Scheme Requests for Implementation **Since ICFP 2018**

Arthur A. Gleckler
SRFI Editor

Scheme and Functional Programming Workshop
International Conference on Functional Programming
Ljubljana Slovenia

2022-09-16

How SRFI can help you, the Scheme programmer

- What is SRFI?
- History
- Flavor of progress since ICFP 2018
- Story
- **Edit: I made a mistake. This should all be relative to ICFP 2019, not 2018, since 2019 is when I gave my previous SRFI talk. The result is that there's a year of overlap. Oops. Still, we've made a lot of progress.**

Scheme standardization

- The Reports
- SRFI

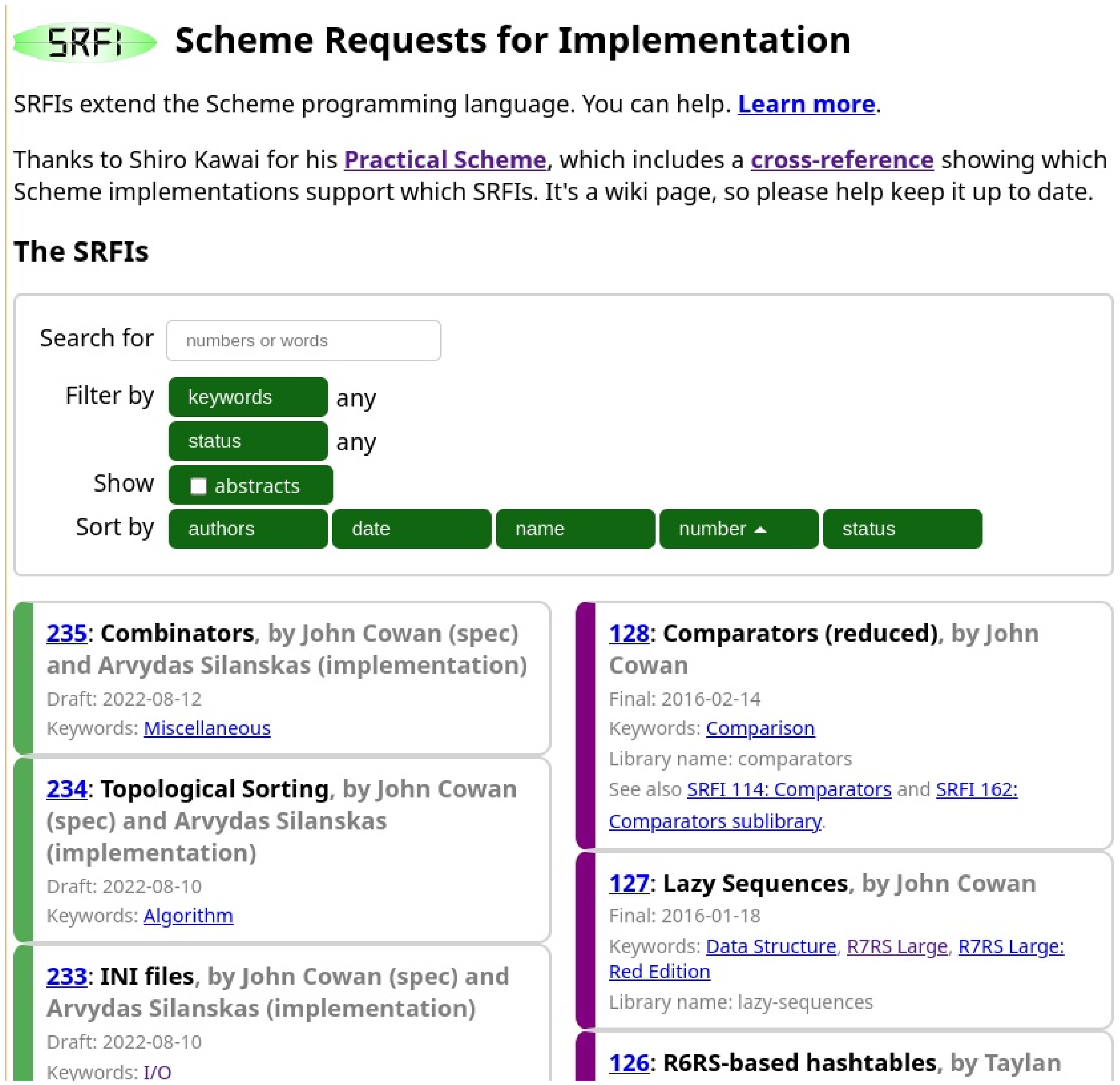
R^{n}RS and IEEE-1178

- Specifies the whole language and libraries.
- Ideally, represent consensus.
- Slower.
- Long history
  - *RS*
  - *R^{2}RS*
  - *R^{3}RS*
  - *IEEE-1178*
  - *R^{4}RS*
  - *R^{5}RS*
  - *R^{6}RS*
  - *R^{7}RS Small*
  - *R^{7}RS Large (in progress)*

SRFI

- Specifies one particular piece of the language or libraries.
- Represent, at a minimum, a request by an individual author.
- Faster.
- Used as part of R^{6}RS and R^{7}RS processes.
- Since 1998 (when R^{5}RS was published).
- I've been editor since 2015.

SRFI home page (srfi.schemers.org)



Scheme Requests for Implementation

SRFIs extend the Scheme programming language. You can help. Learn more.

Thanks to Shiro Kawai for his Practical Scheme, which includes a cross-reference showing which Scheme implementations support which SRFIs. It's a wiki page, so please help keep it up to date.

The SRFIs

235: **Combinators**, by John Cowan (spec) and Arvydas Silanskas (implementation)
Draft: 2022-08-12
Keywords: Miscellaneous

234: **Topological Sorting**, by John Cowan (spec) and Arvydas Silanskas (implementation)
Draft: 2022-08-10
Keywords: Algorithm

233: **INI files**, by John Cowan (spec) and Arvydas Silanskas (implementation)
Draft: 2022-08-10
Keywords: I/O

128: **Comparators (reduced)**, by John Cowan
Final: 2016-02-14
Keywords: Comparison
Library name: comparators
See also SRFI 114: Comparators and SRFI 162: Comparators sublibrary.

127: **Lazy Sequences**, by John Cowan
Final: 2016-01-18
Keywords: Data Structure, R7RS Large, R7RS Large: Red Edition
Library name: lazy-sequences

126: **R6RS-based hashtables**, by Taylan

SRFI web site

- SRFI documents (normative)
- sample implementations
- email archives
    - *one list per SRFI (236 of them)*
    - *`srfi-announce`*
    - *`srfi-discuss`*
    - *`schemedoc`*
    - *`schemeorg`*
    - *`schemepersist`*
    - *`schemeregistry`*
    - *`schemetest`*
    - *`schemeweb`*

SRFI keywords (I)

- Algorithm
- Assignment
- Binding
- Comparison
- Concurrency
- Continuations
- Control Flow
- Data Structure
- Error Handling
- Exceptions
- Features

SRFI keywords (II)

- I/O
- Internationalization
- Introspection
- Lazy Evaluation
- Miscellaneous
- Modules
- Multiple-Value Returns
- Numbers
- Operating System
- Optimization
- Parameters

SRFI keywords (III)

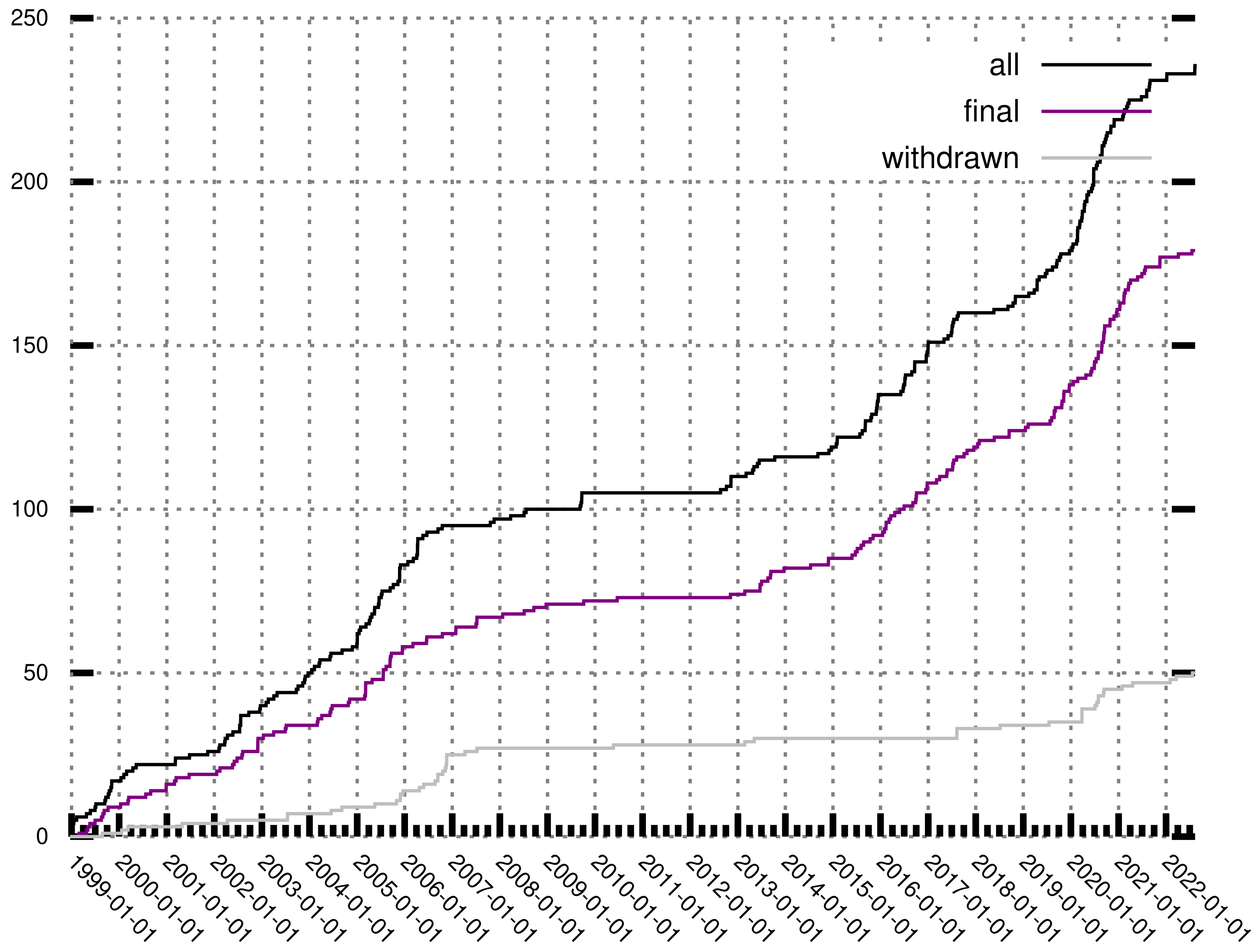
- Pattern Matching
- R6RS process
- R7RS Large
- R7RS Large: Red Edition
- R7RS Large: Tangerine Edition
- Randomness
- Reader Syntax
- SICP
- Superseded
- Syntax
- Testing
- Type Checking

Before ICFP 2018

http://www.schemeworkshop.org/2018/Gleckler.pdf

Earlier talk available on YouTube.

Progress



250
200
150
100
50
0
all
final
withdrawn
1999-01-01
2000-01-01
2001-01-01
2002-01-01
2003-01-01
2004-01-01
2005-01-01
2006-01-01
2007-01-01
2008-01-01
2009-01-01
2010-01-01
2011-01-01
2012-01-01
2013-01-01
2014-01-01
2015-01-01
2016-01-01
2017-01-01
2018-01-01
2019-01-01
2020-01-01
2021-01-01
2022-01-01

SRFIs since ICFP 2018 (I)

- 121 Generators
- 154 First-class dynamic extents
- 155 Promises
- 159 Combinator Formatting
- 160 Homogeneous numeric vector libraries
- 161 Unifiable Boxes
- 162 Comparators sublibrary
- 163 Enhanced array literals
- 164 Enhanced multi-dimensional Arrays
- 165 The Environment Monad
- 166 Monadic Formatting
- 167 Ordered Key Value Store
- 168 Generic Tuple Store Database
- 169 Underscores in numbers
- 170 POSIX API

SRFIs since ICFP 2018 (II)

- 171 Transducers
- 172 Two Safer Subsets of R7RS
- 173 Hooks
- 174 POSIX Timespecs
- 175 ASCII character library
- 176 Version flag
- 177 Portable keyword arguments
- 178 Bitvector library
- 179 Nonempty Intervals and Generalized Arrays (Updated)
- 180 JSON
- 181 Custom ports (including transcoded ports)
- 182 ADBMAL, ALET, and ALET*
- 183 Another format procedure, Fox
- 184 define-record-lambda
- 185 Linear adjustable-length strings

SRFIs since ICFP 2018 (III)

- 186 Transcoders and transcoded ports
- 187 ALAMBDA and ADEFINE
- 188 Splicing binding constructs for syntactic keywords
- 189 Maybe and Either: optional container types
- 190 Coroutine Generators
- 191 Procedure Arity Inspection
- 192 Port Positioning
- 193 Command line
- 194 Random data generators
- 195 Multiple-value boxes
- 196 Range Objects
- 197 Pipeline Operators
- 198 Foreign Interface Status
- 199 POSIX errno manipulation
- 200 Pattern Matching

SRFIs since ICFP 2018 (IV)

- 201 Syntactic Extensions to the Core Scheme Bindings
- 202 Pattern-matching Variant of the and-let* Form that Supports Multiple Values
- 203 A Simple Picture Language in the Style of SICP
- 204 Wright-Cartwright-Shinn Pattern Matcher
- 205 POSIX Terminal Fundamentals
- 206 Auxiliary Syntax Keywords
- 207 String-notated bytevectors
- 208 NaN procedures
- 209 Enums and Enum Sets
- 210 Procedures and Syntax for Multiple Values
- 211 Scheme Macro Libraries
- 212 Aliases
- 213 Identifier Properties
- 214 Flexvectors
- 215 Central Log Exchange

SRFIs since ICFP 2018 (V)

- 216 SICP Prerequisites (Portable)
- 217 Integer Sets
- 218 Unicode Numerals
- 219 Define higher-order lambda
- 220 Line directives
- 221 Generator/accumulator sub-library
- 222 Compound Objects
- 223 Generalized binary search procedures
- 224 Integer Mappings
- 225 Dictionaries
- 226 Control Features
- 227 Optional Arguments
- 228 A further comparator library
- 229 Tagged Procedures
- 230 Atomic Operations

SRFIs since ICFP 2018 (VI)

- 231 Intervals and Generalized Arrays
- 232 Flexible curried procedures
- 233 INI files
- 234 Topological Sorting
- 235 Combinators

R7RS Large (I)

This list includes all, not just *since ICFP 2018*.

- 1 List Library
- 14 Character-set Library
- 41 Streams
- 101 Purely Functional Random-Access Pairs and Lists
- 111 Boxes
- 113 Sets and bags
- 115 Scheme Regular Expressions
- 116 Immutable List Library
- 117 Queues based on lists
- 124 Ephemerons
- 125 Intermediate hash tables
- 127 Lazy Sequences
- 132 Sort Libraries
- 133 Vector Library (R7RS-compatible)

R7RS Large (II)

- 134 Immutable Deques
- 135 Immutable Texts
- 141 Integer division
- 143 Fixnums
- 144 Flonums
- 146 Mappings
- 151 Bitwise Operations
- 158 Generators and Accumulators
- *159 Combinator Formatting*
- *160 Homogeneous numeric vector libraries*

Too many new SRFIs to cover in detail

So I'll tell a story.

Story

animating an airplane's flight

https://commons.wikimedia.org/wiki/File:Slovenia_location_map.svg

https://commons.wikimedia.org/wiki/File:FlyingPete_Icons-Boeing777-300ER.svg

Operating system interface with many optional arguments

SRFI 227 Optional Arguments

```
(define n 1)

(define g
  (opt-lambda (n (m (* n 2)))
    (list n m)))

(g 2) ⇒ (2 2)

(g 2 3) ⇒ (2 3)

(let-optionals '(1)
    (x (y 2) (z 3))
  (list x y z))
```

Hard-coded hex magic numbers in network protocol

SRFI 169 Underscores in numbers

`1234567890` can be written `1_234_567_890`.

`#xfeeddadf00d` can be written `#xfeed_dad_f00d`.

Processing contents of binary data files

SRFI 160 Homogeneous numeric vector libraries

`s8vector`	signed exact integer in the range -2^{7} to $2^{7}-1$
`u8vector`	unsigned exact integer in the range 0 to $2^{8}-1$
`s16vector`	signed exact integer in the range -2^{15} to $2^{15}-1$
`u16vector`	unsigned exact integer in the range 0 to $2^{16}-1$
`s32vector`	signed exact integer in the range -2^{31} to $2^{31}-1$
`u32vector`	unsigned exact integer in the range 0 to $2^{32}-1$
`s64vector`	signed exact integer in the range -2^{63} to $2^{63}-1$
`u64vector`	unsigned exact integer in the range 0 to $2^{64}-1$
`f32vector`	inexact real, typically 32 bits
`f64vector`	inexact real, typically 64 bits
`c64vector`	inexact complex, typically 64 bits
`c128vector`	inexact complex, typically 128 bits

SRFI 160 Homogeneous numeric vector libraries

```
(make-u16vector size [fill]) ⇒ u16vector

(u8vector-unfold-right f length seed) ⇒ u8vector

(f64vector-concatenate list-of-f64vectors) ⇒ f64vector

(u32vector-drop-while pred? u32vec) ⇒ u32vector
```

Using lots of curried procedures

SRFI 219 Define higher-order lambda

```
(define ((greet-with-prefix prefix) suffix)
  (string-append prefix " " suffix))

(define greet (greet-with-prefix "Hello"))

(greet "there!") ⇒ "Hello there!"
```

Airports are numbered

The visible set changes as the display moves.

SRFI 224 Integer Mappings

```
(fxmapping k₁ obj₁ k₂ …) ⇒ fxmapping

(fxmapping->alist
 (fxmapping-unfold (lambda (i) (= i 4))
                   (lambda (i) (values i (make-string i #\a)))
                   (lambda (i) (+ i 1))
                   0))
 ⇒ ((0 . "") (1 . "a") (2 . "aa") (3 . "aaa"))
```

Image processing

SRFI 231 Intervals and Generalized Arrays

About to be finalized.

```scheme
(let* ((greys (pgm-greys test-pgm))
       (edge-array
        (array-copy
         (array-map
          abs
          (array-convolve (pgm-pixels test-pgm)
                          edge-filter))))
       (max-pixel
        (array-foldl max 0 edge-array))
       (normalizer
        (inexact (/ greys max-pixel))))
  (write-pgm
   (make-pgm
    greys
    (array-map (lambda (p)
                 (- greys
                    (round-and-clip (* p normalizer) greys)))
               edge-array))
   "edge-test.pgm"))
```

Generate API documentation

SRFI 166 Monadic Formatting (I)

```
(define func
  '(define (fold kons knil ls)
     (let lp ((ls ls) (acc knil))
       (if (null? ls) acc (lp (cdr ls) (kons (car ls) acc))))))

(define doc
  (string-append
    "The fundamental list iterator.  Applies KONS to each "
    "element of LS and the result of the previous application, "
    "beginning with KNIL.  With KONS as CONS and KNIL as '(), "
    "equivalent to REVERSE."))
```

SRFI 166 Monadic Formatting (II)

```
(show #t (columnar 4 'infinite 'right (line-numbers)
                   (pretty func)
                   " ; "
                   (justified doc)))
```

```
1(define (fold kons knil ls)         ; The   fundamental   list  iterator.
2  (let lp ((ls ls) (acc knil))      ; Applies  KONS  to  each  element of
3    (if (null? ls)                  ; LS  and  the result of the previous
4        acc                         ; application,  beginning  with KNIL.
5        (lp (cdr ls)                ; With  KONS as CONS and KNIL as '(),
6            (kons (car ls) acc))))) ; equivalent to REVERSE.
```

Reading and writing JSON configuration files

SRFI 180 JSON

```
(json-null? obj) ⇒ boolean

json-nesting-depth-limit (parameter)

json-number-of-character-limit (parameter)

(json-generator [port-or-generator]) ⇒ generator

(json-fold proc array-start array-end object-start object-end seed
           [port-or-generator])

(json-read [port-or-generator]) ⇒ object

(json-lines-read [port-or-generator]) ⇒ generator

(json-sequence-read [port-or-generator]) ⇒ generator

(json-accumulator port-or-accumulator) ⇒ procedure

(json-write obj [port-or-accumulator]) ⇒ unspecified
```

Library procedure returns vectors

… but we're passing the result to a procedure that expects multiple values

SRFI 210 Procedures and Syntax for Multiple Values

```
(apply/mv string #\a (values #\b #\c)) ⇒ "abc"

(call/mv string (values #\a #\b) (values #\c #\d)) ⇒ "abcd"

(case-receive (values 'a 'b)
  ((x) #f)
  ((x . y) (list x y))) ⇒ (a (b))

(vector-values #(a b c)) ⇒ a b c
```

Refactoring long series of operations to left-to-right

SRFI 197 Pipeline Operators

```
(chain <initial-value> [<placeholder> [<ellipsis>]] <step> ...)

(chain (a b) (c _ d) (e f _)) ⇒ (let* ((x (a b)) (x (c x d))) (e f x))

(chain (a) (b _ _) (c _)) ⇒ (let*-values (((x1 x2) (a)) ((x) (b x1 x2))) (c
        x))
```

That was just ten SRFIs.

There are 236, and more are coming.

Finis

- We don't have as many libraries and language extensions as Go, Java, or Python, but we're getting there.
  - *And don't forget Akku.scm and Snow.*
- Widely used.
- Pleasant community.
- **Please use them.**
- **Please implement them.**
- **Please help make more.**
  - *Propose them.*
  - *Discuss them.*
  - *Revive them (e.g. SRFI 204 Wright-Cartwright-Shinn Pattern Matcher).*

Q&A

https://srfi.schemers.org/scheme-workshop-2022.pdf