Getting started with C++ Interoperability

This document is designed to get you started with bidirectional API-level interoperability between Swift and C++.

Table of Contents

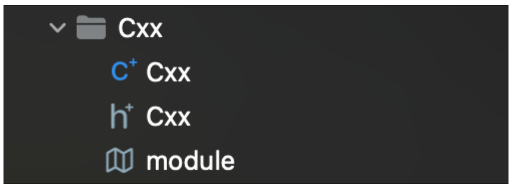
- Creating a Module to contain your C++ source code
- Adding C++ to an Xcode project
- Creating a Swift Package
- Building with CMake

Creating a Module to contain your C++ source code

- Create a new C++ implementation and header file
- For this example we will call the files Cxx, so we should have a Cxx.cpp and Cxx.hpp.
- Next create an empty file and call it `module.modulemap` , in this file create the module for your source code, and define your C++ header (`requires cplusplus` isn't required but it's convention for C++ modules, especially if they use C++ features).

```
//In module.modulemap
module Cxx {
    //note that your header should be the file that containts your method
implementations
    header "Cxx.hpp"
    requires cplusplus
}
```

- Move the newly created files (Cxx.cpp, Cxx.hpp, module.modulemap) into a separate directory (this should remain in your project directory)



Adding C++ to an Xcode project

- In your xcode project, follow the steps Creating a Module to contain your C++ source code in your project directory

Add the C++ module to the include path and enable C++ interop:

- Navigate to your project directory
- In `Project` navigate to `Build Settings` -> `Swift Compiler`
- Under `Custom Flags` -> `Other Swift Flags` add `-Xfrontend -enable-cxx-interop`
- Under `Search Paths` -> `Import Paths` add your search path to the C++ module (i.e, `./ProjectName/Cxx`). Repeat this step in `Other Swift Flags`

```
//Add to Other Swift Flags and Import Paths respectively
-Xfrontend -enable-cxx-interop
```

```
-I./ProjectName/Cxx
```

- This should now allow your to import your C++ Module into any `.swift` file.

```
//In ContentView.swift
import SwiftUI
import Cxx

struct ContentView: View {
    var body: some View {
        Text("Cxx function result: \(cxxFunction(7))")
            .padding()
    }
}
```

```
//In Cxx.hpp

#ifndef Cxx_hpp
#define Cxx_hpp

int cxxFunction(int n) {
    return n;
}
#endif
```

```
//In Cxx.cpp

#include "Cxx.hpp"
int cxxFunction(int n);

```

Creating a Swift Package

After creating your Swift package project, follow the steps Creating a Module to contain your C++ source code in your `Source` directory

- In your Package Manifest, you need to configure the Swift target's dependencies and compiler flags
- In this example the name of the package is `CxxInterop`
- Swift code will be in `Sources/CxxInterop` called `main.swift`
- C++ source code follows the example shown in Creating a Module to contain your C++ source code
- Under targets, add the name of your C++ module and the directory containing the Swift code as a target.
- In the target defining your Swift target, add a `dependencies` to the C++ Module, the `path` , `source` , and `swiftSettings` with `unsafeFlags` with the source to the C++ Module, and enable `-enable-cxx-interop`

```
//In Package Manifest

import PackageDescription

let package = Package(
    name: "CxxInterop",
    platforms: [.macOS(.v12)],
    products: [
        .library(
            name: "Cxx",
            targets: ["Cxx"]),
        .library(
            name: "CxxInterop",
            targets: ["CxxInterop"]),
    ],
    targets: [
        .target(
            name: "Cxx",
            dependencies: []
        ),
        .executableTarget(
            name: "CxxInterop",
            dependencies: ["Cxx"],
            path: "./Sources/CxxInterop",
            sources: [ "main.swift" ],
            swiftSettings: [.unsafeFlags([
                "-I", "Sources/Cxx",
                "-Xfrontend", "-enable-cxx-interop",
            ])]
        ),
    ]
)
```

- We are now able to import our C++ Module into our swift code, and import the package into existing projects

```
//In main.swift

import Cxx

public struct CxxInterop {

    public func callCxxFunction(n: Int32) -> Int32 {
        return cxxFunction(n: n)
    }
}

print(CxxInterop().callCxxFunction(n: 7))
//outputs: 7
```

Building with CMake

After creating your project follow the steps Creating a Module to contain your C++ source code

- Create a `CMakeLists.txt` file and configure for your project
- In `add_library` invoke `cxx-support` with the path to the C++ implementation file

- Add the `target_include_directories` with `cxx-support` and path to the C++ Module `${CMAKE_SOURCE_DIR}/Sources/Cxx`
- Add the `add_executable` to the specific files/directory you would like to generate source, with `SHELL:-Xfrontend -enable-cxx-interop`.
- In the example below we will be following the file structure used in Creating a Swift Package

```
//In CMakeLists.txt

cmake_minimum_required(VERSION 3.18)

project(CxxInterop LANGUAGES CXX Swift)

set(CMAKE_CXX_STANDARD 11)
set(CMAKE_CXX_STANDARD_REQUIRED YES)
set(CMAKE_CXX_EXTENSIONS OFF)

add_library(cxx-support ./Sources/Cxx/Cxx.cpp)
target_compile_options(cxx-support PRIVATE
  -I${SWIFT_CXX_TOOLCHAIN}/usr/include/c++/v1
  -fno-exceptions
  -fignore-exceptions
  -nostdinc++)
target_include_directories(cxx-support PUBLIC
  ${CMAKE_SOURCE_DIR}/Sources/Cxx)

add_executable(CxxInterop ./Sources/CxxInterop/main.swift)
target_compile_options(CxxInterop PRIVATE
  "SHELL:-Xfrontend -enable-cxx-interop"
target_link_libraries(CxxInterop PRIVATE cxx-support)
```

```
//In main.swift

import Cxx

public struct CxxInterop {
    public static func main() {
        let result = cxxFunction(7)
        print(result)
    }
}

CxxInterop.main()
```

- In your project's directory, run `cmake` to generate the systems build files
- To generate an Xcode project run `cmake -GXcode`
- To generate with Ninja run `cmake -GNinja`
- For more information on `cmake` see the 'GettingStarted' documentation: (https://github.com/apple/swift/blob/main/docs/HowToGuides/GettingStarted.md)