Revit .NET 8 Upgrade Tips

Created by Michael Morris, last modified by Mariah Hanning less than a minute ago

Migrating From .NET 4.8 to .NET 8

Revit 2025 is built on .NET 8 for the January 2024 preview release. In this preview release, the Revit API is .NET 8 only, and Revit add-ins need to be recompiled for .NET 8.

The move from .NET 4.8 is to .NET 8 is a relatively large jump. .NET 8 comes from the .NET Core lineage, which has significant differences from .NET 4.8.

Upgrade Process

There are many Microsoft documents and tools to help application developers migrate from .NET 4.8 to .NET Core/5/6/7/8. Following is a list of some helpful documents:

- Overview of porting from .NET Framework to .NET document: https://learn.microsoft.com/en-us/dotnet/core/porting/
- .NET Upgrade Assistant can help with the project migration: https://learn.microsoft.com/en-us/dotnet/core/porting/upgrade-assistant-overview
- The .NET Portability Analyzer - .NET on C# projects to roughly evaluate how much work is required to make the migration as well as dependencies between the assemblies.
- Lists of breaking changes for .NET Core and .NET 5+: https://learn.microsoft.com/en-us/dotnet/core/compatibility/breaking-changes
- The .NET 8 SDK can be installed from here: https://dotnet.microsoft.com/en-us/download/visual-studio-sdks
  - .NET SDK 8.0.100 is used to build the Revit January 2024 preview release.
  - The Revit preview release will install .NET 8 Windows Desktop Runtime x64 8.0.0.33101.
- If you use Visual Studio to build .NET 8 code, you'll need Visual Studio 17.8 or later.

Basic upgrade process for projects

For C# projects (CSPROJ)

1. Convert C# projects to the new SDK-style format: https://learn.microsoft.com/en-us/dotnet/core/project-sdk/overview
   a. The .NET Upgrade Assistant can help with the project migration: https://learn.microsoft.com/en-us/dotnet/core/porting/upgrade-assistant-overview
   b. Convert packages.json into PackageReferences in your CSPROJ. https://learn.microsoft.com/en-us/nuget/consume-packages/migrate-packages-config-to-package-reference
2. Update the target framework for your projects from `<TargetFrameworkVersion>` to `<TargetFramework>net8.0-windows</TargetFramework>`
   a. You can run the .NET Portability Analyzer on C# projects to evaluate how much work is required to make the migration.
   b. The .NET Upgrade Assistant can help with the .NET version migration: https://learn.microsoft.com/en-us/dotnet/core/porting/upgrade-assistant-overview
   c. If your application is a WPF application, then the CSPROJ will need `<TargetFramework>net8.0-windows</TargetFramework>` and `<UseWPF>true</UseWPF>`.
   d. If your application uses Windows forms, then use `<TargetFramework>net8.0-windows</TargetFramework>` and `<UseWindowsForms>true</UseWindowsForms>`.
3. System references can be removed from the CSPROJ, as they are available by default.
4. Then address incompatible packages, library references and obsolete (unsupported) code.

For C++/CLI projects (VCXPROJ)

Refer to Microsoft's guide for migrating C++/CLI projects to .NET Core/5+: https://learn.microsoft.com/en-us/dotnet/core/porting/cpp-cli

1. Replace `<CLRSupport>true</CLRSupport>` with `<CLRSupport>NetCore</CLRSupport>`. This property is often in configuration-specific property groups, so you may need to replace it in multiple places.
2. Replace `<TargetFrameworkVersion>` property with `<TargetFramework>net8.0-windows</TargetFramework>`.
3. Remove any .NET Framework references (like `<Reference Include="System" />`) and add `FrameworkReference` when needed. .NET Core SDK assemblies are automatically referenced when using `NetCore` support.
4. Add FrameworkReferences:
   a. To use Windows Forms APIs, add this reference to the vcxproj file:
   `<FrameworkReference Include="Microsoft.WindowsDesktop.App.WindowsForms" />`
   b. To use WPF APIs, add this reference to the vcxproj file:

`<FrameworkReference Include="Microsoft.WindowsDesktop.App.WPF" />`

c. To use both Windows Forms and WPF APIs, add this reference to the vcxproj file:
`<FrameworkReference Include="Microsoft.WindowsDesktop.App" />`

5. Remove any `<CompileAsManaged>` for cpp files. It will be set as NetCore by default. Any other values may cause issues.
6. Then address incompatible packages, library references and obsolete (unsupported) code.

Global.json

You may need to set `net8.0-windows` as the target in your global.json, if you have one. Refer the link for global.json overview: https://learn.microsoft.com/en-us/dotnet/core/tools/global-json

Component Versions

Your add-in may avoid instability by matching the version of these key components used by the Revit preview release:

- CefSharp
  - "cef.redist.x64" Version="119.4.3"
  - "cef.redist.x86" Version="119.4.3"
  - "CefSharp.Wpf.HwndHost" Version="119.4.30"
  - "CefSharp.Common.NetCore" Version="119.4.30"
  - "CefSharp.Wpf.NetCore" Version="119.1.20"
- Newtonsoft Json
  - "Newtonsoft.Json" Version="13.0.1"

Supporting Multiple Revit Releases

A single code base can support older Revit releases on .NET 4.8 as well as Revit 2025 on .NET 8. See this discussion on the Autodesk forums for ideas on configuring your projects and code to support multi-targeting.

Common Issues

Here are some common issues you may encounter when upgrading to .NET 8:

Build Warning MSB3277

When building code that references RevitAPI or RevitUIAPI, you will see the build warning MSB3277. To fix this, add a reference to the Windows Desktop framework: `<FrameworkReference Include="Microsoft.WindowsDesktop.App"/>`

Build Error CA1416

If your application uses functions that are only available on Windows systems, you may see a CA1416 error. This can be fixed for the project by adding `[assembly: System.Runtime.Versioning.SupportedOSPlatformAttribute("windows")]` to `AssemblyInfo.cs`.

Obsolete Classes and Functions with .NET 8

Your .NET 4.8 application may see compile time or runtime errors if it uses classes or functions that are obsolete or deprecated in .NET Core/5/6/7/8.

Lists of breaking changes for .NET Core/5/6/7/8 are here: https://learn.microsoft.com/en-us/dotnet/core/compatibility/breaking-changes

- BinaryFormatter and SOAPFormatter are obsolete.
  - Resource files that contain images or bitmaps will need to be updated as BinaryFormatter will not be available in .NET 8 to interpret those images/bitmaps.
  - Windows Forms dialogs using ImageList may need to be updated as BinaryFormatter loads the images for the ImageList.
- System.Threading.Thread.Abort is obsolete.
- System.Reflection.AssemblyName.CodeBase and System.Reflection.Assembly.CodeBase are deprecated.
- Delegate.BeginInvoke is not supported.
- Debug.Assert failure or Debug.Fail silently exits the application by default.

Assembly Loading

Your .NET 4.8 application may need updates to help it find and load assemblies:

- .NET 8 uses a different assembly probing approach for DLL loading. When in doubt, try putting the DLL to be loaded in the build output root

directory.
- .NET 8 assembly loading details are different than .NET 4.8.
- .NET 8 projects need runtimeconfig.json files for many DLLs. The **`runtimeconfig.json`** needs to be installed next to the matching DLL, and it configures the behavior of that DLL. These files can be created with **`<GenerateRuntimeConfigurationFiles>true</GenerateRuntimeConfigurationFiles>`**
- .NET 8 projects will create **`deps.json`** files for many DLLs. These **`deps.json`** files can be deleted if dependencies are placed in the same directory as the application. These files can be deleted with **`<GenerateDependencyFile>false</GenerateDependencyFile>`**

Assembly Properties

After updating your application to .NET 8, you may see build errors for your assembly properties. Many assembly properties are now auto-generated and can be removed from AssemblyInfo.cs.

Double Numbers To String

If you have unit tests or integration tests that compare doubles as strings, they may fail when you upgrade to .NET 8. This is because the number of decimal places printed by **`ToString()`** for doubles is different in .NET 4.8 and .NET 8. You can call **`ToString("G15")`** when converting doubles to strings to use the old .NET 4.8 formatting.

String.Compare

String.Compare behavior has changed, see .NET globalization and ICU and Use Globalization and ICU.

Windows Dialogs May Change Appearance

Your dialogs may change appearance with .NET 8.

- WinForms dialogs experience UI layout changes. The workaround is to set `Scale(new SizeF(1.0F, 1.0F));` in the dialog constructor.
- The dialog default font changed from "Microsoft Sans Serif 8 pt" to "Segoe UI". This can change dialog appearance and spacing.

Process.Start() May Fail

If your application is having trouble starting new processes, this may be because System.Diagnostics.Process.Start(url) has a behavior change. The ProcessStartInfo.UseShellExecute Property defaults to **`true`** in .NET 4.8 and **`false`** in .NET 8. Set **`UseShellExecute=true`** to workaround this change.

Encoding.Default Behaves Differently in .NET 8

If your application is having problems getting the text encoding used by Windows, it may be because **`Encoding.Default`** behaves differently in .NET 8. In .NET 4.8 **`Encoding.Default`** would get the system's active code page, but in .NET Core/5/6/7/8 Encoding.Default is always UTF8.

Items Order Differently in Sorted Lists

If you see different orderings of items in sorted lists after updating to .NET 8, this may be because List<T>.Sort() behaves differently in .NET 8 than .NET 4.8. The change fixes a .NET 4.8 bug which affected **`Sort()`** of items of equal value.

System.ServiceModel

System.ServiceModel has been ported to .NET Core through CoreWCF, which is now available through Nuget packages. There are various changes, including **`<System.ServiceModel>`** not being supported in configuration files.

C# Language Updates

If you are building code from .NET 4.8 in .NET 8, you may see build errors or warnings about C# nullable types.

C# has introduced nullable value types and nullable reference types. Prior to .NET 6, new projects used the default **`<Nullable>disable</Nullable>`**. Beginning with .NET 6, new projects include the **`<Nullable>enable</Nullable>`** element in the project file.

You can set **`<Nullable>disable</Nullable>`** if you want to revert to .NET 4.8 behavior.

Environmental Variables

If you use managed .NET to run native C++ code, be aware that environmental variables, including the path variable for DLL loading, are not shared from managed .NET code with native C++ code.

Additional Information for Autodesk Employees:

Autodesk Internal Revit .NET 8 Upgrade Tips

No labels

Autodesk Internal Revit .NET 8 Upgrade Tips