

# R6RS: Scheme

Version 6.1

August 1, 2014

The Revised<sup>6</sup> Report on the Algorithmic Language Scheme defines a dialect of Scheme. We use *R<sup>6</sup>RS* to refer to both the standard and the language defined by the standard.

*R<sup>6</sup>RS* defines both *libraries* and *top-level programs*. Both correspond to Racket *modules* (see §6 “Modules”). That is, although *R<sup>6</sup>RS* defines top-level programs as entry points, you can just as easily treat a library as an entry point when using Racket. The only difference is that an *R<sup>6</sup>RS* top-level program cannot export any bindings to other modules.

See §23 “Dialects of Racket and Scheme” for general information about different dialects of Scheme within Racket.

# Contents

<b>1</b>	<b>Using R<sup>6</sup>RS with DrRacket</b>	<b>4</b>
<b>2</b>	<b>Running Top-Level Programs</b>	<b>5</b>
<b>3</b>	<b>Installing Libraries</b>	<b>6</b>
<b>4</b>	<b>R<sup>6</sup>RS Module Language</b>	<b>8</b>
4.1	Using R <sup>6</sup> RS . . . . .	8
4.2	The Implementation of R <sup>6</sup> RS . . . . .	8
<b>5</b>	<b>Libraries and Collections</b>	<b>9</b>
<b>6</b>	<b>Language Interoperability</b>	<b>10</b>
<b>7</b>	<b>R<sup>6</sup>RS Conformance</b>	<b>11</b>
<b>8</b>	<b>R<sup>6</sup>RS Libraries</b>	<b>13</b>
8.1	( <code>rns base</code> (6)): Base . . . . .	13
8.2	( <code>rns unicode</code> (6)): Unicode . . . . .	13
8.3	( <code>rns bytevectors</code> (6)): Bytevectors . . . . .	13
8.4	( <code>rns lists</code> (6)): List utilities . . . . .	13
8.5	( <code>rns sorting</code> (6)): Sorting . . . . .	13
8.6	( <code>rns control</code> (6)): Control Structures . . . . .	13
8.7	( <code>rns records syntactic</code> (6)): Records: Syntactic . . . . .	14
8.8	( <code>rns records procedural</code> (6)): Records: Procedural . . . . .	14
8.9	( <code>rns records inspection</code> (6)): Records: Inspection . . . . .	14
8.10	( <code>rns exceptions</code> (6)): Exceptions . . . . .	14
8.11	( <code>rns conditions</code> (6)): Conditions . . . . .	14
8.12	( <code>rns io ports</code> (6)): I/O: Ports . . . . .	14
8.13	( <code>rns io simple</code> (6)): I/O: Simple . . . . .	15
8.14	( <code>rns files</code> (6)): File System . . . . .	15
8.15	( <code>rns programs</code> (6)): Command-line Access and Exit Values . . . . .	15
8.16	( <code>rns arithmetic fixnums</code> (6)): Arithmetic: Fixnums . . . . .	15
8.17	( <code>rns arithmetic flonums</code> (6)): Arithmetic: Flonums . . . . .	15
8.18	( <code>rns arithmetic bitwise</code> (6)): Arithmetic: Bitwise . . . . .	15
8.19	( <code>rns syntax-case</code> (6)): Syntax-Case . . . . .	16
8.20	( <code>rns hashtables</code> (6)): Hashtables . . . . .	16
8.21	( <code>rns enums</code> (6)): Enumerations . . . . .	16
8.22	( <code>rns eval</code> (6)): Eval . . . . .	16
8.23	( <code>rns mutable-pairs</code> (6)): Mutable Pairs . . . . .	16
8.24	( <code>rns mutable-strings</code> (6)): Mutable Strings . . . . .	16
8.25	( <code>rns r5rs</code> (6)): R5RS Compatibility . . . . .	17
	<b>Index</b>	<b>18</b>



## 1 Using R<sup>6</sup>RS with DrRacket

To run an R<sup>6</sup>RS program with DrRacket choose Use language declared in source from the language dialog box and add the following line to the top of your program. `#!r6rs`.

Here is a small example R<sup>6</sup>RS program that will work in DrRacket.

```
#!r6rs
(import (rnrs lists (6))
        (rnrs base (6))
        (rnrs io simple (6)))
(display (find even? '(3 1 4 1 5 9)))
```

## 2 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.sps" contains

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

then

```
plt-r6rs hi.sps
```

prints "hello."

- Prefix the program with `#!r6rs`, which counts as a comment from the R<sup>6</sup>RS perspective, but is a synonym for `#lang r6rs` from the Racket perspective. Such files can be run like any other Racket module, such as using `racket`:

```
racket <program-file>
```

or using DrRacket. The file can also be compiled to bytecode using `raco make`:

```
raco make <program-file>
```

For example, if "hi.sps" contains

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

then

```
racket hi.sps
```

prints "hello." Similarly, opening "hi.sps" in DrRacket and clicking Run prints "hello" within the DrRacket interactions window.

### 3 Installing Libraries

To reference an R<sup>6</sup>RS library from a top-level program or another library, it must be installed as a collection-based library in Racket.

One way to produce an R<sup>6</sup>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.sls" 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>
```

You may as well specify an arbitrary collections directory by using the `--collections` flag:

```
plt-r6rs --install --collections <directory> <libraries-file>
```

See §5 “Libraries and Collections” for information on how R<sup>6</sup>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<sup>6</sup>RS library prefixed with `#!r6rs`, then `plt-r6rs ++path <dir>` directs the R<sup>6</sup>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 §5 “Libraries and Collections”. Multiple paths

can be provided with multiple uses of `++path`; the paths are searched in order, and before the installation's collections.

## 4 R<sup>6</sup>RS Module Language

```
#lang r6rs      package: r6rs-lib
```

The `r6rs` language is usually used in the form `#!r6rs`, which is equivalent to `#lang r6rs` and is also valid R<sup>6</sup>RS syntax.

### 4.1 Using R<sup>6</sup>RS

See §1 “Using R<sup>6</sup>RS with DrRacket”, §2 “Running Top-Level Programs”, and §3 “Installing Libraries” for more information on writing and running R<sup>6</sup>RS programs with Racket.

### 4.2 The Implementation of R<sup>6</sup>RS

The R<sup>6</sup>RS language is itself implemented as a module within Racket. The details of that implementation, as provided in this section, are not normally relevant to programmers using R<sup>6</sup>RS; see the links in §4.1 “Using R<sup>6</sup>RS”, instead. The details may be relevant to programmers who are developing new tools or deriving variants of R<sup>6</sup>RS within Racket.

As a Racket module, the `r6rs` module language provides only a `#!/module-begin` binding, which is used to process the entire body of a Racket module (see `module`). The `#!/module-begin` binding from `r6rs` allows the body of a module to use the syntax of either a R<sup>6</sup>RS library or a R<sup>6</sup>RS top-level program.

```
(#!/module-begin
 (library library-name
  (export export-spec ...)
  (import import-spec ...)
  library-body ...))
#!/module-begin
(import import-spec ...)
program-body ...)
```

An `r6rs` module that contains a single `library` form defines an R<sup>6</sup>RS library, while a module body that starts with an `import` form defines an R<sup>6</sup>RS top-level program.

The `library`, `export`, and `import` identifiers are not exported by the `r6rs` library; they are recognized through equivalence to unbound identifiers.



## 5 Libraries and Collections

An R<sup>6</sup>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 Racket module pathname (see §6.3 “Module Paths”) by concatenating the symbols with a `/` separator, and then appending the version integers each with a preceding `-`. As a special case, when an R<sup>6</sup>RS path contains a single symbol (optionally followed by a version), a `main` symbol is effectively inserted after the initial symbol. See below for further encoding considerations.

When an R<sup>6</sup>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<sup>6</sup>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"`, `".sls"`, and `".rkt"`. When resolving version constraints, these extensions are all tried when looking for matches.

To ensure that all R<sup>6</sup>RS library names can be converted to a unique and distinct library module path, the following conversions are applied to each symbol before concatenating them:

- The symbol is encoded using UTF-8, and the resulting bytes are treated as Latin-1 encoded characters. ASCII letters, digits, `+`, `-`, and `_` are left as-is; other characters are replaced by `%` followed by two lowercase hexadecimal digits. Note that UTF-8 encodes ASCII letters, digits, etc. as themselves, so typical library names correspond to readable module paths.
- If the R<sup>6</sup>RS library reference has two symbol elements and the second one is `main` followed by any number of underscores, then an extra underscore is added to that symbol. This conversion avoids a collision between an explicit `main` and the implicit `main` when a library path has a single symbol element.

Examples (assuming a typical Racket installation):

```
(rnrs io simple (6)) means (lib "rnrs/io/simple-6.rkt")
(rnrs)                means (lib "rnrs/main-6.rkt")
(rnrs main)           means (lib "rnrs/main_.rkt")
(rnrs (6))            means (lib "rnrs/main-6.rkt")
(racket base)         means (lib "racket/base.rkt")
(achtung!)             means (lib "achtung%21/main.rkt")
(funco new-λ)         means (lib "funco/new-%ce%bb.rkt")
```

## 6 Language Interoperability

Using the conversion rules in §5 “Libraries and Collections”, and R<sup>6</sup>RS library can refer to modules that are implemented in other dialects supported by Racket, and other Racket modules can refer to libraries that are implemented in R<sup>6</sup>RS.

Beware that a *pair* in R<sup>6</sup>RS corresponds to a *mutable pair* in `racket/base`. Otherwise, R<sup>6</sup>RS libraries and `racket/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 `racket/base` can be used directly as binary ports with R<sup>6</sup>RS libraries, and all R<sup>6</sup>RS ports can be used as ports in `racket/base` programs, but only textual ports created via R<sup>6</sup>RS libraries can be used by other R<sup>6</sup>RS operations that expect textual ports.

## 7 R<sup>6</sup>RS Conformance

Racket's R<sup>6</sup>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-raised without restoring the continuation to the one that raised the exception.

This difference can be made visible using `dynamic-wind`. According to R<sup>6</sup>RS, the following program should print “in” and “out” twice, but each prints once using Racket:

```
(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.

The initial exception handler does not return for non-`&serious` conditions, but `raise` and `raise-continuable` both install an uncaught-exception handler (via `parameterize` and `uncaught-exception-handler`) to one that returns for non-`&serious` conditions.

- 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.
- A custom textual port must represent positions using integers, and the positions must correspond to bytes in a UTF-8 encoding of the port's data. For custom ports (byte or character) that support both input and output, beware that buffered input can create a mismatch between the position implemented by the custom procedures and the port's current position; the result from a custom position procedure is automatically adjusted to account for buffering, and setting the port's position flushes all buffered bytes, but writing after a read does *not* automatically reset the port's position to counteract the effects of buffering.
- The bindings in a namespace produced by `null-environment` or `scheme-report-environment` correspond to R<sup>5</sup>RS bindings instead of R<sup>6</sup>RS bindings. In particular, `=>`, `else`, `_`, and `...` are not bound.

- Bindings for `#!/datum`, `#!/app`, `#!/top`, and `#!/top-interaction` are imported into every library and program, and at every phase level for which the library or program has imports.

Changed in version 6.0.1.4: When an identifier bound by `letrec` or `letrec*` is referenced before it is initialized, an exception is raised, instead of producing `#!<undefined>`.

## 8 R<sup>6</sup>RS Libraries

### 8.1 `(rnrs base (6))`: Base

```
(require rnrs/base-6)      package: r6rs-lib
```

Original specification: Base

### 8.2 `(rnrs unicode (6))`: Unicode

```
(require rnrs/unicode-6)   package: r6rs-lib
```

Original specification: Unicode

### 8.3 `(rnrs bytevectors (6))`: Bytevectors

```
(require rnrs/bytevectors-6) package: r6rs-lib
```

Original specification: Bytevectors

### 8.4 `(rnrs lists (6))`: List utilities

```
(require rnrs/lists-6)     package: r6rs-lib
```

Original specification: List utilities

### 8.5 `(rnrs sorting (6))`: Sorting

```
(require rnrs/sorting-6)   package: r6rs-lib
```

Original specification: Sorting

### 8.6 `(rnrs control (6))`: Control Structures

```
(require rnrs/control-6)   package: r6rs-lib
```

Original specification: Control Structures

## 8.7 `(nrns records syntactic (6))`: **Records: Syntactic**

`(require nrns/records/syntactic-6)` package: r6rs-lib

Original specification: Records: Syntactic

## 8.8 `(nrns records procedural (6))`: **Records: Procedural**

`(require nrns/records/procedural-6)` package: r6rs-lib

Original specification: Records: Procedural

## 8.9 `(nrns records inspection (6))`: **Records: Inspection**

`(require nrns/records/inspection-6)` package: r6rs-lib

Original specification: Records: Inspection

## 8.10 `(nrns exceptions (6))`: **Exceptions**

`(require nrns/exceptions-6)` package: r6rs-lib

Original specification: Exceptions

See also §7 “R<sup>6</sup>RS Conformance”.

## 8.11 `(nrns conditions (6))`: **Conditions**

`(require nrns/conditions-6)` package: r6rs-lib

Original specification: Conditions

## 8.12 `(nrns io ports (6))`: **I/O: Ports**

`(require nrns/io/ports-6)` package: r6rs-lib

Original specification: I/O: Ports

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

`(require rnrs/io/simple-6)`      package: r6rs-lib

Original specification: I/O: Simple

### 8.14 `(rnrs files (6))`: File System

`(require rnrs/files-6)`      package: r6rs-lib

Original specification: File System

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

`(require rnrs/programs-6)`      package: r6rs-lib

Original specification: Command-line Access and Exit Values

### 8.16 `(rnrs arithmetic fixnums (6))`: Arithmetic: Fixnums

`(require rnrs/arithmetic/fixnums-6)`      package: r6rs-lib

Original specification: Arithmetic: Fixnums

### 8.17 `(rnrs arithmetic flonums (6))`: Arithmetic: Flonums

`(require rnrs/arithmetic/flonums-6)`      package: r6rs-lib

Original specification: Arithmetic: Flonums

### 8.18 `(rnrs arithmetic bitwise (6))`: Arithmetic: Bitwise

`(require rnrs/arithmetic/bitwise-6)`      package: r6rs-lib

Original specification: Arithmetic: Bitwise

### 8.19 `(rnrs syntax-case (6))`: Syntax-Case

```
(require rnrs/syntax-case-6)    package: r6rs-lib
```

Original specification: Syntax-Case

### 8.20 `(rnrs hashtables (6))`: Hashtables

```
(require rnrs/hashtables-6)    package: r6rs-lib
```

Original specification: Hashtables

A hashtable is a dictionary in the sense of `racket/dict`, and hash table operations interact with threads in the same way for hash tables created with `make-hash` (e.g., `hashtable-ref` and `hashtable-set!` are thread-safe).

### 8.21 `(rnrs enums (6))`: Enumerations

```
(require rnrs/enums-6)        package: r6rs-lib
```

Original specification: Enumerations

### 8.22 `(rnrs eval (6))`: Eval

```
(require rnrs/eval-6)        package: r6rs-lib
```

Original specification: Eval

### 8.23 `(rnrs mutable-pairs (6))`: Mutable Pairs

```
(require rnrs/mutable-pairs-6) package: r6rs-lib
```

Original specification: Mutable Pairs

### 8.24 `(rnrs mutable-strings (6))`: Mutable Strings

```
(require rnrs/mutable-strings-6) package: r6rs-lib
```

Original specification: Mutable Strings



## 8.25 `(rnrs r5rs (6))`: R5RS Compatibility

`(require rnrs/r5rs-6)`      `package: r6rs-lib`

Original specification: R5RS Compatibility

See also §7 “R<sup>6</sup>RS Conformance”.

## **Index**

















