C++ Interoperability Status

Swift has some experimental ability to interoperate with C++. This document provides an overview of the status of the Swift and C++ interoperability support.

C++ to Swift Interoperability Status

Swift has the experimental ability to import a large subset of C++. This section of the document describes which C++ language and standard library features can be imported and used from Swift in an experimental manner.

Example

The following example demonstrates several interop features. It compiles and runs on main.

```
// cxx-types.h (mapped to CxxTypes module in module.modulemap)
#include <algorithm>
#include <vector>

using V = std::vector<long>;
```

```
// main.swift
import CxxTypes
import std.vector
import std.algorithm

// We can extend C++ types in Swift.
extension V : RandomAccessCollection {
  public var startIndex: Int { 0 }
  public var endIndex: Int { size() }
}

// Create a vector with some data.
var numbers = V(4)
std.fill(numbers.beginMutating(), numbers.endMutating(), 41)

// Transform it using C++.
std.transform(numbers.beginMutating(), numbers.endMutating(),
              numbers.beginMutating()) { (element: Int) in
  return element + 1
}

// Loop over it in Swift.
for (index, element) in numbers.enumerated() {
  print("v[\(index)] = \(element)")
}
```

```
// We can also use anything in RandomAccessCollection, such as map and zip.
let strings = numbers.map { "\($0)" }
for (s, n) in zip(strings, numbers) {
  print("\(s) = \(n)")
}
```

Importing C++

There are currently two experimental ways to import C++ into Swift: - **Clang modules**: can be imported into Swift. This requires a module map. - **Bridging header**: can be imported into Swift. Headers included in the bridging header will be imported. Please note that support for importing C++ 20 modules isn't implemented.

Both CMake and the Swift package manager can be configured to invoke Swift with the correct arguments to import C++ headers.

Note: C++ code is imported using the Objective-C++ language mode on Apple platforms.

Experimental C++ Language Support

This status table describes which of the following C++ language features can be used in Swift:

C++ Language Feature	Implemented Experimental Support For Using It In Swift
Top-level functions	Yes
Enumerations	Yes. That includes `enum class`
Struct / Class types	Yes - as value types, except for types without a copy constructor. Partial experimental support for importing a C++ struct/class as a reference type
Typedefs / Type aliases	Yes
Global Variables	Yes
Namespaces	Yes
Inline Namespaces	Yes, with some known issues (https://bugs.swift.org/browse/SR-15956)
Exceptions	No
Fields	Yes
Member functions	Yes. Some value category overloads aren't imported
Virtual Member Functions	No
Operators	Yes, with some known issues
Subscript Operators	Yes
Constructors	Yes. That includes implicit constructors

C++ Language Feature	Implemented Experimental Support For Using It In Swift
Destructor	Yes. C++ destructors are invoked automatically when the value is no longer used in Swift
Copy constructor / copy assignment operator	Yes. Swift invokes the underlying copy constructor when copying a C++ value
Move constructor / move assignment operator	No
Base class member functions / operators	Yes, with some known issues
Function templates	Yes
Class templates	Yes
Dependent types	Partially: imported as Any
Availability Attributes	Yes

The following C++ code patterns or language features have specific mappings to Swift language features when imported in Swift:

C++ Language Feature	Imported Into Swift
`get`/`set` member functions	Imported as computed property (starting from Swift-5.7)
`const`/non-`const` member function overload set	Both overloads are imported as a method, with non-`const` method being renamed to `mutating...` (starting from Swift-5.7). The renaming logic will change in a future version of Swift, and non-`const` methods won't be renamed

Unless stated otherwise (i.e., imported reference types) all Swift features work with imported types. For example: use in generic contexts, protocol conformance, extensions, etc.

C++ Standard Library Support

Parts of libc++ can be imported and used from Swift. C++ standard library types are bridged directly to Swift, and there is not automatic bridging to native Swift types. This means that if an imported C++ API returns `std::string`, you will get a `std::string` value in Swift as well, and not Swift's `String`.

This status table describes which of the following C++ standard library features have some experimental support for using them in Swift. Please note that this is not a comprehensive list and other libc++ APIs that use the above supported C++ language features could be imported into Swift.

C++ Standard Library Feature	Can Be Used From Swift
`std::string`	Yes
`std::vector`	Yes

Known Issues

Inline Namespaces

- https://bugs.swift.org/browse/SR-15956: Swift's typechecker currently doesn't allow calling a function from an inline namespace when it's referenced through the parent namespace. Example of a test that fails: https://github.com/apple/swift/blob/main/test/Interop/Cxx/namespace/inline-namespace-function-call-broken.swift

Swift to C++ Interoperability Status

This section of the document describes which Swift language and standard library features can be imported and used from C++.

Importing Swift

Swift has some experimental support for generating a header that can be imported by C++.

Swift Language Support

This status table describes which of the following Swift language features have some experimental support for using them in C++.

Functions

Swift Language Feature	Implemented Experimental Support For Using It In C++
Top-level `@_cdecl` functions	Yes
Top-level Swift functions	Partially, only with C compatible types
`inout` parameters	No
Variadic parameters	No
Multiple return values	No