Portable Foreign Function Interface for R7RS Documentation

Portable Foreign Function Interface for R7RS

Portable foreign function interface for R7RS. It is portable in the sense that it supports multiple implementations, as opposed to being portable by conforming to some specification.

Project

Issue trackers

Maling lists

Jenkins

Table of contents

Goals

- Support only R7RS implementations
- Same interface on all implementations
  - Some things that are procedures on one implementation are macros on other, but they must behave the same
- Stability and being boring after 1.0.0 is reached

Non goals

- Compiling of used library C code at any point
  - That is no stubs, no C code generated by the library and so on
  - The pffi library itself may require compilation on installation

Status

In alpha.

Current caveats

- No way to pass structs by value
- Most implementations are missing callback support
- Always pass arguments to pffi functions/macros as ’(1 2 3) and not (list 1 2 3)
- Always pass pffi-define-callback procedure as lambda in place
- No support for variadic function arguments
    - Can be partially worked around by defining multiple versions of same function with different number of arguments

Roadmap

For roadmap to 1.0.0 see issues

Feature mplementation table

Primitives

	c-size-of	**define-c-library**	**c-bytevector?**
Chibi	X	X	X
Chicken	X	X	X
Cyclone	X	X	X
Gambit	X		
Gauche	X	X	X
Gerbil			
Guile	X	X	X
Kawa	X	X	X
Larceny			
Mosh	X	X	X
Racket	X	X	X
Saggittarius	X	X	X
Skint			
Stklos	X	X	X
tr7			
Ypsilon	X	X	X

Built upon

These features are built upon the primitives and if primitives are implemented and work, they should work too.

- make-c-bytevector
- make-c-null
- c-null?
- pffi-pointer-address
- c-free
- pffi-pointer->string
- pffi-string->pointer
- pffi-struct-make
- pffi-struct-pointer
- pffi-struct-offset-get
- pffi-struct-get
- pffi-struct-set!
- pffi-array-allocate
- pffi-array?
- pffi-pointer->array
- pffi-array-get
- pffi-array-set!
- pffi-list->array
- pffi-array->list

Not started

- LIPS
  - Will work on nodejs by using some C FFI library from npm
  - Javascript side needs design
- Biwascheme
  - Will work on nodejs by using some C FFI library from npm
  - Javascript side needs design
- MIT-Scheme
  - Need to study the implementation more
- Airship
  - Need to study the implementation more
- Other gambit targets
  - Gambit compiles to different targets other than C too, for example Javascript. It would be cool and interesting to see if this FFI could also support some of those
  - When LIPS and Biwascheme Javascript side is done then Gambit should be done too
- s48-r7rs
  - Need to study the implementation more
- prescheme
  - Need to study the implementation more

Other

- s7
  - Propably does not need FFI as it is embeddable only

- Loko
    - Desires no C interop, I can respect that

Documentation

Dependencies

Some implementations have extra dependencies/requirements beyond just the library.

Chibi

Building depends on libffi.

Debian/Ubuntu/Mint install with:

```
apt install libffi-dev
```

Chicken

Chicken needs r7rs egg installed. Install it with:

```
chicken-install r7rs
```

Gauche

Building depends on libffi.

Debian/Ubuntu/Mint install with:

```
apt install libffi-dev
```

Racket

Needs racket-r7rs, install with:

```
raco pkg install --auto r7rs
```

Kawa

Kawa Needs at least Java version 22 these flags before any other arguments:

- -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

If you are running kawa.jar with plain java then give same arguments to java without the -J prefix.

Installation

Since the project is under active development is best to clone it from git,

Project local

Linux

Assuming you have a project and your libraries live in directory called snow in it:

```
git clone https://git.sr.ht/~retropikzel/r7rs-pffi
mkdir -p snow
cp -r r7rs-pffi/retropikzel snow/
cd snow/retropikzel/pffi
make <SCHEME>
```

Windows

There is no build scripts yet for Windows, that said many implementations work without compiling anything. If you run this and it says “There is notching to build for SCHEME” then you should be good to go.

System global

Still work in progress.

Reference

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
- callback
    - Callback function

Types

Environment variables

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

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

PFFI_LOAD_PATH

To add more paths to where pffi looks for libraries set PFFI_LOAD_PATH to paths separated by ; on windows, and : on other operating systems.

Procedures and macros

Some of these are procedures and some macros, it might also change implementation to implementation.

pffi-init

pffi-init

Always call this first, on most implementation it does nothing but some implementations might need initialisation run.

c-size-of

c-size-of object -> number

Returns the size of the pffi-struct, pffi-enum or pffi-type.

pffi-align-of

pffi-align-of type -> number

Returns the align of the type.

define-c-library

define-c-library headers shared-object-name [options] -> object

Load given shared object automatically searching many predefined paths.

Takes as argument a list of C headers, these are for the compiler ones. And an shared-object name, used by the dynamic FFI’s. The name of the shared object should not contain suffix like .so or .dll. Nor should it contain any prefix like “lib”.

Additional options argument can be provided, theys should be a pair with a keyword. The options are:

- 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 for

Example:

```
(cond-expand
  (windows (define-c-library libc-stdlib
                                '("stdlib.h")
                                "ucrtbase"
                                '((additional-versions ("0" "6"))
                                  (additiona-paths (".")))))
  (else (define-c-library libc-stdlib
                             (list "stdlib.h")
                             "c"
                             '((additional-versions ("0" "6"))
                               (additiona-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.
- Do pass the headers using quote
    - As '(... and not (list...
- Do pass the options using quote
    - As '(... and not (list...

make-c-null

make-c-null -> pointer

Returns a new NULL pointer.

c-null?

c-null? pointer -> boolean

Returns #t if given pointer is null pointer, #f otherwise.

make-c-bytevector

make-c-bytevector size -> pointer

Returns newly allocated pointer of given size.

pffi-pointer-address

pffi-pointer-address pointer -> pointer

Returns the address of given pointer inside a pointer. This is used when passing pointers to pointers to foreign procedures. This is similar to the c's &. One **important difference** is that after you have passed a pointer to the procedure you must get value from it back to the pointer which address you are passing. Example:

```
(define input-pointer (make-c-bytevector <needed size>))
(define input-pointer-address (pffi-pointer-address input-
pointer))
(<foreign-procedure-that takes &pointer as argument> input-
pointer-address)
(set! input-pointer (pffi-pointer-get input-pointer-address
'pointer 0))
```

c-bytevector?

c-bytevector? object -> boolean

Returns #t if given object is pointer, #f otherwise.

c-free

c-free pointer

Frees given pointer.

pffi-pointer-set!

pffi-pointer-set! pointer type offset value

Sets the value on a pointer on given offset. For example:

```
(define p (make-c-bytevector 128))
(pffi-pointer-set! p 'int 64 100)
```

Would set the offset of 64, on pointer p to value 100.

pffi-pointer-get

pffi-pointer-get pointer type offset -> object

Gets the value from a pointer on given offset. For example:

```
(define p (make-c-bytevector 128))
(pffi-pointer-set! p 'int 64 100)
(pffi-pointer-get p 'int 64)
> 100
```

string->c-bytevector

string->c-bytevector string -> pointer

Makes pointer out of a given string.

c-bytevector->string

c-bytevector->sring pointer -> string

Makes string out of a given pointer.

pffi-struct-make

pffi-struct-make c-type members . pointer -> pffi-struct

Creates a new pffi-struct and allocates pointer for it. The members argument is a list of member names and types. For example:

```
(define color (pffi-struct-make 'color '((int8 . r) (int8 . g)
(int8 . b) (int8 .a ))))
(define test (pffi-struct-make "struct test" '((int8 . r) (int8 .
g) (int8 . b) (int8 .a ))))
```

C-type argument can be symbol or a string.

pffi-struct-pointer

pffi-struct-pointer pffi-struct -> pointer

Returns the pointer that holds the struct content. You need to use this when passing a struct as a pointer to foreign functions.

```
(define s (pffi-struct-make 'test '((int . r) (int . g) (int .
b))))
(pffi-struct-pointer s)
```

pffi-struct-offset-get

pffi-struct-offset-get member-name -> number

Returns the offset of a struct member with given name.

pffi-struct-get

pffi-struct-get pffi-struct member-name -> object

Returns the value of the givens struct member.

pffi-struct-set!

pffi-struct-set! pffi-struct member-name value

Sets the value of the givens struct member. It is up to you to make sure that the type of value is correct.

pffi-array-allocate

pffi-array-allocate type size

Allocates pointer array of given type and size.

pffi-array-pointer

pffi-array-pointer array

Returns the pointer of the array.

pffi-array?

pffi-array? object

Returns #t of given object is array, #f otherwise.

pffi-pointer->array

pffi-pointer->array pointer type size

Converts given pointer to an array of giben type and size.

pffi-array-get

pffi-array-get array index

Returns the value of given index from given array.

pffi-array-set!

pffi-array-set! array index value

Sets the given value of given index in given array.

pffi-list->array

pffi-list->array type list

Converts given list into C array of given type.

pffi-array->list

pffi-array->list type list length

Converts given C array into list of given type and length.

define-c-procedure

define-c-procedure scheme-name shared-object c-name return-type argument-types

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

```
(cond-expand
    (windows (define-c-library libc-stdlib '("stdlib.h")
"ucrtbase" '("")))
    (else (define-c-library libc-stdlib '("stdlib.h")  "c" '(""
"6"))))
(define-c-procedure c-puts libc-stdlib 'puts 'int '(pointer))
(c-puts "Message brought to you by FFI!")
```

pffi-define-callback

pffi-define-callback scheme-name return-type argument-types procedure

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

```
; Load the shared library
(cond-expand
    (windows (define-c-library libc-stdlib '("stdlib.h")
"ucrtbase" '()))
    (else (define-c-library '("stdlib.h") "c" '("" "6"))))

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

; Define our callback
(pffi-define-callback compare
                      'int
                      '(pointer pointer)
                      (lambda (pointer-a pointer-b)
                        (let ((a (pffi-pointer-get pointer-a 'int
0))
                              (b (pffi-pointer-get pointer-b 'int
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-size-of 'int) 3)))
(pffi-pointer-set! array 'int (* (c-size-of 'int) 0) 3)
(pffi-pointer-set! array 'int (* (c-size-of 'int) 1) 2)
(pffi-pointer-set! array 'int (* (c-size-of 'int) 2) 1)

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

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

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