ReadyToRun Compilation

Article • 01/27/2022 • 4 minutes to read • 4 contributors

.NET application startup time and latency can be improved by compiling your application assemblies as ReadyToRun (R2R) format. R2R is a form of ahead-of-time (AOT) compilation.

R2R binaries improve startup performance by reducing the amount of work the just-in-time (JIT) compiler needs to do as your application loads. The binaries contain similar native code compared to what the JIT would produce. However, R2R binaries are larger because they contain both intermediate language (IL) code, which is still needed for some scenarios, and the native version of the same code. R2R is only available when you publish an app that targets specific runtime environments (RID) such as Linux x64 or Windows x64.

To compile your project as ReadyToRun, the application must be published with the PublishReadyToRun property set to `true`.

There are two ways to publish your app as ReadyToRun:

1. Specify the PublishReadyToRun flag directly to the dotnet publish command. See dotnet publish for details.

   .NET CLI — Copy

   ```
   dotnet publish -c Release -r win-x64 -p:PublishReadyToRun=true
   ```

2. Specify the property in the project.

   - Add the `<PublishReadyToRun>` setting to your project.

     XML — Copy

     ```
     <PropertyGroup>
       <PublishReadyToRun>true</PublishReadyToRun>
     </PropertyGroup>
     ```

   - Publish the application without any special parameters.

     .NET CLI — Copy

     ```
     dotnet publish -c Release -r win-x64
     ```

Impact of using the ReadyToRun feature

Ahead-of-time compilation has complex performance impact on application performance, which can be difficult to predict. In general, the size of an assembly will grow to between two to three times larger. This increase in the physical size of the file may reduce the performance of loading the assembly from disk, and increase working set of the process. However, in return the number of methods compiled at run time is typically reduced substantially. The result is that most applications that have large amounts of code receive large performance benefits from enabling ReadyToRun. Applications that have small amounts of code will likely not experience a significant improvement from enabling ReadyToRun, as the .NET runtime libraries have already been precompiled with ReadyToRun.

The startup improvement discussed here applies not only to application startup, but also to the first use of any code in the application. For instance, ReadyToRun can be used to reduce the response latency of the first use of Web API in an ASP.NET application.

Interaction with tiered compilation

Ahead-of-time generated code is not as highly optimized as code produced by the JIT. To address this issue, tiered compilation will replace commonly used ReadyToRun methods with JIT-generated methods.

How is the set of precompiled assemblies chosen?

The SDK will precompile the assemblies that are distributed with the application. For self-contained applications, this set of assemblies will include the framework. C++/CLI binaries are not eligible for ReadyToRun compilation.

To exclude specific assemblies from ReadyToRun processing, use the `<PublishReadyToRunExclude>` list.

XML — Copy

```
<ItemGroup>
  <PublishReadyToRunExclude Include="Contoso.Example.dll" />
</ItemGroup>
```

How is the set of methods to precompile chosen?

The compiler will attempt to pre-compile as many methods as it can. However, for various reasons, it's not expected that using the ReadyToRun feature will prevent the JIT from executing. Such reasons may include, but are not limited to:

- Use of generic types defined in separate assemblies.
- Interop with native code.
- Use of hardware intrinsics that the compiler cannot prove are safe to use on a target machine.
- Certain unusual IL patterns.
- Dynamic method creation via reflection or LINQ.

Symbol generation for use with profilers

When compiling an application with ReadyToRun, profilers may require symbols for examining the generated ReadyToRun files. To enable symbol generation, specify the `<PublishReadyToRunEmitSymbols>` property.

XML — Copy

```
<PropertyGroup>
  <PublishReadyToRunEmitSymbols>true</PublishReadyToRunEmitSymbols>
</PropertyGroup>
```

These symbols will be placed in the publish directory and for Windows will have a file extension of .ni.pdb, and for Linux will have a file extension of .r2rmap. These files are not generally redistributed to end customers, but instead would typically be stored in a symbol server. In general these symbols are useful for debugging performance issues related to startup of applications, as Tiered Compilation will replace the ReadyToRun generated code with dynamically generated code. However, if attempting to profile an application that disables Tiered Compilation the symbols will be useful.

Composite ReadyToRun

Normal ReadyToRun compilation produces binaries that can be serviced and manipulated individually. Starting in .NET 6, support for Composite ReadyToRun compilation has been added. Composite ReadyToRun compiles a set of assemblies that must be distributed together. This has the advantage that the compiler is able to perform better optimizations and reduces the set of methods that cannot be compiled via the ReadyToRun process. However, as a tradeoff, compilation speed is significantly decreased, and the overall file size of the application is significantly increased. Due to these tradeoffs, use of Composite ReadyToRun is only recommended for applications that disable Tiered Compilation or applications running on Linux that are seeking the best startup time with self-contained deployment. To enable composite ReadyToRun compilation, specify the `<PublishReadyToRunComposite>` property.

XML — Copy

```
<PropertyGroup>
  <PublishReadyToRunComposite>true</PublishReadyToRunComposite>
</PropertyGroup>
```

> **Note**
>
> In .NET 6, Composite ReadyToRun is only supported for self-contained deployment.

Cross platform/architecture restrictions

For some SDK platforms, the ReadyToRun compiler is capable of cross-compiling for other target platforms.

Supported compilation targets are described in the table below when targeting .NET 6 and later versions.

SDK platform	Supported target platforms
Windows X64	Windows (X86, X64, ARM64), Linux (X64, ARM32, ARM64), macOS (X64, ARM64)
Windows X86	Windows (X86), Linux (ARM32)
Linux X64	Linux (X64, ARM32, ARM64), macOS (X64, ARM64)
Linux ARM32	Linux ARM32
Linux ARM64	Linux (X64, ARM32, ARM64), macOS (X64, ARM64)
macOS X64	Linux (X64, ARM32, ARM64), macOS (X64, ARM64)
macOS ARM64	Linux (X64, ARM32, ARM64), macOS (X64, ARM64)

Supported compilation targets are described in the table below when targeting .NET 5 and below.

SDK platform	Supported target platforms
Windows X64	Windows X86, Windows X64, Windows ARM64
Windows X86	Windows X86, Windows ARM32
Linux X64	Linux X86, Linux X64, Linux ARM32, Linux ARM64
Linux ARM32	Linux ARM32
Linux ARM64	Linux ARM64
macOS X64	macOS X64

Recommended content

Trim self-contained applications - .NET
Learn how to trim self-contained apps to reduce their size. .NET Core bundles the runtime with an app that is published self-contained and generally includes more of the runtime then is necessary.

.NET Runtime config options - .NET
Learn how to configure the .NET runtime using configuration settings.

Single file application - .NET
Learn what single file application is and why you should consider using this application deployment model.

Runtime roll forward for .NET Core self-contained app deployments. - .NET
Learn about dotnet publish changes for self-contained deployments.

Show more

Feedback

Submit and view feedback for

This product | This page

View all page feedback