

R6RS: Scheme

Version 9.4.0.3

October 30, 2026

The Revised⁶ Report on the Algorithmic Language Scheme defines a dialect of Scheme. We use *R⁶RS* to refer to both the standard and the language defined by the standard.

R⁶RS defines both *libraries* and *top-level programs*. Both correspond to Racket *modules* (see §6 “Modules”). That is, although R⁶RS 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 R⁶RS 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

1	Using R⁶RS with DrRacket	4
2	Running Top-Level Programs	5
3	Installing Libraries	6
4	R⁶RS Module Language	8
4.1	Using R ⁶ RS	8
4.2	The Implementation of R ⁶ RS	8
5	Libraries and Collections	9
6	Language Interoperability	10
7	R⁶RS Conformance	11
8	R⁶RS Libraries	13
8.1	(<code>rnrs base (6)</code>): Base	13
8.2	(<code>rnrs unicode (6)</code>): Unicode	13
8.3	(<code>rnrs bytevectors (6)</code>): Bytevectors	13
8.4	(<code>rnrs lists (6)</code>): List utilities	13
8.5	(<code>rnrs sorting (6)</code>): Sorting	13
8.6	(<code>rnrs control (6)</code>): Control Structures	13
8.7	(<code>rnrs records syntactic (6)</code>): Records: Syntactic	14
8.8	(<code>rnrs records procedural (6)</code>): Records: Procedural	14
8.9	(<code>rnrs records inspection (6)</code>): Records: Inspection	14
8.10	(<code>rnrs exceptions (6)</code>): Exceptions	14

8.11	(rnrs conditions (6)):	Conditions	14
8.12	(rnrs io ports (6)):	I/O: Ports	14
8.13	(rnrs io simple (6)):	I/O: Simple	15
8.14	(rnrs files (6)):	File System	15
8.15	(rnrs programs (6)):	Command-line Access and Exit Values	15
8.16	(rnrs arithmetic fixnums (6)):	Arithmetic: Fixnums	15
8.17	(rnrs arithmetic flonums (6)):	Arithmetic: Flonums	15
8.18	(rnrs arithmetic bitwise (6)):	Arithmetic: Bitwise	15
8.19	(rnrs syntax-case (6)):	Syntax-Case	16
8.20	(rnrs hashtables (6)):	Hashtables	16
8.21	(rnrs enums (6)):	Enumerations	16
8.22	(rnrs eval (6)):	Eval	16
8.23	(rnrs mutable-pairs (6)):	Mutable Pairs	16
8.24	(rnrs mutable-strings (6)):	Mutable Strings	16
8.25	(rnrs r5rs (6)):	R5RS Compatibility	17
	Index		18
	Index		18

1 Using R⁶RS with DrRacket

To run an R⁶RS 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 R⁶RS 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 R⁶RS 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 R⁶RS 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 R⁶RS 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 R⁶RS 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 R⁶RS library prefixed with `#!r6rs`, then `plt-r6rs ++path <dir>` directs the R⁶RS 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 R⁶RS 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 R⁶RS syntax.

4.1 Using R⁶RS

See §1 “Using R⁶RS with DrRacket”, §2 “Running Top-Level Programs”, and §3 “Installing Libraries” for more information on writing and running R⁶RS programs with Racket.

4.2 The Implementation of R⁶RS

The R⁶RS 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 R⁶RS; see the links in §4.1 “Using R⁶RS”, instead. The details may be relevant to programmers who are developing new tools or deriving variants of R⁶RS 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 R⁶RS library or a R⁶RS 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 R⁶RS library, while a module body that starts with an `import` form defines an R⁶RS 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 R⁶RS 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 R⁶RS 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 R⁶RS 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 R⁶RS 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 R⁶RS 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 R⁶RS 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):

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

6 Language Interoperability

Using the conversion rules in §5 “Libraries and Collections”, and R⁶RS 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 R⁶RS.

Beware that a *pair* in R⁶RS corresponds to a *mutable pair* in `racket/base`. Otherwise, R⁶RS 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 R⁶RS libraries, and all R⁶RS ports can be used as ports in `racket/base` programs, but only textual ports created via R⁶RS libraries can be used by other R⁶RS operations that expect textual ports.

7 R⁶RS Conformance

Racket’s R⁶RS support does not conform with the standard in several known ways:

- When guard catches an exception that no clause matches, the exception is re-[raiseed](#) without restoring the continuation to the one that raised the exception.

This difference can be made visible using [dynamic-wind](#). According to R⁶RS, 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 R⁵RS bindings instead of R⁶RS 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 R⁶RS 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 `(rnrs records syntactic (6))`: Records: Syntactic

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

Original specification: Records: Syntactic

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

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

Original specification: Records: Procedural

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

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

Original specification: Records: Inspection

8.10 `(rnrs exceptions (6))`: Exceptions

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

Original specification: Exceptions

See also §7 “R⁶RS Conformance”.

8.11 `(rnrs conditions (6))`: Conditions

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

Original specification: Conditions

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

`(require rnrs/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 `(nnrs syntax-case (6))`: Syntax-Case

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

Original specification: Syntax-Case

8.20 `(nnrs hashtables (6))`: Hashtables

```
(require nnrs/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 `(nnrs enums (6))`: Enumerations

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

Original specification: Enumerations

8.22 `(nnrs eval (6))`: Eval

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

Original specification: Eval

8.23 `(nnrs mutable-pairs (6))`: Mutable Pairs

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

Original specification: Mutable Pairs

8.24 `(nnrs mutable-strings (6))`: Mutable Strings

```
(require nnrs/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 “R⁶RS Conformance”.

Index

```

#%module-begin
&assertion
&condition
&error
&i/o
&i/o-decoding
&i/o-encoding
&i/o-file-already-exists
&i/o-file-does-not-exist
&i/o-file-is-read-only
&i/o-file-protection
&i/o-filename
&i/o-invalid-position
&i/o-port
&i/o-read
&i/o-write
&implementation-restriction
&irritants
&lexical
&message
&no-infinities
&no-nans
&non-continuable
&serious
&syntax
&undefined
&violation
&warning
&who
(rnrs arithmetic bitwise (6)):
  Arithmetic: Bitwise
(rnrs arithmetic fixnums (6)):
  Arithmetic: Fixnums
(rnrs arithmetic flonums (6)):
  Arithmetic: Flonums
(rnrs base (6)): Base
(rnrs bytevectors (6)): Bytevectors
(rnrs conditions (6)): Conditions
(rnrs control (6)): Control Structures
(rnrs enums (6)): Enumerations
(rnrs eval (6)): Eval
(rnrs exceptions (6)): Exceptions
(rnrs files (6)): File System
(rnrs hashtables (6)): Hashtables
(rnrs io ports (6)): I/O: Ports
(rnrs io simple (6)): I/O: Simple
(rnrs lists (6)): List utilities
(rnrs mutable-pairs (6)):  Mutable
  Pairs
(rnrs mutable-strings (6)): Mutable
  Strings
(rnrs programs (6)):  Command-line
  Access and Exit Values
(rnrs r5rs (6)): R5RS Compatibility
(rnrs records inspection (6)):
  Records: Inspection
(rnrs records procedural (6)):
  Records: Procedural
(rnrs records syntactic (6)):
  Records: Syntactic
(rnrs sorting (6)): Sorting
(rnrs syntax-case (6)): Syntax-Case
(rnrs unicode (6)): Unicode
*
+
++path
-
...
...
/
<
<=
=
=>
=>
>
>=
-
-
abs
acos
and
angle
append

```

apply	bytevector-ieee-double-native-
asin	set!
assert	bytevector-ieee-double-ref
assertion-violation	bytevector-ieee-single-native-ref
assertion-violation?	bytevector-ieee-single-native-
assoc	set!
assp	bytevector-ieee-single-ref
assq	bytevector-length
assv	bytevector-s16-native-ref
atan	bytevector-s16-native-set!
begin	bytevector-s16-ref
binary-port?	bytevector-s16-set!
bitwise-and	bytevector-s32-native-ref
bitwise-arithmetic-shift	bytevector-s32-native-set!
bitwise-arithmetic-shift-left	bytevector-s32-ref
bitwise-arithmetic-shift-right	bytevector-s32-set!
bitwise-bit-count	bytevector-s64-native-ref
bitwise-bit-field	bytevector-s64-native-set!
bitwise-bit-set?	bytevector-s64-ref
bitwise-copy-bit	bytevector-s64-set!
bitwise-copy-bit-field	bytevector-s8-ref
bitwise-first-bit-set	bytevector-s8-set!
bitwise-if	bytevector-sint-ref
bitwise-ior	bytevector-sint-set!
bitwise-length	bytevector-u16-native-ref
bitwise-not	bytevector-u16-native-set!
bitwise-reverse-bit-field	bytevector-u16-ref
bitwise-rotate-bit-field	bytevector-u16-set!
bitwise-xor	bytevector-u32-native-ref
boolean=?	bytevector-u32-native-set!
boolean?	bytevector-u32-ref
bound-identifier=?	bytevector-u32-set!
buffer-mode	bytevector-u64-native-ref
buffer-mode?	bytevector-u64-native-set!
bytevector->sint-list	bytevector-u64-ref
bytevector->string	bytevector-u64-set!
bytevector->u8-list	bytevector-u8-ref
bytevector->uint-list	bytevector-u8-set!
bytevector-copy	bytevector-uint-ref
bytevector-copy!	bytevector-uint-set!
bytevector-fill!	bytevector=?
bytevector-ieee-double-native-ref	bytevector?

caar	close-port
cadr	command-line
call-with-bytevector-output-port	complex?
call-with-current-continuation	cond
call-with-input-file	condition
call-with-output-file	condition-accessor
call-with-port	condition-irritants
call-with-string-output-port	condition-message
call-with-values	condition-predicate
call/cc	condition-who
car	condition?
case	cons
case-lambda	cons*
cdddar	cos
cddddr	current-error-port
cdr	current-input-port
ceiling	current-output-port
char->integer	datum->syntax
char-alphabetic?	define
char-ci<=?	define-condition-type
char-ci<?	define-enumeration
char-ci=?	define-record-type
char-ci>=?	define-syntax
char-ci>?	delay
char-downcase	delete-file
char-foldcase	denominator
char-general-category	display
char-lower-case?	div
char-numeric?	div-and-mod
char-title-case?	div0
char-titlecase	div0-and-mod0
char-upcase	do
char-upper-case?	dynamic-wind
char-whitespace?	else
char<=?	else
char<?	endianness
char=?	enum-set->list
char>=?	enum-set-complement
char>?	enum-set-constructor
char?	enum-set-difference
close-input-port	enum-set-indexer
close-output-port	enum-set-intersection

enum-set-member?	fl=?
enum-set-projection	fl>=?
enum-set-subset?	fl>?
enum-set-union	flabs
enum-set-universe	flacos
enum-set=?	flasin
environment	flatan
eof-object	flceiling
eof-object?	flcos
eol-style	fldenominator
eq?	fldiv
equal-hash	fldiv-and-mod
equal?	fldiv0
eqv?	fldiv0-and-mod0
error	fleven?
error-handling-mode	flexp
error?	flexpt
eval	flfinite?
even?	flfloor
exact	flinfinite?
exact->inexact	flinteger?
exact-integer-sqrt	fllog
exact?	flmax
exists	flmin
exit	flmod
exp	flmod0
expt	flnan?
fields	flnegative?
file-exists?	flnumerator
file-options	flodd?
filter	flonum?
find	floor
finite?	flpositive?
fixnum->flonum	flround
fixnum-width	flsin
fixnum?	flsqrt
fl*	fltan
fl+	fltruncate
fl-	flush-output-port
fl/	flzero?
fl<=?	fold-left
fl<?	fold-right

for-all	fxrotate-bit-field
for-each	fxxor
force	fxzero?
free-identifier=?	gcd
fx*	generate-temporaries
fx*/carry	get-bytevector-all
fx+	get-bytevector-n
fx+/carry	get-bytevector-n!
fx-	get-bytevector-some
fx-/carry	get-char
fx<=?	get-datum
fx<?	get-line
fx=?	get-string-all
fx>=?	get-string-n
fx>?	get-string-n!
fxand	get-u8
fxarithmetic-shift	greatest-fixnum
fxarithmetic-shift-left	guard
fxarithmetic-shift-right	hashtable-clear!
fxbit-count	hashtable-contains?
fxbit-field	hashtable-copy
fxbit-set?	hashtable-delete!
fxcopy-bit	hashtable-entries
fxcopy-bit-field	hashtable-equivalence-function
fxdiv	hashtable-hash-function
fxdiv-and-mod	hashtable-keys
fxdiv0	hashtable-mutable?
fxdiv0-and-mod0	hashtable-ref
fxeven?	hashtable-set!
fxfirst-bit-set	hashtable-size
fxif	hashtable-update!
fxior	hashtable?
fxlength	i/o-decoding-error?
fxmax	i/o-encoding-error-char
fxmin	i/o-encoding-error?
fxmod	i/o-error-filename
fxmod0	i/o-error-port
fxnegative?	i/o-error-position
fxnot	i/o-error?
fxodd?	i/o-file-already-exists-error?
fxpositive?	i/o-file-does-not-exist-error?
fxreverse-bit-field	i/o-file-is-read-only-error?

i/o-file-protection-error?	list-ref
i/o-filename-error?	list-sort
i/o-invalid-position-error?	list-tail
i/o-port-error?	list?
i/o-read-error?	log
i/o-write-error?	lookahead-char
identifier-syntax	lookahead-u8
identifier?	magnitude
if	make-assertion-violation
imag-part	make-bytevector
immutable	make-custom-binary-input-port
implementation-restriction-violation?	make-custom-binary-input/output-port
inexact	make-custom-binary-output-port
inexact->exact	make-custom-textual-input-port
inexact?	make-custom-textual-input/output-port
infinite?	port
input-port?	make-custom-textual-output-port
Installing Libraries	make-enumeration
integer->char	make-eq-hashtable
integer-valued?	make-eqv-hashtable
integer?	make-error
irritants-condition?	make-hashtable
lambda	make-i/o-decoding-error
Language Interoperability	make-i/o-encoding-error
latin-1-codec	make-i/o-error
lcm	make-i/o-file-already-exists-error
least-fixnum	make-i/o-file-does-not-exist-error
length	make-i/o-file-is-read-only-error
let	make-i/o-file-protection-error
let*	make-i/o-filename-error
let*-values	make-i/o-invalid-position-error
let-syntax	make-i/o-port-error
let-values	make-i/o-read-error
letrec	make-i/o-write-error
letrec*	make-implementation-restriction-violation
letrec-syntax	make-irritants-condition
lexical-violation?	make-lexical-violation
Libraries and Collections	make-message-condition
list	
list->string	
list->vector	

make-no-infinities-violation	null?
make-no-nans-violation	number->string
make-non-continuable-violation	number?
make-polar	numerator
make-record-constructor-descriptor	odd?
make-record-type-descriptor	opaque
make-rectangular	open-bytevector-input-port
make-serious-condition	open-bytevector-output-port
make-string	open-file-input-port
make-syntax-violation	open-file-input/output-port
make-transcoder	open-file-output-port
make-undefined-violation	open-input-file
make-variable-transformer	open-output-file
make-vector	open-string-input-port
make-violation	open-string-output-port
make-warning	or
make-who-condition	output-port-buffer-mode
map	output-port?
max	pair?
member	parent
memp	parent-rtd
memq	partition
memv	peek-char
message-condition?	port-eof?
min	port-has-port-position?
mod	port-has-set-port-position!?
mod0	port-position
modulo	port-transcoder
mutable	port?
nan?	positive?
native-endianness	procedure?
native-eol-style	protocol
native-transcoder	put-bytevector
negative?	put-char
newline	put-datum
no-infinities-violation?	put-string
no-nans-violation?	put-u8
non-continuable-violation?	quasiquote
nongenerative	quasisyntax
not	quote
null-environment	quotient
	r6rs

R ⁶ RS Conformance	rnrs/bytevectors-6
R ⁶ RS Libraries	rnrs/conditions-6
R ⁶ RS Module Language	rnrs/control-6
R6RS: Scheme	rnrs/enums-6
raise	rnrs/eval-6
raise-continuable	rnrs/exceptions-6
rational-valued?	rnrs/files-6
rational?	rnrs/hashtables-6
rationalize	rnrs/io/ports-6
read	rnrs/io/simple-6
read-char	rnrs/lists-6
real->flonum	rnrs/mutable-pairs-6
real-part	rnrs/mutable-strings-6
real-valued?	rnrs/programs-6
real?	rnrs/r5rs-6
record-accessor	rnrs/records/inspection-6
record-constructor	rnrs/records/procedural-6
record-constructor-descriptor	rnrs/records/syntactic-6
record-field-mutable?	rnrs/sorting-6
record-mutator	rnrs/syntax-case-6
record-predicate	rnrs/unicode-6
record-rtd	round
record-type-descriptor	Running Top-Level Programs
record-type-descriptor?	scheme-report-environment
record-type-field-names	sealed
record-type-generative?	serious-condition?
record-type-name	set!
record-type-opaque?	set-car!
record-type-parent	set-cdr!
record-type-sealed?	set-port-position!
record-type-uid	simple-conditions
record?	sin
remainder	sint-list->bytevector
remove	sqrt
remp	standard-error-port
remq	standard-input-port
remv	standard-output-port
reverse	string
rnrs/arithmetic/bitwise-6	string->bytevector
rnrs/arithmetic/fixnums-6	string->list
rnrs/arithmetic/flonums-6	string->number
rnrs/base-6	string->symbol

string->utf16	syntax-violation-subform
string->utf32	syntax-violation?
string->utf8	tan
string-append	textual-port?
string-ci-hash	The Implementation of R ⁶ RS
string-ci<=?	transcoded-port
string-ci<?	transcoder-codec
string-ci=?	transcoder-eol-style
string-ci>=?	transcoder-error-handling-mode
string-ci>?	truncate
string-copy	u8-list->bytevector
string-downcase	uint-list->bytevector
string-fill!	undefined-violation?
string-foldcase	unless
string-for-each	unquote
string-hash	unquote-splicing
string-length	unsyntax
string-normalize-nfc	unsyntax-splicing
string-normalize-nfd	Using R ⁶ RS
string-normalize-nfkc	Using R ⁶ RS with DrRacket
string-normalize-nfkd	utf-16-codec
string-ref	utf-8-codec
string-set!	utf16->string
string-titlecase	utf32->string
string-upcase	utf8->string
string<=?	values
string<?	vector
string=?	vector->list
string>=?	vector-fill!
string>?	vector-for-each
string?	vector-length
substring	vector-map
symbol->string	vector-ref
symbol-hash	vector-set!
symbol=?	vector-sort
symbol?	vector-sort!
syntax	vector?
syntax->datum	violation?
syntax-case	warning?
syntax-rules	when
syntax-violation	who-condition?
syntax-violation-form	with-exception-handler

with-input-from-file
with-output-to-file
with-syntax
write
write-char
zero?