Objective-C and Swift Interop

This document describes how Swift interoperates with Objective-C code and the Objective-C runtime.

Interactions between the Swift runtime and the Objective-C runtime are implementation details that are subject to change! Don't write code outside the Swift runtime that relies on these details.

This document only applies on platforms where Objective-C interop is available. On other platforms, Swift omits everything related to Objective-C.

Messaging

Swift generates calls to `objc_msgSend` and variants to send an Objective-C message, just like an Objective-C compiler does. Swift methods marked or inferred as `@objc` are exposed as entries in the Objective-C method list for the class.

Classes

All Swift classes are also Objective-C classes. When Swift classes inherit from an Objective-C class, they inherit in exactly the same way an Objective-C class would, by generating an Objective-C class structure whose `superclass` field points to the Objective-C superclass. Pure Swift classes with no superclass generate an Objective-C class that subclasses the internal `SwiftObject` class in the runtime. `SwiftObject` implements a minimal interface allowing these objects to be passed around through Objective-C code with correct memory management and basic functionality.

Compiler-Generated Classes

Swift classes can be generated as part of the binary's static data, like a normal Objective-C class would be. The compiler lays down a structure matching the Objective-C class structure, followed by additional Swift-specific fields.

Dynamically-Generated Classes

Some Swift classes (for example, generic classes) must be generated at runtime. For these classes, the Swift runtime allocates space using `MetadataAllocator`, fills it out with the appropriate class structures, and then registers the new class with the Objective-C runtime by using the SPI `objc_readClassPair`.

Stub Classes

Note: stub classes are only supported on macOS 10.15+, iOS/tvOS 13+, and watchOS 6+. The Objective-C runtime on older OSes does not have the necessary

calls to support them. The Swift compiler will only emit stub classes when targeting an OS version that supports them.

Stub classes can be generated for dynamically-generated classes that are known at compile time but whose size can't be known until runtime. This unknown size means that they can't be laid down statically by the compiler, since the compiler wouldn't know how much space to reserve. Instead, the compiler generates a *stub class*. A stub class consists of:

```
uintptr_t dummy;
uintptr_t one;
SwiftMetadataInitializer initializer;
```

The `dummy` field exists to placate the linker. The symbol for the stub class points at the `one` field, and uses the `.alt_entry` directive to indicate that it is associated with the `dummy` field.

The `one` field exists where a normal class's `isa` field would be. The Objective-C runtime uses this field to distinguish between stub classes and normal classes. Values between 1 and 15, inclusive, indicate a stub class. Values other than 1 are currently reserved.

An Objective-C compiler can refer to these classes from Objective-C code. A reference to a stub class must go through a `classref`, which is a pointer to a class pointer. The low bit of the class pointer in the `classref` indicates whether it needs to be initialized. When the low bit is set, the Objective-C runtime treats it as a pointer to a stub class and uses the initializer function to retrieve the real class. When the low bit is clear, the Objective-C runtime treats it as a pointer to a real class and does nothing.

The compiler must then access the class by generating a call to the Objective-C runtime function `objc_loadClassref` which returns the initialized and relocated class. For example, this code:

```
[SwiftStubClass class]
```

Generates something like this code:

```
static Class *SwiftStubClassRef =
  (uintptr_t *)&_OBJC_CLASS_$_SwiftStubClassRef + 1;
Class SwiftStubClass = objc_loadClassref(&SwiftStubClassRef);
objc_msgSend(SwiftStubClass, @selector(class));
```

The initializer function is responsible for setting up the new class and returning it to the Objective-C runtime. It must be idempotent: the Objective-C runtime takes no precautions to avoid calling the initializer multiple times in a multi-threaded environment, and expects the initializer function itself to take care of any such needs.

The initializer function must register the newly created class by calling the `_objc_realizeClassFromSwift` SPI in the Objective-C runtime. It must pass

the new class and the stub class. The Objective-C runtime uses this information to set up a mapping from the stub class to the real class. This mapping allows Objective-C categories on stub classes to work: when a stub class is realized from Swift, any categories associated with the stub class are added to the corresponding real class.

To Document

This document is incomplete. It should be expanded to include:

- Information about the ABI of ObjC and Swift class structures.
- The is-Swift bit.
- Legacy versus stable ABI and is-Swift bit rewriting.
- Objective-C runtime hooks used by the Swift runtime.
- And more?