as the filemap instead of the default (`FILEMAP`). If this option is used in conjunction with the **`--implicit-filemap`** flag, then the file map is consulted first when mapping module names to files.

`-i` *lib*, **`--use=`***lib*
Add the MOBY library "*lib*" to the libraries used by the source file.

`--implicit-filemap`
Use an implicit mapping from module and signature names to file names. A module or signature "`Foo`" is assumed to be defined in the file "`Foo.mby`."

`-I` *path*
Add the directory specified by *path* to the search path for MOBY libraries.

`-l` *library*
Add the system library "*lib*" to the list of objects when linking.

`-L` *path*
: Add the directory specified by *path* to the search path for system libraries.

`--main=`*Module*
: Specifies that “*Module*” is the main module of the MOBY program (the default is `Main`).

`--version`
: Print the compiler version number and then exit.

`-v`
: Run the compiler driver in verbose mode. This option can also be used with the **`--help`** flag to get a more detailed list of options.

1

The MOBY runtime system recognizes a number of command-line options that allow one to configure the execution environment and to aid in debugging the compiler and runtime. The options that control runtime parameters are:

`--moby-alloc-pages`=*n*
: Specifies the number of pages allocated to the “*nursery*” (*i.e.*, used for small-object allocation). The default is 64 pages (512Kb).

`--moby-task-pages`=*n*
: Specifies the number of allocation pages given to a task at one time. The default is 8 pages (64Kb).

The debugging flags allow various aspects of the runtime system to be traced. These flags are not available if the **–nodebug** linking option was specified.

`--moby-debug`=*file*
: Specifies a file for debugging output.

`--moby-debug-help`
: Prints a it of the debuggin options and exits.

`--moby-debug-all`
: Enables all runtime-system tracing.

`--moby-debug-gc`
: Enables tracing of the garbage collector.

`--moby-debug-heap`
: Enables tracing of the heap.

`--moby-debug-pcmaps`
: Enables tracing of the PC map registration and searching.

`--moby-debug-static`
: Enables tracing of the static-data registration.

SEEALSO

charon(1)