Title

Another format procedure, Fox

Author

Joo ChurlSoo

Status

This SRFI is currently in *withdrawn* status. Here is an explanation of each status that a SRFI can hold. To provide input on this SRFI, please send email to srfi-183@srfi.schemers.org. To subscribe to the list, follow these instructions. You can access previous messages via the mailing list archive.

- Received: 2020-02-21
- Draft #1 published: 2020-02-22
- Withdrawn: 2020-03-27

Abstract

This SRFI introduces the formatting procedure Fox ("format of X"), which takes one required argument and a variable number of additional arguments and returns a formatted string.

Changes from SRFI 54

This SRFI is based on SRFI 54. The optional arguments of the original CAT procedure were divided into three groups: arguments only for the number type of <object>, arguments for all types except the number type of <object>, and arguments for all types of <object>. This complexity could confuse users.

The optional arguments of this revised SRFI are divided into two groups: arguments only for the number type of <object> and arguments for all types of <object>. This simplicity also makes <writer> able to substitute for <converter>. The <precision> actually serves as ~G of Common Lisp's FORMAT and %G of C's PRINTF. The <point> of this revised one, an additional optional argument, serves as ~F or ~E of Common Lisp's FORMAT and %F or %E of C's PRINTF. This revised CAT procedure, Fox, is for speedup of non-numeric object types. For this purpose, the optional arguments only for the number type of object integrate into a list-type argument, <list-for-number>.

Additional extensions:

1. All optional arguments can be applied to the number type of <object>.
2. The default value of <writer> is the DISPLAY procedure.

3. The <width> is an integer, and the <precision> is a non-negative exact integer.
4. The new optional arguments <point> ('fixed or 'floating), <pre-string>, and <post-string> are added.
5. The <converter> is integrated into the <writer>.
6. The <take> and <pipe> arguments are integrated into a new argument, <converter>.
7. The <separator> is changed from a list to a vector whose elements are a string and a non-zero exact integer.
8. The infinities and NaNs of R6RS are supported.

Rationale

It is difficult to gain a complete consensus on the design of a generic formatting procedure that performs a variety of the functions provided by C's PRINTF and Common Lisp's FORMAT.

One way is to devise a free non-sequent method that easily handles optional arguments, in contrast to the conventional fixed sequent method, in order to obtain a handy optional and functional interface.

Specification

```
(fox <object>
    [<pre-string>]                              ;string
    [<port>]                                    ;port or boolean
    [<width>]                                   ;integer
    [<char>]                                    ;char
    [<writer>]                                  ;procedure
    [<list-for-number>]                         ;list
    [<converter>]                               ;pair
    [<separator>]                               ;vector
    [<post-string>] ...)                        ;string
```

The <list-for-number> is a list whose elements are <precision>, <point>, <radix>, <sign>, and <exactness>. They are effective only for the number type of <object>.

Except for <string>s, the order of all other optional arguments does not matter. When there is a <string> or <string>s without the other optional arguments, the <string> or <string>s are <post-string>.

1. The <object> is any Scheme object.

2. The <width> is an integer whose absolute value specifies the width of the resulting string. When the resulting string has fewer characters than the absolute value of <width>, it is placed rightmost with the rest being padded with <char>s, if <width> is positive, or it is placed leftmost with the rest being padded with <char>s, if <width> is negative, or it is placed in the center (near to right in case of positive <width>, or near to left in case of negative <width>) with the rest being padded with <char>s, if <width> is an inexact integer. On the

other hand, when the resulting string has more characters than the absolute value of <width>, the <width> is ignored. The default value is 0.

3. The <writer> is a procedure of two arguments, <object> and a string port. It writes <object> to the string port. The default value is the DISPLAY procedure. If you want any objects to be displayed in your own way, you have to define your own <writer>. Otherwise, they are displayed simply in their evaluated forms. When a <writer> other than the DISPLAY and WRITE procedures is used, the optional arguments that are effective only for the number type of <object> become ineffective.
4. The <port> is an output port or a boolean. If an output port is specified, the resulting string is output to the port. If <port> is #t, the output port is the current output port. If <port> is #f, the resulting string is returned. The default value is #f.
5. The <char> is a padding character. The default value is #\space.
6. The <converter> is a pair whose car and cdr values are exact integers or strings or procedures; m and n, and the absolute values of m and n are M and N, respectively. First, when the car element is an exact integer, the resulting string takes from the left m-characters, if it is positive, or all the characters but M-characters, if non-positive. When the car element is a string, the string is prefixed. When the car element is a procedure, the procedure takes a string argument and returns a string as a pipe. Second, when the cdr element is an exact integer, the resulting string takes from the right n-characters of the string that is processed by the car element, if it is positive, or all the characters but N-characters, if non-positive. When the cdr element is a string, the string is postfixed to the string that is processed by the car element. When the cdr element is a procedure, the procedure takes the string processed by the car element as an argument and returns a string.
7. The <separator> is a vector whose elements are a string serving as a separator, and a non-zero exact integer; n, and its absolute value is N. The resulting string is separated in every N-characters of the resulting string from right end, if n is positive, or from left end, if n is negative. Even if n is a negative integer, its absolute value is used for the number type of <object>. When the number of elements is one, the <separator> is effective only for the number type of <object>. The default values of the elements are "," and 3 respectively.
8. The <point> is a symbol: fixed or floating. Each returns a string of decimal fraction or exponential representation.
9. The <precision> is a non-negative exact integer that specifies the number of decimal digits after a decimal point.
10. The <radix> is a symbol: binary, octal, decimal, or hexadecimal. Each radix sign except decimal is prefixed to the resulting string. The default value is decimal.
11. If <sign> is a symbol that takes the form of 'sign, and <object> is a positive number without a positive sign, the positive sign is prefixed to the resulting string.
12. The <exactness> is a symbol: exact or inexact. Each returns a string of exact or inexact representation.

13. The resulting string is prefixed with <pre-string> and postfixed with <post-string>s.

Examples

```
(fox 129.995 -10 '(1))                  -> "130.0     "
(fox 129.995 10 '(1))                   -> "     130.0"
(fox 129.995 -10. #\* '(1))             -> "**130.0***"
(fox 129.995 10. #\* '(1))              -> "***130.0**"
(fox 129.995 10. #\* '(2))              -> "**130.00**"

(fox 4048 10 #\* '(hexadecimal))        -> "*****#xfd0"
(fox 4048 10 #\* '(hexadecimal sign))   -> "****#x+fd0"
(fox 4048 10 #\0 '(hexadecimal sign))   -> "#x+0000fd0"
(fox 4048 10 #\0 '(hexadecimal sign) `(,string-upcase . 0))      -> "#X+0000FD0"
(fox 4048 10 #\B '(hexadecimal sign) `(0 . ,string-upcase))      -> "#X+BBBBFD0"
(fox 4048 10 #\Z '(hexadecimal sign) `(,string-upcase . 0))      -> "ZZZZ#X+FD0"

(fox 129.995 10 '(2 sign) '("$" . 0))   -> "  $+130.00"
(fox 129.995 10 '(2 sign) '("$" . -3))  -> "     $+130"

(fox 123000000 '(floating))             -> "1.23e+8"
(fox 123000000 '(5 floating))           -> "1.23000e+8"
(fox 1.23456789e+20 '(fixed))           -> "123456789000000000000.0"

(fox 123456789.012 '(sign) '#(","))     -> "+123,456,789.012"
(fox 123456789 '(sign) '#("," 4))       -> "+1,2345,6789"
(fox "abcdefg" '(sign) '#(","))         -> "abcdefg"
(fox "abcdefg" '(sign) '#("::" 2))      -> "a::bc::de::fg"
(fox "abcdefg" '(sign) '#("::" -2))     -> "ab::cd::ef::g"

(fox '(#\a "str" s))                    -> "(a str s)"
(fox '(#\a "str" s) write)              -> "(#\\a \"str\" s)"

(fox 129.995 10. #\* '(0) '("|" . "|")) ->  "**|130.|**"
(fox 129.995 "|" 10. #\* '(0) "|")      -> "|***130.***|"

(fox 'String "^" (current-output-port) 10 #\* "$")      ->  ^****String$
(fox 'String "^" #t 10 #\* "$")                         ->  ^****String$
(fox 'String "^" #f 10 #\* "$")                         -> "^****String$"
(fox 'String "^" 10 #\* "$")                            -> "^****String$"

(define-record-type example
  (make-example num str)
  example?
  (num get-num set-num!)
  (str get-str set-str!))

(define (record-writer object string-port)
  (if (example? object)
      (begin (display (get-num object) string-port)
             (display "-" string-port)
```

```
              (display (get-str object) string-port))
        (display object string-port)))

(define (record-display object string-port)
  (display (get-num object) string-port)
  (display "-" string-port)
  (display (get-str object) string-port))

(let ((plus 12345678.901)
      (minus -123456.789)
      (ex (make-example 1234 "ex"))
      (file "today.txt"))
  (for-each
   (lambda (x y)
     (fox x #t 10 (fox y 15 '(2) record-writer '#(",")))
     (fox x #t 10)
     (fox y #t 15 '(2) (if (example? y) record-display display) '#(","))
     (newline))
   (list "plus: " "minus: " "net: " "ex: " "file: ")
   (list plus minus (+ plus minus) ex file)))
->
    plus:  12,345,678.901    plus:   12,345,678.90
   minus:    -123,456.789   minus:     -123,456.79
     net:  12,222,222.112     net:   12,222,222.11
      ex:         1234-ex      ex:         1234-ex
    file:       today.txt    file:       today.txt
```

Sample Implementation

The sample implementation available both in the Github repo and in this `.tgz` file.

References

[R5RS] Richard Kelsey, William Clinger, and Jonathan Rees: Revised(5) Report on the Algorithmic Language Scheme.
http://www.schemers.org/Documents/Standards/R5Rs/

[R6RS] Michael Sperber, R. Kent Dybvig, Matthew Flatt, and Anton von Straaten: Revised(6) Report on the Algorithmic Language Scheme.
http://www.r6rs.org

[SRFI 54] Joo ChurlSoo: Formatting.
http://srfi.schemers.org/srfi-54/

Copyright

Editor: Arthur A. Gleckler