# Chez Scheme Version 7.4 Release Notes

December 2007

## 1. Overview

This document outlines the changes made to *Chez Scheme* for Version 7.4 since Version 7.0.

Version 7.4 is available for the following platforms:

- Intel 80x86 Linux, threaded and nonthreaded
- Intel 80x86 Windows, 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
- Intel 80x86 OpenBSD, 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. Expression Editor (7.4)

A new *expression editor* has been added and runs by default whenever *Chez Scheme* or *Petite Chez Scheme* is run in a shell window. The expression editor permits entry and editing of single- and multiple-line expressions, automatically indents expressions as they are entered, and supports name-completion based on the identifiers defined in the interactive environment. The expression editor also maintains a history of expressions typed during and across sessions and supports tcsh-like history movement and search commands. Details can be found in *Chez Scheme Version 7 User's Guide* and the manual page.

### 2.2. Profiling support (7.4)

The new procedure `profile-dump-html` allows profiling information to be written to HTML files, including one file (profile.html) containing summary information and one additional file for each source file showing a listing of the source file, color-coded according to relative execution counts.

The parameter `profile-palette` may be used to control the colors used, including the number of colors.

The new procedure `profile-dump-list` returns profiling information as a list of entries, each of which records the execution count, pathname, beginning file position, and ending file position for each profiled block of code.

Details can be found in *Chez Scheme Version 7 User's Guide*.

These procedures are all available in *Petite Chez Scheme* as well as in *Chez Scheme*, as are `profile-dump` and `profile-clear`, which were previously available only in *Chez Scheme*. This allows profiling information to be gathered for code compiled with profiling enabled in *Chez Scheme* and subsequently run in *Petite Chez Scheme*.

## 2.3. New Eq Hash Table Support (7.4)

Several R6RS hashtable procedures have been added:

```
make-eq-hashtable
hashtable?
hashtable-size
hashtable-ref
hashtable-set!
hashtable-delete!
hashtable-contains?
hashtable-update!
hashtable-copy
hashtable-clear!
hashtable-keys
hashtable-entries
hashtable-mutable?
```

The R6RS procedures below have *not yet* been added:

```
make-eqv-hashtable
make-hashtable
hashtable-equivalence-function
hashtable-hash-function
equal-hash
string-hash
string-ci-hash
symbol-hash
```

Thus, hashtable support is presently limited to "eq" hashtables.

Weak eq hashtables are supported via the following procedures:

```
make-weak-eq-hashtable
eq-hashtable-weak?
```

In addition, several procedures specific to eq hashtables have been added:

```
eq-hashtable?
eq-hashtable-cell
eq-hashtable-contains?
eq-hashtable-delete!
eq-hashtable-ref
eq-hashtable-set!
eq-hashtable-update!
```

These are like their generic counterparts above but work only for eq hash tables and have somewhat less overhead, especially at optimize-level 3.

The old eq hash-table implementation is still available through the existing procedures `make-hash-table`, `get-hash-table`, `put-hash-table!`, and `remove-hash-table`. These will be changed to use the new implementation in a subsequent release.

The new implementation of eq hashing is "generation friendly," which means there is no overhead for older objects that do not move on a given collection. The old implementation rehashes all objects on the first failed access after any garbage collection.

With the new implementation, hash tables also shrink when many keys have been deleted (or for weak hash tables, become inaccessible). In the old implementation, once a hash table has grown to a certain size, it never shrinks.

Details can be found in *Chez Scheme Version 7 User's Guide*.

## 2.4. Terminating `equal?` (7.4)

The `equal?` procedure now terminates with behavior required by R6RS. It is coded in such a way that it remains fast for trees and acyclic graphs while also being fast when cycles are present.

## 2.5. Miscellaneous R6RS functionality (7.4)

Several R6RS primitives and syntactic forms have been added:

- `assp`, `memp`, and `remp`
- `datum->syntax` (synonym for `datum->syntax-object`) and `syntax->datum` (`syntax-object->datum`)
- `exact` (synonym for `inexact->exact`) and `inexact` (`exact->inexact`)
- `boolean=?` and `symbol=?`
- `eof-object`
- `for-all` (synonym for `andmap`) and `exists` (`ormap`)
- `let*-values`
- `list-sort`, `vector-sort`, and `vector-sort!`
- `vector-map` and `vector-for-each`
- `find`, `filter`, and `partition`
- `fold-left`, `fold-right`
- `cons*` (synonym for `list*`)

Details can be found in *Chez Scheme Version 7 User's Guide*.

## 2.6. Filesystem operators (7.4)

The `file-exists?` procedure now takes an additional *follow?* argument: if true (the default), `file-exists?` follows symbolic links; otherwise it does not.

The new procedures `file-regular?`, `file-directory?`, and `file-symbolic-link?` have been added to query the type of a file. The first two accept a string pathname and the optional *follow?* argument; the latter accepts only a string pathname and never follows symbolic links.

The procedure `delete-file` now takes an additional *error?* argument; if true, `delete-file` signals an error if the file does not exist or cannot be deleted; otherwise (the default), it returns the boolean value `#t` on success and `#f` on failure.

The related procedure `delete-directory` has been added, with the same interface.

The new procedure `directory-list` takes a string pathname naming a directory (folder) and returns a list of the files found in that directory.

The new procedure `directory-separator?` accepts a character argument and returns `#t` if the character is a valid directory separator and `#f` otherwise. The character `#\/` is a valid directory separator on all current machine types and `#\\` is a valid directory separator under Windows.

The new procedure `directory-separator` returns the preferred directory separator for the current machine type, which is `#\\` for Windows and `#\/` for other systems.

The new procedures `path-extension`, `path-last`, `path-parent`, `path-root` each take a string argument and return a component of the path represented by the string.

Details can be found in *Chez Scheme Version 7 User's Guide*.

## 2.7. Date and time procedures (7.4)

A set of procedures for handling times and dates have been added. The set is a subset of those listed in SRFI 19. It includes the following time-handling procedures.

```
current-time, make-time, time?
time-type, set-time-type!
time-second, set-time-second!
time-nanosecond, set-time-nanosecond!
time=?, time<?, time<=?, time>=?, time>?
```

It also includes the following date-handling procedures.

```
current-date, make-date, date?
date-second, date-nanosecond, date-minute, date-hour
date-day, date-month, date-year
date-week-day, date-year-day,
date-zone-offset
```

The time types `utc`, `monotonic`, `duration`, `process`, and `thread` are supported. Details can be found in *Chez Scheme Version 7 User's Guide*.

## 2.8. New `meta-cond` syntax (7.4)

A new syntactic form, `meta-cond`, has been added. It is similar to `cond` except the tests are evaluated when the form is expanded. `meta-cond` may be used to choose, at expansion time, from among a set of possible forms, e.g., based on optimization level, machine type, or system configuration. Details can be found in *Chez Scheme Version 7 User's Guide*.

## 2.9. Windows scheme.h, dllimport, and static libraries (7.4)

Under Windows, defining `SCHEME_IMPORT` before including scheme.h now causes scheme.h to declare its entry points using `extern declspec (dllimport)` rather than `extern declspec (dllexport)` (which remains

the default). Not defining `SCHEME_IMPORT` and instead defining `SCHEME_STATIC` causes scheme.h to declare exports using just `extern`. The static kernel csv*v*_s.lib, (where *v* is the Scheme version) and custom_s.obj are built using `SCHEME_STATIC`.

## 2.10. OpenBSD 'x86 support (7.4)

Support for running *Chez Scheme* under OpenBSD on Intel 'x86 architectures has been added. The non-threaded version uses the machine type "i3ob," and the threaded version uses the machine type "ti3ob."

## 2.11. Windows environment procedures (7.4)

Under Windows, `getenv` now looks for an environment binding using the `GetEnvironmentVariable` Windows API function before the standard C library `getenv`. Similarly, `putenv` now sets the environment variable using both `SetEnvironmentVariable` and the C library `putenv`. This should circumvent some problems where two versions of the C library, each of which has its own copy of the environment, are used in the same process.

## 2.12. `iota` and `enumerate` (7.4)

The new procedure `iota` takes a nonnegative integer $n$ and returns a list of integers starting at 0 and ending at $n-1$. The new procedure `enumerate` takes a list $ls$ and returns a list of integers starting at 0 and ending at $n-1$, where $n$ is the length of the list.

## 2.13. Prettier unique names for gensyms (7.4)

Unique names now start with a lower-case letter and contain only lower-case letters and digits, with a hyphen separating the portion of the name that is common to all names created during a session and the portion that is unique to the specific name. This allows them to be printed without the enclosing vertical bars and generally makes them more readable (and appear less like cartoon cussing).

## 2.14. `read-token` and datum comments (7.4)

The procedure `read-token` no longer scans past datum comments but rather returns a token with type `quote` and value `datum-comment`. This allows programs that use `read-token` to treat a commented-out datum as it would any other datum. The expression editor exploits this feature to indent commented-out expressions properly.

## 2.15. Inspector interface (7.4)

The interactive inspector `ref` (`r`) command now allows selection of frame and closure elements by name as well as number, when names appear. If multiple elements have the same name, the name will select one, and the others must be selected by number.

The `forward` (`f`) and `back` (`b`) commands may now be used to move backward and forward among record fields after a record field is selected by number.

The inspector `eval` (`e`), `=>`, and `set!` commands employ the new expression editor when the argument expression is not typed on the same line as the command.

Record inspector objects now accept a `length` message, which returns the number of fields.

## 2.16. `fresh-line` and `set-port-bol!` (7.4)

The new procedure `fresh-line` has been added. It is like `newline` but has no effect if the port is believed already to be positioned at the beginning of a line. The related procedure `set-port-bol!` has been added, which marks a port as being positioned at the beginning of a line. (This is primarily useful when dealing with generic ports.)

## 2.17. Nested transcripts (7.4)

It is now possible to run a transcript within a transcript using either `transcript-on` or `transcript-cafe`.

## 2.18. 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.

## 2.19. 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.

## 2.20. 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++.

## 2.21. 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 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`.

## 2.22. `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.

## 2.23. `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.

## 2.24. 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.”

## 2.25. 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.”

## 2.26. `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.

## 2.27. Fasl write of eq hashtables (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`.

## 2.28. FreeBSD ’x86 support (7.1)

Support for running *Chez Scheme* under FreeBSD on Intel ’x86 architectures has been added. The nonthreaded version uses the machine type “i3fb,” and the threaded version uses the machine type “ti3fb.”

## 2.29. Exec shield support (7.1)

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

### 2.30. `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.

### 2.31. `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.

### 2.32. 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.

### 2.33. 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.

### 2.34. 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.

### 2.35. 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. Error-case bug in heap search-path processing (7.4)

A broken `fprintf` control-string used in a warning about extra percent signs in a heap search path has been fixed.

### 3.2. `mutex-acquire` compiler problem (7.4)

The compiler mistakenly assumed that `mutex-acquire` always returns true, but this is not necessarily the case when the optional *block?* flag is false. This problem has been fixed.

### 3.3. `Sscheme_deinit` problem (7.4)

`Sscheme_deinit` was not properly recording the fact that the system was deinitialized so that a subsequent call to `Sscheme_init` would fail. This problem has been fixed.

### 3.4. 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.

### 3.5. 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.

### 3.6. `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.

### 3.7. `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.

### 3.8. 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.

### 3.9. `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.

### 3.10. 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.

### 3.11. 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.

### 3.12. 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.

### 3.13. 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.]

### 3.14. 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.]

### 3.15. 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.]

### 3.16. `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. Storage management (7.4)

The default setting of `collect-trip-bytes` has also been increased from $2^20$ to $2^22$, reflecting the larger sizes of contemporary memories. This makes collections occur less frequently. The number of generations has been increased from 4 to 5 so that collection of long-lived objects occurs even less frequently.

### 4.2. PowerPC performance (7.3)

The performance of fixnum and generic arithmetic on some PowerPC systems has been improved. See Section 3.4.

## 4.3. 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 2.21 and 2.26.)

## 4.4. 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 the 1993 ACM PLDI paper on Guardians.

## 4.5. 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.

## 4.6. `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.