# R6RS Status Report


## Kent Dybvig, Will Clinger, Matthew Flatt, Mike Sperber, and Anton van Straaten


**June 13, 2006**

## 1. Overview

This status report describes the current state of the R$^6$RS standardization effort. It covers principles we have outlined to guide the effort, decisions we have made to date, our work in progress, and the process by which we intend to complete the Revised$^6$ Report on Scheme.

## 2. Change Log

Here is a brief overview of the important changes to this document since the February 2006 version.

Section 4.2 (new): describes the forms which portable code can take.

Section 4.3: now lists `interaction-environment` and top-level definitions and expressions among the eliminated features and `scheme-report-environment` and `null-environment` among those that have been relegated to an R$^5$RS compatibility library.

Section 4.4 lists several additional changes. (All but the first four listed are new.)

Section 4.5 lists several added features. (All but the first six listed are new.)

Section 4.6 lists two new features to be added: scripts and a byte-vector datatype. Read/write invariance is now covered in Section 4.4.

Section 4.7 lists several newly reaffirmed features. (All but the first three listed are new.)

Section 4.8 lists several features that are officially not under consideration for R$^6$RS. (All but the first four listed are new.)

Section 5 (new) describes the editors' commitment to provide reference implementations for the major subsystems included in R$^6$RS.

Section 6.2 documents that we have now withdrawn the record SRFI as planned, after receiving valuable community input, and that support for records will be based on this SRFI. It also describes decisions we have made regarding some issues left open by the SRFI.

Section 6.3 documents that we have now withdrawn the Unicode SRFI as planned, after receiving valuable community input, and that support for Unicode will be based on this SRFI.

Section 6.4 documents that the arithmetic SRFI has undergone revisions.

Section 6.5 documents that we have decided to base the R$^6$RS I/O system on SRFI's 34 and 35.

Section 6.6 documents that we have decided to base the R$^6$RS I/O system on SRFI's 79 and 81.

Section 6.8 (new) documents that we have decided to base R$^6$RS byte vectors on SRFI 74.

Section 6.9 now lists enumerations and `eval` among possible features and changes. Some of the previously listed items are no longer under consideration and are now listed as "beyond R$^6$RS" in Section 4.8:

- external representation for (possibly cyclic) graph structures
- syntax for the eof-object, if any
- `cond-expand`
- homogeneous numeric vectors

- support for regular expressions
- formatted output
- adding support for weak pointers
- support for gensyms and uids

Some are now mentioned in Section 4.7:

- making quotation of empty list optional (reaffirmed that `()` is not a valid expression)

Some are listed as changes to be made or features added or to be added:

- `#t`, `#f`, and characters must be followed by a delimiter (Section 4.4)
- `case-lambda` (Section 4.5)
- bitwise operations on exact integers (Section 6.4)
- adding a void object to replace the "unspecified value" (as "unspecified" rather than "void"; Sections 4.4 and 4.5)
- `let-values` or other multiple-value binding construct(s) (both `let-values` and `let*-values`; Section 4.5)

Section 7 now lists Sperber and Clinger as the editors in charge of byte vectors.

# 3. Guiding Principles

To help guide the standardization effort, the editors have adopted a set of principles, presented below. They are, like R$^6$RS itself, a work in progress and still subject to change.

Like R$^5$RS Scheme, R$^6$RS Scheme should:

- derive its power from simplicity, a small number of generally useful core syntactic forms and procedures, and no unnecessary restrictions on how they are composed;
- allow programs to define new procedures and new hygienic syntactic forms;
- support the traditional s-expression representation of program source code as data;
- make procedure calls powerful enough to express any form of sequential control, and allow programs to perform non-local control operations without the use of global program transformations;
- allow interesting, purely functional programs to run indefinitely without terminating or running out of memory on finite-memory machines;
- allow educators to use the language to teach programming effectively, at various levels and with a variety of pedagogical approaches; and
- allow researchers to use the language to explore the design, implementation, and semantics of programming languages.

In addition, R$^6$RS Scheme should:

- allow programmers to create and distribute substantial programs and libraries, e.g., SRFI implementations, that run without modification in a variety of Scheme implementations;
- support procedural, syntactic, and data abstraction more fully by allowing programs to define hygiene-bending and hygiene-breaking syntactic abstractions and new unique datatypes along with procedures and hygienic macros in any scope;
- allow programmers to rely on a level of automatic run-time type and bounds checking sufficient to ensure type safety while also providing a standard way to declare whether such checks are desired; and
- allow implementations to generate efficient code, without requiring programmers to use implementation-specific operators or declarations.

In general, R$^6$RS should include building blocks that allow a wide variety of libraries to be written, include commonly used user-level features to enhance portability and readability of library and application code, and exclude features that are less commonly used and easily implemented in separate libraries.

R$^6$RS Scheme should also be backward compatible with programs written in R$^5$RS Scheme to the extent possible without compromising the above principles and future viability of the language. With respect to future viability, we operate under the assumption that many more Scheme programs will be written in the future than exist in the present, so the future programs are those with which we must be most concerned.

# 4. Decisions

This section outlines the decisions made to date.

## 4.1. Language structure

The R$^6$RS language consists of a core language and set of additional libraries.

The following features are definitely in the core language:

- *none yet identified*

The following features are definitely not in the core language:

- `delay` and `force`
- hash tables (see Section 4.6)
- `case-lambda` (see Section 4.5)
- `when` and `unless` (see Section 4.5)

## 4.2. Programs

R$^6$RS programs exist only in the form of portable libraries and scripts. A library consists of a single top-level library form. Portable libraries may import variable and keyword bindings from other libraries (standard or user-defined) and may export variable and keyword bindings. A portable script consists of a standard script header and a single top-level library. All definitions and expressions must appear within a library form; R$^6$RS has no notion of a top-level definition or expression. The `eval` procedure may, however, will likely allow the evaluation of an expression (but not a definition) within the scope of a specified set of library bindings.

## 4.3. Features eliminated

The following features of R$^5$RS have been eliminated.

- `transcript-on` and `transcript-off`
- `interaction-environment`
- top-level definitions and expressions (see Section 4.2)

The following features of R$^5$RS are deprecated but will be available in an R$^5$RS compatibility library:

- `scheme-report-environment`
- `null-environment`

## 4.4. Changes

The following syntactic and semantic changes have been made to existing features.

- Syntax is case sensitive.
- Internal defines now follow `letrec*` semantics.
- There is now a single unique end-of-file object.
- Continuations created by `begin` must accept any number of values. (This was optional in R$^5$RS.)
- Any character or boolean must be followed by a delimiter.
- The new syntax `#!r6rs` is treated as a declaration that a source library or script contains only r6rs-compatible lexical constructs. It is otherwise treated as a comment by the reader.
- An implementation may or may not signal an error when it sees `#!`*symbol*, for any symbol *symbol* that is not `r6rs`. Implementations are encouraged to use specific `#!`-prefixed symbols as flags that subsequent input contains extensions to the standard lexical syntax.
- All other lexical errors must be signaled, effectively ruling out any implementation-dependent extensions unless identified by a `#!`-prefixed symbol.
- Expressions that would have evaluated to some "unspecified value" in R$^5$RS evaluate to a new unique (in the sense of `eq?`) "unspecified" value.
- Character comparison routines are now n-ary. (This was optional in R$^5$RS.)
- The *in* and *out* thunks of a `dynamic-wind` are considered "outside" of the `dynamic-wind`; that is, escaping from either does not cause the *out* thunk to be invoked, and jumping back in does not cause the *in* thunk to be invoked.
- Most standard procedures are required to raise a specific condition (in the default "safe" mode) when given invalid inputs, except in certain specific cases where the answer can be determined in spite of the invalid input and the additional work involved may be extraordinary. For example, `map` must raise an exception if its first argument is not a procedure or if its other arguments are not (proper) lists of the same length. On the other hand, `(memq` $x$ $ls$`)` must raise an exception only if, before it finds a tail of $ls$ whose car is eq? to $x$, it encounters a non-list tail or cycle in $ls$.
- When given a value $x$ that can be represented as a datum, `write` must print $x$ as a datum for which `read` would produce a value that is equivalent (in the sense of equal?) to $x$ (read/write invariance).
- Every symbol, string, character, and number that can be created via standard operators has at least one standard datum representation.

## 4.5. Features added

The following features have been added:

- `letrec*` (`letrec` with left-to-right evaluation order)
- block comments bracketed by `#|` and `|#`
- expression comments prefixed by `#;`
- matched square brackets ("`[`" and "`]`"); equivalent to matched parentheses for list data and list-structured forms
- symbols of the form `->`*subsequent** are now allowed
- `eof-object` constructor to obtain the end-of-file object
- `unspecified` procedure that returns the unspecified value
- `let-values` and `let*-values` multiple-value binding forms
- `(define var)` syntax: abbreviation for `(define var (unspecified))`
- `when` and `unless` library syntax
- `case-lambda` library syntax
- `call/cc` as a second name for `call-with-current-continuation`
- Alan Bawden's PEPM '99 nested `quasiquote` extensions
- new list-processing procedures (inspired by SRFI 1): `exists`, `forall`, `fold-left`, `fold-right`, `filter`, `partition`, `iota`, `find`, `remq`, `remv`, `remove`, `generalized-member`, `generalized-remove`, and `generalized-assoc`

## 4.6. Features to be added

The following features will be added, but the details have yet to be fully worked out.

- top-level libraries
- record types and record definitions
- exception handling
- safe (default) and unsafe modes
- `syntax-case` macros
- hash tables (as a library)
- Unicode support
- new string escape characters, including `\n` for newline (part of Unicode support)
- byte-vector datatype and operations
- scripts

## 4.7. Reaffirmations

The following features of R$^5$RS are reaffirmed for R$^6$RS.

- support for multiple values
- unspecified evaluation order for applications, `let` bindings, and `letrec` bindings
- `set-car!` and `set-cdr!`
- `read-char` and `peek-char` return the eof object
- `(begin)` is still an invalid expression
- `case` still uses `memv`
- one-armed `if` remains in the language
- `append` copies all but last argument, even if last argument is `()`
- as an expression, `()` is invalid syntax
- `()` is still an invalid expression

## 4.8. Beyond R$^6$RS

The following features are definitely not under consideration for R$^6$RS. We encourage anyone interested in seeing any of these features in R$^7$RS to make concrete proposals via the SRFI process.

- processes
- network programming
- object-oriented programming
- box datatype
- formatted output
- graph printing (printed representation for shared structure and cycles)
- `rec` form, `(rec id e) => (letrec ([id e]) id)`
- vector-length prefix: `#`$n$`(`
- gensyms / uids
- external syntax for the eof object, e.g., `#!eof`
- external syntax for the unspecified value, e.g., `#!unspecified`
- SRFI 0 `cond-expand`
- homogeneous numeric vectors
- weak pointers
- support for regular expressions

# 5. Reference implementations

The editors will publish, along with the revised report proper, nonnormative, portable (with implementation-dependent hooks as necessary), and reasonably efficient reference implementations of the major subsystems of R$^6$RS, including the library, record, Unicode, arithmetic, exceptions, I/O, and macro subsystems. The editors may publish reference implementations of selected additional features as well.

# 6. Work in Progress

Most of the standardization effort is currently focused on several subsystems. Sections 6.1–6.8 list for each subsystem any informal requirements the editors have identified, the current status, and open questions.

In several cases, a subsystem is up for discussion as a SRFI in order to give the editors a chance to inform the community of the ongoing work and obtain valuable feedback from the community. The final mechanism adopted for R$^6$RS may, however, differ in minor or significant ways from the published SRFI.

A list of other items up for consideration is given in Section 6.9. These have not received as much attention to date, usually because they involve less complex or far-reaching changes or are considered to be of lower priority.

## 6.1. Libraries

Informal requirements: support distribution of portable libraries, support identification of library location, namespace management, export/import of macros, permit separate but dependent analysis and compilation, support generation of efficient compiled code, ability to define new libraries.

Support for libraries is under community discussion via SRFI 83 (R6RS Library Syntax). Two big issues have arisen: the need to clarify phases, e.g., for compile-time modules that import at compile-time, and how library names are written (coding as strings is controversial). Still up in the air are the extent to which the syntax of `import` and `export` forms is tied down, what built-in libraries besides `r6rs` there might be, and how to support subsetting and supersetting of libraries.

## 6.2. Records

Informal requirements: disjoint types, syntactic interface, mutable fields.

Support for records will be based on SRFI 76 (R6RS Records), which has now been withdrawn as planned after revisions based in part on community input. While the SRFI did not fully specify the generativity of ordinary record definitions, we have decided that they should be "run-time" generative unless declared nongenerative. We have also eliminated the restriction that the parent of a nongenerative record be a nongenerative record, and we decided to keep the "sealed" feature.

Additionally, we have decided to allow an implementation to treat any or all of its built-in types as records, i.e., `record?` may or may not return true for an object of a built-in type.

## 6.3. Unicode

Informal requirements: provision for Unicode characters and character syntax, Unicode strings and string syntax; Unicode character I/O; `integer->char` and `char->integer` are inverse operations and support Unicode-specific text encodings; write/read invariance for every datum, including symbols.

Support for Unicode will be based on SRFI 75 (R6RS Unicode data), which has now been withdrawn as planned after revisions based in part on community input.

## 6.4. Arithmetic

Informal requirements: support for IEEE zeros, infinities, and NaNs, clean up behavior of `eqv?` wrt numbers, fix certain arithmetic operations, transparency.

Changes for R$^6$RS arithmetic, including support for fixnum-specific, flonum-specific, and bitwise operators and IEEE arithmetic, are under community discussion via SRFI 77 (Preliminary Proposal for R6RS Arithmetic), which has recently been revised based in part on community input.

## 6.5. Exceptions

Informal requirements: clarify the meaning of "is an error," view exception handling as a means of communication between parts of the program.

The editors have decided to adopt SRFI 34 (Exception Handling for Programs) as the basis for the R$^6$RS exception-handling system and SRFI 35 (Conditions) as the basis for the R$^6$RS condition system.

## 6.6. I/O

Informal requirements: `read-byte` and `write-byte`, ports that support binary I/O, byte-vector datatype, block read/write operations.

The editors have decided to adopt SRFI 79 (Primitive I/O) and SRFI 81 (Port I/O) as the basis for the R$^6$RS I/O system.

The byte-vector datatype requirement is addressed by the binary block datatype (Section 6.8).

## 6.7. Macros

Informal requirements: specify expansion semantics, specify interaction with modules, allow procedural transformers, hygiene-breaking operations, maintain support for syntax-rules.

The editors have decided to adopt `syntax-case` as currently implemented in Chez Scheme and MzScheme, with various differences to be worked out by Dybvig and Flatt. Also, the underscore identifier ("_") will no longer be a pattern variable but instead a special identifier that matches any input, and underscore will be allowed in place of the keyword naming a macro in a `syntax-rules` pattern.

## 6.8. Binary block datatype

The editors have decided to adopt SRFI 74 (Octet-Addressed Binary Blocks) as the basis for byte-vector functionality in R$^6$RS.

## 6.9. Other possible changes

The following possible features and changes have been discussed without resolution.

- improving the semantics of `eqv?` and `equal?`
- support for file operations
- support system operations
- R$^5$RS compatibility library
- support for enumerations
- changes to `eval` to reflect the existence of libraries and other R$^6$RS changes

# 7. Completion Process

We intend to deliver a draft R$^6$RS to the Steering Committee by September 1, 2006. In order to meet this target, we plan to wrap up work on the various subsystems, decide on the core language/library split, and create a rough internal (editors only) draft of the R$^6$RS by mid-June.

For each of the subsystems, the core/library split, and the safe/unsafe mode mechanism and semantics, we have assigned a single editor to be responsible for ensuring progress. We have also assigned one or more additional editors to help. These assignments are shown below.

| **subsystem** | **primary editor** | **additional editors** |
|---|---|---|
| libraries | Flatt | Dybvig |
| records | Sperber | Dybvig, van Straaten |
| arithmetic | Clinger | Sperber |
| Unicode | Flatt | Clinger |
| macros | Dybvig | Flatt |
| exceptions | Sperber | Clinger |
| I/O | Sperber | van Straaten |
| byte vectors | Sperber | Clinger |
| core/library split | van Straaten | Dybvig |
| hash tables | van Straaten | Clinger |
| safe/unsafe mode | Clinger | Sperber |

As time permits, we will also discuss as a group the other possible features and changes described in Section 6.9, as well as additional ones that may arise, and decide which are to be incorporated into R$^6$RS.

Responsibility for making sure that the editors complete their work and communicate effectively lies with the chair (Dybvig) and responsibility for creating the R$^6$RS drafts lies with the project editor (Sperber).