The Revised R6RS Status Report

Marc Feeley
Université de Montréal
feeley@iro.umontreal.ca

Editors' note: This article is the revised version of the progress report submitted by the Scheme Language Editors Committee to the Scheme Language Steering Committee on September 2, 2004. It takes into account the Editors Committee meetings in Snowbird prior to the Scheme workshop. We have included the revised version in the workshop proceedings to represent the concluding presentation of the workshop on the state of the standardisation effort by the editors committee.

The members of the Scheme Language Editors Committee are:

Marc Feeley, editor in chief (Université de Montréal)
Will Clinger (Northeastern University)
Kent Dybvig (Indiana University)
Matthew Flatt (University of Utah)
Richard Kelsey (Ember Corporation)
Manuel Serrano (INRIA)
Michael Sperber (DeinProgramm)

The members of the Scheme Language Steering Committee are:

Alan Bawden (Brandeis University)
Guy L. Steele Jr. (Sun Microsystems)
Mitch Wand (Northeastern University)

–Waddell & Shivers

1 Creation of the Committees

At the 2003 Scheme workshop in November, the strategy committee (Alan Bawden, Will Clinger, Kent Dybvig, Matthew Flatt, Richard Kelsey, Manuel Serrano, Mike Sperber) was given a mandate to nominate a steering committee and an editors committee to work on the R6RS standard. In January 2004, the editors committee was nominated: Feeley (editor in chief), Clinger, Dybvig, Flatt, Kelsey, Serrano, and Sperber.

Fifth Workshop on Scheme and Functional Programming. September 22, 2004, Snowbird, Utah, USA.

2 Work Prior to the Snowbird Meetings

On January 19, a private mailing list was created to keep a record of the email exchanges between the editors. Although some editors suggested that a more open process would be desirable, we chose to keep this mailing list private to avoid outside interference and keep the process disciplined and focused. Sometime in the future the archive of the discussions will be made public so that the reasons for the design decisions are clear.

Because of the expected difficulty in managing productive discussions for a seven-member committee by email, we adopted some ground rules for ensuring progress. If an editor does not participate in an email discussion within a reasonable time limit (which was set to seven days), then the other editors may assume that editor does not have an opinion on the subject (or does not want to voice his opinion), and can be ignored (in the discussion, in a vote, *etc.*). I think this has been helpful to create a certain pressure to keep up-to-date in the discussion.

The subject of backward compatibility was also discussed early on. That is, will R5RS code work unchanged in an R6RS-compliant Scheme implementation? Our position is that backward compatibility is desirable but that there may be some incompatibilities (for example at the lexical syntax level) that prevent R5RS code from working under R6RS. Our first objective is to improve the Scheme language. Backward compatibility, while important, is a secondary objective.

We then set off on our first technical task: come up with a list of goals that is more precise than the one in the draft charter (which had four items: produce a core Scheme specification, define a module system, define a macro system, and designate library modules). Our plan was to use this list of goals (1) to organize the design process, and (2) to identify which changes were uncontroversial (and thus easier to standardize) and which would require considerable effort (and where consensus might not be achievable during the R6RS design process). The committees main goals for R6RS are to improve portability and consistency between implementations, so that a broad class of programs can be written using one implementation of Scheme and used without modification in another implementation of Scheme.

All the editors were polled to get a list of specific issues that they thought needed to be addressed in the R6RS design process (*i.e.*, features the committee should consider adding/removing). At the end of March, we had merged all the editors' lists into a single list with each editor's position. At this point, there had been very little technical discussion of these issues (on purpose), so that we

could order the issues and discuss them in a disciplined way. As suggested by Will Clinger, the list was organized into the following categories:

- Deletions of R5RS-Scheme features;
- Incompatible changes to R5RS Scheme;
- Extensions that could be entirely compatible with R5RS Scheme
  - but would break some implementation-specific extensions;
  - but would be controversial and aren't worth it;
  - that are controversial or difficult but necessary;
  - that are probably uncontroversial.

Below is the current list of issues, without each editor's position. Note that this list is still open to be expanded as new issues arise in the design process.

Deletions from R5RS

- remove `transcript-on` and `transcript-off`
- remove `force` and `delay`
- remove multiple values

Incompatible changes to R5RS

- make syntax case-sensitive

Extensions that would break implementation-specific features

- specify evaluation order
- support for processes
- support for network programming
- object-oriented programming
- external representation for records
- serialization

Extensions to R5RS (controversial and probably unnecessary)

- pattern matching / destructuring
- abstract data type for continuations
- composable continuations
- box types
- uninterned symbols
- extended symbol syntax
- add `letrec*`, define internal `define` in terms of it
- optional and keyword arguments as in DSSSL

Extensions to R5RS (controversial or difficult but necessary)

- module system
- non-hygienic macros
- records
- mechanism for new primitive types
- Unicode support
- errors and exceptions
- require a mode where "it is an error" means "an error is signaled"

Extensions to R5RS (probably not terribly controversial)

- multiline comments
- external representation for circular structures
- `#!eof`
- more escape characters
- require that `#f`, `#t`, and characters be followed by a delimiter
- `case-lambda`
- `cond-expand`
- allow the name of the macro being defined in `syntax-rules` to be arbitrary (or _)
- allow continuations created by `begin` to accept any number of values
- tighten up specification of `eq?` and `eqv?` (or otherwise address their portability problems)
- tighten up specification of inexact arithmetic
- add `+0.`, `-0.`, `+inf.0`, `-inf.0`, `+nan.0`
- bitwise operations on exact integers
- SRFI 4 homogeneous numeric vectors
- specify dynamic environment
- operations on files
- binary I/O or new I/O subsystem entirely
- string code
- regular expressions
- command-line parsing
- hash tables
- library for dates
- system operations

Editorial changes

- split language into core and libraries

Additional extensions

- support for expression comments `#;...`
- make `[ ]` equivalent to `( )`
- subset of Common Lisp `format` (in a library)
- remove `set-car!` and `set-cdr!`
- make quotation of empty list optional
- allow symbols to start with `->`
- add `weak-cons`
- add void object

Because of the central role of the module system and its probable use in splitting the Scheme language into a core and libraries, we decided that the most pressing issue was the design of the module

system. Our starting point was the "strawman module system" proposed by Flatt, which is based on the MzScheme system. Various aspects of the proposed system were discussed, mainly to understand it better and to add constructive criticism. Because many aspects are interrelated, we did not achieve consensus on any specific aspect (nor did we really try to achieve it given that this is early in the design process).

Over May and June, the discussion on the module system was slow and only two of the seven editors were active. At the end of June, I suggested that the reason for this apathy might be a lack of practical experience with the proposed module system (the two editors that were active both had experience with the MzScheme module system). I proposed that we should work on building a portable implementation of the module system so that the editors can all experiment with it in our own Scheme implementations. This would get the editors more involved in the details of the module system, allow proposed changes to be made and evaluated on-the-fly by changing the portable implementation, and the resulting public-domain code would greatly increase acceptance of R6RS by other implementors. It still remains to be seen if this portable implementation becomes a reality, as it represents quite a bit of work.

Dybvig noted that there are few differences between the module system proposed by Flatt and the one in Chez Scheme. This prompted an effort by Dybvig and Flatt to design a new module system that combines both systems. There has been a very active discussion on this topic from that point.

3 The Snowbird Meetings

Because all the editors except for Kelsey were going to the 2004 Scheme workshop in Snowbird, we made arrangements to have a whole-day meeting September 18, and we also met on two other days before the Scheme workshop. These face-to-face meetings were very useful and allowed efficient discussion on several issues. Each editor present in Snowbird prepared a presentation and notes on a major issue (portability, modules, records, `syntax-case` macros, hashtables, exceptions) to allow the other editors to prepare for the meeting.

In general, the committee's design procedure consists in tackling each issue separately: gathering proposal(s) for solutions (from existing implementations, SRFI's, old RnRS proposals, new ideas, etc), discussing and modifying the proposals, and then taking a preliminary decision on the course of action which can be: dropping the issue, adopting a specific solution, forming a subgroup of editors to work out details and come back with a revised or alternate proposal for the whole committee, or mandating an editor or subgroup of editors to write up a specification to be included in R6RS. Even though the committee is always careful to think of possible interactions between issues, the committee will be taking a final vote on the various changes to ensure overall consistency of the R6RS.

An important concern we have is the tight schedule given in the charter for writing the draft R6RS. This has forced the committee to take the complexity of an issue into account when considering the course of action. Time-consuming issues whose solution would marginally contribute to our goals were quickly dropped to give more time for other issues which are more likely to be solved within the charter's time frame. We believe that some of these issues are best left to the next standardization effort (R7RS) and hope the user community will have started proposing solutions, possibly in the form of SRFIs, to be considered by the committee.

In Snowbird, several but not all of the topics on the committee's list of issues were discussed and some lead to a preliminary decision. We give here the list of these issues with the preliminary decision that was taken.

Editorial changes

- split language into core and libraries
  The committee believes that it is important to keep the core language small. The goal is thus to split the R6RS into a core language that would more-or-less have the size and features of R5RS, and a set of libraries. This need to modularize the functionality of the R6RS is one of the motivations for a module system. We have not worked out the details of how this will be expressed in the R6RS.

Deletions from R5RS

- remove `transcript-on` and `transcript-off`
  These procedures will be removed.
- remove `force` and `delay`
  These features will remain but may be moved out of the core.
- remove multiple values
  Multiple values will remain in R6RS.

Incompatible changes to R5RS

- make syntax case-sensitive
  Discussed but no decision taken.

Extensions that would break implementation-specific features

- support for processes
  Issue left for R7RS.
- support for network programming
  Issue left for R7RS.
- object-oriented programming
  Issue left for R7RS.
- external representation for records
  Discussed but no decision taken.
- serialization
  Any Scheme datum should be serializable (in particular symbols). No decision taken on other types of objects.
- specify evaluation order
  Evaluation order will be unspecified for procedure calls and `let` *form.*

Extensions to R5RS (controversial and probably unnecessary)

- add `letrec*`, define internal `define` in terms of it
  The `letrec*` *form will be added and internal definitions will be expressed in terms of it (leading to a left-to-right evaluation order for internal definitions).*
- abstract data type for continuations
  Discussed but no decision taken.
- composable continuations
  Discussed but no decision taken.
- box types
  The box type will not be added to R6RS.
- uninterned symbols
  Discussed but no decision taken.

- extended symbol syntax
  Discussed but no decision taken.
- optional and keyword arguments as in DSSSL
  Discussed but no decision taken.

Extensions to R5RS (controversial or difficult but necessary)

- module system
  A module system will definitely be part of R6RS. We have discussed this topic extensively and are now converging on a specific proposal.
- non-hygienic macros
  Discussed but no decision taken.
- records
  Discussed but no decision taken.
- Unicode support
  Discussed but no decision taken.
- errors and exceptions
  Discussed but no decision taken.
- require a mode where "it is an error" means "an error is signaled"
  Discussed but no decision taken.

Extensions to R5RS (probably not terribly controversial)

- multiline comments
  Multiline comments will be included in R6RS.
- external representation for circular structures
  Discussed but no decision taken.
- `#!eof`
  There will be a single end-of-file object and a constructor to get the end-of-file object.
- more escape characters
  Escape characters will be added, in particular `\n`.
- require that `#f`, `#t`, and characters be followed by a delimiter
  Discussed but no decision taken.
- tighten up specification of inexact arithmetic
  Discussed but no decision taken.
- add `+0.`, `-0.`, `+inf.0`, `-inf.0`, `+nan.0`
  Discussed but no decision taken.
- hash tables
  Will be added as a library.

Additional extensions

- support for expression comments `#;...`
  Expression comments will be added to R6RS.
- make `[ ]` equivalent to `( )`
  Balanced square brackets will be equivalent to balanced parentheses.

4 The Next 6 Months

The next 6 months will be critical to the R6RS design process. Discussion will have to continue on the remaining issues, and most issues will have to be resolved. We will continue email discussions and set up a "wiki" site to separate discussion on each issue and allow collaborative development of specifications. We are planning other face-to-face meetings to tackle some of the most difficult issues. We also plan to start developing a portable implementation of the module, macro and record system. This implementation will help us understand these systems, and allow Scheme implementors to more easily build R6RS conformant implementations.