R6RS: Standard Language

Version 4.0.2

July 4, 2008

The The Revised^6 Report on the Algorithmic Language Scheme defines a dialect of Scheme. We use *R^{6}RS* to refer to both the standard and the language defined by the standard.

R^{6}RS defines both *libraries* and *top-level programs*. Both correspond to PLT Scheme *modules* (see §6 "Modules"). That is, although R^{6}RS defines top-level programs as entry points, you can just as easily treat a library as an entry point when using PLT Scheme. The only difference is that an R^{6}RS top-level program cannot export any bindings to other modules.

Contents

1 Running Top-Level Programs

To run a top-level program, either:

- Use the `plt-r6rs` executable, supplying the file that contains the program on the command line:

  `plt-r6rs` ⟨*program-file*⟩

  Additional command-line arguments are propagated as command-line arguments to the program (accessed via `command-line`).

  To compile the file to bytecode (to speed future runs of the program), use `plt-r6rs` with the `--compile` flag:

  `plt-r6rs --compile` ⟨*program-file*⟩

  The bytecode file is written in a `"compiled"` sub-directory next to ⟨*program-file*⟩.

  For example, if `"hi.scm"` contains

```
(import (rnrs))
(display "hello\n")
```

  then

  `plt-r6rs hi.scm`

  prints "hello."

- Prefix the program with `#!r6rs`, which counts as a comment from the R^{6}RS perspective, but is a synonym for `#lang r6rs` from the PLT Scheme perspective. Such files can be run like any other PLT Scheme module, such as using `mzscheme`:

  `mzscheme` ⟨*program-file*⟩

  or using DrScheme with the Module language. The file can also be compiled to bytecode using `mzc`:

  `mzc` ⟨*program-file*⟩

  For example, if `"hi.ss"` contains

```
#!r6rs
(import (rnrs))
(display "hello\n")
```

  then

  `mzscheme hi.ss`

  prints "hello." Similarly, opening `"hi.ss"` in DrScheme and clicking Run prints "hello" within the DrScheme interactions window.

2 Installing Libraries

To reference an R^{6}RS library from a top-level program or another library, it must be installed as a collection-based library in PLT Scheme.

One way to produce an R^{6}RS installed library is to create in a collection a file that starts with `#!r6rs` and that contains a `library` form. For example, the following file might be created in a `"hello.ss"` file within a `"examples"` collection directory:

```
#!r6rs
(library (examples hello)
  (export greet)
  (import (rnrs))

  (define (greet)
    (display "hello\n")))
```

Alternately, the `plt-r6rs` executable with the `--install` flag accepts a sequence of `library` declarations and installs them into separate files in a collection directory, based on the declared name of each library:

`plt-r6rs --install` ⟨*libraries-file*⟩

By default, libraries are installed into the user-specific collection directory (see `find-user-collects-dir`). The `--all-users` flag causes the libraries to be installed into the main installation, instead (see `find-collects-dir`):

`plt-r6rs --install --all-users` ⟨*libraries-file*⟩

See §3 “Libraries and Collections” for information on how R^{6}RS library names are turned into collection-based module paths, which determines where the files are written. Libraries installed by `plt-r6rs --install` are automatically compiled to bytecode form.

One final option is to supply a `++path` flag to `plt-r6rs`. A path added with `++path` extends the set of directories that are searched to find a collection (i.e., it sets `current-library-collection-paths`). If ⟨*dir*⟩ contains `"duck"` and `"cow"` sub-directories with `"duck/feather.sls"` and `"cow/bell.sls"`, and if each file is an R^{6}RS library prefixed with `#!r6rs`, then `plt-r6rs ++path` ⟨*dir*⟩ directs the R^{6}RS library references `(duck feather)` and `(cow bell)` to the files. Note that this technique does not support accessing `"duck.sls"` directly within ⟨*dir*⟩, since the library reference `(duck)` is treated like `(duck main)` for finding the library, as explained in §3 “Libraries and Collections”. Multiple paths can be provided with multiple uses of `++path`; the paths are search in order, and before the installation’s collections.

3 Libraries and Collections

An R^{6}RS library name is sequence of symbols, optionally followed by a version as a sequence of exact, non-negative integers. Roughly, such a name is converted to a PLT Scheme module pathname (see §6.3 "Module Paths") by concatenating the symbols with a `/` separator, and then appending the version integers each with a preceeding `-`. As a special case, when an R^{6}RS path contains a single symbol followed by a version, a `main` symbol is effectively inserted after the initial symbol.

When an R^{6}RS library or top-level program refers to another library, it can supply version constraints rather than naming a specific version. Version constraints are always resolved at compile time by searching the set of installed files.

In addition, when an R^{6}RS library path is converted, a file extension is selected at compile time based on installed files. The search order for file extensions is `".mzscheme.ss"`, `".mzscheme.sls"`, `".ss"`, and `".sls"`. When resolving version constraints, these extensions are all tried when looking for matches.

Examples (assuming a typical PLT Scheme installation):

```
(rnrs io simple (6))  means  (lib "rnrs/io/simple-6.ss")
(rnrs)                means  (lib "rnrs/main-6.ss")
(rnrs (6))            means  (lib "rnrs/main-6.ss")
(scheme base)         means  (lib "scheme/base.ss")
```

4 Scheme Interoperability

Using the conversion rules in §3 "Libraries and Collections", and R^{6}RS library can refer to modules that are implemented in other dialects supported by PLT Scheme, and other PLT Scheme modules can refer to libraries that are implemented in R^{6}RS.

Beware that a *pair* in R^{6}RS corresponds to a *mutable pair* in `scheme/base`. Otherwise, R^{6}RS libraries and `scheme/base` share the same datatype for numbers, characters, strings, bytevectors (a.k.a. byte strings), vectors, and so on. Hash tables are different. Input and output ports from `scheme/base` can be used directly as binary ports with R^{6}RS libraries, and all R^{6}RS ports can be used as ports in `scheme/base` programs, but only textual ports created via R^{6}RS libraries can be used by other R^{6}RS operations that expect textual ports.

5 R^{6}RS Conformance

PLT Scheme's R^{6}RS support does not conform with the standard in several known ways:

- When `guard` catches an exception that no clause matches, the exception is re-`raise`ed without restoring the continuation to the one that raised the exception.

  This difference can be made visible using `dynamic-wind`. According to R^{6}RS, the following program should print "in" and "out" twice, but each prints once using PLT Scheme:

```
(guard (exn [(equal? exn 5) 'five])
  (guard (exn [(equal? exn 6) 'six])
    (dynamic-wind
      (lambda () (display "in") (newline))
      (lambda () (raise 5))
      (lambda () (display "out") (newline)))))
```

  Along similar lines, continuation capture and invocation within an exception handler is restricted. Unless the exception is raised through `raise-continuable`, a handler can escape only through a continuation that is a tail of the current continuation, and a continuation captured within the handler cannot be invoked after control escapes from the raise.

- Currently, inexact numbers are printed without a precision indicator, and precision indicators are ignored on input (e.g., `0.5|7` is read the same as `0.5`).

- Word boundaries for `string-downcase`, `string-upcase`, and `string-titlecase` are not determined as specified by Unicode Standard Annex #29.

- When an identifier bound by `letrec` or `letrec*` is referenced before it is bound, an exception is not raised; instead, the reference produces `#<undefined>`.

- The bindings in a namespace produced by `null-environment` or `scheme-report-environment` correspond to R^{5}RS bindings instead of R^{6}RS bindings. In particular, `=>`, `else`, `_`, and `...` are not bound.

6 R^{6}RS Libraries

6.1 (rnrs base (6)): Base

```
(require rnrs/base-6)
```

Original specification: Base

6.2 (rnrs unicode (6)): Unicode

```
(require rnrs/unicode-6)
```

Original specification: Unicode

6.3 (rnrs bytevectors (6)): Bytevectors

```
(require rnrs/bytevectors-6)
```

Original specification: Bytevectors

6.4 (rnrs lists (6)): List utilities

```
(require rnrs/lists-6)
```

Original specification: List utilities

6.5 (rnrs sorting (6)): Sorting

```
(require rnrs/sorting-6)
```

Original specification: Sorting

6.6 (rnrs control (6)): Control Structures

```
(require rnrs/control-6)
```

Original specification: Control Structures

6.7 `(rnrs records syntactic (6))`: Records: Syntactic

```
(require rnrs/records/syntactic-6)
```

Original specification: Records: Syntactic

6.8 `(rnrs records procedural (6))`: Records: Procedural

```
(require rnrs/records/procedural-6)
```

Original specification: Records: Procedural

6.9 `(rnrs records inspection (6))`: Records: Inspection

```
(require rnrs/records/inspection-6)
```

Original specification: Records: Inspection

6.10 `(rnrs exceptions (6))`: Exceptions and Conditions

```
(require rnrs/exceptions-6)
```

Original specification: Exceptions and Conditions

See also §5 “R[6]RS Conformance”.

6.11 `(rnrs conditions (6))`: Exceptions and Conditions

```
(require rnrs/conditions-6)
```

Original specification: Exceptions and Conditions

6.12 `(rnrs io ports (6))`: I/O: Ports

```
(require rnrs/io/ports-6)
```

Original specification: I/O: Ports

6.13 `(rnrs io simple (6))`: I/O: Simple

```
(require rnrs/io/simple-6)
```

Original specification: I/O: Simple

6.14 `(rnrs files (6))`: File System

```
(require rnrs/files-6)
```

Original specification: File System

6.15 `(rnrs programs (6))`: Command-line Access and Exit Values

```
(require rnrs/programs-6)
```

Original specification: Command-line Access and Exit Values

6.16 `(rnrs arithmetic fixnums (6))`: Arithmetic: Fixnums

```
(require rnrs/arithmetic/fixnums-6)
```

Original specification: Arithmetic: Fixnums

6.17 `(rnrs arithmetic flonums (6))`: Arithmetic: Flonums

```
(require rnrs/arithmetic/flonums-6)
```

Original specification: Arithmetic: Flonums

6.18 `(rnrs arithmetic bitwise (6))`: Arithmetic: Bitwise

```
(require rnrs/arithmetic/bitwise-6)
```

Original specification: Arithmetic: Bitwise

6.19 (rnrs syntax-case (6)): Syntax-Case

```
(require rnrs/syntax-case-6)
```

Original specification: Syntax-Case

6.20 (rnrs hashtables (6)): Hashtables

```
(require rnrs/hashtables-6)
```

Original specification: Hashtables

A hashtable is a dictionary in the sense of scheme/dict.

6.21 (rnrs enums (6)): Enumerations

```
(require rnrs/enums-6)
```

Original specification: Enumerations

6.22 (rnrs eval (6)): Eval

```
(require rnrs/eval-6)
```

Original specification: Eval

6.23 (rnrs mutable-pairs (6)): Mutable Pairs

```
(require rnrs/mutable-pairs-6)
```

Original specification: Mutable Pairs

6.24 (rnrs mutable-strings (6)): Mutable Strings

```
(require rnrs/mutable-strings-6)
```

Original specification: Mutable Strings

6.25 `(rnrs r5rs (6))`: R5RS Compatibility

```
(require rnrs/r5rs-6)
```

Original specification: R5RS Compatibility

See also §5 "R^{6}RS Conformance".

Index