UMB Scheme Release Notes

last updated July 12, 1991
$Revision: 2.5 $

William R Campbell

GNU GENERAL PUBLIC LICENSE

Version 1, February 1989

Preamble

The license agreements of most software companies try to keep users at the mercy of those companies. By contrast, our General Public License is intended to guarantee your freedom to share and change free software—to make sure the software is free for all its users. The General Public License applies to the Free Software Foundation's software and to any other program whose authors commit to using it. You can use it for your programs, too.

When we speak of free software, we are referring to freedom, not price. Specifically, the General Public License is designed to make sure that you have the freedom to give away or sell copies of free software, that you receive source code or can get it if you want it, that you can change the software or use pieces of it in new free programs; and that you know you can do these things.

To protect your rights, we need to make restrictions that forbid anyone to deny you these rights or to ask you to surrender the rights. These restrictions translate to certain responsibilities for you if you distribute copies of the software, or if you modify it.

For example, if you distribute copies of a such a program, whether gratis or for a fee, you must give the recipients all the rights that you have. You must make sure that they, too, receive or can get the source code. And you must tell them their rights.

We protect your rights with two steps: (1) copyright the software, and (2) offer you this license which gives you legal permission to copy, distribute and/or modify the software.

Also, for each author's protection and ours, we want to make certain that everyone understands that there is no warranty for this free software. If the software is modified by someone else and passed on, we want its recipients to know that what they have is not the original, so that any problems introduced by others will not reflect on the original authors' reputations.

The precise terms and conditions for copying, distribution and modification follow.

TERMS AND CONDITIONS

1. This License Agreement applies to any program or other work which contains a notice placed by the copyright holder saying it may be distributed under the terms of this General Public License. The "Program", below, refers to any such program or work, and a "work based on the Program" means either the Program or any work containing the Program or a portion of it, either verbatim or with modifications. Each licensee is addressed as "you".
2. You may copy and distribute verbatim copies of the Program's source code as you receive it, in any medium, provided that you conspicuously and appropriately publish on each copy an appropriate copyright notice and disclaimer of warranty; keep intact all the notices that refer to this General Public License and to the absence of any warranty; and give any other recipients of the Program a copy of this General Public License along with the Program. You may charge a fee for the physical act of transferring a copy.
3. You may modify your copy or copies of the Program or any portion of it, and copy and distribute such modifications under the terms of Paragraph 1 above, provided that you also do the following:
   - cause the modified files to carry prominent notices stating that you changed the files and the date of any change; and
   - cause the whole of any work that you distribute or publish, that in whole or in part contains the Program or any part thereof, either with or without modifications, to be licensed at no charge to all third parties under the terms of this General Public License (except that you may choose to grant warranty protection to some or all third parties, at your option).
   - If the modified program normally reads commands interactively when run, you must cause it, when started running for such interactive use in the simplest and most usual way, to print or display an announcement including an appropriate copyright notice and a notice that there is no warranty (or else, saying that you provide a warranty) and that users may redistribute the program under these conditions, and telling the user how to view a copy of this General Public License.
   - You may charge a fee for the physical act of transferring a copy, and you may at your option offer warranty protection in exchange for a fee.

   Mere aggregation of another independent work with the Program (or its derivative) on a volume of a storage or distribution medium does not bring the other work under the scope of these terms.
4. You may copy and distribute the Program (or a portion or derivative of it, under Paragraph

2) in object code or executable form under the terms of Paragraphs 1 and 2 above provided that you also do one of the following:

- accompany it with the complete corresponding machine-readable source code, which must be distributed under the terms of Paragraphs 1 and 2 above; or,
- accompany it with a written offer, valid for at least three years, to give any third party free (except for a nominal charge for the cost of distribution) a complete machine-readable copy of the corresponding source code, to be distributed under the terms of Paragraphs 1 and 2 above; or,
- accompany it with the information you received as to where the corresponding source code may be obtained. (This alternative is allowed only for noncommercial distribution and only if you received the program in object code or executable form alone.)

Source code for a work means the preferred form of the work for making modifications to it. For an executable file, complete source code means all the source code for all modules it contains; but, as a special exception, it need not include source code for modules which are standard libraries that accompany the operating system on which the executable file runs, or for standard header files or definitions files that accompany that operating system.

5. You may not copy, modify, sublicense, distribute or transfer the Program except as expressly provided under this General Public License. Any attempt otherwise to copy, modify, sublicense, distribute or transfer the Program is void, and will automatically terminate your rights to use the Program under this License. However, parties who have received copies, or rights to use copies, from you under this General Public License will not have their licenses terminated so long as such parties remain in full compliance.
6. By copying, distributing or modifying the Program (or any work based on the Program) you indicate your acceptance of this license to do so, and all its terms and conditions.
7. Each time you redistribute the Program (or any work based on the Program), the recipient automatically receives a license from the original licensor to copy, distribute or modify the Program subject to these terms and conditions. You may not impose any further restrictions on the recipients' exercise of the rights granted herein.
8. The Free Software Foundation may publish revised and/or new versions of the General Public 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.

   Each version is given a distinguishing version number. If the Program specifies a version number of the license which applies to it and "any later version", you have the option of following the terms and conditions either of that version or of any later version published by the Free Software Foundation. If the Program does not specify a version number of the license, you may choose any version ever published by the Free Software Foundation.
9. If you wish to incorporate parts of the Program into other free programs whose distribution conditions are different, write to the author to ask for permission. For software which is copyrighted by the Free Software Foundation, write to the Free Software Foundation; we

sometimes make exceptions for this. Our decision will be guided by the two goals of preserving the free status of all derivatives of our free software and of promoting the sharing and reuse of software generally.

NO WARRANTY

10. BECAUSE THE PROGRAM IS LICENSED FREE OF CHARGE, THERE IS NO WARRANTY FOR THE PROGRAM, TO THE EXTENT PERMITTED BY APPLICABLE LAW. EXCEPT WHEN OTHERWISE STATED IN WRITING THE COPYRIGHT HOLDERS AND/OR OTHER PARTIES PROVIDE THE PROGRAM "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESSED OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. THE ENTIRE RISK AS TO THE QUALITY AND PERFORMANCE OF THE PROGRAM IS WITH YOU. SHOULD THE PROGRAM PROVE DEFECTIVE, YOU ASSUME THE COST OF ALL NECESSARY SERVICING, REPAIR OR CORRECTION.
11. IN NO EVENT UNLESS REQUIRED BY APPLICABLE LAW OR AGREED TO IN WRITING WILL ANY COPYRIGHT HOLDER, OR ANY OTHER PARTY WHO MAY MODIFY AND/OR REDISTRIBUTE THE PROGRAM AS PERMITTED ABOVE, BE LIABLE TO YOU FOR DAMAGES, INCLUDING ANY GENERAL, SPECIAL, INCIDENTAL OR CONSEQUENTIAL DAMAGES ARISING OUT OF THE USE OR INABILITY TO USE THE PROGRAM (INCLUDING BUT NOT LIMITED TO LOSS OF DATA OR DATA BEING RENDERED INACCURATE OR LOSSES SUSTAINED BY YOU OR THIRD PARTIES OR A FAILURE OF THE PROGRAM TO OPERATE WITH ANY OTHER PROGRAMS), EVEN IF SUCH HOLDER OR OTHER PARTY HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

END OF TERMS AND CONDITIONS

1. Why We Implemented UMB Scheme

Our reasons form implementing yet another version of Scheme are the following:

1. We use Scheme for modeling the operational semantics of programming languages in our undergraduate languages course. We wanted a Scheme system that was easily ported to the various architectures on campus and that gave reasonable run-time performance.
2. We wanted to produce a relatively large piece if code as an example of good program organization for our undergraduate and graduate software engineering students.
3. We wanted to involve students in all phases of the implementation effort, including initial development, testing, extensions, and performance tuning.
4. We wanted to illustrate that object-oriented design might successfully be applied to programs written in vanilla programming languages such as C.

Although this effort was meant as an exercise in design and programming, we have ended up with what we think to be a rather nice interpreter. It runs relatively fast (performance comparisons with MIT Scheme are made below) and we (as well as others not involved in development) have found the code very easy to maintain and extend.

2. What's Supported by UMB Scheme

UMB Scheme is an implementation of the language described in Jonathan Rees and William Clinger (Editors), Revised (3.99) Report on the Algorithmic Language Scheme, Draft (August 1989).

All syntax, variables and procedures are implemented. Integers are implemented as fixnums and bignums, rationals as pairs of integers, (inexact) reals as double-precision floats, and (inexact) complex numbers as pairs of double-precision floats.

The following files are loaded in order at startup:

If the variable SCHEME_INIT is set in the user's environment by executing

```
setenv SCHEME_INIT file
```

then file is loaded.

If SCHEME_INIT is not set and if a file *.scheme* exists in the user's home directory then it is loaded.

The files named as optional arguments are loaded from left to right.

The primitive procedure *edit* may be used for editing files during a scheme session.

```
(edit filename)
(edit)
```

Filename is a string object specifying the file to be edited. If no filename is given then that file that was most recently edited is assumed. The editor used is taken from the shell variable, EDITOR, in the user's environment; if this variable is not set then *vi(1)* is used by default. The user can make sure EDITOR is always set by putting a setenv in his *.login* file; e.g.

```
setenv EDITOR /usr/ucb/emacs
```

Upon leaving the editor, that file specified by filename is automatically loaded using the primitive procedure *loadv*, which causes the interpreter output to be sent to the current output port, normally the user's terminal. If verbose loading is not desired use the commands

```
(edits filename)
(edits)
```

causing filename to be loaded silently by the primitive procedure *load*.

Load and *loadv* can be used to load any file in silent or verbose mode respectively:

```
(load filename)
(loadv filename).
```

UMB Scheme has *property lists:*

```
(put symbol property-name object)
(get symbol property-name)
```

where *property-names* are symbols.

UMB Scheme has a simple debugger. Throughout a session one is in one of two modes: *top-level mode* or *debugging mode.* In general, one works in top-level mode. If the debugger has been turned on, an error raised during an evaluation (or an explicit call to break) causes a *break* which puts the user into debugging mode. The user can place explicit calls to break or error in his code:

```
(break obj ...)  ; Print the objects and break the evaluation.
(error obj ...)  ; Print the objects and raise an error.
```

In debugging mode, certain primitives apply for finding the cause of the offense. Notice that syntax errors cause a *reset*, returning the user to top-level mode. When the debugger is turned off, all errors simply cause a reset to top-level mode.

```
(debug) ; turn on the debugger
(debug-off) ; turn off the debugger.
```

NB: When the debugger is turned on, UMB Scheme is no longer properly tail-recursive as required by the language definition. For this reason, the debugger is turned *off* by default. One can insure the debugger is always turned on by putting a call to debug in the Scheme Init file (e.g. .scheme).

Any scheme expression may be evaluated in debugging mode. It is evaluated in the environment that existed when the break occurred in the top level computation; this makes it easy to find the bindings for local variables. In addition, the following primitives apply.

```
(reset) ; Return to the top-level read-eval-print loop.
Control-D ; Typing Control-D causes a (reset).

(show-env) ; Show the bindings of all local environments.
(show-env k) ; Show-env for only the k most recent frames.
(show-globals) ; Show bindings for all user-defined globals.
(show-proc-env proc) ; Show a procedure's environment.
(how symbol) ; Show the expression causing symbol's binding.

(where) ; Show an enumerated backtrace of the
; computation being debugged.
(where k) ; Show the most recent k steps of the backtrace.

(go obj) ; Resume the computation being debugged,
; substituting the value of obj for the most recent
; step (as indicated by a call to where).
(go k obj) ; Likewise, but obj is substituted for the k-th step
; as enumerated in the backtrace by (where).
```

Tracing a procedure involves interrupting evaluation when either the procedure is about to be applied or the procedure is about to return with a value. Upon such an interruption, the call or the returned value is printed, and the user is put in debugging mode.

```
(trace proc ...)  ; Trace named procedures.
(trace) ; Trace all procedures.
(untrace proc ...)  ; Cancel tracing for named procedures.
(untrace) ; Cancel tracing for all procedures.
```

```
Control-D ; Resume the computation interrupted in the trace.
```

Stepping through a computation involves interrupting evaluation at every k-th expression, for a given k. The expression in question is printed and the user is put in debugging mode.

```
(step k) ; Interrupt evaluation at every k-th expression.
(step 0) ; Inhibit stepping altogether.

Control-D ; Resume the evaluation broken in the stepping.
```

Notice that, when stepping or tracing is in effect, any one of a number of events (the application of a procedure or the k-th expression being reached) will interrupt evaluation. Since typing Control-D resumes the interrupted computation, one can step through such a computation by repeatedly typing Control-D.

Finally, errors or explicit calls to break arising while in debugging mode simply leave the user in debugging mode. Unlike some other implementations, UMB Scheme does not support nested debugging sessions. (Keep it simple.)

3. Remarks on Performance

So far as speed is concerned, UMB Scheme loses to MIT's C Scheme in pure evaluation but performs significantly better than C Scheme when it comes to compile-load-and-go.

For example, consider the following *tak* benchmark code from Richard P Gabriel, *Performance and Evaluation of Lisp Systems*, MIT Press, 1985.

```
; tak
; function-call-heavy; test of function call and recursion

(define (tak x y z)
  (if (not (< y x))
      z
      (tak (tak (- x 1) y z)
   (tak (- y 1) z x)
   (tak (- z 1) x y))))

(tak 18 12 6)
```

On our Sun 3/140, C Scheme gets through this in 75 seconds while UMB Scheme takes 98 seconds; in this instance C Scheme runs faster.

But consider a definition-heavy file of code such as the following.

```
(define (fac x) (if (= x 0) 1 (* x (fac (- x 1))) ))
(fac 5)

(define (fac x) (if (= x 0) 1 (* x (fac (- x 1))) ))
(fac 5)

(define (fac x) (if (= x 0) 1 (* x (fac (- x 1))) ))
(fac 5)

;; ...  1000 copies
```

To get through these 1000 definitions of (and calls to) *fac*, C Scheme takes 432 seconds but UMB Scheme takes only 19 seconds; in this instance UMB Scheme runs 22 times faster than C Scheme.

Therefore, UMB Scheme appears to do much better in a heavy compile-load-and-go environment (such as one would find in an undergraduate languages course) than in raw computation.

Space is more difficult to address when comparing Scheme systems since it depends on one's choice for a heap size. UMB Scheme's load module has a size of 216K bytes; C Scheme's load module is 279K bytes. As they are currently configured, UMB Scheme runs in a 1600K region as compared to C Scheme's 3944K (as measured by top(1) on Unix). UMB Scheme's storage allocator begins with a small (200K byte) heap and increases heap size as necessary (and as permitted by the operating system) at garbage collection time.

4. Contributors to UMB Scheme

- The interpreter is based on the Explicit-Control Evaluator described in the textbook: Harold Abelson and Gerald Jay Sussman, *Structure and Interpretation of Computer Programs*, MIT Press, Cambridge, Massachusetts, 1985.
- Bill Campbell was the primary author and project manager.
- The idea of organizing the control of the interpreter around objects comes from Richard Schooler.
- Karl Berry and Kathy Hargreaves helped with the initial programming.
- Bill McCabe coded some performance improvements, particularly lexical variable addressing.
- Mary Glaser, Tim Holt, Long Nguyen and Thang Quoc Tran worked on numbers.
- Ira Gerstein and Jeyashree Sivasubram help bring UMB Scheme up to R4RS standard.
- Barbara Dixey, Susan Quina and Bela Sohoni coded some additional performance improvements, handling varying numbers of arguments to functions in C.
- Many undergraduate students at UMB wrote test programs and offered suggestions for improvement.

5. Compiling and Installing UMB Scheme

UMB Scheme has been installed successfully using both *cc* and *gcc* (the Gnu C compiler) on the following machines and operating systems:

- Sun 3 running Sun Unix 3.4 and 3.5.
- Sun 386i running Sun Unix 4.0.1.
- DEC Vax 11/750 running Unix 4.3 BSD.

UMB Scheme was written so as to be easily ported to other systems. Some modifications may be required.

To install UMB Scheme one of the systems listed above, simply go into the distribution directory and do the following.

1. Compile UMB Scheme using the '`Makefile`' provided. If you have the Gnu C Compiler, *gcc*, then simply type

   ```
   make
   ```

   or,

   ```
   make CC=gcc
   ```

   Otherwise, to use *cc*, type

   ```
   make CC=cc CFLAGS=-O
   ```

   This will compile all necessary source files and create the executable file '`scheme`'.
2. Install the executable file in some public bin, e.g.

   ```
   cp scheme /usr/local/bin/scheme
   ```

3. Install the UMB Scheme *standard prelude* in the public library.

   ```
   mkdir /usr/local/lib/scheme
   cp prelude.scheme /usr/local/lib/scheme
   ```

This pathname is wired into UMB Scheme. If necessary, you can change the definition of `STANDARD_PRELUDE_FILENAME` in the source file '`steering.c`'.

4. Copy the manual page to the appropriate directory, e.g.

```
cp scheme.l /usr/spool/man/man1
```

Some shops maintain a separate '`manl`' directory for local manual pages; consult your local system administrator.

That should do it!

6. Using UMB Scheme

Invoking UMB Scheme is straightforward; simply type

```
scheme [names of any scheme files to be loaded]
```

To get out of UMB Scheme type `Control-D`.

See the man pages (scheme.l) for details.

7. What To Do About Bugs

UMB Scheme has been used by many undergraduate students and several bugs have been exposed and repaired. No doubt, bugs remain. The author would appreciate hearing about any bugs found (and any fixes made).

Send reports to


```
bill@cs.umb.edu
```


or, by postal service, to


```
Bill Campbell
Department of Mathematics and Computer Science
University of Massachusetts at Boston
Harbor Campus
Boston, MA 02125

617-287-6449
```

8. Distribution files

```
Distribution Files for UMB Scheme Version 1.1

README              This list of files.
scheme.texinfo      UMB Scheme Release Notes.
Makefile            For compiling UMB Scheme.
prelude.scheme      The Scheme prelude, primitives loaded at start.
scheme.1            UMB Scheme manual page (in man(1) format).


Source files:

portable.h          Portability considerations.

steering.c          UMB Scheme steering -- including main().
steering.h

debug.c             Debugger steering and debug primitives.
debug.h

architecture.c      Registers, stacks, heap, name handling.
architecture.h

object.c            Scheme objects.
object.h

io.c                Read, display, print, file handling.
io.h

compiler.c          Compiler: Expression -> Graph.
compiler.h

eval.c              Eval().
eval.h

primitive.c         (Non-numeric) primitives implemented in C.
primitive.h

number.c            Number primitives.
number.h

bignum.c            Bignum (integers of arbitrary magnitude).
bignum.h

complex.c           Complex number support.
complex.h

fixnum.c            Fixnum (smaller integers) support.
```

```
fixnum.h

real.c              Real number (C double) support.
real.h

rational.c          NB: rationals not yet implemented (skeletons).
rational.h

See the Release Notes (scheme.texinfo) for installation.
```

Table of Contents