# FFI (Foreign function Interface) Library

Mark Tarver

April 27th 2012

This is a very simple slot-and-filler approach to specifying an FFI to Shen based on my posts to qilang.

Formally, if you are going to pass the evaluation of any Shen expression E to a foreign language F, you need four things.

1. A syntax mapping from Shen to F
2. A means of sending the result of this mapping to F
3. A means of receiving the response of F
4. A means of translating, if needed, this response in terms Shen understands.

The FFI code in the Shen library is very short and is based on this model. There are two user functions.

1. ffi.
2. call-ffi

## An Example

Here is an example where we can set up a simple system based on a call to an external TCL/tk process.

```
(ffi tcl/tk (@p shen->tcl send-tcl) (@p tcl->shen receive-tcl))
```

which says

'To build an FFI to TCL/tk, use the syntax mapping 'shen->tcl' and use 'send-tcl' to send the object to TCL/tk; to get the result back, use 'receive-tcl' and apply 'tcl->shen' to read it'

The ffi declaration simply pushes this spec onto a global G, creating G if needed. **All the code in red is part of the FFI package and hence you do not have to define it yourself.**

```
(define ffi
Language OutSpec InSpec -> (push [Language OutSpec InSpec] *ffi*))

(define push
X Y -> (set Y [X | (trap-error (value Y) (/. E []))]))
```

Now we define a macro for calling the FFI.

```
(defmacro call-ffi-macro
[call-ffi Foreign-Language Code] -> (let Spec (assoc Foreign-Language (value *ffi*))
                                         (if (empty? Spec)
                                             (error "we don't know how to talk to ~A~%" Foreign-
Language)
                                             (process-ffi-call Spec Code))))
```

Given a call to the FFI, we look first for the appropriate spec to tell us how to talk to the foreign language. If we cannot find it, then we raise an error. Otherwise we use the spec to process the FFI call.

```
(define process-ffi-call
[_ (@p SyntaxOutF SendF) (@p SyntaxInF ReceiveF)] Code
-> [SyntaxInF [ReceiveF [SendF [SyntaxOutF (quote Code)]]]])
```

'process-ffi-call' takes the components of the spec and generates an application which applies them in the correct order. The 'quote' function simply turns the evaluable expression into a list structure; doing what ’ (QUOTE) does in Lisp.

```
(define quote
  [X | Y] -> [cons (quote X) (quote Y)]
  X -> X)
```

That's all there is to the FFI.

To make our *specific* FFI declaration work, we need to define

1. shen->tcl
2. send-tcl
3. tcl->shen
4. receive-tcl

Our TCL FFI reads strings, so we coerce the input into a string. I'm going to wildly simplify and assume TCL is being called perform simple calculations. Hence we have a simple conversion to infix.

```
(define shen->tcl
X -> (make-string "set x [expr ~A]; puts $x~%" (tcl-maths-expr X)))

(define tcl-maths-expr
[Op X Y] -> (make-string "(~A ~A ~A)" (tcl-maths-expr X) Op (tcl-maths-expr Y))
X -> X)
```

'send-tcl' is

```
(define send-tcl
  Data -> (do (pr Data (tcl-stream)) (FINISH-OUTPUT (tcl-stream))))
```

‘pr’ is a Shen primitive; it prints a string to a stream.

```
(define tcl-stream
-> (trap-error (value *tcl-stream*) (/. E (create-tcl-stream))))

(define create-tcl-stream
-> (set *tcl-stream* (open-process-stream
                "C:/TCL/BIN/WISH85.EXE" ["-name" "root"])))
```

And for open-process-stream we need a bit of CLisp.

```
(DEFUN open-process-stream (CMD &OPTIONAL ARGUMENTS)
(SYSTEM::RUN-PROGRAM CMD
:ARGUMENTS ARGUMENTS
:INPUT :STREAM
:OUTPUT :STREAM))
```

'receive-tcl' reads TCL output. Since this stream is a CLisp stream, I need a bit of Lisp.

```
(define receive-tcl
_ -> (READ (value *tcl-stream*)))
```

tcl->shen doesn't need to do much here; it is simply passing on a number.

```
(define tcl->shen
Data -> Data)
```

If we load all this and call it; we can add 12 and 14 in an extremely silly way. First we declare to Shen the FFI to TCL/tk ....

```
(2-) (ffi tcl/tk (@p shen->tcl send-tcl) (@p tcl->shen receive-tcl))
[[tcl/tk (@p shen->tcl send-tcl) (@p tcl->shen receive-tcl)]]
```

... then we use it to tell TCL/tk to add two numbers (note Shen is *not* doing the addition here)

```
(3-) (time (call-ffi tcl/tk (+ 12 14)))
run time: 0.0 secs
26
```