Supported Features and Limitations of C++ Interoperability

Swift supports bidirectional interoperability with C++. This page describes which C++ interoperability features are supported in the upcoming Swift 5.9 release. It also talks about the limitations in the current support for C++ interoperability. Additionally, it lists the set of known issues that are related to C++ interoperability support.

C++ interoperability is an actively evolving feature of Swift. Certain aspects of its design and functionality might change in future releases of Swift, as the Swift community gathers feedback from real-world adoption of C++ interoperability in mixed Swift and C++ codebases. This page is going to be updated whenever a new release of Swift changes the supported set of C++ interoperability features.

Platform Support

C++ interoperability is supported for development and deployment on all platforms that Swift supports.

Compiler Support

C++ interoperability is supported in Swift 5.9 and above.

Swift's support for bidirectional interoperability relies on a header generated by the Swift compiler that can then be included by C++ code that wants to use Swift APIs. This header uses Swift-specific compiler extensions that are supported only by the following C++ compilers:

- Clang (starting with LLVM 11 and above)
- Xcode's Apple Clang

C++ code built with other compilers cannot call Swift functions or use Swift types from C++.

C++ Standard Library Support

Swift compiler uses the platform's default C++ standard library when interoperating with C++. This table shows which C++ standard library is used when building Swift code for a specific deployment platform:

Platform running Swift application	Default C++ Standard Library
macOS, iOS, watchOS, tvOS	libc++
Ubuntu, CentOS, Amazon Linux	libstdc++
Windows	Microsoft C++ Standard Library (msvcprt)

Swift does not currently support selecting an alternative standard library for platforms that support alternative standard libraries. For example, you can't use libc++ when building Swift code for Ubuntu, even though libc++ can be used when building C++ code for Ubuntu.

Mixed Swift and C++ code must use the same C++ standard library.

Supported C++ APIs

This section describes which C++ APIs are supported in Swift.

C++ Functions Supported in Swift

Swift supports calling most non-templated:

- Top-level functions.
- Functions inside of namespaces.
- Member functions, both instance and static.
- Constructors.

Functions and constructors that use r-value reference types are not yet available in Swift.

Virtual member functions are not yet available in Swift.

Swift supports calling some C++ function templates. Any function or function template that uses a dependent type in its signature, or a universal reference (`T &&`) is not available in Swift. Any function template with non-type template parameters is not available in Swift. Variadic function templates are not available in Swift.

A C++ function whose return type is not supported in Swift, or with a parameter whose type is not supported in Swift is not available in Swift.

C++ Types Supported in Swift

The following C++ types can be used in Swift:

- Primitive types, like `int` and `bool`.
- Pointers.
- C++ references (including r-value reference / universal reference parameters).
- Type aliases, only when the underlying type is supported in Swift.
- Copyable structures and classes.
  - Swift 5.9 does not support C++ structures and classes that have a deleted copy constructor, including move-only structures and classes.
    - The only exception are the non-copyable C++ structures and classes that have been explicitly mapped to Swift reference types.
- Enumerations. That includes scoped enumerations (`enum class`).

C++ types that become value types in Swift can be constructed and passed around by value.

C++ types that become reference types can't be constructed directly by Swift code. They can be passed around freely between Swift and C++.

C++ types defined inside of a C++ `namespace` are available in Swift.

Class and structure templates are not directly available in Swift. The instantiated specializations of a class or structure template are available in Swift.

Public data members of a C++ structure or class are available in Swift when the type of such data member is supported in Swift.

C++ Standard Library Types Supported in Swift

The following C++ standard library types are supported in Swift:

- `std::string`, `std::u16string`.
- Specializations of `std::pair`.
- Specializations of `std::vector`.
- Specializations of `std::map` and `std::unordered_map`.
- Specializations of `std::set` and `std::unordered_set`.
- Specializations of `std::optional`.
- Specializations of `std::shared_ptr`.
- Specializations of `std::array`.

Specializations of `std::unique_ptr`, `std::function` and `std::variant` are not yet supported in Swift.

Other C++ Features Handled by Swift

C++ Exceptions

Swift can interoperate with C++ code that throws exceptions. However, Swift does not support catching C++ exceptions. Instead, the running program terminates with a fatal error when a C++ exception that's not caught by C++ code reaches Swift code.

Swift's strict program termination enforcement for any uncaught exceptions is not supported when running Swift code built with Swift 5.9 on Windows. Any mixed language program running on Windows should always terminate when a C++ exception propagates through Swift code as the program's stack is unwound. Any attempt to recover from such uncaught exception can lead to undefined behavior in your program.

Clang's Availability Attributes

C++ APIs annotated with Clang's availability attributes receive the same availability annotation in Swift.

Supported Swift APIs

This section describes which Swift APIs get exposed to C++ in the generated header.

Swift Structures Supported by C++

Swift can generate C++ representation for most top-level Swift structures. The following Swift structures are not yet supported:

- Zero-sized structures that don't have any stored properties.
- Non-copyable structures.
- Generic structures with generic constraints, or with more than 3 generic parameters, or that have variadic generics.

Swift currently does not expose nested structures to C++.

Swift Classes and Actors Supported by C++

Swift can generate C++ representation for most top-level Swift classes and actors. The following Swift classes are not yet supported:

- Generic classes and actors.

Swift currently does not expose nested classes and actors to C++.

Swift Enumerations Supported by C++

Swift can generate C++ representation for most top-level Swift enumerations that do not have associated values, and some top-level Swift enumerations that have associated values. The following Swift enumerations are not yet supported:

- Non-copyable enumerations.
- Generic enumerations with generic constraints, or with more than 3 generic parameters, or that have variadic generics.
- Enumerations that have an enumeration case with more than one associated value.
- Indirect enumerations.

Additionally, the types of all the associated values of an enumeration must be representable in C++. The exact set of representable types is described below, in the section that describes the representable parameter or return types.

Swift currently does not expose nested enumerations to C++.

Swift Functions and Properties Supported by C++

Any function, property, or initializer is exposed to C++ only when Swift can represent all of its parameter and return types in C++. A parameter or return type can be represented in C++ only when:

- it is a Swift structure / class / enumeration that is defined in the same Swift module.
- or, it is a C++ structure, class or enumeration.
- or, it is one of the supported Swift standard library types.
  - if it's a generic type, like `Array`, its generic parameters must be bound to one of the types listed here.
- or, it is an `UnsafePointer`/`UnsafeMutablePointer`/`Optional<UnsafePointer>`/`Optional<UnsafeMutablePointer>` that points to any type from the supported three type categories listed above.

Functions or initializers that have a parameter type or a return type that's not listed above can not be represented in C++ yet. Properties of type that's not listed above can not be represented in C++ yet.

Additionally, the following Swift functions, properties and initializers can not yet be represented in C++:

- Asynchronous functions / properties.
- Functions / properties / initializers that `throw`.
- Generic functions / properties / initializers with generic constraints or variadic generics.
- Functions that return an opaque type.
- Functions / properties / initializers with the `@_alwaysEmitIntoClient` attribute.

Supported Swift Standard Library Types

Swift is able to represent the following Swift standard library types in C++:

- Primitive types, such as `Bool`, `Int`, `Float` and their C variants like `CInt`.
  - The full list of supported primitive types is provided below.
- Pointer types, like `OpaquePointer`, `UnsafeRawPointer`, `UnsafeMutableRawPointer`, `UnsafeBufferPointer` and `UnsafeMutableBufferPointer`.
- `String` type.
- `Array` type.
- `Optional` type.

List Of Primitive Swift Types Supported by C++

This table lists the primitive Swift types defined in Swift's standard library that can be represented in C++:

Swift Type	Corresponding C++ type
`Bool`	`bool`
`Int`	`swift::Int`
`UInt`	`swift::UInt`
`Int8`	`int8_t`
`Int16`	`int16_t`
`Int32`	`int32_t`
`Int64`	`int64_t`
`UInt8`	`uint8_t`
`UInt16`	`uint16_t`
`UInt32`	`uint32_t`
`UInt64`	`uint64_t`
`Float`	`float`
`Double`	`double`
`Float32`	`float`
`Float64`	`double`
`CBool`	`bool`
`CChar`	`char`
`CWideChar`	`wchar_t`
`CChar16`	`char16_t`
`CChar32`	`char32_t`
`CSignedChar`	`signed char`
`CShort`	`short`
`CInt`	`int`
`CLong`	`long`
`CLongLong`	`long long`
`CUnsignedChar`	`unsigned char`
`CUnsignedShort`	`unsigned short`
`CUnsignedInt`	`unsigned int`
`CUnsignedLong`	`unsigned long`
`CUnsignedLongLong`	`unsigned long long`
`CFloat`	`float`
`CDouble`	`double`

Known Issues

Swift 5.9 has some known issues and limitations related to C++ interoperability support. All of the known issues are listed on github.

Known Swift Package Manager Issues

A Swift target that enables C++ interoperability in Swift package manager requires its dependencies to enable C++ interoperability as well. The following issue tracks the status of this limitation:

- Swift should provide support for internal imports and resilience for all platforms (that can be enabled in SwiftPM) to allow Swift modules to depend on C++ modules without requiring that the clients enable C++ interoperability.

The other known Swift package manager issues are listed here:

- The C++ language standard that's specified in the package manifest is not passed to the Swift compiler when C++ interoperability is enabled for Swift code

Known Performance Issues and Limitations

Swift's current support for C++ container types does not provide explicit performance guarantees. Most notably, Swift can make a deep copy of a collection when it's used in a `for-in` loop in Swift.

The following issue tracks the status of this performance limitation:

- Swift should provide language affordances that make it possible to avoid copying a C++ container when traversing through it in a for-in loop, or when using collection methods like map and filter