

Chez Scheme Version 7.3 Release Notes

Copyright © 2007 Cadence Research Systems

All Rights Reserved

February 2007

1. Overview

This document outlines the changes made to *Chez Scheme* for Version 7.3 since Version 7.0.

Version 7.3 is available for the following platforms:

- Intel 80x86 Linux, threaded and nonthreaded
- Intel 80x86 Windows 95/98/ME/NT/2000/XP, threaded and nonthreaded
- Apple PowerPC Mac OS X 10.4, threaded and nonthreaded
- Apple Intel Mac OS X 10.4, threaded and nonthreaded
- Intel 80x86 FreeBSD, threaded and nonthreaded
- Sun Sparc Solaris, 32-bit threaded and nonthreaded
- Sun Sparc Solaris, 64-bit threaded and nonthreaded

This document contains three sections describing significant (1) functionality changes, (2) bugs fixed, and (3) performance enhancements. A version number listed in parentheses in the header for a change indicates the first minor release or internal prerelease to support the change.

More information on *Chez Scheme* and *Petite Chez Scheme* can be found at <http://www.scheme.com>, and extensive documentation is available in *The Scheme Programming Language, 3rd edition* and the *Chez Scheme Version 7 User's Guide*.

2. Functionality Changes

2.1. Thread-safe compilation and loading (7.3)

Various changes have been made to support the concurrent use of the compiler and loader, e.g., via `eval`, `compile`, `compile-file`, and `load` in multiple threads.

Compiled code loaded or evaluated by a given thread can be used reliably in that thread or threads subsequently forked, directly or indirectly, by the thread. It cannot be used reliably in other threads, since the data (in this case, the machine code produced by the compiler or loaded from a compiled file) written by one thread is not necessarily available immediately in other threads.

2.2. Source-path handling (7.3)

The `with-source-path` procedure no longer adds the `"/` prefix when a file is found in the current directory, even if `."` is one of several included in list of source directories, i.e., the list value of the `source-directories` parameter. It also treats the empty directory `"` as equivalent to `."` rather than as equivalent to `"/`.

The `load` procedure and `include` syntax now record as part of the loaded code's inspector information the file name given to them rather than the full path derived via a search of the source directories, to avoid the unintentional inclusion of host-machine paths in code that might be run on other systems.

The inspector and various error handlers now try harder to locate source files from recorded inspector information. For absolute pathnames starting with a / (or \ or a directory specifier under Windows), they try the absolute pathname first, then look for the last (filename) component of the path in the list of source directories. For pathnames starting with ./ (or .\ under Windows) or ../ (or ..\ under Windows), they look in "." or ".." first, as appropriate, then for the entire .- or ..-prefixed pathname in the source directories, then for the last (filename) component in the source directories. For other (relative) pathnames, they look for the entire relative pathname in the list of source directories, then the last (filename) component in the list of source directories.

2.3. `scheme.h` now C++ compatible (7.3)

The "C" modifier is now inserted before extern function declarations in `scheme.h` when the `__cplusplus` C preprocessor variable is set, to allow the entries declared in the include file to be used directly from C++.

2.4. Fixnum-only vectors (fxvectors) (7.3)

Support for "fxvectors," i.e., vectors of fixnums has been added, with operators that parallel the vector operators: `fxvector`, `make-fxvector` `fxvector?`, `fxvector-length`, `fxvector-ref`, `fxvector-set!`, `fxvector-fill!`, `fxvector-copy`, `list->fxvector`, and `fxvector->list`. Fxvectors are written with the `#vfx` prefix in place of the `#` prefix for vectors, e.g., `#vfx(1 2 3)` or `#10vfx(2)`. The `read-token` procedure can now return the two new fxvector tokens `vfxparen` and `vfxnparen` corresponding to the vector tokens `vparen` and `vnparen`.

2.5. `quasiquote` allocation guarantee (7.3)

The `quasiquote` form now guarantees that new pairs or vectors will be allocated any time a nonempty `unquote` or `unquote-splicing` form is used, even if the unquoted object is itself a constant. For example, while `'(a . b)` is equivalent to `'(a . b)` and returns a constant pair (the same one each time the `quasiquote` expression is evaluated, e.g., in a loop or procedure called multiple times) `'(, 'a . , 'b)` is equivalent to `(cons 'a 'b)` so that a new pair is allocated each time the `quasiquote` expression is evaluated.

2.6. `record-reader` extension (7.3)

The `record-reader` procedure now allows the first argument to be a record-type descriptor when second is `#f` so that the association can be removed by passing in the record-type descriptor as well as by passing in the record name.

2.7. Intel Mac support (7.2)

Support for running *Chez Scheme* under Mac OS 10.4 on Intel Macs has been added. The nonthreaded version uses the machine type "i3osx," and the threaded version uses the machine type "ti3osx."

2.8. Threaded PPC Mac support (7.2)

Support for running the threaded version of *Chez Scheme* under Mac OS 10.4 on PPC Macs has been added. The threaded version uses the machine type "tppcosx."

2.9. `vector-set-fixnum!` procedure (7.2)

A new procedure, `vector-set-fixnum!`, has been added. It works just like `vector-set!` but requires the new value (third argument) to be a fixnum. It is faster to store a fixnum than an arbitrary value, since the system has to record potential assignments from older to younger objects to support generational garbage collection. Care must be taken to ensure that the argument is indeed a fixnum, however; otherwise, the collector may not properly track the assignment. The primitive performs a fixnum check on the argument except at optimization level 3.

2.10. Fasl write of eq hash tables (7.2)

The representation of eq hash tables has been altered to allow them to be written using `fasl-write`, provided that they keys and values are suitable for `fasl-write`.

2.11. FreeBSD 'x86 support (7.1)

Support for running *Chez Scheme* under FreeBSD on Intel 'x86 architectures has been added. The non-threaded version uses the machine type "i3fb," and the threaded version uses the machine type "ti3fb."

2.12. Exec shield support (7.1)

Support for running *Chez Scheme* under Linux kernels with the "exec shield" feature enabled has been added.

2.13. `quasisyntax` (7.1)

New `quasisyntax`, `unsyntax`, and `unsyntax-splicing` syntactic forms have been added. A `quasisyntax` form is like as `syntax` form except that the portions encapsulated within an `unsyntax` or `unsyntax-splicing` form are evaluated and their values inserted into the output, as with `quasiquote`. Hash-backquote (`#'`), hash-comma (`#,`), and hash-comma-at (`#,@`) abbreviations may be used by analogy with the similar `quasiquote` abbreviations.

2.14. `syntax->vector` (7.1)

The new procedure `syntax->vector` takes a syntax object representing a vector-structured form and returns a vector of syntax-objects, each representing the corresponding subform of the input form, in a manner similar to the existing `syntax->list` procedure.

2.15. MacOS X `dlopen` support (7.1)

The foreign interface, including `load-shared-object`, now use the `dlopen` C library function and related features recently added to MacOS X for loading foreign code and looking up foreign entry points.

2.16. Universal foreign-callable support (7.0a)

Support for `foreign-callable` in Version 7.0 and prior releases was limited to the Intel Windows and Linux environments (threaded and nonthreaded). `foreign-callable` is now supported for the threaded and nonthreaded Solaris (32- and 64 bit) and PowerPC MacOS X.

2.17. New command-line parameter (7.0a)

The existing `command-line-arguments` parameter is set to a list of the command-line arguments by the default value of the `scheme-script` parameter whenever a Scheme shell script is run. The new `command-line` is similar, but is set to include as well the name of the script as the first element. Thus, `(car (command-line))` can be used to determine the script, and `(cdr (command-line))` can be used to determine the command-line arguments.

2.18. Socket example (7.0a)

The socket example found in `examples/socket.ss` in the release directory has been made more robust. The updated code also appears in the *Chez Scheme Version 7 User's Guide*.

3. Bug Fixes

3.1. PowerPC generic and fixnum addition and subtraction (7.3)

The compiler no longer generates code that uses the PowerPC `mcrxr` instruction in the implementation of the fixnum and generic addition and subtraction operators. The instruction is now an optional part of the PowerPC architecture and is not supported by some PowerPC implementations, notably the processors used in the Apple G5. Instead of passing along an illegal instruction trap, the O/S silently implements the instruction in software at a cost hundreds of times greater than the cost of a typical instruction. With a different mechanism now in place, most programs will run noticeably faster, with some possibly running more than twice as fast.

3.2. Large allocation requests (7.3)

A bug that sometimes resulted in an invalid memory reference when a large allocation request was made, e.g., when an attempt was made to allocate a vector or string with length equal to the most-positive fixnum, has been fixed.

3.3. bytes-allocated (7.3)

A bug that caused the `bytes-allocated` procedure to return a negative number for heaps exceeding 2GB on 32-bit machines has been fixed.

3.4. letrec* internal compiler error (7.3)

A bug that sometimes resulted in an internal compiler error when a reference to a possibly undefined variable occurred in a `letrec` or `letrec*` expression appearing in a `letrec*` binding has been fixed.

3.5. Cache flush or memory protection error (7.3)

A bug that on rare occasions resulted in a bad `mprotect` argument error or an invalid memory reference while synchronizing data and instruction caches has been fixed.

3.6. `Sactivate_thread` bug (7.3)

A bug in `Sactivate_thread` that could result in memory faults and other undesirable behavior when the initial (main) thread is actively running Scheme code has been fixed.

3.7. Multiple-value handling bug (7.1)

A bug in the inliner's handling of `call-with-values` that could result in an invalid memory reference at higher optimization levels has been fixed.

3.8. Removed `open-input-file` exclusive option (7.1)

The `exclusive` option has been eliminated from the file open operations, including `open-input-file`, since the underlying mechanism used on many operating systems does not support exclusive access to read-only files. The option remains for output and input/output files.

3.9. Unbound meta variables (7.1)

A bug that caused variables defined with `meta define` to be unbound in environments other than the interaction environment has been fixed.

3.10. Unbound compiler-related variables (7.1)

The variables `make-boot-header` and `compile-script` previously evaluated to the value `#<unbound>` in *Petite Chez Scheme*. They are now bound to procedures that report that the compiler is not loaded, as with other compiler-related variables. [This bug dated back to Version 6.9c.]

3.11. Logical test operations (7.0a)

A bug in the optimizer's treatment of `logtest`, `logbit?`, `fxlogtest`, and `fxlogbit?`, which caused it to treat their return values as always true in test contexts, has been fixed. [This bug dated back to Version 6.9d.]

3.12. Format “\$” bug (7.0a)

A bug that caused `format` to reject exact real numbers has been fixed, and `format` also now produces a more appropriate error message when passed a non-real number. [This bug dated back to Version 6.9b.]

3.13. `thread?` for nonthreaded versions (7.0a)

The `thread?` procedure, like other threading procedures, is no longer defined in the nonthreaded versions of the system. [This bug dated back to Version 6.5.]

4. Performance Enhancements

4.1. PowerPC performance (7.3)

The performance of `fixnum` and generic arithmetic on some PowerPC systems has been improved. See Section 3.1.

4.2. Vector improvements (7.3)

When reading a vector with an unspecified length, e.g., `#(1 2 3)` rather than `#3(1 2 3)`, the reader now performs less memory allocation, resulting in less overall storage management cost and less overall overhead, especially for files with numerous small vectors. Filling of vectors either by `make-vector` or `vector-fill!` is now more efficient as well. (Additional performance can be gained via `fxvectors` or the `vector-set-fixnum!` procedure, described in Sections 2.4 and 2.9.)

4.3. Guardian registration (7.3)

Registering older-generation objects with an older-generation guardian now results in less collection overhead, e.g., when older objects are reregistered as part of the implementation of the transport guardians described in the 1993 ACM PLDI paper on Guardians.

4.4. Record accessors (7.1)

The speed of record field accessors and mutators at optimization levels 2 and below has been improved by inlining the actual memory loads and stores.

4.5. `foreign-callable` for nonthreaded versions (7.0a)

The speed of a call into Scheme via `foreign-callable` has been improved slightly for nonthreaded versions of the system. The difference is likely to be noticeable only for calls to Scheme procedures that execute quickly.