SRFIs: Libraries

Version 8.9

May 7, 2023

The Scheme Requests for Implementation (a.k.a. *SRFI*) process allows individual members of the Scheme community to propose libraries and extensions to be supported by multiple Scheme implementations.

Racket is distributed with implementations of many SRFIs, most of which can be implemented as libraries. To import the bindings of SRFI *n*, use

```
(require srfi/n)
```

This document lists the SRFIs that are supported by Racket and provides links to the original SRFI specifications (which are also distributed as part of Racket's documentation).

Contents

SRFI 1: List Library **6**

SRFI 2: AND-LET*: an AND with local bindings... **7**

SRFI 4: Homogeneous numeric vector datatypes **8**

SRFI 5: A compatible let form with signatures and rest arguments **9**

SRFI 6: Basic String Ports **12**

SRFI 7: Feature-based program configuration language **13**

SRFI 8: RECEIVE: Binding to multiple values **14**

SRFI 9: Defining Record Types **15**

SRFI 11: Syntax for receiving multiple values **16**

SRFI 13: String Libraries **17**

SRFI 14: Character-set Library **18**

SRFI 16: Syntax for procedures of variable arity **19**

SRFI 17: Generalized set! **20**

SRFI 19: Time Data Types and Procedures **21**

SRFI 23: Error reporting mechanism **22**

SRFI 25: Multi-dimensional Array Primitives **23**

SRFI 26: Notation for Specializing Parameters without Currying **24**

SRFI 27: Sources of Random Bits **25**

SRFI 28: Basic Format Strings **26**

SRFI 29: Localization **27**

SRFI 30: Nested Multi-line Comments **28**

SRFI 31: A special form rec for recursive evaluation **29**

SRFI 34: Exception Handling for Programs **30**

SRFI 35: Conditions **31**

SRFI 38: External Representation for Data With Shared Structure **32**

SRFI 39: Parameter objects **33**

SRFI 40: A Library of Streams **34**

SRFI 41: Streams **35**

SRFI 42: Eager Comprehensions **36**

SRFI 43: Vector Library **37**

SRFI 45: Primitives for Expressing Iterative Lazy Algorithms **38**

SRFI 48: Intermediate Format Strings **39**

SRFI 54: Formatting **40**

SRFI 57: Records **41**

SRFI 59: Vicinity **42**

SRFI 60: Integers as Bits **43**

SRFI 61: A more general cond clause **44**

SRFI 62: S-expression comments **45**

SRFI 63: Homogeneous and Heterogeneous Arrays **46**

SRFI 64: A Scheme API for test suites **47**

SRFI 66: Octet Vectors **48**

SRFI 67: Compare Procedures **49**

SRFI 69: Basic hash tables **50**

SRFI 71: Extended LET-syntax for multiple values **51**

SRFI 74: Octet-Addressed Binary Blocks **52**

SRFI 78: Lightweight testing **53**

SRFI 86: MU & NU simulating VALUES & CALL-WITH-VALUES... **54**

SRFI 87: => in case clauses **55**

SRFI 98: An interface to access environment variables **56**

Index **57**

Index **57**

SRFI 1: List Library

```
(require srfi/1)
```
package: `srfi-lite-lib`

Original specification: SRFI 1

This SRFI works with pairs and lists as in `racket`, which are immutable, so it does not export `set-car!` and `set-cdr!`. The other provided bindings that end in `!` are equivalent to the corresponding bindings without `!`. Functions that are documented in the SRFI in bold (but not bold italic) correspond to `racket` functions, while the others are distinct from same-named `racket` functions.

SRFI 2: AND-LET*: an AND with local bindings...

```
(require srfi/2)
```
package: `srfi-lib`

Original specification: SRFI 2

SRFI 4: Homogeneous numeric vector datatypes

`(require srfi/4)` package: `srfi-lib`

Original specification: SRFI 4

This SRFI's reader and printer syntax is not supported. The bindings are also available from `scheme/foreign`.

SRFI 5: A compatible let form with signatures and rest arguments

```
(require srfi/5)
```
package: srfi-lib

Original specification: SRFI 5

For historical reasons, the SRFI 5 specification document has a restrictive license and is not included in the main Racket distribution.

The implementation in `srfi/5` and this documentation are distributed under the same license as Racket: only the original specification document is restrictively licensed.

```
(let ([id init-expr] ...)
  body ...+)
(let ([id init-expr] ...+ rest-binding)
  body ...+)
(let loop-id ([id init-expr] ... maybe-rest-binding)
  body ...+)
(let (loop-id [id init-expr] ... maybe-rest-binding)
  body ...+)

maybe-rest-binding =
                   | rest-binding

      rest-binding = rest-id rest-init-expr ...
```

Like `let` from `racket/base`, but extended to support additional variants of named `let`.

As with `let` from `racket/base`, SRFI 5's `let` form conceptually expands to the immediate application of a function to the values of the *init-expr*s: the *id*s are bound in the *body*s (but not in any *init-expr*s or *rest-init-expr*s), and *loop-id*, if present, is bound in the *body*s to the function itself, allowing it to be used recursively. An *id* or a *rest-id* can shadow *loop-id*, but the *rest-id* (if given) and all *id*s much be distinct.

SRFI 5's `let` adds support for a syntax like `define`'s function shorthand, which allows the bindings to be written in a syntax resembling an application of the function bound to *loop-id*.

Additionally, SRFI 5's `let` adds support for rest arguments. If a *rest-id* is present, the function bound to *loop-id* (or the conceptual anonymous function, if *loop-id* is not used) will accept an unlimited number of additional arguments after its required by-position arguments, and the *rest-id* will be bound in the *body*s (but not in any *init-expr*s or *rest-init-expr*s) to a list of those additional arguments. The values of the *rest-init-expr*s are supplied as arguments to the initial, implicit application when the `let` form is evaluated, so the initial value bound to *rest-id* is `(list rest-init-expr ...)`.

Unlike the *kw-formals* of `lambda` and `define` or the *formals* of `case-lambda`, the bindings of SRFI 5's `let`, with or without a *rest-binding*, are always a proper (syntactic) list.

A *rest-binding* can be used with both the `define`-like and the named-`let`–like variants of `let`. It is also possible to use *rest-id* without any *loop-id*; however, as specified in the grammar, at least one *id–init-expr* pair is required in that case. (Otherwise, there would be an ambiguity with the `define`-like variant).

Examples:

```
; define-like bindings
> (define (factorial n)
    (let (fact [n n] [acc 1])
      (if (zero? n)
          acc
          (fact (sub1 n) (* n acc)))))
> (factorial 5)
120
> (factorial 11)
39916800
; rest arguments with named-let--like bindings
> (let reverse-onto ([lst '(a b c)]
                     tail)
    (if (null? lst)
        tail
        (apply reverse-onto (cdr lst) (car lst) tail)))
'(c b a)
> (let reverse-onto ([lst '(a b c)]
                     tail 'x 'y 'z)
    (if (null? lst)
        tail
        (apply reverse-onto (cdr lst) (car lst) tail)))
'(c b a x y z)
> (let no-evens (lst 1 2 3 4 5)
    (cond
      [(null? lst)
       '()]
      [(even? (car lst))
       (apply no-evens (cdr lst))]
      [else
       (cons (car lst) (apply no-evens (cdr lst)))]))
'(1 3 5)
; rest arguments with define-like bindings
> (let (reverse-onto [lst '(a b c)] tail)
    (if (null? lst)
        tail
        (apply reverse-onto (cdr lst) (car lst) tail)))
'(c b a)
> (let (reverse-onto [lst '(a b c)]    tail 'x 'y 'z)
```

```
    (if (null? lst)
        tail
        (apply reverse-onto (cdr lst) (car lst) tail)))
'(c b a x y z)
> (let (loop [continue? 0] args 'a 'a1 'a2)
    (case continue?
      [(0) (cons args (loop 1 'b))]
      [(1) (cons args (loop 2 'c 'd))]
      [else (list args)]))
'((a a1 a2) (b) (c d))
; rest arguments without any loop-id
> (let ([x 1]
        [y 2]
        z 3 4 5 6 7)
    (list* x y z))
'(1 2 3 4 5 6 7)
```

SRFI 6: Basic String Ports

```
(require srfi/6)
```
package: srfi-lib

Original specification: SRFI 6

This SRFI's bindings are also available in `racket/base`.

SRFI 7: Feature-based program configuration language

`(require srfi/7)` package: `srfi-lib`

Original specification: SRFI 7

SRFI 8: RECEIVE: Binding to multiple values

```
(require srfi/8)
```
package: `srfi-lite-lib`

Original specification: SRFI 8

SRFI 9: Defining Record Types

`(require srfi/9)` package: `srfi-lib`

Original specification: SRFI 9

SRFI 11: Syntax for receiving multiple values

```
(require srfi/11)
```
package: srfi-lib

Original specification: SRFI 11

This SRFI's bindings are also available in `racket/base`, but without support for dotted "rest" bindings.

SRFI 13: String Libraries

`(require srfi/13)` package: `srfi-lite-lib`

Original specification: SRFI 13

SRFI 14: Character-set Library

```
(require srfi/14)
```
package: `srfi-lite-lib`

Original specification: SRFI 14

SRFI 16: Syntax for procedures of variable arity

```
(require srfi/16)
```
package: srfi-lib

Original specification: SRFI 16

This SRFI's bindings are also available in `racket/base`.

SRFI 17: Generalized set!

```
(require srfi/17)
```
package: `srfi-lib`

Original specification: SRFI 17

SRFI 19: Time Data Types and Procedures

```
(require srfi/19)
```
package: srfi-lite-lib

Original specification: SRFI 19

The date structure produced by this SRFI library is identical to the one provided by `racket/base` in most cases (see `date`).

For backwards compatibility, when an invalid date field value is provided to the SRFI constructor, the constructor will produce a *lax date structure*. A lax date structure is *not* compatible with functions from `racket/base` or `racket/date`. SRFI functions such as `string->date` may return a lax date structure depending on the format string. The predicate `lax-date?` recognizes lax dat structures.

As an extension, Racket's implementation of `string->date` supports `~?` as a conversion specifier: it parses one- and two-digit years like `~y` and three- and four-digit years like `~Y`.

Examples:

```
> (string->date "4-1-99" "~d-~m-~?")
(date* 0 0 0 4 1 1999 1 3 #f -14400 0 "")
> (string->date "4-1-1999" "~d-~m-~?")
(date* 0 0 0 4 1 1999 1 3 #f -14400 0 "")
```

```
(lax-date? v) → boolean?
  v : any/c
```

Returns `#t` if `v` is a lax date structure. Otherwise, returns `#f`.

Examples:

```
> (lax-date? (make-date 0 19 10 10 14 "bogus" "bogus" 0))
#t
> (lax-date? (make-date 0 19 10 10 14 1 2013 0))
#f
> (lax-date? (string->date "10:21:00" "~H:~M:~S"))
#t
```

SRFI 23: Error reporting mechanism

```
(require srfi/23)
```
package: srfi-lib

Original specification: SRFI 23

This SRFI's bindings are also available in `racket/base`.

SRFI 25: Multi-dimensional Array Primitives

`(require srfi/25)` package: `srfi-lib`

Original specification: SRFI 25

SRFI 26: Notation for Specializing Parameters without Currying

`(require srfi/26)` package: `srfi-lib`

Original specification: SRFI 26

SRFI 27: Sources of Random Bits

```
(require srfi/27)
```
package: `srfi-lib`

Original specification: SRFI 27

SRFI 28: Basic Format Strings

```
(require srfi/28)
```
package: srfi-lib

Original specification: SRFI 28

This SRFI's bindings are also available in racket/base.

SRFI 29: Localization

```
(require srfi/29)
```
package: `srfi-lite-lib`

Original specification: SRFI 29

SRFI 30: Nested Multi-line Comments

```
(require srfi/30)
```
package: srfi-lib

Original specification: SRFI 30

This SRFI's syntax is part of Racket's default reader.

SRFI 31: A special form rec for recursive evaluation

```
(require srfi/31)
```
package: `srfi-lib`

Original specification: SRFI 31

SRFI 34: Exception Handling for Programs

```
(require srfi/34)
```
package: srfi-lib

Original specification: SRFI 34

An `else` is recognized as either the one from `racket/base` or as an identifier with the symbolic form `'else` and no binding.

SRFI 35: Conditions

```
(require srfi/35)
```
package: `srfi-lib`

Original specification: SRFI 35

SRFI 38: External Representation for Data With Shared Structure

```
(require srfi/38)
```
package: srfi-lib

Original specification: SRFI 38

This SRFI's syntax is part of Racket's default reader and printer.

SRFI 39: Parameter objects

```
(require srfi/39)
```
package: srfi-lib

Original specification: SRFI 39

This SRFI's bindings are also available in `racket/base`.

SRFI 40: A Library of Streams

```
(require srfi/40)
```
package: `srfi-lib`

Original specification: SRFI 40

Superceded by `srfi/41`.

SRFI 41: Streams

```
(require srfi/41)
```
package: srfi-lib

Original specification: SRFI 41

The `stream-cons` operation from `srfi/41` is the same as from `racket/stream`.

SRFI 42: Eager Comprehensions

```
(require srfi/42)
```
package: srfi-lib

Original specification: SRFI 42

Forms that syntactically detect `if` recognize both `if` from `racket/base` and `if` from `mzscheme`.

SRFI 43: Vector Library

```
(require srfi/43)
```
package: `srfi-lib`

Original specification: SRFI 43

SRFI 45: Primitives for Expressing Iterative Lazy Algorithms

```
(require srfi/45)
```
package: srfi-lib

Original specification: SRFI 45

Additional binding:

```
(promise? v) → boolean?
  v : any/c
```

Returns #t if v is a promise, #f otherwise.

SRFI 48: Intermediate Format Strings

```
(require srfi/48)
```
package: `srfi-lib`

Original specification: SRFI 48

SRFI 54: Formatting

```
(require srfi/54)
```
package: `srfi-lib`

Original specification: SRFI 54

SRFI 57: Records

```
(require srfi/57)
```
package: `srfi-lib`

Original specification: SRFI 57

SRFI 59: Vicinity

```
(require srfi/59)
```
package: `srfi-lib`

Original specification: SRFI 59

SRFI 60: Integers as Bits

```
(require srfi/60)
```
package: `srfi-lib`

Original specification: SRFI 60

SRFI 61: A more general cond clause

```
(require srfi/61)
```
package: `srfi-lib`

Original specification: SRFI 61

SRFI 62: S-expression comments

Original specification: SRFI 62

This SRFI's syntax is part of Racket's default reader (no `require` is needed).

SRFI 63: Homogeneous and Heterogeneous Arrays

```
(require srfi/63)
```
package: `srfi-lib`

Original specification: SRFI 63

SRFI 64: A Scheme API for test suites

`(require srfi/64)` package: `srfi-lib`

Original specification: SRFI 64

SRFI 66: Octet Vectors

```
(require srfi/66)
```
package: `srfi-lib`

Original specification: SRFI 66

SRFI 67: Compare Procedures

`(require srfi/67)` package: `srfi-lib`

Original specification: SRFI 67

SRFI 69: Basic hash tables

```
(require srfi/69)
```
package: `srfi-lib`

Original specification: SRFI 69

SRFI 71: Extended LET-syntax for multiple values

```
(require srfi/71)
```
package: srfi-lib

Original specification: SRFI 71

SRFI 74: Octet-Addressed Binary Blocks

```
(require srfi/74)
```
package: `srfi-lib`

Original specification: SRFI 74

SRFI 78: Lightweight testing

`(require srfi/78)` package: `srfi-lib`

Original specification: SRFI 78

SRFI 86: MU & NU simulating VALUES & CALL-WITH-VALUES...

`(require srfi/86)` package: `srfi-lib`

Original specification: SRFI 86

SRFI 87: => in case clauses

```
(require srfi/87)
```
package: srfi-lib

Original specification: SRFI 87

SRFI 98: An interface to access environment variables

```
(require srfi/98)
```
package: `srfi-lib`

Original specification: SRFI 98

Index

->char-set, 18
:, 36
:char-range, 36
:dispatched, 36
:do, 36
:generator-proc, 36
:integers, 36
:let, 36
:list, 36
:parallel, 36
:port, 36
:range, 36
:real-range, 36
:string, 36
:until, 36
:vector, 36
:while, 36
add-duration, 21
add-duration!, 21
alist-cons, 6
alist-copy, 6
alist-delete, 6
alist-delete!, 6
and, 36
and-let*, 7
any, 6
any?-ec, 36
append, 6
append!, 6
append-ec, 36
append-map, 6
append-map!, 6
append-reverse, 6
append-reverse!, 6
array, 23
array-end, 23
array-rank, 23
array-ref, 23
array-set!, 23
array-start, 23
array?, 23
assoc, 6
assq, 6
assv, 6
begin, 36
break, 6
break!, 6
car, 6
car+cdr, 6
case-lambda, 19
cddadr, 6
cddddr, 6
cdr, 6
char-set, 18
char-set->list, 18
char-set->string, 18
char-set-adjoin, 18
char-set-adjoin!, 18
char-set-any, 18
char-set-complement, 18
char-set-complement!, 18
char-set-contains?, 18
char-set-copy, 18
char-set-count, 18
char-set-cursor, 18
char-set-cursor-next, 18
char-set-delete, 18
char-set-delete!, 18
char-set-diff+intersection, 18
char-set-diff+intersection!, 18
char-set-difference, 18
char-set-difference!, 18
char-set-every, 18
char-set-filter, 18
char-set-filter!, 18
char-set-fold, 18
char-set-for-each, 18
char-set-hash, 18
char-set-intersection, 18
char-set-intersection!, 18
char-set-map, 18
char-set-ref, 18

char-set-size, 18
char-set-unfold, 18
char-set-unfold!, 18
char-set-union, 18
char-set-union!, 18
char-set-xor, 18
char-set-xor!, 18
char-set:ascii, 18
char-set:blank, 18
char-set:digit, 18
char-set:empty, 18
char-set:full, 18
char-set:graphic, 18
char-set:hex-digit, 18
char-set:iso-control, 18
char-set:letter, 18
char-set:letter+digit, 18
char-set:lower-case, 18
char-set:printing, 18
char-set:punctuation, 18
char-set:symbol, 18
char-set:title-case, 18
char-set:upper-case, 18
char-set:whitespace, 18
char-set<=, 18
char-set=, 18
char-set?, 18
check-substring-spec, 17
circular-list, 6
circular-list?, 6
concatenate, 6
concatenate!, 6
cons, 6
cons*, 6
copy-time, 21
count, 6
current-country, 27
current-date, 21
current-julian-day, 21
current-language, 27
current-locale-details, 27
current-modified-julian-day, 21
current-time, 21
cut, 24
cute, 24
date->julian-day, 21
date->modified-julian-day, 21
date->string, 21
date->time-monotonic, 21
date->time-tai, 21
date->time-utc, 21
date-day, 21
date-hour, 21
date-minute, 21
date-month, 21
date-nanosecond, 21
date-second, 21
date-week-day, 21
date-week-number, 21
date-year, 21
date-year-day, 21
date-zone-offset, 21
date?, 21
declare-bundle!, 27
default-random-source, 25
define-record-type, 15
define-stream, 35
delay, 38
delete, 6
delete!, 6
delete-duplicates, 6
delete-duplicates!, 6
do-ec, 36
dotted-list?, 6
drop, 6
drop-right, 6
drop-right!, 6
drop-while, 6
eager, 38
eighth, 6
end-of-char-set?, 18
error, 22
every, 6
every?-ec, 36

f32vector, 8
f64vector, 8
fifth, 6
filter, 6
filter!, 6
filter-map, 6
find, 6
find-tail, 6
first, 6
first-ec, 36
fold, 6
fold-ec, 36
fold-right, 6
fold3-ec, 36
for-each, 6
force, 38
format, 26
format, 39
fourth, 6
generator, 36
get-environment-variable, 56
get-environment-variables, 56
get-output-string, 12
getter-with-setter, 20
guard, 30
home-vicinity, 42
if, 36
implementation-vicinity, 42
in-vicinity, 42
iota, 6
julian-day->date, 21
julian-day->time-monotonic, 21
julian-day->time-tai, 21
julian-day->time-utc, 21
kmp-step, 17
last, 6
last-ec, 36
last-pair, 6
lax date structure, 21
lax-date?, 21
lazy, 38
length, 6
length+, 6
let, 9
let*-values, 16
let-string-start+end, 17
let-values, 16
library-vicinity, 42
list, 6
list->char-set, 18
list->char-set!, 18
list->stream, 35
list->string, 17
list->vector, 37
list-copy, 6
list-ec, 36
list-index, 6
list-ref, 6
list-tabulate, 6
load-bundle!, 27
localized-template, 27
lset-adjoin, 6
lset-diff+intersection, 6
lset-diff+intersection!, 6
lset-difference, 6
lset-difference!, 6
lset-intersection, 6
lset-intersection!, 6
lset-union, 6
lset-union!, 6
lset-xor, 6
lset-xor!, 6
lset=, 6
make-array, 23
make-date, 21
make-kmp-restart-vector, 17
make-list, 6
make-parameter, 33
make-random-source, 25
make-string, 17
make-time, 21
make-vector, 37
make-vicinity, 42
map, 6

map!, 6
map-in-order, 6
max-ec, 36
member, 6
memq, 6
memv, 6
min-ec, 36
modified-julian-day->date, 21
modified-julian-day->time-monotonic, 21
modified-julian-day->time-tai, 21
modified-julian-day->time-utc, 21
nested, 36
ninth, 6
not, 36
not-pair?, 6
null-list?, 6
null?, 6
open-input-string, 12
open-output-string, 12
or, 36
pair-fold, 6
pair-fold-right, 6
pair-for-each, 6
pair?, 6
parameterize, 33
partition, 6
partition!, 6
pathname->vicinity, 42
port->stream, 35
product-ec, 36
program, 13
program-vicinity, 42
promise?, 38
proper-list?, 6
raise, 30
random-integer, 25
random-real, 25
random-source-make-integers, 25
random-source-make-reals, 25
random-source-pseudo-randomize!, 25
random-source-randomize!, 25
random-source-state-ref, 25
random-source-state-ref!, 25
random-source?, 25
read-with-shared-structure, 32
rec, 29
receive, 14
reduce, 6
reduce-right, 6
remove, 6
remove!, 6
reverse, 6
reverse!, 6
reverse-list->string, 17
reverse-list->vector, 37
reverse-vector->list, 37
s16vector, 8
s32vector, 8
s64vector, 8
s8vector, 8
second, 6
set!, 20
set-time-nanosecond!, 21
set-time-second!, 21
set-time-type!, 21
seventh, 6
shape, 23
share-array, 23
sixth, 6
span, 6
span!, 6
split-at, 6
split-at!, 6
SRFI, 1
SRFI 11: Syntax for receiving multiple values, 16
SRFI 13: String Libraries, 17
SRFI 14: Character-set Library, 18
SRFI 16: Syntax for procedures of variable arity, 19
SRFI 17: Generalized set!, 20
SRFI 19: Time Data Types and Procedures, 21

SRFI 1: List Library, 6
SRFI 23: Error reporting mechanism, 22
SRFI 25: Multi-dimensional Array Primitives, 23
SRFI 26: Notation for Specializing Parameters without Currying, 24
SRFI 27: Sources of Random Bits, 25
SRFI 28: Basic Format Strings, 26
SRFI 29: Localization, 27
SRFI 2: AND-LET*: an AND with local bindings..., 7
SRFI 30: Nested Multi-line Comments, 28
SRFI 31: A special form rec for recursive evaluation, 29
SRFI 34: Exception Handling for Programs, 30
SRFI 35: Conditions, 31
SRFI 38: External Representation for Data With Shared Structure, 32
SRFI 39: Parameter objects, 33
SRFI 40: A Library of Streams, 34
SRFI 41: Streams, 35
SRFI 42: Eager Comprehensions, 36
SRFI 43: Vector Library, 37
SRFI 45: Primitives for Expressing Iterative Lazy Algorithms, 38
SRFI 48: Intermediate Format Strings, 39
SRFI 4: Homogeneous numeric vector datatypes, 8
SRFI 54: Formatting, 40
SRFI 57: Records, 41
SRFI 59: Vicinity, 42
SRFI 5: A compatible let form with signatures and rest arguments, 9
SRFI 60: Integers as Bits, 43
SRFI 61: A more general cond clause, 44
SRFI 62: S-expression comments, 45
SRFI 63: Homogeneous and Heterogeneous Arrays, 46
SRFI 64: A Scheme API for test suites, 47
SRFI 66: Octet Vectors, 48
SRFI 67: Compare Procedures, 49
SRFI 69: Basic hash tables, 50
SRFI 6: Basic String Ports, 12
SRFI 71: Extended LET-syntax for multiple values, 51
SRFI 74: Octet-Addressed Binary Blocks, 52
SRFI 78: Lightweight testing, 53
SRFI 7: Feature-based program configuration language, 13
SRFI 86: MU & NU simulating VALUES & CALL-WITH-VALUES..., 54
SRFI 87: => in case clauses, 55
SRFI 8: RECEIVE: Binding to multiple values, 14
SRFI 98: An interface to access environment variables, 56
SRFI 9: Defining Record Types, 15
srfi/1, 6
srfi/11, 16
srfi/13, 17
srfi/14, 18
srfi/16, 19
srfi/17, 20
srfi/19, 21
srfi/2, 7
srfi/23, 22
srfi/25, 23
srfi/26, 24
srfi/27, 25
srfi/28, 26
srfi/29, 27
srfi/30, 28
srfi/31, 29
srfi/34, 30
srfi/35, 31
srfi/38, 32
srfi/39, 33
srfi/4, 8
srfi/40, 34
srfi/41, 35
srfi/42, 36
srfi/43, 37
srfi/45, 38
srfi/48, 39

srfi/5, 9
srfi/54, 40
srfi/57, 41
srfi/59, 42
srfi/6, 12
srfi/60, 43
srfi/61, 44
srfi/63, 46
srfi/64, 47
srfi/66, 48
srfi/67, 49
srfi/69, 50
srfi/7, 13
srfi/71, 51
srfi/74, 52
srfi/78, 53
srfi/8, 14
srfi/86, 54
srfi/87, 55
srfi/9, 15
srfi/98, 56
SRFIs: Libraries, 1
store-bundle, 27
stream, 34
stream, 35
stream->list, 35
stream-append, 35
stream-car, 34
stream-car, 35
stream-cdr, 34
stream-cdr, 35
stream-concat, 35
stream-cons, 34
stream-cons, 35
stream-constant, 35
stream-delay, 34
stream-drop, 35
stream-drop-while, 35
stream-filter, 34
stream-filter, 35
stream-fold, 35
stream-for-each, 34
stream-for-each, 35
stream-from, 35
stream-iterate, 35
stream-lambda, 35
stream-length, 35
stream-let, 35
stream-map, 34
stream-map, 35
stream-match, 35
stream-null, 34
stream-null, 35
stream-null?, 34
stream-null?, 35
stream-of, 35
stream-pair?, 35
stream-range, 35
stream-ref, 35
stream-reverse, 35
stream-scan, 35
stream-take, 35
stream-take-while, 35
stream-unfold, 35
stream-unfoldn, 34
stream-zip, 35
stream?, 34
stream?, 35
string, 17
string->char-set, 18
string->char-set!, 18
string->date, 21
string->list, 17
string-any, 17
string-append, 17
string-append-ec, 36
string-append/shared, 17
string-ci<, 17
string-ci<=, 17
string-ci<>, 17
string-ci=, 17
string-ci>, 17
string-ci>=, 17
string-compare, 17

string-compare-ci, 17
string-concatenate, 17
string-concatenate-reverse, 17
string-concatenate-reverse/shared, 17
string-concatenate/shared, 17
string-contains, 17
string-contains-ci, 17
string-copy, 17
string-copy!, 17
string-count, 17
string-delete, 17
string-downcase, 17
string-downcase!, 17
string-drop, 17
string-drop-right, 17
string-ec, 36
string-every, 17
string-fill!, 17
string-filter, 17
string-fold, 17
string-fold-right, 17
string-for-each, 17
string-for-each-index, 17
string-hash, 17
string-hash-ci, 17
string-index, 17
string-index-right, 17
string-join, 17
string-kmp-partial-search, 17
string-length, 17
string-map, 17
string-map!, 17
string-null?, 17
string-pad, 17
string-pad-right, 17
string-parse-final-start+end, 17
string-parse-start+end, 17
string-prefix-ci?, 17
string-prefix-length, 17
string-prefix-length-ci, 17
string-prefix?, 17
string-ref, 17
string-replace, 17
string-reverse, 17
string-reverse!, 17
string-set!, 17
string-skip, 17
string-skip-right, 17
string-suffix-ci?, 17
string-suffix-length, 17
string-suffix-length-ci, 17
string-suffix?, 17
string-tabulate, 17
string-take, 17
string-take-right, 17
string-titlecase, 17
string-titlecase!, 17
string-tokenize, 17
string-trim, 17
string-trim-both, 17
string-trim-right, 17
string-unfold, 17
string-unfold-right, 17
string-upcase, 17
string-upcase!, 17
string-xcopy!, 17
string<, 17
string<=, 17
string<>, 17
string=, 17
string>, 17
string>=, 17
string?, 17
sub-vicinity, 42
substring-spec-ok?, 17
substring/shared, 17
subtract-duration, 21
subtract-duration!, 21
sum-ec, 36
take, 6
take!, 6
take-right, 6
take-while, 6

take-while!, 6
tenth, 6
third, 6
time-difference, 21
time-difference!, 21
time-duration, 21
time-monotonic, 21
time-monotonic->date, 21
time-monotonic->julian-day, 21
time-monotonic->modified-julian-day, 21
time-monotonic->time-tai, 21
time-monotonic->time-tai!, 21
time-monotonic->time-utc, 21
time-monotonic->time-utc!, 21
time-nanosecond, 21
time-process, 21
time-resolution, 21
time-second, 21
time-tai, 21
time-tai->date, 21
time-tai->julian-day, 21
time-tai->modified-julian-day, 21
time-tai->time-monotonic, 21
time-tai->time-monotonic!, 21
time-tai->time-utc, 21
time-tai->time-utc!, 21
time-thread, 21
time-type, 21
time-utc, 21
time-utc->date, 21
time-utc->julian-day, 21
time-utc->modified-julian-day, 21
time-utc->time-monotonic, 21
time-utc->time-monotonic!, 21
time-utc->time-tai, 21
time-utc->time-tai!, 21
time<=?, 21
time<?, 21
time=?, 21
time>=?, 21
time>?, 21
time?, 21
u16vector, 8
u32vector, 8
u64vector, 8
u8vector, 8
ucs-range->char-set, 18
ucs-range->char-set!, 18
unfold, 6
unfold-right, 6
unzip1, 6
unzip2, 6
unzip3, 6
unzip4, 6
unzip5, 6
user-vicinity, 42
vector, 37
vector->list, 37
vector-any, 37
vector-append, 37
vector-binary-search, 37
vector-concatenate, 37
vector-copy, 37
vector-copy!, 37
vector-count, 37
vector-ec, 36
vector-empty?, 37
vector-every, 37
vector-fill!, 37
vector-fold, 37
vector-fold-right, 37
vector-for-each, 37
vector-index, 37
vector-index-right, 37
vector-length, 37
vector-map, 37
vector-map!, 37
vector-of-length-ec, 36
vector-ref, 37
vector-reverse!, 37
vector-reverse-copy, 37
vector-reverse-copy!, 37
vector-set!, 37

vector-skip, 37
vector-skip-right, 37
vector-swap!, 37
vector-unfold, 37
vector-unfold-right, 37
vector=, 37
vector?, 37
vicinity:suffix?, 42
with-exception-handler, 30
write-with-shared-structure, 32
xcons, 6
xsubstring, 17
zip, 6