

UnifiedFFI

A common language to talk with the outside world

Now with demo!





Esteban Lorenzano

(The stabilisation fairy)





What is UnifiedFFI

- A front-end to express Foreign Function Interface (FFI) calls in Pharo.
- Uses ThreadedFFIPlugin, but should be able to plug others in the future.
- It shares same philosophy as NativeBoost
 - keep as much as possible in the image
 - no need to modify VM to add functionality
- But it is not ASM: Just plain Smalltalk.



But, what happened with NativeBoost?

- It was not working on Spur
- It was hard to maintain (and we need a different implementation for each architecture)
- Since we were using NB exclusively for FFI, we decided to replace it with a different one, using the VM plugin



UnifiedFFI goals

- Keep NativeBoost syntax
 - ▶ Because is cool :)
 - ▶ Provide backward compatibility for most uses
- Enhance documentation and self-documentation
- Be the unified base for future FFI backends implementations



How does a call looks like?

```
char *getenv(const char *)
```

```
getEnv: variable
```

```
  ^ self
```

```
    ffiCall: #( String getenv( String variable ))
```

```
    module: LibC
```

(People who know NativeBoost will find this very familiar....)



How does a call looks like?

```
char *getenv(const char *)
```

```
getEnv: variable
```

```
^ self  
  ffiCall: #( String getenv( String variable ))  
  module: LibC
```

A regular Pharo method with one argument



How does a call looks like?

```
char *getenv(const char *)
```

```
getEnv: variable
```

```
  ^ self
```

```
    ffiCall: #( String getenv( String variable ))
```

```
    module: LibC
```

A literal array to represent C function



How does a call looks like?

`char *getenv(const char *)`

```
getEnv: variable
```

```
^ self
```

```
ffiCall: #(String getenv( String variable ))
```

```
module: LibC
```


Types annotation used to generate marshalling code



How does a call looks like?

```
char *getenv(const char *)
```

```
getEnv: variable  
  ^ self  
  ffiCall: #( String getenv( String variable ))  
  module: LibC
```



The value to be passed when calling out



How does a call looks like?

```
char *getenv(const char *)
```

```
getEnv: variable
```

```
  ^ self
```

```
    ffiCall: #( String getenv( String variable ))
```

```
    module: LibC
```

The library to lookup C function



FFILibrary

- A very simple abstraction to define module names that can be different each platform.
- Can be used also as a place to store C function definitions (like a real library :).



FFILibrary: LibC

```
FFILibrary subclass: #LibC
```

```
LibC>>unixModuleName  
^ 'libc.so.6'
```

```
LibC>>macModuleName  
^ 'libc.dylib'
```

```
LibC>>win32ModuleName  
^ 'msvcrt.dll'
```

```
LibC>>memCopy: src to: dest size: n  
^ self ffiCall: #(void *memcpy(void *dest, const void *src, size_t n))
```



Insights to UnifiedFFI

```
getEnv: variable
```

```
  ^ self
```

```
    ffiCall: #( String getenv( String variable ))
```

```
    module: LibC
```



Insights to UnifiedFFI

```
char *getenv(const char *)
```

```
getEnv: variable
```

```
^ self
```

```
ffiCall: #( String getenv( String variable ))  
module: LibC
```

1. Generate bytecodes for marshalling
2. Re-send the method execution



How does a call looks like? (bytecode version)

```
char *getenv(const char *)
```

```
21 <20> pushConstant: <cdecl: char* 'getenv' (char*) module: 'libc.dylib'>  
22 <10> pushTemp: 0  
23 <8A 81> pop 1 into (Array new: 1)  
25 <E1> send: invokeWithArguments:  
26 <7C> returnTop
```



Types

- Support for standard C types: int, float, etc.
- Support for type aliases (map a name to one of the defined types)
- Complex types:
 - FFIExternalObject: External addresses (objects)
 - FFIOpaqueObject: Opaque C types/structures
 - FFIExternalStructure
 - FFIExternalArray, FFITypeArray
 - FFIExternalEnumeration
 - FFIExternalValueHolder: Buffers (to pass referenced data, e.g. “double *”)
 - FFIConstantHandle: Windows HANDLE (constant addresses)



FFIExternalObject (1)

```
cairo_surface_t *  
cairo_image_surface_create (cairo_format_t format,  
                             int width,  
                             int height);
```

```
AthensSurface subclass: #AthensCairoSurface  
  uses: TCairoLibrary  
  instanceVariableNames: 'handle context builder id ftFontRenderer session'  
  classVariableNames: ''  
  poolDictionaries: 'AthensCairoDefs'  
  package: 'Athens-Cairo'
```



FFIExternalObject (2)

```
cairo_surface_t *  
cairo_image_surface_create (cairo_format_t format, int width, int height);
```

```
AthensCairoSurface class>>primImage: aFormat width: aWidth height: aHeight  
^self ffiCall: #(AthensCairoSurface cairo_image_surface_create (  
    int aFormat,  
    int aWidth,  
    int aHeight) )
```



```
unsigned char *  
cairo_image_surface_get_data (cairo_surface_t *surface);
```

```
AthensCairoSurface>>getDataPtr  
  "get a pointer to surface bitmap data"  
  
  ^self ffiCall: #( void* cairo_image_surface_get_data ( self ) )
```



FFICallback

```
int (*compar)(const void*,const void*)
```

```
callback := FFICallback  
signature: #(int (const void *arg1, const void *arg2))  
block: [ :arg1 :arg2 |  
        ((arg1 doubleAt: 1) - (arg2 doubleAt: 1)) sign ].
```



FFICallback

```
int (*compar)(const void*,const void*)
```

```
callback := FFICallback  
signature: #(int (const void *arg1, const void *arg2))  
block: [ :arg1 :arg2 |  
        ((arg1 doubleAt: 1) - (arg2 doubleAt: 1)) sign ].
```

A literal array to represent C **anonymous** function



FFICallback: qsort

```
Cqsort>>primQsort: array with: count with: size with: compare
self
  ffiCall: #(void qsort (FFIExternalArray array, size_t count, size_t size, FFIcallback compare))
  module: LibC

Cqsort>>example
| array callback |

array := FFIExternalArray newType: 'double' size: 100.
1 to: 100 do: [ :i | array at: i put: (100 atRandom asFloat) ].
callback := FFIcallback
  signature: #(int (const void *arg1, const void *arg2))
  block: [ :arg1 :arg2 |
    ((arg1 doubleAt: 1) - (arg2 doubleAt: 1)) sign ].

self
  primQsort: array
  with: 100
  with: (FFIExternalType sizeOf: 'double')
  with: callback.
```



Status

- Feature complete (but I keep adding stuff when requested)
- Documentation ongoing
- Still can accept a lot of optimisations (but that will be for versions 1.x)



Summary

- Replace for NativeBoost keeping its philosophy
- Simple Callout/Callback system
- Very easy to extend (no need of ASM knowledge)
- Is ready to use in Pharo 5.0



Thanks!

Smalltalk quitSession.

