Setting Up Mixed-Language Swift and C++ Project

Swift supports bidirectional interoperability with C++. This page describes how to set up a mixed-language Swift and C++ project using one of the supported IDEs or build systems. It also describes how other build systems can enable C++ interoperability by describing how to use C++ interoperability when invoking Swift compiler directly.

Mixing Swift and C++ Using Swift Package Manager

The Swift package manager allows Swift code to use C++ APIs in Swift.

> As of Swift 5.9, Swift package manager does not yet provide support for using Swift APIs in C++.

Enabling C++ Interoperability in a Package Target

A specific target in a Swift package must enable C++ interoperability in order to be able to import and use C++ APIs in Swift. The `interoperabilityMode` Swift build setting is used to enable C++ interoperability for a target. For example, the following package manifest shows how to enable C++ interoperability for a library target:

```
let package = Package(
    name: "LibraryThatUsesCxx",
    products: [
        .library(
            name: "libraryUsesCxx",
            targets: ["libraryUsesCxx"])
    ],
    targets: [
        .target(
            name: "libraryUsesCxx",
            swiftSettings: [.interoperabilityMode(.Cxx)])
    ]
)
```

Importing Headers from a C++ Package Target

Swift imports C++ headers using Clang modules. Swift package manager can generate a module map file automatically for a C++ target that contains an **umbrella** header. The generated module map file allows a Swift target that depends on such C++ target to import the C++ headers from such target.

The umbrella header used by the C++ target must:

- Use the name of the C++ target (with additional extension) as its file name.
- Be placed in the `include` directory in the target.

The umbrella header can then include other C++ headers in the project, which will then be imported into Swift.

For example, the following Swift package builds a Swift command line tool that uses a C++ library:

```
let package = Package(
    name: "CommandLineSwiftToolUsesCxx",
    products: [
        .library(
            name: "cxxLibrary",
            targets: ["cxxLibrary"]),
        .executable(
            name: "swiftCLITool",
            targets: ["swiftCLITool"])
    ],
    targets: [
        .target(
            name: "cxxLibrary")
        .executableTarget(
            name: "swiftCLITool",
            dependencies: ["cxxLibrary"],
            swiftSettings: [.interoperabilityMode(.Cxx)])
    ]
)
```

Swift package manager will automatically generate a module map for the C++ library in this package, as it can find an umbrella header in the sources:

```
Sources
├── swiftCLITool
└── cxxLibrary
    ├── include
    │   ├── cxxLibrary.h   [This is the umbrella header]
    │   └── classHeader.h
    ├── cxxLibrary.cpp
    └── classHeader.cpp
```

The umbrella header `cxxLibrary.h` contains some declarations and also includes the other headers in the C++ target:

```
// Header file `cxxLibrary.h`
#pragma once

#include <cxxLibrary/classHeader.h>
```

The Swift code in the `swiftCLITool` can import `cxxLibrary` directly:

```
import cxxLibrary
```

All of the supported C++ APIs declared in the `classHeader.h` header file will then be available in Swift.

Vending Packages That Enable C++ Interoperability

Enabling C++ interoperability for a Swift package manager target will need other targets that depend on such target to enable C++ interoperability as well.

Enabling C++ interoperability is a breaking change for an existing package, and so it must be done only in a new major semver version. Please bump up the package's major version when you enable C++ interoperability!

If you'd like to vend a package with a target that enables C++ interoperability, we recommend that you:

- Clearly communicate to clients that they have to enable C++ interoperability when depending on targets from such package.

Mixing Swift and C++ Using Other Build Systems

This section describes how to enable and use C++ interoperability when invoking the Swift compiler directly. This should allow other build systems to configure a mixed-language Swift and C++ project.

Enabling C++ Interoperability in the Swift Compiler

The `-cxx-interoperability-mode=` build flag is used to enable C++ interoperability in the Swift compiler. It receives the interoperability compatibility version as its value. The only supported value right now is `default`. The `default` value implies that the interoperability compatibility version used by Swift matches the Swift language version.

Importing a C++ Clang Module When Invoking Compiler Directly

The following build flag allows Swift to find the C++ headers:

- `-I <path>`: This flag tells Swift that it should look for imports in the directory specified by the given path. This path should contain a `module.modulemap` file when you want to import a C++ Clang module into Swift.

The `-Xcc` flag is used to pass additional C++ build settings to the C++ Clang compiler embedded in the Swift compiler. For example, you can use Clang's `-std=` flag to import C++ headers that require C++20 into Swift:

```
swiftc ... -Xcc -std=c++20 ...
```

Putting it all together, the following Swift compiler invocation lets you compile a Swift file that imports a Clang module whose module map file is located in the `include` directory:

```
swiftc main.swift -cxx-interoperability-mode=default -I include -o main
```