

foreign-c a portable foreign function interface for R7RS Schemes

foreign-c

foreign-c is a C foreign function interface (FFI) library for R7RS Schemes. It is portable in the sense that it supports multiple implementations, as opposed to being portable by conforming to some specification.

[Issue tracker](#)

[Mailing lists](#)

[Jenkins](#)

- [Installation](#)
- [Documentation](#)
 - [Types](#)
 - [Primitives 1](#)
 - c-type-size
 - define-c-library
 - define-c-procedure
 - c-bytevector?
 - c-bytevector-u8-set!
 - c-bytevector-u8-ref
 - c-bytevector-pointer-set!
 - c-bytevector-pointer-ref
 - [Primitives 2](#)
 - define-c-callback
 - [c-bytevector](#)
 - make-c-null
 - c-null?
 - c-free
 - make-c-bytevector
 - call-with-address-of
 - native-endianness
 - c-bytevector-s8-set!
 - c-bytevector-s8-ref
 - c-bytevector-s16-set!
 - c-bytevector-s16-ref
 - c-bytevector-s16-native-set!
 - c-bytevector-s16-native-ref
 - c-bytevector-u16-set!
 - c-bytevector-u16-ref
 - c-bytevector-u16-native-set!
 - c-bytevector-u16-native-ref

- c-bytevector-s32-set!
- c-bytevector-s32-ref
- c-bytevector-s32-native-set!
- c-bytevector-s32-native-ref
- c-bytevector-u32-set!
- c-bytevector-u32-ref
- c-bytevector-u32-native-set!
- c-bytevector-u32-native-ref
- c-bytevector-s64-set!
- c-bytevector-s64-ref
- c-bytevector-s64-native-set!
- c-bytevector-s64-native-ref
- c-bytevector-u64-set!
- c-bytevector-u64-ref
- c-bytevector-u64-native-set!
- c-bytevector-u64-native-ref
- c-bytevector-sint-set!
- c-bytevector-sint-ref
- c-bytevector-uint-set!
- c-bytevector-uint-ref
- c-bytevector-ieee-single-set!
- c-bytevector-ieee-single-native-set!
- c-bytevector-ieee-single-ref
- c-bytevector-ieee-single-native-ref
- c-bytevector-ieee-double-set!
- c-bytevector-ieee-double-native-set!
- c-bytevector-ieee-double-ref
- c-bytevector-ieee-double-native-ref
- bytevector->c-bytevector
- c-bytevector->bytevector
- string->c-utf8
- c-utf8->string
- [Environment variables](#)

Implementation support tables

Required versions:

- Chibi > 0.11
 - At the only 0.11 is out so build from git
- Chicken >= 5.4.0 < 6
- Gauche >= 0.9.15
 - Does not yet work with snow-chibi install
- Guile >= 3
 - Does not yet work with snow-chibi install
 - Has include bug, might not work on all situations
- Kawa >= 3.11 and Java >= 22
 - Needs arguments to enable FFI
 - -J-add-exports=java.base/jdk.internal.foreign.abi=ALL-UNNAMED

- -J-add-exports=java.base/jdk.internal.foreign.layout=ALL-UNNAMED
 - -J-add-exports=java.base/jdk.internal.foreign=ALL-UNNAMED
 - -J-enable-native-access=ALL-UNNAMED
 - -J-enable-preview
- All needed arguments on one line for copy pasting
 - -J-add-exports=java.base/jdk.internal.foreign.abi=ALL-UNNAMED -J-add-exports=java.base/jdk.internal.foreign.layout=ALL-UNNAMED -J-add-exports=java.base/jdk.internal.foreign=ALL-UNNAMED -J-enable-native-access=ALL-UNNAMED -J-enable-preview
- So that snow-chibi installed library is found
 - -Dkawa.import.path=/usr/local/share/kawa
 - -Dkawa.import.path=/usr/local/share/kawa/lib
- Mosh >= 0.2.9-rc1
- Racket >= 8.16 [cs]
- Sagittarius >= 0.9.13
- STklos > 2.10
 - At the time only 2.10 is out so build from git

Primitives 1 table

| | c-type-size | c-bytevector-u8-set! | c-byt |
|--------------------|--------------------|-----------------------------|--------------|
| Chibi | X | X | |
| Chicken | X | X | |
| Gauche | X | X | |
| Guile | X | X | |
| Kawa | X | X | |
| Mosh | X | X | |
| Racket | X | X | |
| Sagittarius | X | X | |
| STklos | X | X | |
| Ypsilon | X | X | |

Primitives 2 table

Chibi
Chicken
 Gauche
Guile
 Kawa
Mosh
Racket
Saggittarius

STklos
Ypsilon

Test files pass

| | primitives.scm |
|---------------------|-----------------------|
| Chibi | X |
| Chicken | X |
| Gauche | X |
| Guile | X |
| Kawa | X |
| Mosh | X |
| Racket | X |
| Saggittarius | X |
| STklos | X |
| Ypsilon | X |

Installation

Snow-fort

<https://snow-fort.org/>

snow-chibi -impls=IMPLEMENTATION install "(foreign c)"

You can test that library is found by your implementation like this:

```
cp tests/hello.scm /tmp/hello.scm
cd /tmp
IMPLEMENTATION hello.scm
```

Manual

Either download the latest release from <https://git.sr.ht/~retropikzel/foreign-c/refs> or git clone, tag, and copy the foreign directory to your library directory.

Example assuming libraries in directory snow:

```
git clone https://git.sr.ht/~retropikzel/foreign-c --branch
LATEST_VERSION
cd foreign-c
make SCHEME_IMPLEMENTATION_NAME
cd ..
mkdir -p snow
cp -r foreign-c/foreign snow/
```

With most implementations the make command does not compile anything. When that is the case it will say “Nothing to build on SCHEME_IMPLEMENTATION_NAME.”

Documentation

Types

Types are given as symbols, for example 'int8 or 'pointer.

- int8
- uint8
- int16
- uint16
- int32
- uint32
- int64
- uint64
- char
- unsigned-char
- short
- unsigned-short
- int
- unsigned-int
- long
- unsigned-long
- float
- double
- pointer
 - c-bytevector on Scheme side
- callback
 - Callback function
- void
 - Can not be argument type, only return type

Primitives 1

(**c-type-size** type)

Returns the size of given C type.

(**define-c-library** scheme-name headers object-name options)

Takes a scheme-name to bind the library to, list of C headers as strings, shared-object name and options.

The C header strings should not contain “<” or “>”, they are added automatically.

The name of the shared object should not contain suffix like .so or .dll. Nor should it contain any prefix like “lib”.

Options:

- additional-versions
 - Search for additional versions of shared object, given shared object "c" and additional versions "6" "7" on linux the files "libc", "libc.6", "libc.7" are searched for.
 - Can be either numbers or strings
- additional-paths
 - Give additional paths to search shared objects from

Example:

```
(define-c-library libc
  (list "stdlib.h")
  libc-name
  '((additional-versions (" " "0" "6"))
    (additional-paths ("."))))
```

Notes

- Do not cond-expand inside the arguments, that might lead to problems on some implementations.
- Do not store options in variables, that might lead to problems on some implementations.
- Pass the headers using quote
 - As '(...) and not (list...)
- Pass the options using quote
 - As '(...) and not (list...)

(define-c-procedure scheme-name shared-object c-name return-type argument-type)

Takes a scheme-name to bind the C procedure to, shared-object where the function is looked from, c-name of the function as symbol, return-type and argument-types.

Defines a new foreign function to be used from Scheme code.

Example:

```
(define-c-library libc '("stdlib.h") libc-name '("6"))
(define-c-procedure c-puts libc 'puts 'int '(pointer))
(c-puts "Message brought to you by foreign-c!")
```

Notes

- Pass the return-types using quote
 - As '(...) and not (list...)

(c-bytevector? obj)

Returns **#t** if obj is c-bytevector, otherwise returns **#f**.

(c-bytevector-u8-set! c-bytevector k byte)

If K is not a valid index of c-bytevector the behaviour is undefined.

Stores the byte in element k of c-bytevector.

(c-bytevector-u8-ref c-bytevector k)

If K is not a valid index of c-bytevector the behaviour is undefined.

Returns the byte at index k of c-bytevector.

(c-bytevector-pointer-set! c-bytevector k pointer)

If K is not a valid index of c-bytevector the behaviour is undefined.

Stores the pointer(which is also c-bytevector) in element k of c-bytevector.

(c-bytevector-pointer-ref c-bytevector k pointer)

If K is not a valid index of c-bytevector the behaviour is undefined.

Returns the pointer(which is also c-bytevector) at index k of c-bytevector.

Primitives 2

(define-c-callback scheme-name return-type argument-types procedure)

Takes scheme-name to bind the Scheme procedure to, return-type, argument-types and procedure as in place lambda.

Defines a new Sceme function to be used as callback to C code.

Example:

```
; Load the shared library
(define-c-library libc-stdlib ("stdlib.h") libc-name '("") "6")

; Define C function that takes a callback
(define-c-procedure qsort libc-stdlib 'qsort 'void '(pointer int
int callback))

; Define our callback
(define-c-callback compare
  'int
  '(pointer pointer)
  (lambda (pointer-a pointer-b)
    (let ((a (c-bytevector-sint-get pointer-a
(native-endianness) 0))
          (b (c-bytevector-sint-get pointer-b
(native-endianness) 0)))
      (cond ((> a b) 1)
            ((= a b) 0)
            (< a b) -1))))))
```

```

; Create new array of ints to be sorted
(define array (make-c-bytevector (* (c-type-size 'int) 3)))
(c-bytevector-s32-native-set! array (* (c-type-size 'int) 0) 3)
(c-bytevector-s32-native-set! array (* (c-type-size 'int) 1) 2)
(c-bytevector-s32-native-set! array (* (c-type-size 'int) 2) 1)

(display array)
(newline)
;> (3 2 1)

; Sort the array
(qsort array 3 (c-type-size 'int) compare)

(display array)
(newline)
;> (1 2 3)

```

c-bytevector

Foreign-c c-bytevector interface is copied from R6RS bytevectors, with some added functionality for C null pointers and manual memory management.

(make-c-null)

Returns a null C pointer.

(c-null? obj)

Returns **#t** if obj is a null C pointer, otherwise returns **#f**.

(c-free c-bytevector)

Frees c-bytevector from memory.

(call-with-address-of c-bytevector thunk)

Calls thunk with address pointer of c-bytevector.

Since the support for calling C functions taking pointer address arguments, ones prefixrd with & in C, varies, some additional ceremony is needed on the Scheme side.

Example:

Calling from C:

```

//void func(int** i);
func(&i);

```

Calling from Scheme:


```
(define cbv (make-bytevector (c-type-size 'int)))
(call-with-address-of
 cbv
 (lambda (address)
  (func address)))
; Use cbv here
```

The passed c-bytevector, in example named cbv, should only be used **after** call to call-with-address-of ends.

(bytevector->c-bytevector bytevector)

Returns a newly allocated c-bytevector of the bytes of bytevector.

(c-bytevector->bytevector)

Returns a newly allocated bytevector of the bytes of c-bytevector.

(native-endianness)

Returns the endianness symbol associated implementation's preferred endianness (usually that of the underlying machine architecture). This may be any <endianness symbol>, including a symbol other than big and little.

(make-c-bytevector k)

(make-c-bytevector k fill)

Returns a newly allocated c-bytevector of k bytes.

If the fill argument is missing, the initial contents of the returned c-bytevector are unspecified.

If the fill argument is present, it's value must confine to C uint8_t values , it specifies the initial value for the bytes of the c-bytevector

(c-bytevector-s8-set! c-bytevector k byte)

If k is not a valid index of c-bytevector the behaviour is undefined.

Stores the byte in element k of c-bytevector.

(c-bytevector-s8-ref c-bytevector k)

If k is not a valid index of c-bytevector the behaviour is undefined.

Returns the byte at index k of c-bytevector.

(c-bytevector-char-set! c-bytevector k char)

If k is not a valid index of c-bytevector the behaviour is undefined.

Stores the char in element k of c-bytevector.

(c-bytevector-char-ref c-bytevector k)

If k is not a valid index of `c-bytevector` the behaviour is undefined.

Returns the char at index k of `c-bytevector`.

(c-bytevector-uchar-set! `c-bytevector` k `char`)

If k is not a valid index of `c-bytevector` the behaviour is undefined.

Stores the unsigned char in element k of `c-bytevector`.

(c-bytevector-uchar-ref `c-bytevector` k)

If k is not a valid index of `c-bytevector` the behaviour is undefined.

Returns the unsigned char at index k of `c-bytevector`.

(c-bytevector-uint-ref `c-bytevector` k `endianness` `size`)

(c-bytevector-sint-ref `c-bytevector` k `endianness` `size`)

(c-bytevector-uint-set! `c-bytevector` k n `endianness` `size`)

(c-bytevector-sint-set! `c-bytevector` k n `endianness` `size`)

Size must be a positive exact integer object. If $k, \dots, k + \text{size} - 1$ is not valid indices of `c-bytevector` the behavior is unspecified.

The `c-bytevector-uint-ref` procedure retrieves the exact integer object corresponding to the unsigned representation of size `size` and specified by `endianness` at indices $k, \dots, k + \text{size} - 1$.

The `c-bytevector-sint-ref` procedure retrieves the exact integer object corresponding to the two's-complement representation of size `size` and specified by `endianness` at indices $k, \dots, k + \text{size} - 1$. For `c-bytevector-uint-set!`, n must be an exact integer object in the interval $\{0, \dots, 256^{\text{size}} - 1\}$.

The `c-bytevector-uint-set!` procedure stores the unsigned representation of size `size` and specified by `endianness` into `c-bytevector` at indices $k, \dots, k + \text{size} - 1$.

The `...-set!` procedures return unspecified values.

Examples:

```
(define cbv (make-c-bytevector (c-type-size 'int)))
(c-bytevector-sint-set! cbv 0 100 (native-endianness) (c-type-size 'int))
(c-bytevector-sint-ref cbv 0 (native-endianness) (c-type-size 'int))
> 100
```

(c-bytevector-u16-ref `c-bytevector` k `endianness`)

(c-bytevector-s16-ref `c-bytevector` k `endianness`)

(c-bytevector-u16-native-ref `c-bytevector` k)

(c-bytevector-s16-native-ref `c-bytevector` k)

(c-bytevector-u16-set! `c-bytevector` k n `endianness`)

(c-bytevector-s16-set! `c-bytevector` k n `endianness`)

(c-bytevector-u16-native-set! c-bytevector k n)
(c-bytevector-s16-native-set! c-bytevector k n)

K must be a valid index of c-bytevector ; so must $k + 1$. For c-bytevector-u16-set! and c-bytevector-u16-native-set!, n must be an exact integer object in the interval $\{0, \dots, 216 - 1\}$. For c-bytevector-s16-set! and c-bytevector-s16-native-set!, n must be an exact integer object in the interval $\{-215, \dots, 215 - 1\}$.

These retrieve and set two-byte representations of numbers at indices k and $k + 1$, according to the endianness specified by endianness. The procedures with u16 in their names deal with the unsigned representation; those with s16 in their names deal with the two's-complement representation.

The procedures with native in their names employ the native endianness, and work only at aligned indices: k must be a multiple of 2.

The ...-set! procedures return unspecified values.

(c-bytevector-u32-ref c-bytevector k endianness)
(c-bytevector-s32-ref c-bytevector k endianness)
(c-bytevector-u32-native-ref c-bytevector k)
(c-bytevector-s32-native-ref c-bytevector k)
(c-bytevector-u32-set! c-bytevector k n endianness)
(c-bytevector-s32-set! c-bytevector k n endianness)
(c-bytevector-u32-native-set! c-bytevector k n)
(c-bytevector-s32-native-set! c-bytevector k n)

K, ..., $k + 3$ must be valid indices of bytevector. For c-bytevector-u32-set! and bytevector-u32-native-set!, n must be an exact integer object in the interval $\{0, \dots, 232 - 1\}$. For bytevector-s32-set! and bytevector-s32-native-set!, n must be an exact integer object in the interval $\{-231, \dots, 232 - 1\}$.

These retrieve and set four-byte representations of numbers at indices k, ..., $k + 3$, according to the endianness specified by endianness. The procedures with u32 in their names deal with the unsigned representation; those with s32 with the two's-complement representation.

The procedures with native in their names employ the native endianness, and work only at aligned indices: k must be a multiple of 4.

The ...-set! procedures return unspecified values.

(c-bytevector-u64-ref c-bytevector k endianness)
(c-bytevector-s64-ref c-bytevector k endianness)
(c-bytevector-u64-native-ref c-bytevector k)
(c-bytevector-s64-native-ref c-bytevector k)
(c-bytevector-u64-set! c-bytevector k n endianness)
(c-bytevector-s64-set! c-bytevector k n endianness)
(c-bytevector-u64-native-set! c-bytevector k n)
(c-bytevector-s64-native-set! c-bytevector k n)

$K, \dots, k + 7$ must be valid indices of `c-bytevector`. For `c-bytevector-u64-set!` and `c-bytevector-u64-native-set!`, n must be an exact integer object in the interval $\{0, \dots, 264 - 1\}$. For `c-bytevector-s64-set!` and `c-bytevector-s64-native-set!`, n must be an exact integer object in the interval $\{-263, \dots, 264 - 1\}$.

These retrieve and set eight-byte representations of numbers at indices $k, \dots, k + 7$, according to the endianness specified by `endianness`. The procedures with `u64` in their names deal with the unsigned representation; those with `s64` with the two's-complement representation.

The procedures with `native` in their names employ the native endianness, and work only at aligned indices: k must be a multiple of 8.

The `...-set!` procedures return unspecified values.

(c-bytevector-ieee-single-native-ref)
(c-bytevector-ieee-single-ref)

$K, \dots, k + 3$ must be valid indices of `c-bytevector`. For `c-bytevector-ieee-single-native-ref`, k must be a multiple of 4.

These procedures return the inexact real number object that best represents the IEEE-754 single-precision number represented by the four bytes beginning at index k .

(c-bytevector-ieee-double-native-ref)
(c-bytevector-ieee-double-ref)

$K, \dots, k + 7$ must be valid indices of `c-bytevector`. For `c-bytevector-ieee-double-native-ref`, k must be a multiple of 8.

These procedures return the inexact real number object that best represents the IEEE-754 double-precision number represented by the eight bytes beginning at index k .

(c-bytevector-ieee-single-native-set!)
(c-bytevector-ieee-single-set!)

$K, \dots, k + 3$ must be valid indices of `c-bytevector`. For `c-bytevector-ieee-single-native-set!`, k must be a multiple of 4.

These procedures store an IEEE-754 single-precision representation of x into elements k through $k + 3$ of `bytevector`, and return unspecified values.

(c-bytevector-ieee-double-native-set!)
(c-bytevector-ieee-double-set!)

$K, \dots, k + 7$ must be valid indices of `bytevector`. For `c-bytevector-ieee-double-native-set!`, k must be a multiple of 8.

These procedures store an IEEE-754 double-precision representation of x into elements k through $k + 7$ of `bytevector`, and return unspecified values.

(string->c-utf8 string)

Returns a newly allocated (unless empty) c-bytevector that contains the UTF-8 encoding of the given string.

(c-utf8->string c-bytevector)

Returns a newly allocated (unless empty) string whose character sequence is encoded by the given c-bytevector.

Utilities

libc-name

Name of the C standard library on the current operating system. Supported OS:

- Windows
- Linux
- Haiku

See `foreign/c/libc.scm` to see which headers are included and what shared libraries are loaded.

Example:

```
(define-c-library libc ("stdlib.h") libc-name '("") "6")
(define-c-procedure c-puts libc 'puts 'int '(pointer))
(c-puts "Message brought to you by foreign-c!")
```

Environment variables

Setting environment variables like this on Windows works for this library:

```
set "FOREIGN_C_LOAD_PATH=C:\Program Files (x86)/foo/bar"
```

FOREIGN_C_LOAD_PATH

To add more paths to where foreign c looks for libraries set `FOREIGN_C_LOAD_PATH` to paths separated by `;` on windows, and `:` on other operating systems.