Working with C/C++ & Swift

Richard Topchii

Mobile Week 23 August 2023

Agenda

- Overview of methods for working with C/C++
- Pros & Cons, Examples
- Case Study

C++ & Swift Interop

New in 2023

0:00 / 17:44

Overview Transcript Code

Mix Swift and C++

Learn how you can use Swift in your C++ and Objective-C++ projects to make your code safer, faster, and easier to develop. We'll show you how to use C++ and Swift APIs to incrementally incorporate Swift into your app.

WWDC 2023 session 10172

Motivation

Benefits of brining C/C++ into codebase

Cons

- Build issues are harder to debug
- Dependency management is usually per platform

Pros

- Cross-Platform (Apple, Windows, Android…)
- Great tooling support
- Reuse existing code
- Use C/C++ libraries
- Verify/Test code only once

Multiple ways of working

C Interop

Objective-C++

ObjC

C++ Interop

C Interop

- Allows calling C functions from Swift
- … and vice versa
- Available since ~2015
- Very flexible & reliable

Keychain API

Using C in Swift

```
var query = [String: Any]()
query[kSecAttrAccount as String] = key
query[kSecReturnData as String] = kCFBooleanTrue
query[kSecMatchLimit as String] = kSecMatchLimitOne
query[kSecReturnAttributes as String] = kCFBooleanTrue
query[kSecUseAuthenticationContext as String] = context


var queryResult: CFTypeRef?
let status = SecItemCopyMatching(query as CFDictionary, &queryResult)

guard [errSecItemNotFound, errSecSuccess]
    .contains(status) else { return nil }

if let result = queryResult as? [String: Any] {
    return result[kSecValueData as String] as? Data
}
```

Endpoint Security

Using C in Swift

```
// Create the client.
es_client_t *client = NULL;
es_new_client_result_t newClientResult =
es_new_client(&client,
              ^(es_client_t * client, const es_message_t * message) {
    switch (message->event_type) {
        case ES_EVENT_TYPE_AUTH_EXEC:
            es_respond_auth_result(client, message, ES_AUTH_RESULT_ALLOW, true);
            break;
        default:
            panic("Found unexpected event type: %i", message->event_type);
            break;
    }
});
```

Objective-C++

- C++ code is called from Objective-C
- Objective-C is called from Swift using Interop
- Need to maintain an ObjC <> C++ „bridge“
- Very little documentation & support (compared to other methods)

Objective-C++

Texture/AsyncDisplayKit

- Performance-oriented UI framework
- Written in Objective-C++
- C++ is used with Objective-C types
- Interface is fully compatible with Objective-C

C++ Interop

- Available starting 2023
- Newest API
- Allows directly calling C++ from Swift and vice versa
- Not everything can be imported / called from Swift

C++ Interop

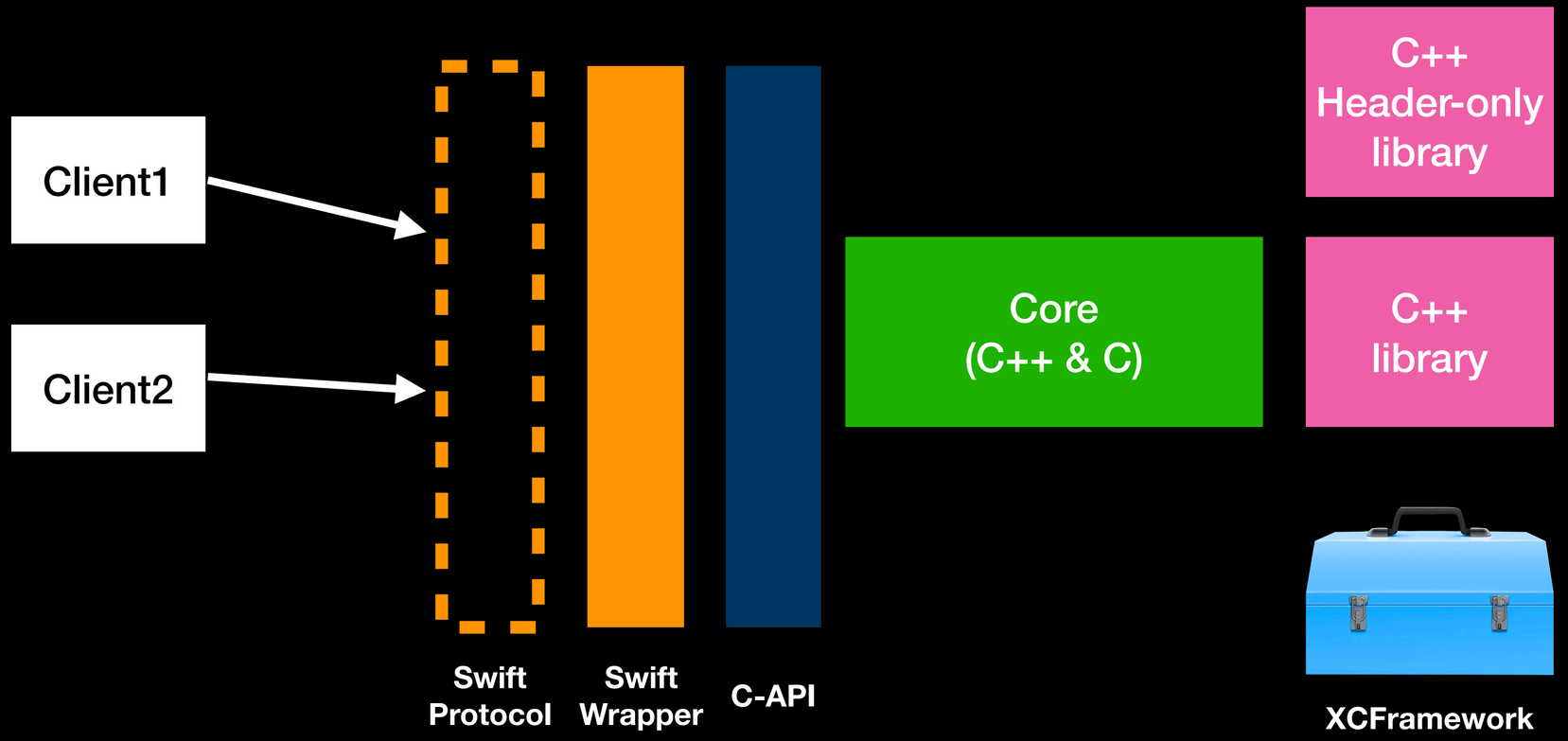
```cpp
class FibonacciCalculatorCplusplus {
public:
    FibonacciCalculatorCplusplus(bool printInvocation);
    double fibonacci(double value) const;
private:
    bool printInvocation;
};
```

```
// Create the C++ `FibonacciCalculatorCplusplus` class and invoke its `fibonacci` method.
let cxxCalculator = FibonacciCalculatorCplusplus(printInvocation)
return cxxCalculator.fibonacci(value - 1.0) + cxxCalculator.fibonacci(value - 2.0)
```

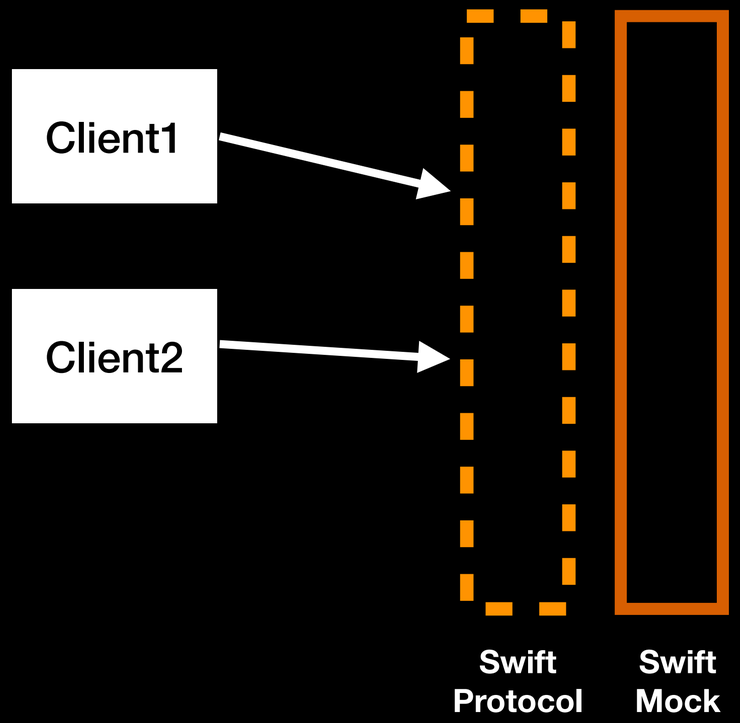
Example case

- A C++ library with C interface (Cross-platform)
- Tested with GoogleTest
- Uses pre-built frameworks
- Outside looks like just a Swift library

Example case
Architecture



Example case
Architecture



Swift Wrapper

Hides C/C++ Complexity

```swift
public func save(_ value: String) throws -> String {
    var returnValue: UnsafePointer<CChar>?
    defer {
        FreeString(returnValue)
    }
    let result = withUnsafeMutablePointer(to: &returnValue) {
        Core_Save(handle, value, $0)
    }
    if result != 0 {
        throw NSError(domain: "Core", code: -1)
    }

    return String(cString: returnValue)
}
```

Swift Wrapper

Hides C/C++ Complexity

```swift
public func save(_ value: String) throws -> String {
    var returnValue: UnsafePointer<CChar>?
    defer {
        FreeString(returnValue)
    }
    let result = withUnsafeMutablePointer(to: &returnValue) {
        Core_Save(handle, value, $0)
    }
    if result != 0 {
        throw NSError(domain: "Core", code: -1)
    }

    return String(cString: returnValue)
}
```

C-API Call

Swift Wrapper

Hides C/C++ Complexity

Allows writing to pointer

```swift
public func save(_ value: String) throws -> String {
    var returnValue: UnsafePointer<CChar>?
    defer {
        FreeString(returnValue)
    }
    let result = withUnsafeMutablePointer(to: &returnValue) {
        Core_Save(handle, value, $0)
    }
    if result != 0 {
        throw NSError(domain: "Core", code: -1)
    }

    return String(cString: returnValue)
}
```

C-API Call

Combine / Asynchronous code

- C Code can call the Swift code back (function pointer)
- Function pointers are imported as a closures with **@convention(c)**

Combine / Asynchronous code

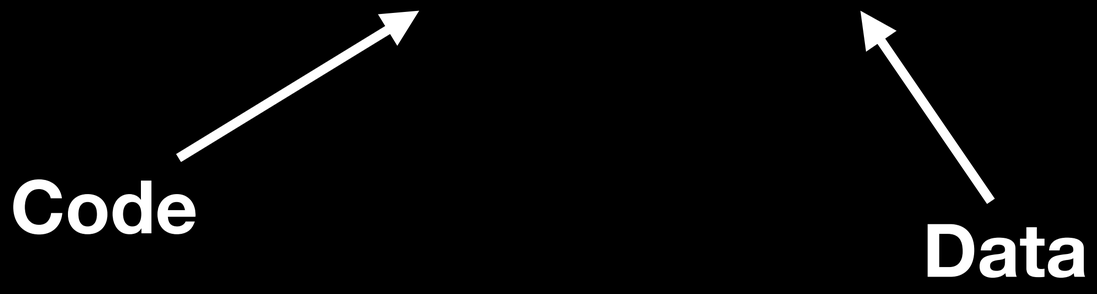
```
typedef void(CALLBACK_TYPE *Callback)(
    void *context,
    const char *outputValue,
    ErrorCode result);


ErrorCode AsyncOperation(CoreHandle handle, Callback cb, void *context, const char *inputValue);
```

Combine / Asynchronous code

```
typedef void(CALLBACK_TYPE *Callback)(
    void *context,
    const char *outputValue,
    ErrorCode result);


ErrorCode AsyncOperation(CoreHandle handle, Callback cb, void *context, const char *inputValue);
```



Combine / Asynchronous code

Helper tool

```
private final class PromiseWrapper<Output, Failure: Error> {
    let promise: Future<Output, Failure>.Promise

    init(_ value: @escaping Future<Output, Failure>.Promise) {
        self.promise = value
    }
}
```

Combine / Asynchronous code

Helper tool

```
private final class PromiseWrapper<Output, Failure: Error> {
    let promise: Future<Output, Failure>.Promise

    init(_ value: @escaping Future<Output, Failure>.Promise) {
        self.promise = value
    }
}
```

Just stores the promise

Combine / Asynchronous code

```
public func asyncOperation(_ inputValue: String) -> Future<Void, Error> {
    let handle = self.handle
    return Future { promise in
        typealias CCallback = @convention(c)(UnsafeMutableRawPointer?, CoreError) -> Void
        let callback: CCallback = {context, result in
            if let context {
                let promise = Unmanaged<PromiseWrapper<Void, Error>>
                    .fromOpaque(context)
                    .takeRetainedValue()
                    .promise
                promise(.success(()))
                // promise(.failure(error))
            }
        }

        let wrapper = PromiseWrapper(promise)
        let context = Unmanaged
            .passRetained(wrapper)
            .toOpaque()
        Core_AsyncOperation(handle,
                            callback,
                            context,
                            inputValue)
    }
}
```

Combine / Asynchronous code

```
public func asyncOperation(_ inputValue: String) -> Future<Void, Error> {
    let handle = self.handle
    return Future { promise in
        typealias CCallback = @convention(c)(UnsafeMutableRawPointer?, CoreError) -> Void
        let callback: CCallback = {context, result in
            if let context {
                let promise = Unmanaged<PromiseWrapper<Void, Error>>
                    .fromOpaque(context)
                    .takeRetainedValue()
                    .promise
                promise(.success(()))
                // promise(.failure(error))
            }
        }

        let wrapper = PromiseWrapper(promise)
        let context = Unmanaged
            .passRetained(wrapper)
            .toOpaque()
        Core_AsyncOperation(handle,
                            callback,
                            context,
                            inputValue)

    }
}
```

1. Function invocation

Combine / Asynchronous code

```
public func asyncOperation(_ inputValue: String) -> Future<Void, Error> {
    let handle = self.handle
    return Future { promise in
        typealias CCallback = @convention(c)(UnsafeMutableRawPointer?, CoreError) -> Void
        let callback: CCallback = {context, result in
            if let context {
                let promise = Unmanaged<PromiseWrapper<Void, Error>>
                    .fromOpaque(context)
                    .takeRetainedValue()
                    .promise
                promise(.success(()))
                // promise(.failure(error))
            }
        }

        let wrapper = PromiseWrapper(promise)
        let context = Unmanaged
            .passRetained(wrapper)
            .toOpaque()
        Core_AsyncOperation(handle,
                            callback,
                            context,
                            inputValue)

    }
}
```

1. Function invocation

Combine / Asynchronous code

```
public func asyncOperation(_ inputValue: String) -> Future<Void, Error> {
    let handle = self.handle
    return Future { promise in
        typealias CCallback = @convention(c)(UnsafeMutableRawPointer?, CoreError) -> Void
        let callback: CCallback = {context, result in
            if let context {
                let promise = Unmanaged<PromiseWrapper<Void, Error>>
                    .fromOpaque(context)
                    .takeRetainedValue()
                    .promise
                promise(.success(()))
                // promise(.failure(error))
            }
        }

        let wrapper = PromiseWrapper(promise)
        let context = Unmanaged
            .passRetained(wrapper)
            .toOpaque()
        Core_AsyncOperation(handle,
                            callback,
                            context,
                            inputValue)

    }
}
```

2. Callback invocation

1. Function invocation

Q&A

Richard Topchii

- LinkedIn: Richard Topchii
- YouTube: @richardtopchii
- Twitter: @richardtop_iOS
- GitHub: @richardtop