R6RS Status Report

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

February 24, 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 R^{6}RS.

2. 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.

3. Decisions

This section outlines the decisions made to date.

3.1. Structural changes

R6RS will consist of a core language and set of separate libraries.

The following features are definitely in the core language:

- *none yet identified*

The following features are definitely in a separate library.

- `delay` and `force`
- hash tables (see Section 3.5)

3.2. Features eliminated

The following features have been eliminated.

- `transcript-on` and `transcript-off`

3.3. 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

3.4. 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
- allow symbols to start with `->`
- `eof-object` constructor to obtain the end-of-file object
- require continuations created by `begin` to accept any number of values

3.5. Features to be added

The following features will be added once the details have been 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)
- serialization (read-write invariance) for every datum (part of Unicode support)

3.6. 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!`

3.7. Beyond R^{6}RS

The following features are definitely not under consideration for R^{6}RS.

- processes
- network programming
- object-oriented programming
- box datatype

4. Work in Progress

Most of the standardization effort is currently focused on several subsystems: libraries, records, Unicode, arithmetic, exceptions, I/O, modules, and hash tables. Sections 4.1–4.7 list for each subsystem a set of 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 4.8. 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.

4.1. 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. 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 whether there is to be support for user-defined libraries.

4.2. Records

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

Support for records is under community discussion via SRFI 76. Still to be settled is whether generativity should be specified, e.g., as expand-time or run-time and also whether to elide or provide a rationale for the "sealed" feature.

4.3. 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 is under community discussion via SRFI 75. Open issues include what normalization and character representation to use. We will probably use normalization form "C," and Scheme characters will likely correspond to Unicode scalar values (which can be represented by a 21-bit fixed-length encoding, but other representations are also possible).

4.4. 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 are under community discussion via SRFI 77. There is general agreement to require the full tower and to require that real? implies an exact zero imaginary part. Among the open questions are whether fixnum, flonum, exact-only, and inexact-only operations should be in separate libraries rather than in the core language.

4.5. Exceptions

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

Proposals for this subsystem are currently under discussion. No R^{6}RS-specific SRFIs have been published, and no decisions have been made. There is, however, general agreement to use SRFI 34 as a basis for the R^{6}RS exception-handling system.

4.6. I/O

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

This subsystem actually addresses two separable issues here: potential additions changes to I/O and the inclusion of a byte-vector datatype. The byte-vector datatype is necessary to support block read/write operations.

Proposals for this subsystem are currently under discussion. No R^{6}RS-specific SRFIs have been published, and no decisions have been made.

4.7. 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 Dr. Scheme, 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.

4.8. Other possible changes

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

- external representation for (possibly cyclic) graph structures
- syntax for the eof-object, if any
- whether `#t`, `#f`, and characters must be followed by a delimiter
- `case-lambda`
- `cond-expand`
- improving the semantics of `eqv?` and `equal?`
- bitwise operations on exact integers
- homogeneous numeric vectors
- support for file operations
- support for regular expressions
- support system operations
- formatted output

- making quotation of empty list optional
- adding support for weak pointers
- adding a void object to replace the "unspecified value"
- support for gensyms and uids
- `let-values` or other multiple-value binding construct(s)
- R^{5}RS compatibility library

5. 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
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 4.8, 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).