Choosing C++ or C# for native code

Edit

React Native for Windows supports writing native code in both C++ and C#, but there are trade-offs with each language. The choice of language can impact the compatibility, developer experience, and performance of your project. So whether you're building an app or native module, you should choose the native language that best meets your requirements.

> **Note**: In this document, C++ refers specifically to C++/WinRT.

Common Considerations

Compatibility Matrix

While the React Native for Windows toolchain does support app and module projects written in both languages, there are limitations when mixing such projects.

The following table lists several compatibility scenarios, listing whether apps written in a certain language can consume modules written in a certain language.

- ✅ means the scenario works
- 🟨 means the scenario partially works (i.e. with specific limitations)
- 🟥 means the scenario does not work

~~C++ Module w/ C++ NuGet Dependency~~	✅	✅
C++ Module w/ C# NuGet Dependency	🟥	🟥
C# Module	🟨	✅
C# Module w/ C# NuGet Dependency	🟥	✅
C# Module w/ C++ NuGet Dependency	🟥	✅

So if you're writing an app, you'll need to ask yourself if you need to consume any native modules. If you are, you'll want to double-check the languages of those modules when deciding the language for your app.

Now, if you're writing a native module, you'll need to ask yourself if you need to consume packages from NuGet. If you are, you'll want to double-check the languages of those packages when deciding the language for your module.

Finally, if you have any existing native (source) code you want reuse, you'll want to keep that in mind as well.

> **Note:** We are actively working to fix the bugs blocking these scenarios and improve the mixing of C# and C++ projects. For more details on the actual bugs blocking compatibility, see issue #6819.

Developer Experience

C# development brings engineering efficiencies to writing native code for Windows, including:

- An extensive standard library
- A rich ecosystem of tools, libraries, and code examples
- A variety of syntax options for modern coding conventions

- Access to the broader ecosystem of portable C++ code
- A rich ecosystem of tools
- Full control of memory management

If you're an existing C++ developer, or planning on using existing C++ code, you may find that developing in C++/WinRT to be the most productive.

Performance

Generally, code written in C++ can run faster and use less memory than code written in C#. This is because with C++ you have more control over your memory allocations, and C++ compiles directly into native machine code for the target platform (X86, X64, ARM, etc). With C#, garbage collection can cause performance drops, and while C# does eventually get compiled into machine code (we'll get to this in a second) it first compiles into intermediate language, or IL.

In the past, C# apps relied on just-in-time (JIT) compilation of their IL into machine code at runtime. This means the IL is delivered to end user machines and this JIT process does not occur until the user runs the app, which can negatively impact the app's performance and startup time.

However, this is not the case for UWP apps, which instead default to using the .NET Native toolchain. The .NET Native toolchain (enabled when building a "Release" configuration of your app) will cross-compile your IL code into native machine code at build time. As such, when you submit your app to the Microsoft store, only the finished, "native-ized" code will be delivered to end user machines. This improves the performance and startup times of these apps versus their non-"native-ized" counterparts. For more information about performance in .NET Native apps, see Measuring startup improvement with .NET Native.

In the end, only proper performance profiling and testing of your actual code can determine whether the performance differences of the two languages are significant and/or noticeable

Parity Note

Ultimately, while we strive to maintain parity between developing with C++ and C#, each has their trade-offs, and neither is best for all scenarios. The core and vast majority of React Native for Windows is itself written in C++/WinRT, in order to both maximize performance and facilitate the sharing of C++ code with other projects on other platforms, including standard React Native and React Native for macOS.

For similar reasons, Microsoft developers prefer contributing to community modules in C++ rather than C#.

However, if your app or module already uses C#, you should feel empowered to continue to use C#.

Additional Resources

Getting started with UWP for iOS Developers: Choosing a programming language

‹ Working with native code on Windows

Marshaling Data ›

Components and APIs

More Resources

Get Started with macOS

React Native Windows Components and APIs

Native Modules

Native UI Components

GitHub

Samples