Objective C Bridge User Guide

Release 1.0

Dylan Hackers

December 15, 2018

CONTENTS

The Objective C / Dylan bridge is designed to work with the modern Objective C 2.0 run-time as found on recent versions of Mac OS X and iOS. The bridge library itself can be found at https://github.com/dylan-foundry/objc-dylan/. A pre-release version of Open Dylan 2014.1 is currently required as the compiler has been extended to support this library.

CHAPTER

ONE

QUICK USAGE

Objective C selectors (method definitions can be described using the *objc-selector-definer* macro. Messages can be sent to Objective C classes and instances using the *send* macro.

This demonstrates the definition of 3 standard selectors and how to invoke them:

```
define objc-selector @alloc
  parameter target :: <objc/class>;
  result objc-instance :: <objc/instance>;
  selector: "alloc";
end;

define objc-selector @init
  parameter target :: <objc/instance>;
  result objc-instance :: <objc/instance>;
  selector: "init";
end;

define objc-selector @retain-count
  parameter target :: <objc/instance>;
  result retain-count :: <C-int>;
  selector: "retainCount";
end;

begin
  let inst = send(send($NSObject, @alloc), @init);
  let count = send(inst, @retain-count);
end;
```

New subclasses of Objective C classes can be created from Dylan and methods added to them:

```
define objc-method finished-launching
    (self, cmd, notification :: <NSNotification>)
 => ()
  c-signature: (self :: <MyDelegate>, cmd :: <objc/selector>,
                notification :: <NSNotification>) => ()
  format-out("Application has launched.\n");
end;

define objc-class <MyDelegate> (<NSObject>) => MyDelegate
  bind @applicationDidFinishLaunching/ => finished-launching;
end;
```

See *objc-class-definer* and *objc-method-definer* for more details.

CHAPTER

TWO

DESIGN

Type Safety

This bridge library provides a type-safe mechanism for accessing Objective C libraries. A library binding must set up a hierarchy of Dylan classes that mirror the Objective C class hierarchy using the *objc-shadow-class-definer* macro.

When used correctly, instances of Objective C classes should be able to correctly participate in Dylan's method dispatch mechanisms.

To help make this more clear, in Objective C, `NSNumber` is a subclass of `NSValue`. In Dylan, we represent this relationship:

```
define objc-shadow-class <ns/value> (<ns/object>, <<ns/copying>>,
                                     <<ns/secure-coding>>)
 => NSValue;
define objc-shadow-class <ns/number> (<ns/value>) => NSNumber;
```

Now, instances of `NSNumber` will appear to be of type `<ns/number>` and instances of `NSValue` would be of type `<ns/value>`. This allows method definitions such as these to work:

```
define method print-object (value :: <ns/value>, stream :: <stream>)
  ...
end;

define method print-object (number :: <ns/number>, stream :: <stream>)
  ...
end;
```

Performance

Attempts have been made to keep the overhead from using the bridge to a minimum. Most Dylan method dispatch has been eliminated (with the exception of *objc/make-instance*), and inlining has been used as needed to further reduce overhead.

CHAPTER

THREE

LIMITATIONS

Memory Management

A design is not yet in place for simplifying Objective C memory management or integrating it with the Dylan garbage collection. This will be provided in a future version of this library.

Not Yet Supported

We do not yet support these features of the Objective C run-time:

- Doing much of anything with *<objc/method>*.
- Protocol introspection or representing Objective C protocols on the Dylan side.
- Properties of classes or instances.
- Access to instance variables.

Patches are welcome. Details about many of these things can be found in `/usr/include/objc/runtime.h`.

Unsupported

Functions available in earlier versions of the Objective C run-time that have been deprecated or removed are not supported.

The GNU Objective C run-time is also not supported.

CHAPTER

FOUR

NAMING SCHEME

Selectors are commonly named with a prefix of `@` such as `@alloc`, `@description`. Colons within a selector name can be converted to `/` as in `@perform-selector/with-object/`. That example also demonstrates the conversion of the changes in case to the more Dylan-like use of hyphenated names.

Shadow classes are named in typical Dylan fashion with the name of the Objective C class wrapped in `<...>`: `<NSObject>`, `NSApplication`. The actual underlying Objective C class is available as a constant: `$NSObject`, `$NSApplication`.

Protocols are commonly named with double << and >> as with `<<NSObject>>` to distinguish them from a regular class or shadow class.

CHAPTER

FIVE

TYPE ENCODING

The types correspond to the Dylan C-FFI types as follows:

Type	Enc
<objc/instance>	'@'
<objc/class>	'#'
<objc/selector>	':'
<C-character>	'c'
<C-unsigned-char>	'C'
<C-short>	's'
<C-unsigned-short>	'S'
<C-int>	'i'
<C-unsigned-int>	'I'
<C-long>	'l'
<C-unsigned-long>	'L'
<C-float>	'f'
<C-double>	'd'
<C-boolean>	'B'
<C-void>	'v'
poiniter to	'^'
<C-string>	'*'

CHAPTER

SIX

THE OBJECTIVE-C MODULE

Macros

`send` Macro

Sends an Objective C message to a target.

Macro Call

```
send(*target*, *selector*, *args*)
```

Discussion The selector must be the binding name that refers to the Objective C selector as the name given here is used literrally to construct a function call.

Example This example:

```
let inst = send($NSObject, @alloc);
```

expands to:

```
let inst = %send-@alloc($NSObject);
```

`objc-class-definer` Macro

Defines a new Objective C class, creates the corresponding shadow class, and allows binding method implementations to the class.

Macro Call

```
define objc-class *class-name* (*superclass*) => *objective-c-name*
  bind *selector* => *objc-method*;
  ...
end;
```

Parameters

- **`class-name`** – the name of the Dylan shadow class.
- **`superclass`** – the names of the Dylan shadow superclass.
- **`objective-c-name`** – the name of the Objective C class being created.
- **`selector`** – The selector to be bound to a method implementation.
- **`objc-method`** – The method defined via *`objc-method-definer`*.

Discussion

Defines a new Objective C class and the corresponding Dylan shadow class. The new class can only have a single super-class (named by *superclass*). Protocol support will be added in the future.

Methods may be bound to the new Objective C class using the bind syntax:

```
bind *selector* to *objc-method* (*type-encoding*);
```

Example

```
define objc-selector @adder
  parameter target :: <ns/object>;
  parameter a :: <C-int>;
  result r :: <C-int>;
  selector: "adder:";
end;

define objc-method adder
    (self, selector, a :: <integer>)
 => (r :: <integer>)
  c-signature: (self :: <objc/instance>, selector :: <objc/selector>,
                a :: <C-int>) => (r :: <C-int>);
  assert-true(instance?(self, <test-class>));
  a + 1
end;

define objc-class <test-class> (<ns/object>) => DylanTestClass
  bind @adder => adder;
end;
```

`objc-method-definer` Macro

Defines a function in Dylan and the associated information needed to invoke it from Objective C.

Macro Call

```
define objc-method *name* (*args*) => (*result*)
  c-signature: (*cffi-args*) => (*cffi-result*);
  *body*
end;
```

Parameters

- **`name`** – The name of the method.
- **`args`** – A typical parameter list for a method.
- **`result`** – A typical result definition. If no result, then just `()`.
- **`cffi-args`** – A typical parameter list for a method, but all parameters must have a C-FFI type associated with them.
- **`cffi-result`** – A typical result defintion, but the type must be one of the C-FFI types. If no result, then just `()`.
- **`body`** – The body of the method.

Discussion

Defines a function in Dylan and the associated information needed to invoke it from Objective C. This is used in combination with *`objc-class-definer`* to add methods to an Objective C class.

Example

```
define objc-method finished-launching
    (self, cmd, notification :: <NSNotification>)
 => ()
  c-signature: (self :: <MyDelegate>, cmd :: <objc/selector>,
                notification :: <NSNotification>) => ()
  format-out("Application has launched.\n");
end;
```

`objc-protocol-definer` Macro

Macro Call

```
define objc-protocol *protocol-name*;
```

Discussion

> **Note:** This will change in the near future when we introduce improved support for Objective C protocols.

Example

```
define objc-protocol <<ns/copying>>;
```

This currently expands to:

```
define abstract class <<ns-copying>> (<object>)
end;
```

`objc-selector-definer` Defining Macro

Describe Objective C selectors to the *c-ffi*.

Macro Call

```
define objc-selector *name*
  [*parameter-spec*; ...]
  [*result-spec*;]
  [*function-option*, ...;]
end [C-function] [*name*]
```

Parameters

- **`name`** – A Dylan variable name.
- **`parameter-spec`** –
- **`result-spec`** –
- **`function-option`** – A property list.

Discussion

Describes an Objective C selector to the C-FFI. In order for a selector to be invoked correctly by Dylan, the same information about the selector must be given as is needed by C callers, including the selector's name and the types of its parameters and results.

The result of processing a `define objc-selector` definition is a Dylan function and a constant bound to *name*. This function takes Dylan objects as arguments, converting them to

their C representations according to the types declared for the parameters of the C function before invoking the selector with them. If the corresponding Objective C method returns results, these results are converted to Dylan representations according to the declared types of those results before being returned to the Dylan caller of the function. By default the function created is a raw function, not a generic function. A generic function method can defined by using the *generic-function-method:* option.

The *selector:* function option must be supplied with a constant string value for the name of the selector.

There must be at least one parameter specification. The first parameter specifies the target of the method, so it should be either an Objective C class or an object instance.

A parameter-spec has the following syntax:

```
[*adjectives*] parameter name :: *c-type* #key *c-name*
```

If only the target parameter is specified, the selector is taken to have no arguments.

The adjectives can be either *output*, *input*, or both. The calling discipline is specified by the *input* and *output* adjectives.

By itself, *input* indicates that the argument is passed into the function by value. This option is the default and is used primarily to document the code. There is a parameter to the generated Dylan function corresponding to each *input* parameter of the C function.

The *output* adjective specifies that the argument value to the C function is used to identify a location into which an extra result of the C function will be stored. There is no parameter in the generated Dylan function corresponding to an *output* parameter of the C function. The C-FFI generates a location for the extra return value itself and passes it to the C function. When the C function returns, the value in the location is accessed and returned as an extra result from the Dylan function. The C-FFI allocates space for the output parameter's referenced type, passes a pointer to the allocated space, and returns `pointer-value` of that pointer. A struct or union type may not be used as an output parameter.

If both *input* and *output* are supplied, they specify that the argument value to the C function is used to identify a location from which a value is accessed and into which an extra result value is placed by the C function. There is a parameter to the generated Dylan function corresponding to each *input output* parameter of the C function that is specialized as the union of the export type of the referenced type of the type given for the parameter in `define c-function`, and `#f`. When the C function returns, the value in the location is accessed and returned as an extra result from the Dylan function. If an *input output* parameter is passed as `#f` from Dylan then a `NULL` pointer is passed to the C function, and the extra value returned by the Dylan function will be `#f`.

Note that neither *output* nor *input output* affects the declared type of an argument: it must have the same type it has in C and so, because it represents a location, must be a pointer type.

A result-spec has the following syntax:

```
result [name :: c-type]
```

If no *result* is specified, the Dylan function does not return a value for the C result, and the C function is expected to have a return type of *void*.

Each *function-option* is a keyword–value pair.

The *generic-function-method:* option may be either `#t` or `#f`, indicating whether to add a method to the generic function name or to bind a bare constant method directly to name. The default value for *generic-function-method:* is `#f`.

The option *C-modifiers:* can be used to specify alternate versions of `objc_msgSend` to use. For example, if a selector needs to be sent using `objc_msgSend_fpret`, then you would use `C-modifiers: "_fpret"`.

In effect, a `define objc-selector` such as:

```
define objc-selector @alloc
  parameter objc-class :: <objc/class>;
  result instance :: <objc/instance>;
  c-name: "alloc";
end;
```

expands into something like:

```
define constant @alloc = objc/register-selector("alloc", "@#:");
define function %send-@alloc (target)
  let c-target = %as-c-representation(<objc/class>,
                                      target);
  let c-selector = %as-c-representation(<objc/selector,
                                        @alloc);
  let c-result = %objc-msgsend(c-target, c-selector);
  %as-dylan-representation(<objc/instance>, c-result)
end;
```

with the declared type.

Example

```
define objc-selector @alloc
  parameter target :: <objc/class>;
  result objc-instance :: <objc/instance>;
  selector: "alloc";
end;
```

`objc-shadow-class-definer` Macro

Macro Call

```
define objc-shadow-class *class-name* (*superclasses*)
  => *objective-c-class*;
```

Parameters

- **`class-name`** – the name of the dylan shadow class.
- **`superclasses`** – the names of the dylan shadow superclasses and protocols.
- **`objective-c-class`** – the name of the objective c class being shadowed.

Discussion The shadow class hierarchy is an important part of how we enable a type-safe binding to an Objective C library.

Example

```
define objc-shadow-class <ns/value> (<ns/object>, <<ns/copying>>,
                                     <<ns/secure-coding>>)
 => NSValue;
define objc-shadow-class <ns/number> (<ns/value>) => NSNumber;
```

The definition of `<ns/number>` would expand to a couple of important definitions:

```
define constant $NSNumber = objc/get-class("NSNumber");
define class <ns/number> (<ns/value>)
  inherited slot instance-objc-class, init-value: $NSNumber;
end;
objc/register-shadow-class($NSNumber, <ns/number>);
```

We can see that the important elements are:

- The constant, `$NSNumber`, that represents the Objective C class.
- The shadow class, `<ns/number>`.
- The shadow class is registered so that *objc/make-instance* can work.

Classes

<objc/class> Class

The Dylan representation of an Objective C class object.

Superclasses <c-statically-typed-pointer>

Discussion

This class is not meant to be inherited from. To represent an instance of an Objective C class, a subclass of *<objc/instance>* as created by a hierarchy of *objc-shadow-class-definer* calls would be used.

Messages may be sent to Objective C classes using instances of this class.

objc/class-name Function

Returns the name of an Objective C class.

Signature objc/class-name (objc-class) => (objc-class-name)

Parameters

- **objc-class** – An instance of *<objc/class>*.

Values

- **objc-class-name** – An instance of `<string>`.

objc/super-class Function

Returns the superclass of an Objective C class.

Signature objc/super-class (objc-class) => (objc-super-class?)

Parameters

- **objc-class** – An instance of *<objc/class>*.

Values

- **objc-super-class?** – An instance of `false-or(<objc/class>)`.

objc/class-responds-to-selector? Function

Returns whether or not an Objective C class responds to the given selector.

Signature objc/class-responds-to-selector? (objc-class selector) => (well?)

Parameters

- **objc-class** – An instance of *<objc/class>*.

- **selector** – An instance of *<objc/selector>*.

Values

- **well?** – An instance of <boolean>.

objc/get-class Function

Looks up an Objective C class, given its name.

Signature objc/get-class (name) => (objc-class)

Parameters

- **name** – An instance of <string>.

Values

- **objc-class** – An instance of false-or(<objc/class>).

objc/get-class-method Function

Signature objc/get-class-method (objc-class selector) => (method?)

Parameters

- **objc-class** – An instance of *<objc/class>*.
- **selector** – An instance of *<objc/selector>*.

Values

- **method?** – An instance of false-or(<objc/method>).

objc/get-instance-method Function

Signature objc/get-instance-method (objc-class selector) => (method?)

Parameters

- **objc-class** – An instance of *<objc/class>*.
- **selector** – An instance of *<objc/selector>*.

Values

- **method?** – An instance of false-or(<objc/method>).

Instances

<objc/instance> Abstract Class

Represents an instance of an Objective C class.

Superclasses <c-statically-typed-pointer>

Discussion

Direct instances of this class are not used. Instead, use instances of subclasses created with *objc-shadow-class-definer*.

When this class is used as the result type for a selector, the value will be mapped back into the correct instance of a subclass of *<objc/instance>*. This requires that the actual class has been correctly set up as a shadow class or an error will be signaled.

$nil Constant

objc/instance-class Function

Signature objc/instance-class (objc-instance) => (objc-class)

Parameters

- **objc-instance** – An instance of *<objc/instance>*.

Values

- **objc-class** – An instance of *<objc/class>*.

objc/instance-class-name Function

Signature objc/instance-class-name (objc-instance) => (objc-class-name)

Parameters

- **objc-instance** – An instance of *<objc/instance>*.

Values

- **objc-class-name** – An instance of <string>.

objc/instance-size Function

Signature objc/instance-size (objc-class) => (objc-instance-size)

Parameters

- **objc-class** – An instance of *<objc/class>*.

Values

- **objc-instance-size** – An instance of <integer>.

objc/make-instance Function

Signature objc/make-instance (raw-instance) => (objc-instance)

Parameters

- **raw-instance** – An instance of <machine-word>.

Values

- **objc-instance** – An instance of *<objc/instance>*.

Methods

<objc/method> Class

Represents an Objective C method object.

Superclasses <c-statically-typed-pointer>

objc/method-name Function

Signature objc/method-name (objc-method) => (objc-method-selector)

Parameters

- **objc-method** – An instance of *<objc/method>*.

Values

- **objc-method-selector** – An instance of *<objc/selector>*.

Selectors

<objc/selector> Class
Represents an Objective C selector.

Superclasses <c-statically-typed-pointer>

objc/register-selector Function
Returns an *<objc/selector>* for the given selector name.

Signature objc/register-selector (name, type-encoding) => (objc-selector)

Parameters

- **name** – An instance of <string>.
- **type-encoding** – An instance of <string>.

Values

- **objc-selector** – An instance of *<objc/selector>*.

Discussion

This will not usually be called in user code. Instead, the selector is usually defined using *objc-selector-definer*.

See *Type Encoding* for more details on the *type-encoding* parameter.

objc/selector-name Function
Returns the name of the given selector.

Signature objc/selector-name (objc-selector) => (selector-name)

Parameters

- **objc-selector** – An instance of *<objc/selector>*.

Values

- **selector-name** – An instance of <string>.

Associated Objects

objc/associated-object Generic function

Signature objc/associated-object (objc-instance key) => (objc-instance)

Parameters

- **objc-instance** – An instance of *<objc/instance>*.
- **key** – An instance of either a <string> or a <symbol>.

Values

- **objc-instance** – An instance of *<objc/instance>*.

objc/remove-associated-objects Function

Signature objc/remove-associated-objects (objc-instance) => ()

Parameters

- **objc-instance** – An instance of *<objc/instance>*.

$objc-association-assign Constant

$objc-association-copy Constant

$objc-association-copy-nonatomic Constant

$objc-association-retain-nonatomic Constant

$objc-association-return Constant

objc/set-associated-object Generic function

Signature objc/set-associated-object (objc-instance key value association-policy) => ()

Parameters

- **objc-instance** – An instance of *<objc/instance>*.
- **key** – An instance of either a <string> or a <symbol>.
- **value** – An instance of *<objc/instance>*.
- **association-policy** – An instance of <integer>.

Protocols

<objc/protocol> Class

Represents an Objective C protocol.

Superclasses <C-statically-typed-pointer>

objc/get-protocol Function

Looks up an Objective C protocol, given its name.

Signature objc/get-protocol (name) => (objc-protocol)

Parameters

- **name** – An instance of <string>.

Values

- **objc-protocol** – An instance of false-or(<objc/protocol>).

objc/protocol-name Function

Signature objc/protocol-name (objc-protocol) => (objc-protocol-name)

Parameters

- **objc-protocol** – An instance of *<objc/protocol>*.

Values

- **objc-protocol-name** – An instance of <string>.

objc/conforms-to-protocol? Generic function

Signature objc/conforms-to-protocol? (object) => (conforms?)

Parameters

- **object** – An instance of *<objc/class>* or *<objc/protocol>*.

Values

- **conforms?** – An instance of <boolean>.

Core Foundation Bindings

<<ns/object>> Abstract Class

Superclasses <object>

<ns/object> Class

Superclasses *<objc/instance>*, *<<ns/object>>*

N

O

S

Symbols

O

S