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

- To have every possible FFI feature
- 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

Currently the interface of the library is in okay shape. It propably will not change much but no guarantees are being made just yet.

Due to supporting many different Scheme implementations, different parts of this software are in different stage. As a whole it is still in **alpha** stage. That said the interface should not be changing anymore and some implementations are in **beta**.

Current caveats

- No way to pass structs by value
- Most implementations are missing callback support

Implementation table

Beta

	pffi-init	pffi-size-of	pffi-shared-object-auto-load	pffi-shared-object-load	pffi-pointer-null	pffi-pointer-null?	pffi-pointer-allocate	pffi-pointer-address	pffi-pointer?
Chibi	X	X	X	X	X	X	X	X	X
Gauche	X	X	X	X	X	X	X		X
Guile	X	X	X	X	X	X	X		X
Kawa	X	X	X	X	X	X	X		X
Racket	X	X	X	X	X	X	X		X
Saggittarius	X	X	X	X	X	X	X		X

Alpha

	pffi-init	pffi-size-of	pffi-shared-object-auto-load	pffi-shared-object-load	pffi-pointer-null	pffi-pointer-null?	pffi-pointer-allocate	pffi-pointer-address	pffi-pointer?	p
Chicken-5	X	X	X	X	X	X	X		X	
Cyclone	X	X	X	X	X	X	X		X	
Gambit	X	X						X		
Gerbil	X									
Larceny	X									
Mosh	X	X	X	X	X	X	X		X	
Skint	X									
Stklos	X	X	X	X	X	X	X		X	
tr7										
Ypsilon										

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

Usage

Chibi

Needs libffi-dev, on Debina/Ubuntu/Mint install with:

```
apt install libffi-dev
```

Build with:

```
chibi-ffi retropikzel/r7rs-pffi/r7rs-pffi-chibi.stub
gcc -o retropikzel/r7rs-pffi/r7rs-pffi-chibi.so -fPIC -shared
retropikzel/r7rs-pffi/r7rs-pffi-chibi.c -lchibi-scheme -lffi
```

Chicken

Needs r7rs egg, install with:

```
chicken-install r7rs
```

Racket

Needs racket-r7rs, install with:

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

Kawa

Kawa Needs at least Java version 22

Needs jvm flags:

- –add-exports java.base/jdk.internal.foreign.abi=ALL-UNNAMED
- –add-exports java.base/jdk.internal.foreign.layout=ALL-UNNAMED
- –add-exports java.base/jdk.internal.foreign=ALL-UNNAMED
- –enable-native-access=ALL-UNNAMED

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

Procedures and macros

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

pffi-init

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

pffi-size-of object -> number

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

pffi-align-of type -> number

Returns the align of the type.

pffi-shared-object-auto-load 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:

```
(define libc-stdlib
  (cond-expand
    (windows (pffi-shared-object-auto-load (list "stdlib.h")
"ucrtbase"))
    (else (pffi-shared-object-auto-load (list "stdlib.h")
                                        "c"
                                        '(additional-versions .
("6"))
```

```
                                          '(additional-search-
paths . (".")))))))
```

pffi-shared-object-load headers path [options]

It is recommended to use the pffi-shared-object-auto-load instead of this directly.

Headers is a list of strings needed to be included, for example

```
(list "curl/curl.h")
```

Path is the full path of the shared object without any “lib” prefix or “.so/.dll” suffix. For example:

```
"curl"
```

Options:

- additional-versions
  - List of different versions of library to try, for example (list “.0” “.1”)

pffi-pointer-null -> pointer

Returns a new NULL pointer.

pffi-pointer-null? pointer -> boolean

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

pffi-pointer-allocate size -> pointer

Returns newly allocated pointer of given size.

pffi-pointer-address pointer -> number

Returns the address of given pointer as number.

pffi-pointer? object -> boolean

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

pffi-pointer-free pointer

Frees given pointer.

pffi-pointer-set! pointer type offset value

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

```
(define p (pffi-pointer-allocate 128))
(pffi-pointer-set! p 'int 64 100)
```

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

pffi-pointer-get pointer type offset -> object

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

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

pffi-string->pointer string -> pointer

Makes pointer out of a given string.

pffi-pointer->string pointer -> string

Makes string out of a given pointer.

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

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 member-name -> number

Returns the offset of a struct member with given name.

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

Returns the value of the givens struct member.

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

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

```
(define libc-stdlib
    (cond-expand
        (windows (pffi-shared-object-auto-load (list "stdlib.h")
(list) "ucrtbase" (list "")))
        (else (pffi-shared-object-auto-load (list "stdlib.h")
(list) "c" (list "" "6")))))
(pffi-define c-puts libc-stdlib 'puts 'int (list 'pointer))
(c-puts "Message brought to you by FFI!")
```

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
(define libc-stdlib
    (cond-expand
        (windows (pffi-shared-object-auto-load (list "stdlib.h")
(list) "ucrtbase" (list "")))
        (else (pffi-shared-object-auto-load (list "stdlib.h")
(list) "c" (list "" "6")))))
```

```
; Define C function that takes a callback
(pffi-define qsort libc-stdlib 'qsort 'void (list 'pointer 'int
'int 'callback))

; Define our callback
(pffi-define-callback compare
                      'int
                      (list '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 (pffi-pointer-allocate (* (pffi-size-of 'int) 3)))
(pffi-pointer-set! array 'int (* (pffi-size-of 'int) 0) 3)
(pffi-pointer-set! array 'int (* (pffi-size-of 'int) 1) 2)
(pffi-pointer-set! array 'int (* (pffi-size-of 'int) 2) 1)

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

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

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