# Chez Scheme Version 7.9.3 Release Notes

August 2009

## 1. Overview

This document outlines the changes made to *Chez Scheme* for Version 7.9.3 since Version 7.4, most of which are focused on converting *Chez Scheme* into an implementation of the new Scheme standard described in Revised[6] Report on Scheme.

Version 7.9.3 is available for the following platforms:

- Linux x86, threaded and nonthreaded
- Linux x86_64, threaded and nonthreaded
- MacOS X x86, threaded and nonthreaded
- MacOS X x86_64, threaded and nonthreaded
- MacOS X PowerPC, threaded and nonthreaded
- Windows x86, threaded and nonthreaded
- OpenBSD x86, threaded and nonthreaded
- OpenBSD x86_64, threaded and nonthreaded
- FreeBSD x86, threaded and nonthreaded
- Solaris 32-bit Sparc, threaded and nonthreaded
- Solaris 64-bit Sparc, 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.

Version 7.9.3 is a prerelease of Version 8 and has not been fully tested or tuned. Several additional changes are expected in Version 8, and some may be incompatible with Version 7.9.3. More information on *Chez Scheme* and *Petite Chez Scheme* can be found at http://www.scheme.com, and extensive documentation is available in the recently published *The Scheme Programming Language, 4th edition* (available directly from MIT Press or from online and local retailers) and the draft *Chez Scheme Version 8 User's Guide*. Online versions of both books can be found at http://www.scheme.com.

## 2. Functionality Changes

### 2.1. New `compile-program` procedure (7.9.3)

A new `compile-program` procedure has been added. It is similar to `compile-script` but differs in that it implements the semantics of RNRS top-level programs, while `compile-script` implements the semantics of the interactive top-level. The resulting compiled program will also run faster than if compiled via `compile-file` or `compile-script`.

## 2.2. `#!fold-case` and `#!no-fold-case` (7.9.3)

The reader now recognizes two new comment directives, analogous to the existing `#!r6rs` and `#!chezscheme` directives. If `#!fold-case` has been seen (more recently than `#!no-fold-case`) in an input stream, subsequent reads from port are case-insensitive, i.e., symbols and character names are case-folded, as if by `string-foldcase`. If `#!no-fold-case` has been seen (more recently than `#!fold-case`) in an input stream, subsequent reads from the port are case-sensitive. If neither has been seen in an input stream, case-sensitivity is determined as before by the `case-sensitive` parameter, whose value defaults to `#t`. The case sensitive parameter is not consulted if either `#!fold-case` or `#!no-fold-case` has been seen. `#!fold-case`, `#!no-fold-case`, and `case-sensitive` are all ignored when vertical bars or slashes (excluding Unicode hex escapes) are seen. Like `#!r6rs` and `#!chezscheme`, `#!fold-case` and `#!no-fold-case` do not require delimiting, so `#!fold-caseaBc` reads as the symbol `abc`.

## 2.3. New mechanisms for specifying library directories and extensions (7.9.3)

The `--libdirs` option can be used to specify additional directories for `import` to search. The format of the `--libdirs` argument string used to specify directories is a sequence of substrings separated by a colon (:) or, under Windows, a semi-colon (;). If a colon (semi-colon under Windows) appears at the end of the string, the default libraries are searched as well (after the ones specified). Otherwise, the system searches only for those specified by the option argument. The default set of libraries can be determined by calling the Scheme parameter library-directories, without arguments, and the effect of specifying the `--libdirs` option is to set this parameter. If no `--libdirs` option appears and the CHEZSCHEMELIBDIRS environment variable is set, the string value of CHEZSCHEMELIBDIRS is treated as if it were specified by a `--libdirs` option.

Similarly, if a library upon which a top-level program depends has an extension other than one of the standard extensions, the `--libexts` option can be used to specify additional extensions to search. The format and treatment of its argument is the same as for the `--libdirs` option, the corresponding environment variable is CHEZSCHEMELIBEXTS, and the corresponding parameter is library-extensions.

The `library-directories` and `library-extensions` parameters have been extended to accept a string in the format described above, as an alternative to a list of strings naming individual directories.

## 2.4. New library extensions (7.9.3)

By default, `import` now searches for files with the extensions `".chezscheme.so"` and `".chezscheme.sls"` before files with other extensions to facilitate the use of libraries specialized to *Chez Scheme*. The default set of extensions can be overridden with the `library-extensions` parameter.

## 2.5. New `(chezscheme)` library (7.9.3)

The new `(chezscheme)` library is identical to the existing `(scheme)` library. The new name can be used to make clear that *Chez Scheme* extensions are involved.

## 2.6. New `--optimize-level` command-line option (7.9.3)

The new `--optimize-level` option sets the initial value of the optimize-level parameter to 0, 1, 2, or 3. The value is 0 by default.

## 2.7. Procedures for manipulating top-level keyword bindings (7.9.3)

The new procedure `define-top-level-syntax`, which accepts a symbol representing a keyword, a transformer, and an optional environment defaulting to the current interaction environment, can be used to

establish a top-level keyword binding in an interactive environment, just as `define-top-level-value` can be used to establish a top-level variable binding. The new procedure `top-level-syntax`, which accepts a symbol and an optional environment, can be used to retrieve the transformer associated with a keyword in an environment.

### 2.8. `file-buffer-size` parameter (7.9.3)

A new parameter, `file-buffer-size`, has been added. This parameter controls the size of a buffer used when a file is opened with anything by buffer mode `none`. The default value of this parameter is currently set at the minimum of 4096 and the value of the underlying operating system's stdio `BUFSIZ` constant.

### 2.9. `current-exception-state` parameter (7.9.3)

The new `current-exception-state` may be used to save and later restore the state of the exception system.

### 2.10. `fork-thread` and multiple values (7.9.3)

`fork-thread` now permits its thunk argument to return zero values or more than one value.

### 2.11. `system` return value (7.9.3)

The `system` procedure, which runs a command in a newly forked process and waits for it to terminate, now raises an exception if the subprocess creation fails, if the exit status of the forked process cannot be determined, or if the forked process is terminated by a signal. Otherwise, it returns the exit status of the child process.

### 2.12. `abort` optional argument (7.9.3)

The `abort` procedure now takes an optional argument, like `exit`, that may be used to specify the exit code for the Scheme process.

### 2.13. New inspector commands (7.9.3)

The interactive inspector now supports `reset` (`r`) and `abort` (`a`) commands, which may be used to reset to the current REPL or abort from the system.

For characters and strings, the interactive inspector also supports a new `unicode` command that displays the Unicode scalar values of the character or string elements.

### 2.14. `inspect/object` extensions (7.9.3)

The continuation and code inspector-object source-path message now returns a position as well as a file name if the file name is known but the (unmodified) file cannot be found. Previously, it returned a single value in this case, the file name.

### 2.15. New I/O procedures and support for nonblocking I/O (7.9.3)

New input procedures `get-string-some`, `get-string-some!`, and `get-bytevector-some!` have been added. `get-string-some` is like the R6RS `get-bytevector-some` but operates on textual input ports and re-

turns a string rather than a bytevector. `get-string-some!` and `get-bytevector-some!` are like the R6RS `get-string-n!` and `get-bytevector-n!` except they may return less than the requested count.

The new procedures `put-string-some` and `put-bytevector-some` are like `put-string` and `put-bytevector` except they do not guarantee to write the entire string or bytevector, and they return a count of the actual number of elements written.

The procedure `set-port-nonblocking!` may be used to put a port into nonblocking mode. When a port is in nonblocking mode, `get-string-some` may return an empty string, `get-bytevector-some` may return an empty bytevector, and `get-string-some!` and `get-bytevector-some!` may return 0, in each case indicating that no input is available, i.e., an attempt to read would block. Similarly, `put-string-some` and `put-bytevector-some` may return 0, indicating that an attempt write to the port would block.

The procedure `port-nonblocking?` may be used to determine if a port is in nonblocking mode. The predicates `port-has-port-nonblocking??` and `port-has-set-port-nonblocking!?` return `#t` if the nonblocking status of a port may be queried or set and `#f` otherwise. They should be called before attempting to use the `set-port-nonblocking!` or `port-nonblocking?` procedures.

## 2.16. Expression editor delimiter handling (7.9.3)

The expression editor no longer changes the corresponding close delimiter when an open delimiter is inserted into existing code, since this often led to surprising results.

## 2.17. `file-directory?` and `file-exists?` change under Windows (7.9.3)

The `file-directory?` and `file-exists?` predicates now return `#t` under Windows for existing drive names like `c:`, mounted volumes like `//server/mount`, and directories specified with (or without) a trailing slash (and regardless of whether forward or backward slashes are used).

## 2.18. R6RS (7.9.2)

(Initial support was included in Version 7.9.1.)

Version 7.9.3 implements the entire language of the Revised[6] Report on Scheme (R6RS) and associated libraries, including all of the syntactic requirements and each export of the standard libraries, namely:

```
(rnrs)
(rnrs arithmetic bitwise)
(rnrs arithmetic fixnums)
(rnrs arithmetic flonums)
(rnrs base)
(rnrs bytevectors)
(rnrs conditions)
(rnrs control)
(rnrs enums)
(rnrs eval)
(rnrs exceptions)
(rnrs files)
(rnrs hashtables)
(rnrs io ports)
(rnrs io simple)
(rnrs lists)
(rnrs mutable-pairs)
(rnrs mutable-strings)
(rnrs programs)
(rnrs r5rs)
```

```
(rnrs records procedural)
(rnrs records syntactic)
(rnrs records inspection)
(rnrs sorting)
(rnrs syntax-case)
(rnrs unicode)
```

Top-level programs are supported via the `--program` command-line option, i.e., the shell command:

`scheme --program` *pathname*

runs the top-level program contained in the file named by *pathname*. To create an executable R6RS top-level program on Unix-based system, insert:

```
#! /usr/bin/scheme --program
```

at the front of the top-level program (adjusting the path for `scheme` or replacing `scheme` with `petite` as appropriate) and give the file execute permissions. To make use of existing executable top-level programs containing the "shebang" line

```
#! /usr/bin/env scheme-script
```

recommended by R6RS nonnormative appendix D.1, create a copy or symbolic link of the Scheme executable to `scheme-script` and copies or symbolic links of the "scheme" or "petite" boot file to `scheme-script.boot`. Place the former somewhere in the path searched by `/usr/bin/env` and the latter in a standard directory for *Chez Scheme* boot files, e.g., the same place where `scheme.boot` and/or `petite.boot` already reside. (This may already be done as part of the installation process.)

## 2.19. Interaction environment and R6RS (7.9.2)

The default interaction environment used for any code that occurs outside of an R6RS top-level program or library (including such code typed at the REPL or loaded from a file) contains all of the bindings of the `(scheme)` library (or `scheme` module, which exports the same set of bindings). This set contains a number of bindings that are not in the R6RS libraries. It also contains a number of bindings that extend the R6RS counterparts in some way and are thus not strictly compatible with the R6RS bindings for the same identifiers. To replace these with bindings strictly compatible with R6RS, simply import the `rnrs` libraries into the interaction environment by typing the following into the REPL or loading it from a file:

```
(import
  (rnrs)
  (rnrs eval)
  (rnrs mutable-pairs)
  (rnrs mutable-strings)
  (rnrs r5rs))
```

To obtain an interaction environment that contains all *and only* R6RS bindings, use the following.

```
(interaction-environment
  (copy-environment
    (environment
      '(rnrs)
      '(rnrs eval)
      '(rnrs mutable-pairs)
      '(rnrs mutable-strings)
      '(rnrs r5rs))
    #t))
```

The read-eval-print loop (REPL) and files loaded using `load` or included using `include` also support various Chez Scheme lexical extensions, by default. To disable these extensions, `#!r6rs` can be inserted at the front of a file to be loaded or included or before the start of an expression typed into the REPL. The `#!r6rs` mode is implicit for libraries loaded as a result of an `import` form and for R6RS top-level programs. To enable *Chez Scheme* extensions in libraries and R6RS top-level programs, the prefix `#!chezscheme` can be used.

To be useful for most purposes, `library` and `import` should probably also be included, from the `(scheme)` library.

```
(interaction-environment
  (copy-environment
    (environment
      '(rnrs)
      '(rnrs eval)
      '(rnrs mutable-pairs)
      '(rnrs mutable-strings)
      '(rnrs r5rs)
      '(only (scheme) library import))
    #t))
```

It might also be useful to include `debug` in the set of identifiers imported from `(scheme)` to allow the debugger to be entered after an exception is raised.

Most of the identifiers bound in the default interaction environment that are not strictly compatible with the R6RS are variables bound to procedures with extended interfaces, i.e., optional arguments or extended argument domains. The others are keywords bound to transformers that extend the R6RS syntax in some way. This should not be a problem except for R6RS programs that count on exceptions being raised in cases that coincide with the extensions. For example, if a program passes the `=` procedure a single numeric argument and expects an exception to be raised, it will fail in the initial interaction environment because `=` returns `#t` when passed a single numeric argument.

The procedures that are not strictly compatible include the following, which return `#t` for one argument (while the R6RS versions require two or more):

`<`, `<=`, `=`, `>`, `>=`, `char<=?`, `char<?`, `char=?`, `char>=?`, `char>?`, `string<=?`, `string<?`, `string=?`, `string>=?`, `string>?`, `char-ci<=?`, `char-ci<?`, `char-ci=?`, `char-ci>=?`, `char-ci>?`, `string-ci<=?`, `string-ci<?`, `string-ci=?`, `string-ci>=?`, `string-ci>?`;

the following, which are extended to accept zero or more arguments:

`fx*`, `fx+`, `exit`;

the following, which is extended to accept one or more arguments:

`fx-`;

the following, which allow radixes 3, 5–7, 9–15, and 17–36:

`string->number`, `number->string`;

the following, which accepts either one or two arguments and allows the first argument to represent a definition:

`eval`;

the following, for which the argument is optional and defaults to the current output port:

`flush-output-port`;

the following, which are parameters and thus may be used to alter the encapsulated value:

`command-line`, `current-error-port`, `current-input-port`, `current-output-port`, `standard-error-port`, `standard-input-port`, `standard-output-port`;

the following, which accept an "options" symbol or list:

`call-with-input-file`, `call-with-output-file`, `open-input-file`, `open-output-file`,

`with-input-from-file`, `with-output-to-file`;

and the following, which accept optional arguments:

`delete-file`, `dynamic-wind`, `file-exists?`, `record?`.

The only keyword that is not strictly compatible is `syntax-rules`, which allows fenders.

For details on the `(scheme)` library versions of these procedures and syntactic forms, see the *Chez Scheme Version 7 User's Guide*.

## 2.20. Incompatible Changes (7.9.2)

Although we have strived to maintain backward compatibility with earlier versions of Scheme wherever possible, the following incompatible changes have been made to support R6RS.

- The reader now recognizes R6RS hex Unicode escapes in characters, strings, and symbols. In some cases, characters, strings, and symbols containing next Unicode escapes would have parsed as some different character, string, or symbol.
- The setting of `(case-sensitive)` is now `#t` by default, so the case is distinguished in symbols and character names.
- The C function `Sstring_value` has been eliminated because strings are no longer represented as byte strings but rather using an internal representation that supports Unicode. Routines that accepted Scheme strings as arguments and treated them as nul-terminated byte strings should instead be passed Scheme bytevectors instead. Bytevectors are nul-terminated (with the nul byte being stored in the first byte beyond the last element of the bytevector and not counted in the length), and the address of the start of a bytevector's data can be obtained with `Sbytevector_data`, which is analogous to the old `Sstring_value`.
- For any foreign-procedure argument declared as a `string`, a bytevector or a string may be passed. If a bytevector is passed, the foreign procedure receives the address of the first byte of the bytevector. Bytevectors are nul-terminated as described above to facilitate the use of bytevectors as arguments to foreign procedures that expect nul-terminated strings.

  If a string is passed, it is copied to a freshly allocated bytevector (since strings are no longer represented internally as byte arrays) using the equivalent of `string->utf8`. Modifications of the object by the foreign procedure affect the freshly allocated bytevector only and are not reflected back to the original string.

  In most cases, it is preferable to pass bytevectors rather than strings to avoid the copying overhead and so that modifications made by the foreign procedure are visible to the caller. Even when the argument is represented by a string in Scheme and no modifications are made by the foreign procedure, it may still be appropriate for the caller to convert the string to a bytevector explicitly when the implicit UTF-8 conversion is not appropriate.
- R6RS does not treat complex numbers with inexact zero (`+0.0` or `-0.0`) imaginary parts as real numbers, nor does it treat infinities and nans as rational numbers. Thus, `integer?`, `rational?`, and `real?` now return `#f` for complex numbers with inexact zero imaginary parts, while `integer?` and `rational?` now return `#f` for `+inf.0` and `-inf.0`, `rational?` now returns `#f` for `+nan.0`. Correspondingly, procedures that accept only integer, only rational, and only real arguments now reject those that no longer qualify as integer, rational, or real.
- Binary data (bytes) cannot be read from or written to textual ports, and textual data (characters) cannot be read from or written to binary ports. Source (textual) and compiled (binary) code can thus no longer be loaded from the same file. Similarly, (binary) fasl data can no longer be combined with textual data, and fasl data must be read with `fasl-read`, not `read`. The port argument of `fasl-write`

must now be a binary output port and is no longer optional, since there is no current binary output port.

Applications requiring the mixing of textual and binary data should use a binary port and convert the portions of the file representing textual data to and from characters or strings using an appropriate conversion procedure. It is is also possible, when the textual portions are represented using characters in the Latin-1 character set, to open the port as a textual port using a transcoder constructed from the `latin-1` codec with eol-style `none`, then convert the binary portions to and from bytes using `char->integer` and `integer->char`.

- The ordering of *old* and *new* ids in the module `import rename` syntax is now *old* first, *new* second to match the `import` syntax for libraries.
- The `letrec` and `letrec*` forms assume the continuations of right-hand-side expressions are invoked at most once.
- The `(scheme)` library and default interaction-environment bindings for `record-type-descriptor`, `record-type-field-names`, `record-type-name`, and `record-type-mutable?` have their R6RS semantics, which conflicts with the old *Chez Scheme* semantics. Bindings for these identifiers that are compatible with the old *Chez Scheme* semantics, along with bindings for the related procedures `record-field-accessible?`, `record-field-accessor` and `record-field-mutator`, as well as `record-type-symbol` and `record-type-field-decls`, which have all been superseded by R6RS versions, are available only in the `(scheme csv7)` compatibility library.
- The `error-handler` and `warning-handler` parameters have been eliminated because their semantics are in direct conflict with the R6RS exception handling mechanism.
- The `(scheme)` library and thus interaction-environment binding for `error` is compatible with the R6RS error, which means the second argument is not treated as a format string. When the argument is a format string, use `errorf` instead, which is like `error` except the second argument *is* treated as a format string (and the additional arguments as arguments to be consumed while formatting). For consistency, the same change has been made for `warning`, and the corresponding procedure `warningf` has also been added. Similarly, the new procedure `assertion-violationf` is like the R6RS `assertion-violation` but treats its second argument as a format string.
- The thread-system `condition?` procedure has been renamed `thread-condition?`, since the R6RS `condition?` predicate is used to test for the condition objects that are passed to exceptions.
- As required by R6RS, of the standard data types, only booleans, numbers, characters, strings, and bytevectors count as expressions if they are not quoted. In particular, vectors and the empty list must be quoted. Of the nonstandard data types, fxvectors, `#!eof`, and `#!bwp` need not be quoted. Previously, *Chez Scheme* treated any unquoted value other than a pair or symbol as a literal.
- Also, as required by R6RS, the hash character ( `#` ) is now a delimiter, so, for example, `abc#def` is no longer a valid symbol. Although some non-r6rs symbols, like `1+`, are still accepted by default on input, i.e., except after `#!r6rs`, they are printed using r6rs syntax, e.g., `\x31;+`. In particular, while `1+` and `1-` are still defined as variables bound to increment and decrement procedures, their names print in a funny way.

## 2.21. C library routines for 32- and 64-bit integers (7.9.2)

Four new C library operators have been added for converting 32- and 64-bit integers from their Scheme representations:

- `Sinteger32_value(`$x$`)` returns the 32-bit integer value of the Scheme exact integer $x$;
- `Sunsigned32_value(`$x$`)` returns the 32-bit unsigned integer value of the Scheme exact integer $x$;

- `Sinteger64_value(`$x$`)` returns the 64-bit integer value of the Scheme exact integer $x$; and
- `Sunsigned64_value(`$x$`)` returns the 64-bit unsigned integer value of the Scheme exact integer $x$.

An exception is raised if a 32-bit value is not in the range $-2^{31}$ through $2^{32}-1$ or a 64-bit values is not in the range $-2^{63}$ through $2^{64}-1$.

Similarly, four new C library operators have been added for converting 32- and 64-bit integers to their Scheme representations:

- `Sinteger32(`$x$`)`,
- `Sunsigned32(`$x$`)`,
- `Sinteger64(`$x$`)`, and
- `Sunsigned64(`$x$`)`.

Each returns a Scheme exact integer whose value is $x$. The arguments are signed or unsigned 32- or 64-bit C integers, as appropriate.

## 2.22. New nonstandard character names (7.9.2)

The characters `#\nel` and `#\ls`, representing the Unicode next-line and line-separator characters, are recognized by the reader (except after `#!r6rs`).

## 2.23. Fewer "invalid context for definition" errors (7.9.2)

The expander now expands library body forms left-to-right even after discovering an apparent expression so that a spelling error in a definition keyword (e.g., `defnie` for `define`) results in an "unbound identifier" error rather than an "invalid context for definition" error for some subsequent definition.

## 2.24. Windows library change (7.9.1/7.9.2)

Windows builds of *Chez Scheme* now link against `msvcr90.dll`. Static libraries built using libcmt are also available.

## 2.25. Saved heaps not currently supported (7.9.1)

None of the operating systems upon which *Chez Scheme* runs provide the guarantees about storage allocation needed to restore a saved heap with absolute addresses, and while restoration used to work on some operating systems even without the guarantee, this is generally no longer the case as the operating systems are randomizing the addresses of data and even code for security purposes. We are looking into ways to address this problem, including relocatable saved heaps, but for the present, support for saved heaps has been removed from the system.

## 2.26. Thread-safe console ports (7.9.1)

Unbuffered file output ports are no longer thread-safe. The default console output port remains thread-safe, however, and the default console input port is now thread-safe as well. There is a substantial performance cost in both cases, so programs that need faster I/O should create their own ports and either avoid using them from multiple threads or arrange for appropriate synchronization. (This is relevant only for the threaded versions of Chez Scheme.)

## 2.27. `input-port-ready?` and `char-ready?` (7.9.1)

A new `input-port-ready?` procedure has been that is similar to the old `char-ready?` but works on both binary and textual ports. The `char-ready?` procedure still works for textual ports.

Under Windows, these procedures properly handle files and ports, while in previous releases, `char-ready?` always returned `#t`. In cases where the system cannot determine if input is ready, e.g., for a console port that has no data already buffered, each of these procedures raises an exception so that the application can control what happens in such cases. Thus, these procedures either (a) return `#f` if no input is ready and the port is not at end of file, (b) return `#t` if input is ready or the port is at end of file, or (c) raise an exception if the ready status cannot be determined.

## 2.28. X86_64 Support (7.9.1)

Support for running *Chez Scheme* with 64-bit pointers on the x86_64 architecture under Linux, MacOS X, and OpenBSD with machine types a6le, a6osx, and a6ob for nonthreaded and ta6le, ta6osx, and ta6ob for threaded, has been added. C code intended to be linked with these versions of the system should be compiled using the Gnu C compiler's `-m64` option, which may or may not be the default.

## 2.29. Additional byte-vector operations (7.9.1)

In addition to the standard R6RS bytevector operations, *Chez Scheme* also supports the following procedures:

`(bytevector->s8-list` *bytevector*`)` ⇒ *list*
`(s8-list->bytevector` *s8-list*`)` ⇒ *bytevector*
`(bytevector` *fill* `...)` ⇒ *bytevector*

where each element of *s8-list* is an exact integer value ranging from −128 through +128 and each *fill* is an exact integer value ranging from −128 through +255.

As with vectors and bytevectors, the reader allows an optional length to appear in the bytevector prefix, e.g.:

```
(bytevector? #10vu8(0))
```

is a bytevector containing 10 zero bytes. (This cannot be used following `#!r6rs`, which limits the syntax recognized by the reader to R6RS-only features.)

The `read-token` procedure returns token types `vu8paren` and `vu8nparen` for bytevector prefixes, analogous to the `vparen` and `vnparen` token types for vector prefixes.

The C library interface supports four new bytevector operators:

- `Smake_bytevector(`*len*`,`*n*`)`, which returns a new bytevector of length *len* with each element set to *n*;
- `Sbytevector_u8_ref(`*bv*`,`*i*`)`, which returns the *i*th byte of *bv*;
- `Sbytevector_u8_set(`*bv*`,`*i*`,`*n*`)`, which sets the *i*th byte of *bv* to *n*; and
- `Sbytevector_data(`*bv*`)`, which returns a pointer to *bv*'s data.

## 2.30. `meta-cond` generalization (7.9.1)

The `meta-cond` syntax has been extended to allow it to be used in definition contexts. `meta-cond` still expands into `(void)` if no clause's test evaluates to true and no `else` clause is present, however, which is not suitable in a definition context. To avoid this problem, use an explicit `else` clause and expand into `(begin)`.

### 2.31. Default heap/boot search path (7.9.1)

The default heap and boot search path on Unix-based systems now starts with `~/lib/csv%v/%m` if the home directory can be determined. The default heap and boot search path is still determined by a registry setting under Windows.

### 2.32. Tilde-prefixed paths under Windows (7.9.1)

The filename prefix `~/` now expands to the user's home directory under Windows, as on Unix-based systems, if the `HOMEDRIVE` and `HOMEPATH` environment variables are set.

### 2.33. Months range from 1 to 12 (7.5)

The month field of a date record, e.g., one returned by `current-date`, runs from 1 through 12 rather than 0 through 11, for compatibility with SRFI 19.

### 2.34. Monotonic time (7.5)

Requesting monotonic no longer fails on Linux systems that do not support monotonic clocks; on such systems, the real-time clock is now used instead. All of the high-precision time procedures now fall back on older time mechanisms when requesting high-precision timers fails.

### 2.35. New and altered expression-editor key bindings (7.5)

The expression editor now binds the common delete-key sequence (`Esc[3~`) to `ee-delete-char` and escape followed by the same sequence to `ee-delete-sexp`. The effect of this is that the delete key should now delete forward rather than backward. The backspace key should still delete backward.

New bindings have also been added for the home and end keys of certain terminal emulators, including the Gnome terminal.

### 2.36. Top-level value handling (7.5)

The `top-level-value` and `top-level-bound?` procedures now recognize as bound a variable assigned by a file compiled in one session and loaded into another session—even if the variable was not defined.

## 3. Bug Fixes

### 3.1. Work-around for automatic margins on MacOS X Terminal (7.9.3)

The expression editor disables automatic margins (i.e., automatic line wrapping) when possible so it can use the last column without scrolling the display. Unfortunately, when automatic margins are disabled in the MacOS X Terminal application, they are disabled simultaneously for all windows, not just for the current window, causing problems when the expression editor is running in one window and long lines are printed in another. The expression editor now disables automatic margins only temporarily while writing text to the last column, reducing (though not entirely eliminating) the possibility that other windows are affected.

### 3.2. Invalid memory reference using hashtables (7.9.3)

A bug that caused corruption of the heap and typically resulted in an "unrecoverable invalid memory reference" while using the R6RS eq hashtable interface has been fixed.

### 3.3. Bug in `format` tabulation (7.9.3)

A bug resulting in an "undefined for zero" error when the optional column width specifier in a tabulate directive is zero, e.g., `~1,0t`, has been fixed.

### 3.4. Expansion internal error (7.9.3)

A bug in the expander that sometimes resulted in an internal error ("source-wrap ae/x mismatch") has been fixed.

### 3.5. Expression editor datum comment handling (7.5)

The expression editor no longer returns end-of-file for an entry containing only a single commented-out datum.

### 3.6. Expression editor clipboard bug under Windows (7.5)

The expression editor now properly closes the clipboard after a paste (control-v) operation, i.e., no longer prevents new items from being copied to the clipboard.

### 3.7. Bug in `ash` (7.5)

A bug that caused `ash` to return the wrong result for word-sized or greater negative (right) shifts of large powers of two has been fixed.

### 3.8. Bug in `inexact` (7.5)

A bug that could cause `inexact` (aka `exact->inexact`) to improperly round an exact value halfway between two floating-point values has been fixed.

### 3.9. Bug in handling of `compile` (7.5)

A bug in the compiler that caused it to treat the return value of `compile` as true in test context, no matter what it actually returns, has been fixed.

## 4. Performance Enhancements

### 4.1. Reduced `letrec*` undefined-variable checking overhead (7.9.3)

A improvement in the handling of `letrec*`, which also affects internal definitions, including library and top-level program definitions, causes the compiler to produce fewer checks for undefined variables. Improvements in safe code can be substantial (up to 15% or so), but the amount of improvement is highly dependent on the structure of the code. Unsafe code (code compiled at optimize-level 3) is not affected, since these checks are not performed in unsafe code.

## 4.2. Improved register assignment for x86 (7.9.1)

A better choice of register assignments has resulted in measurable improvement in x86 performance. Actual improvements differ from one program to another, and we've seen anything from -9% decreases in speed to 36% increases, with increases of 5-15% appearing typical.

## 4.3. Improved recursive bindings (7.9.1)

The compiler now generates better code for some `letrec` expressions, `letrec*` expressions, and bodies containing internal definitions by taking advantage of the R6RS restriction that right-hand-side expressions should not return to their continuations multiple times.

## 4.4. Improved procedural record interface (7.9.1)

Record constructors, predicates, accessors, and mutators created with the procedural record interface are often faster than before because the compiler now tracks record-type information through to the sites where constructors, predicates, accessors, and mutators are created in order to generate specific and inlinable code for them. The effects of this optimization can be seen with `expand/optimize`. (This optimization is not performed by *Petite Chez Scheme*.)