.Net Framework based Inventor projects upgrade to .Net 8

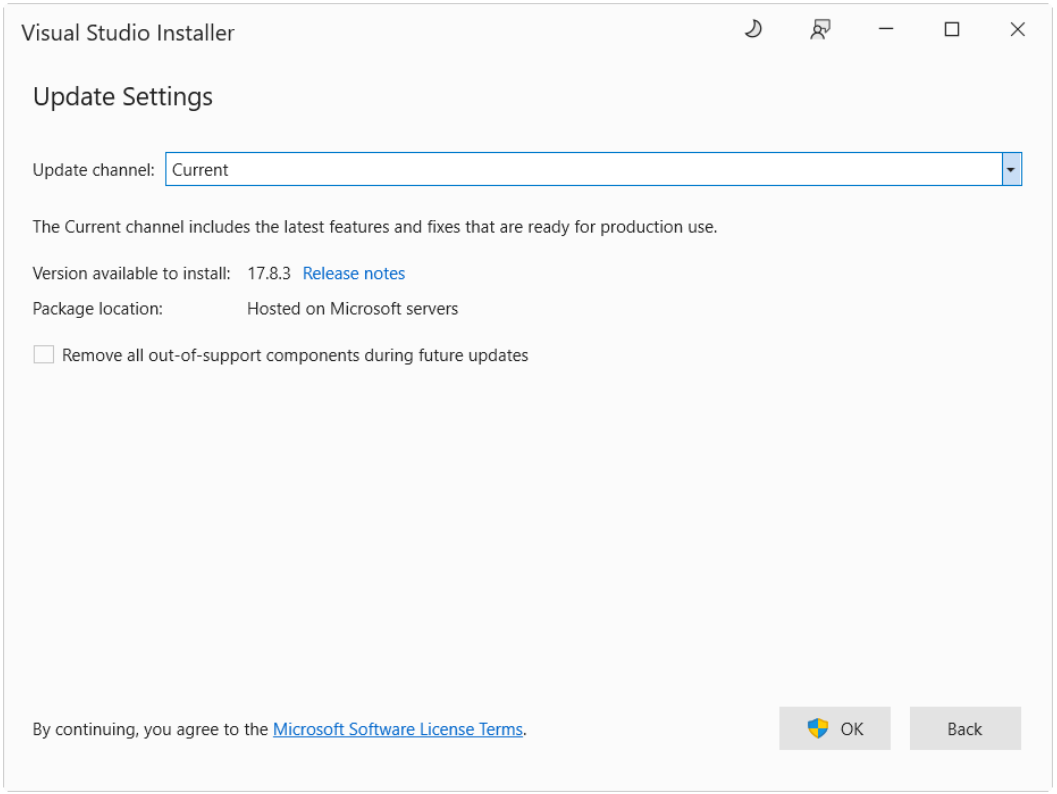
January 5, 2024

1. Check if a .Net Framework-based project needs to migrate to .Net 8 or not:

- Functional testing: Change project reference to Autodesk.Inventor.interop.dll in Inventor 2025 (C:\Program Files\Autodesk\Inventor 2025\Bin), and re-compile project and load your addin or launch your executable and test every function with Inventor 2025 Beta build (https://feedback.autodesk.com/). If the newly compiled DLL or Exe still can't work well with Inventor 2025 you need to consider migrating the project to .Net 8.

2. Environment and Tools Preparation:

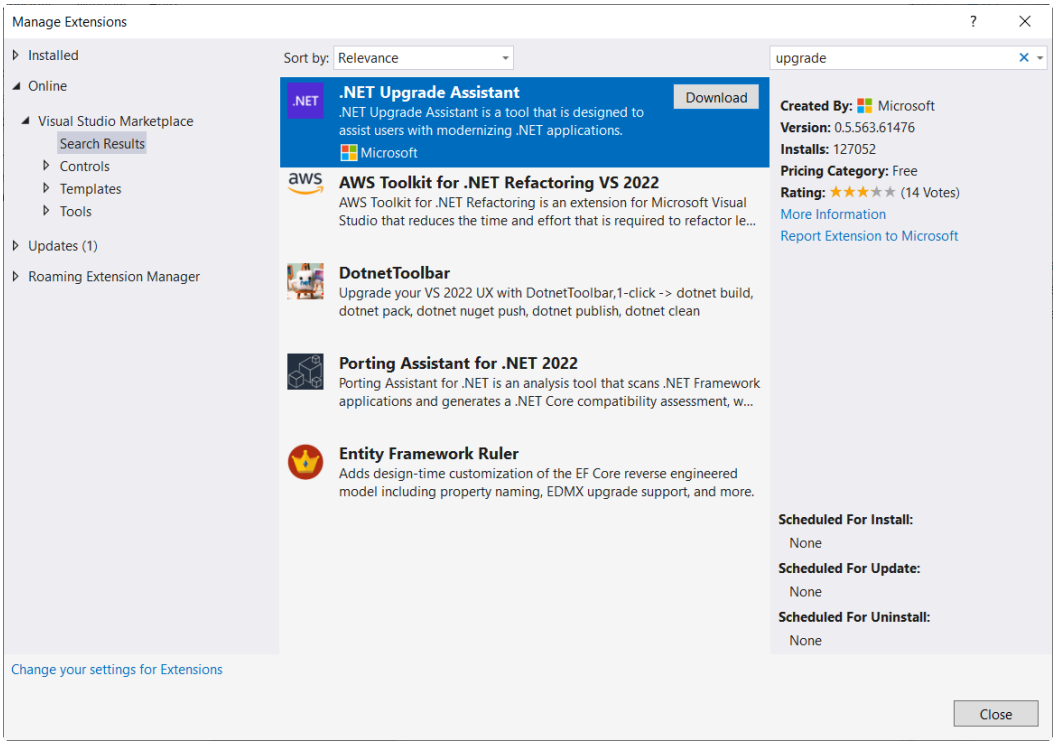
- **Visual Studio 2022:**
  Update Visual Studio 2022 to 17.8 version (17.7 version can be used for Preview only).
  See version requirements for .Net 8: https://learn.microsoft.com/en-us/dotnet/core/compatibility/sdk/8.0/version-requirements
  Visual Studio->Help->Check for Updates->choose Current for Update channel:



- **.Net 8 SDK:**
  Install .Net 8 from: https://dotnet.microsoft.com/en-us/download/visual-studio-sdks?cid=getdotnetsdk
- **.NET Upgrade Assistant:**
  Install the Visual Studio extension ".NET Upgrade Assistant":

https://learn.microsoft.com/en-us/dotnet/core/porting/upgrade-assistant-install#install-the-visual-studio-extension

(.NET Portability Analyzer (extension for VS2017 and VS2019 only) is included in .NET Upgrade Assistant)

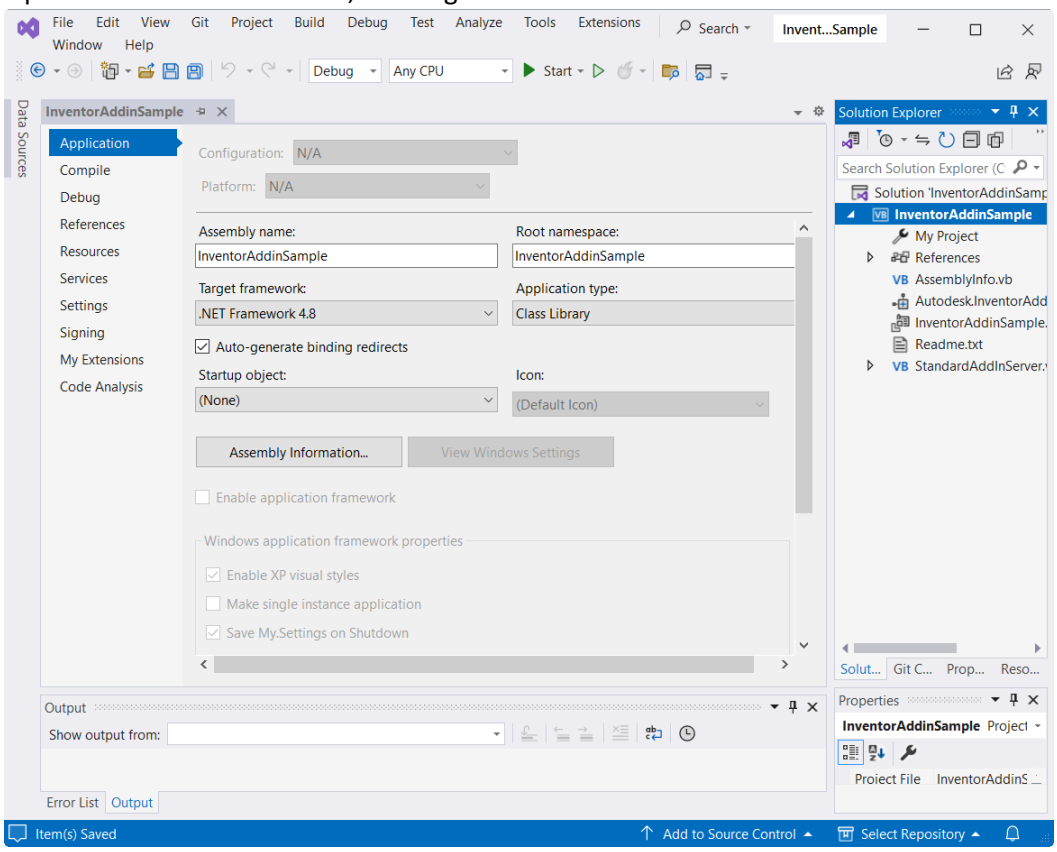


3. Upgrade Process (VB/C# projects):

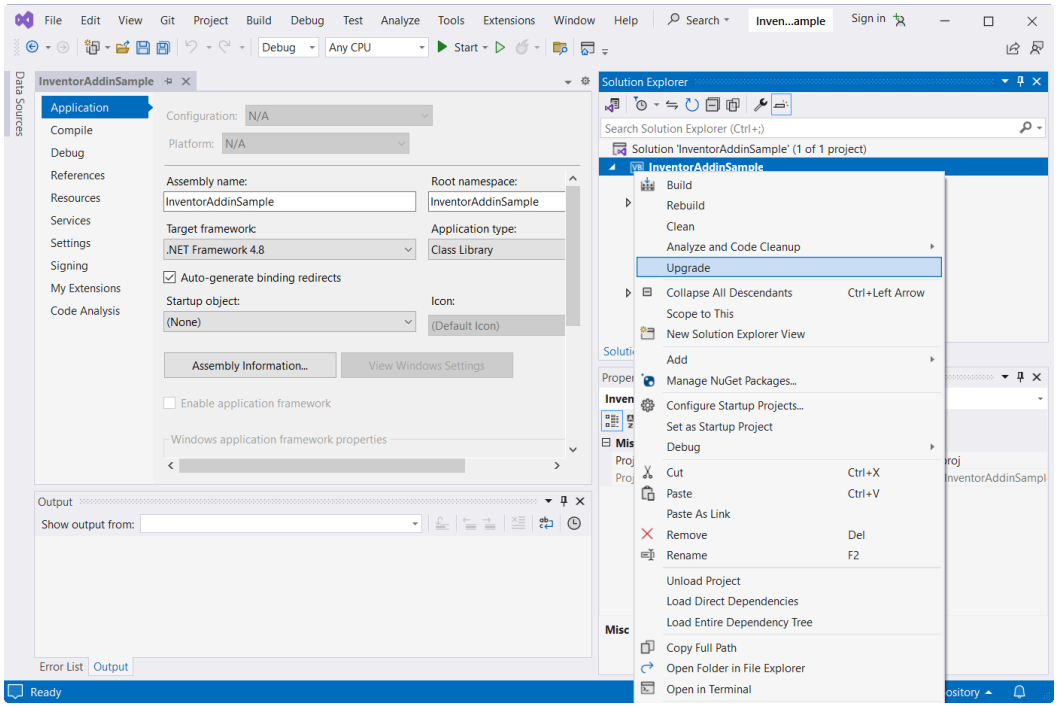
Note: Below project makes use of a simple VB.net addin project as example, for your own project you can create a copy before migrating it. The process to upgrade a C# project is similar.

a) Unzip attached InventorAddinSample.zip. This is a simple VB.net project for Inventor addin.

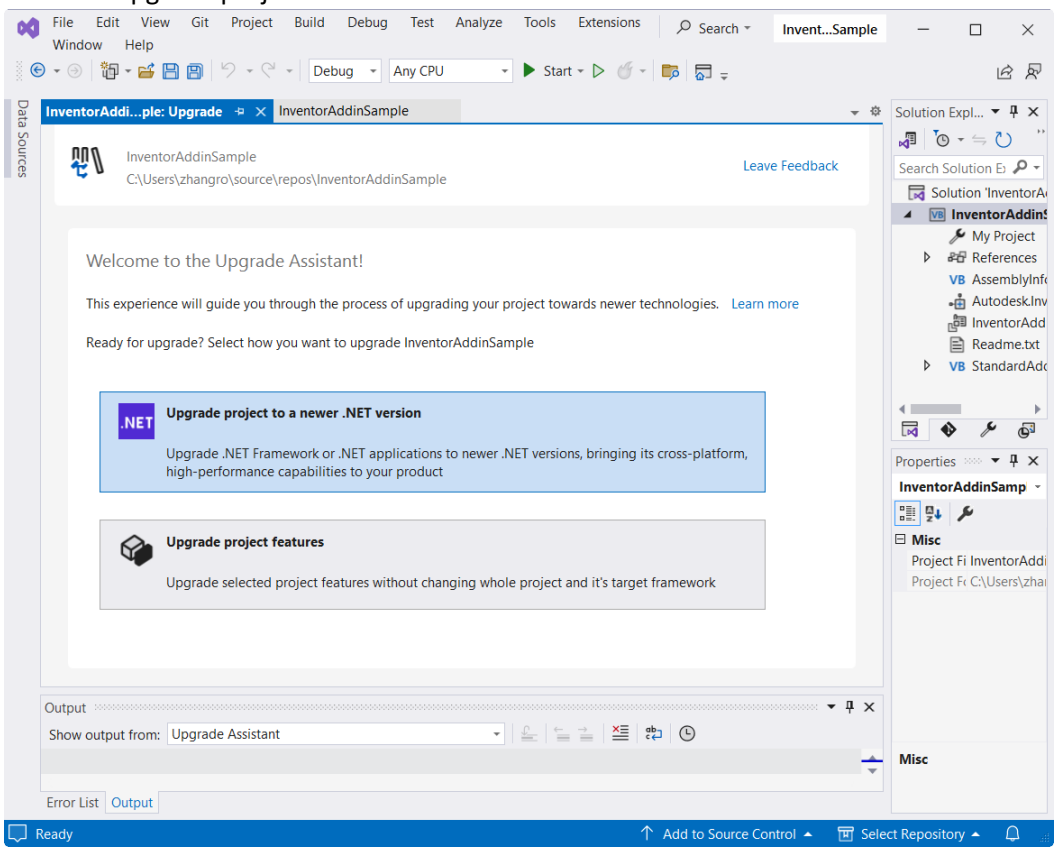
b) Open it in Visual Studio 2022, it is targeted to .Net Framework 4.8.



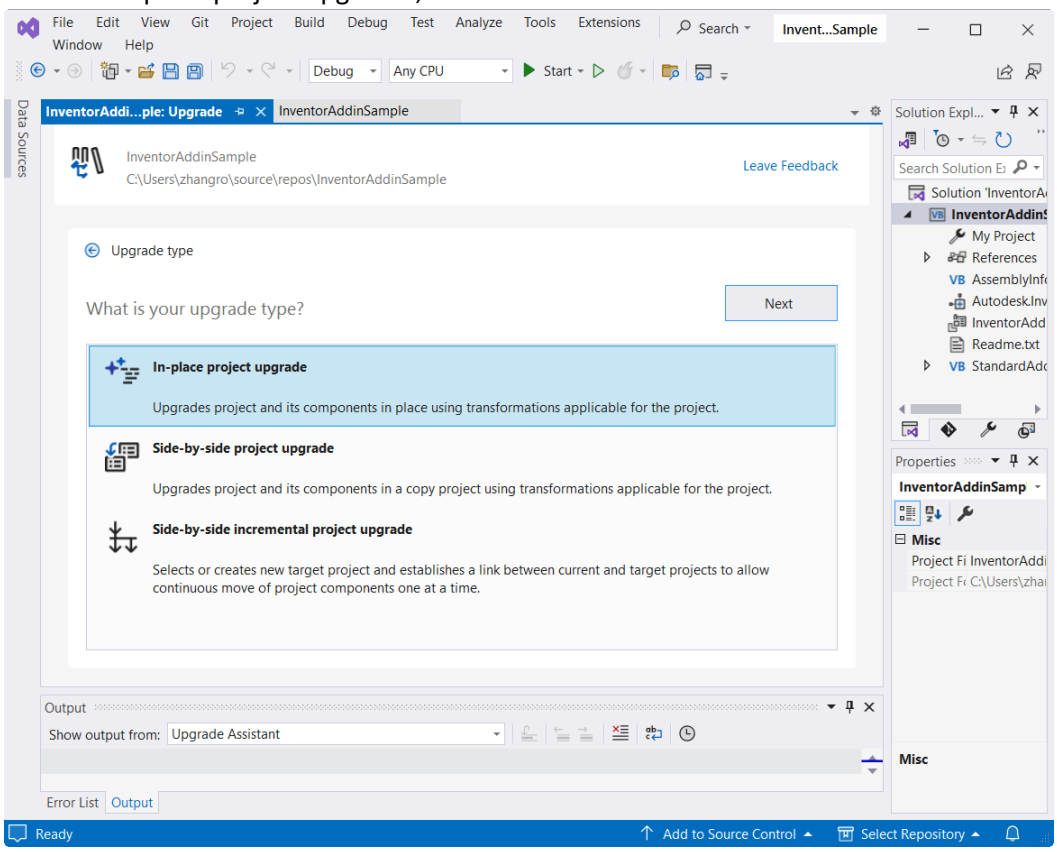
c) Right click the project, and choose Upgrade command:



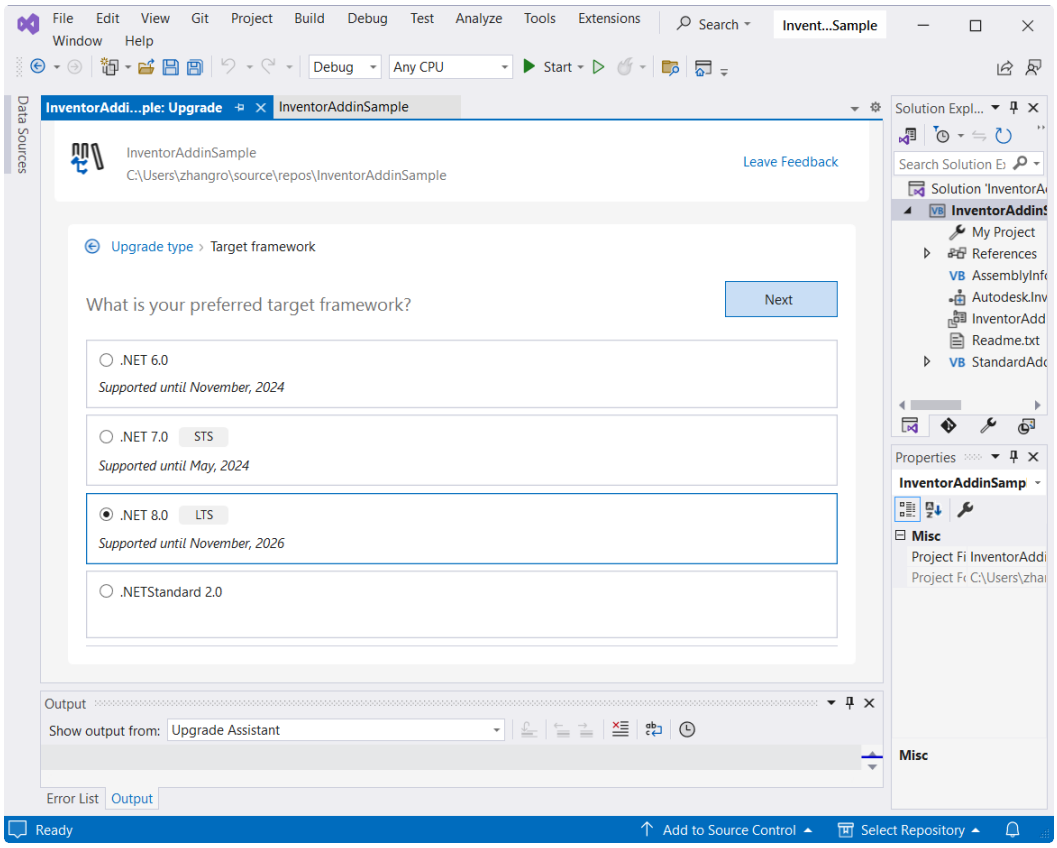
d) Choose “Upgrade project to a newer .NET version”:



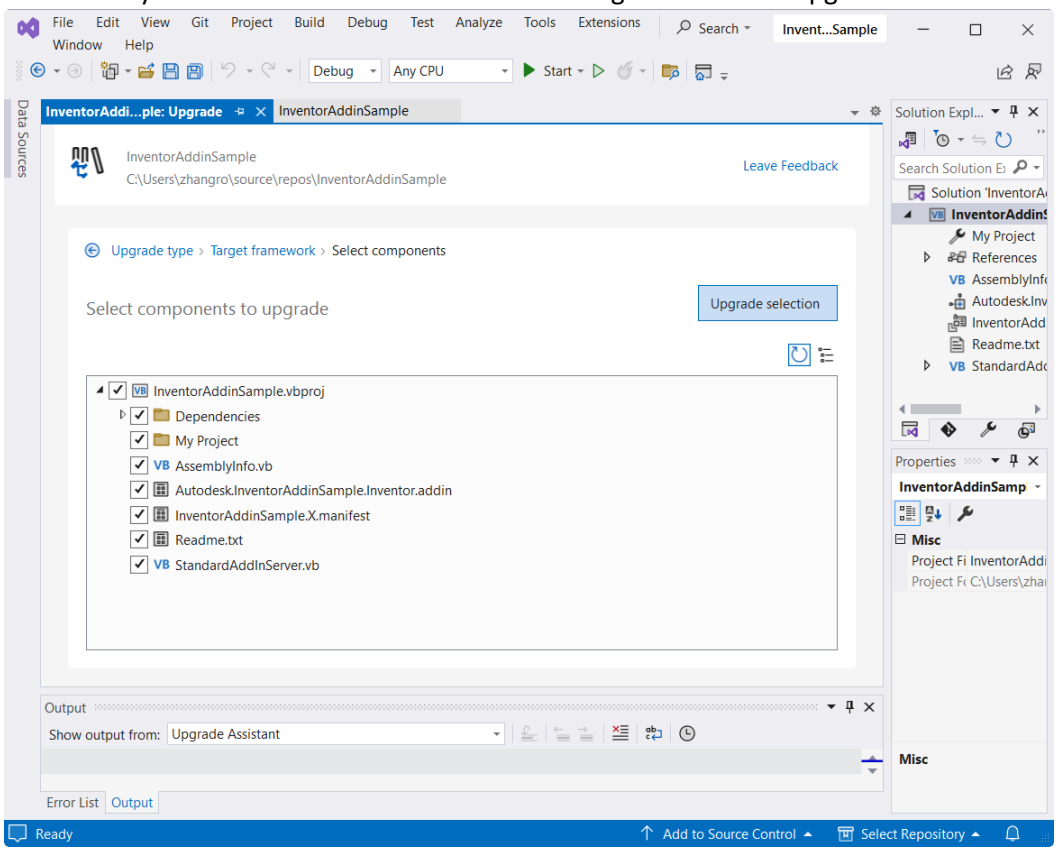
e) Choose “In-place project upgrade”, and click Next:



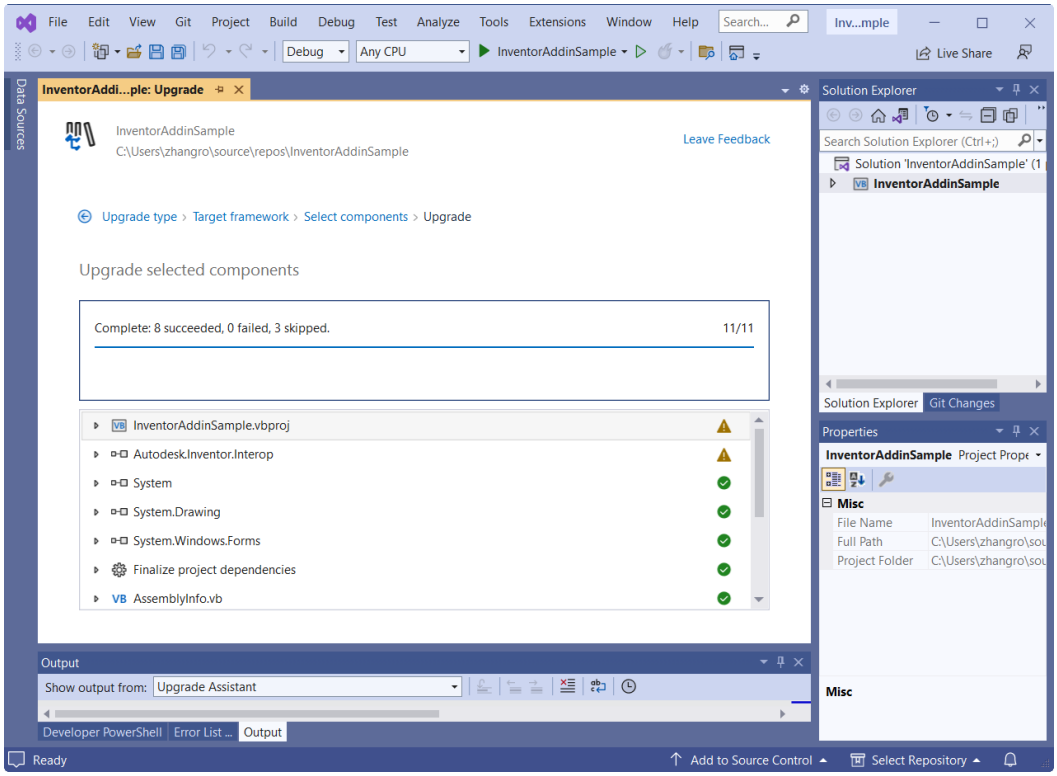
f) Choose ".NET 8.0" and click Next:



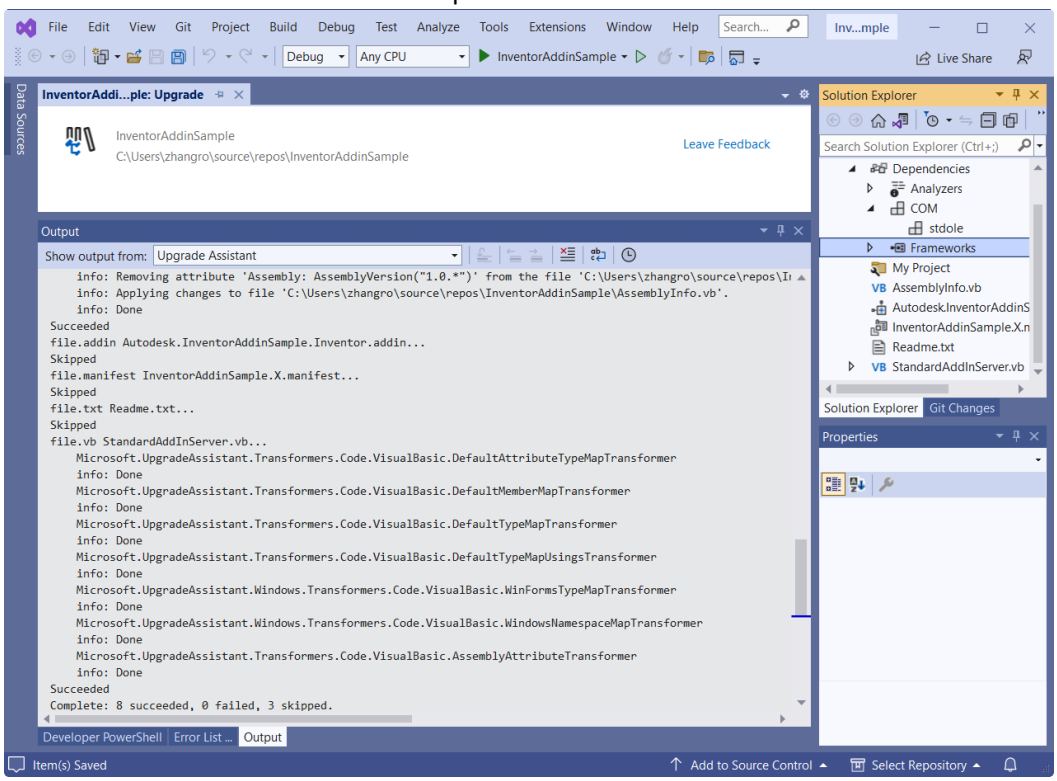
g) Select all if you are not sure which ones need to migrate and click Upgrade selection:



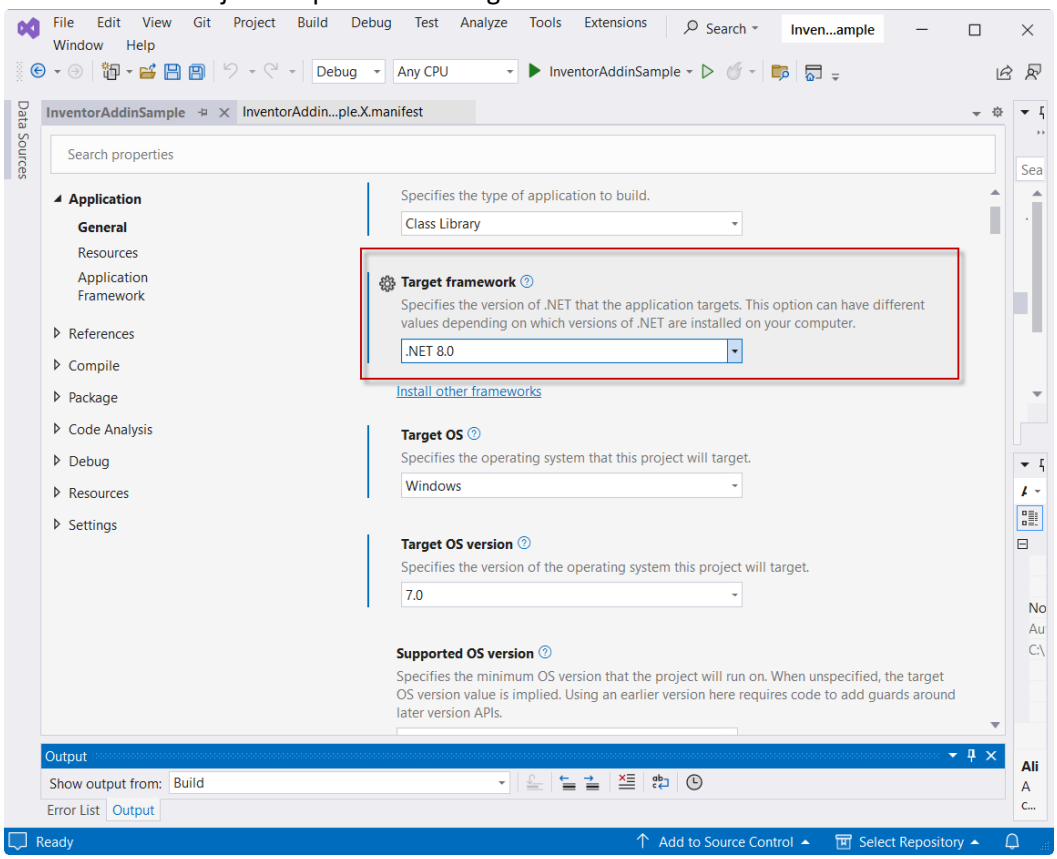
h) Now the migration result is as below:



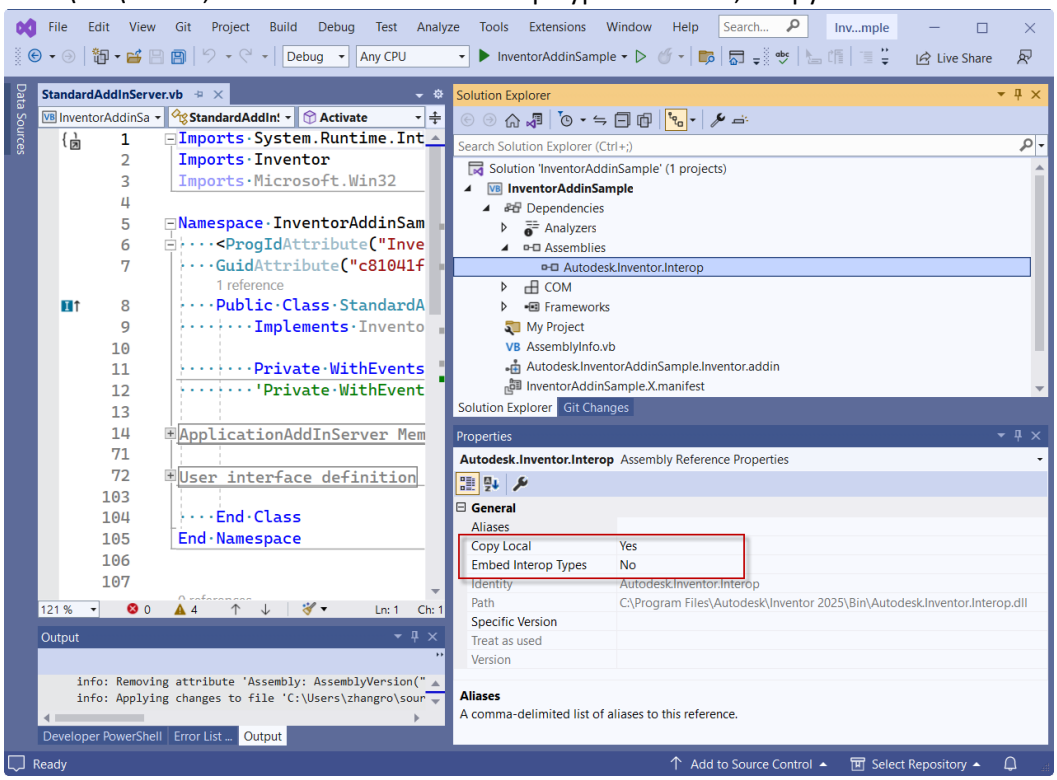
You can also view the result from Output window:



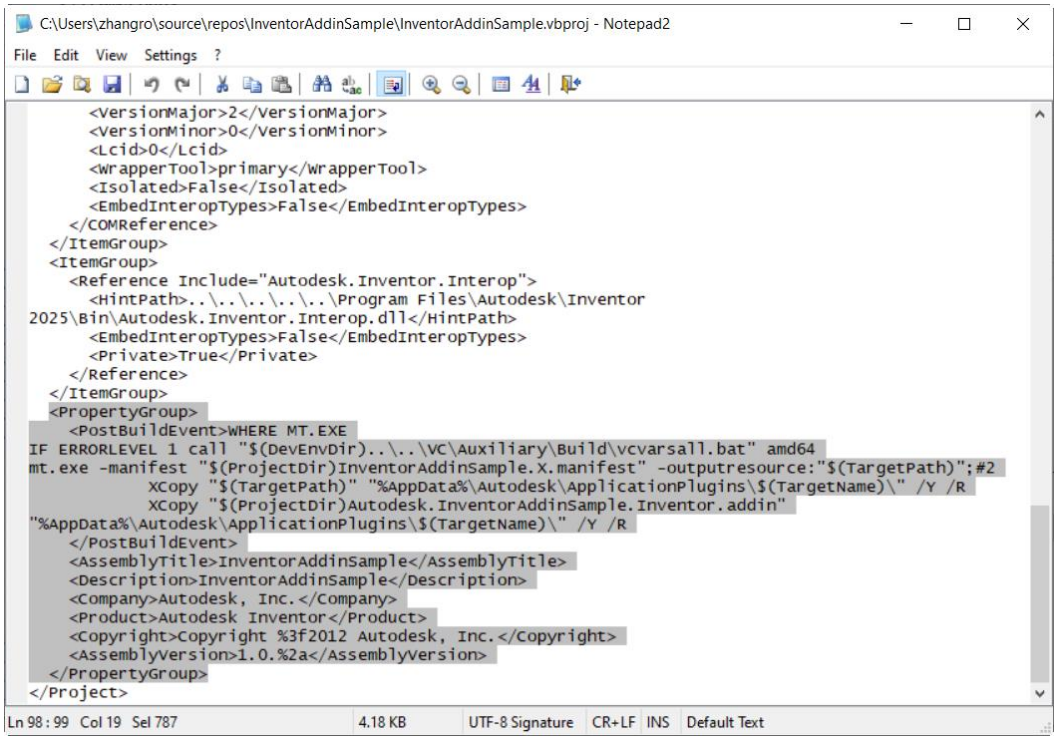
And check the Project Properties the Target framework is .NET 8.0 now:



i) Re-reference Autodesk.Inventor.interop.dll from C:\Program Files\Autodesk\Inventor 2025\Bin\ folder, and set the “Embed Interop Types” to False, “Copy Local” to True:



j) Edit the InventorAddinSample.vbproj using a text editor, and delete the below <PropertyGroup> tag for Post Build Event:




```
<PropertyGroup>
```

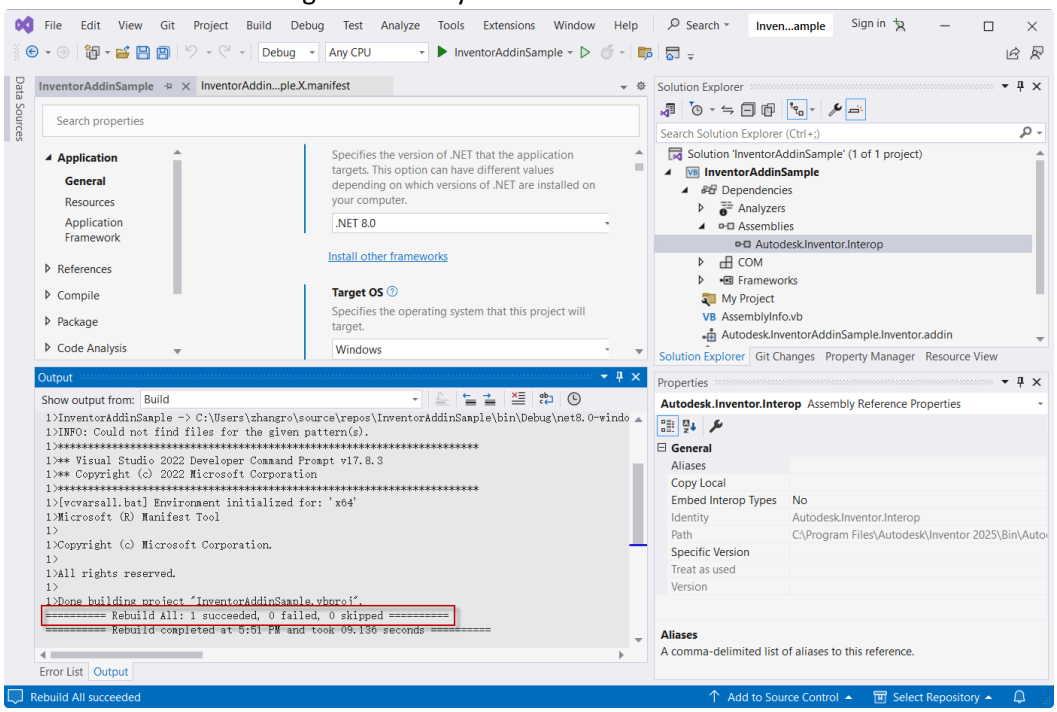
```
    <PostBuildEvent>WHERE MT.EXE
IF ERRORLEVEL 1 call "$(DevEnvDir)..\..\VC\Auxiliary\Build\vcvarsall.bat" amd64
mt.exe -manifest "$(ProjectDir)InventorAddinSample.X.manifest" -
outputresource:"$(TargetPath)";#2
                                XCopy "$(TargetPath)"
"%AppData%\Autodesk\ApplicationPlugins\$(TargetName)\" /Y /R
                        XCopy
"$(ProjectDir)Autodesk.InventorAddinSample.Inventor.addin"
"%AppData%\Autodesk\ApplicationPlugins\$(TargetName)\" /Y /R
            </PostBuildEvent>
    <AssemblyTitle>InventorAddinSample</AssemblyTitle>
    <Description>InventorAddinSample</Description>
    <Company>Autodesk, Inc.</Company>
    <Product>Autodesk Inventor</Product>
    <Copyright>Copyright %3f2012 Autodesk, Inc.</Copyright>
    <AssemblyVersion>1.0.%2a</AssemblyVersion>
  </PropertyGroup>
```

k) Add below <Target> tag to do the PostBuild:

Note: For you own project you need to update the below highlighted text to specify correct manifest file name:

```
<Target Name="PostBuild" AfterTargets="PostBuildEvent">
      <Exec Command="WHERE MT.EXE&#xD;&#xA;IF ERRORLEVEL 1 call
   &quot;$(DevEnvDir)..\..\VC\Auxiliary\Build\vcvarsall.bat&quot;
   amd64&#xD;&#xA;mt.exe -manifest
   &quot;$(ProjectDir)InventorAddinSample.X.manifest&quot; -
   outputresource:&quot;$(TargetPath)&quot;;#2" />
 </Target>
```

l) Rebuild the solution and get the binary:



The upgraded project is InventorAddinSample_NET8.zip

m) Deploy your addin to proper locations:

Place the Autodesk.InventorAddinSample.Inventor.addin to C:\Program Files\Autodesk\Inventor 2025\Bin\Addins\ folder.

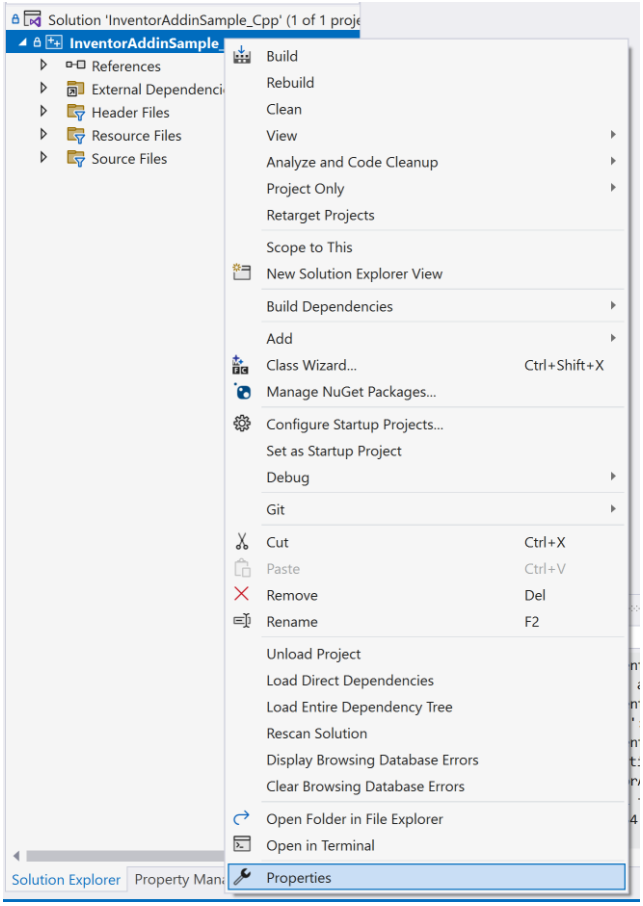
Place the binary \InventorAddinSample\bin\Debug\net8.0-windows7.0\InventorAddinSample.dll to C:\Program Files\Autodesk\Inventor 2025\Bin\ folder.

n) Launch Inventor, and check if the addin works well.
For this sample addin, when it is loaded it will just create an assembly document.

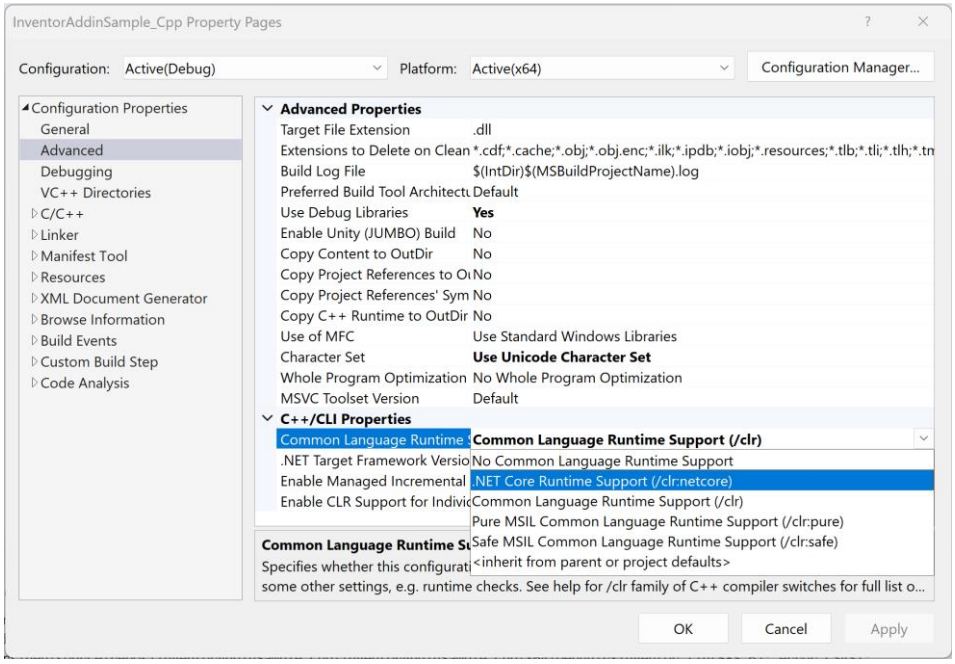
4. Upgrade Process (C++/CLI projects):

a) The ".NET Upgrade Assistant" doesn't work for C++/CLI projects. Therefore, the project needs to be upgraded manually.

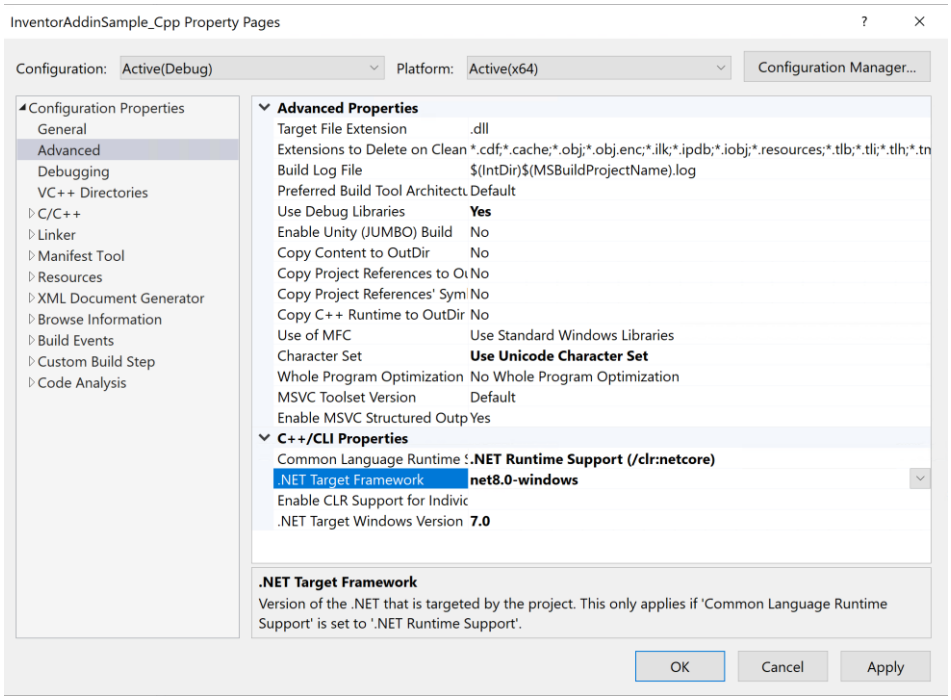
b) Open it in Visual Studio 2022. Right click the project, and choose “Properties”.



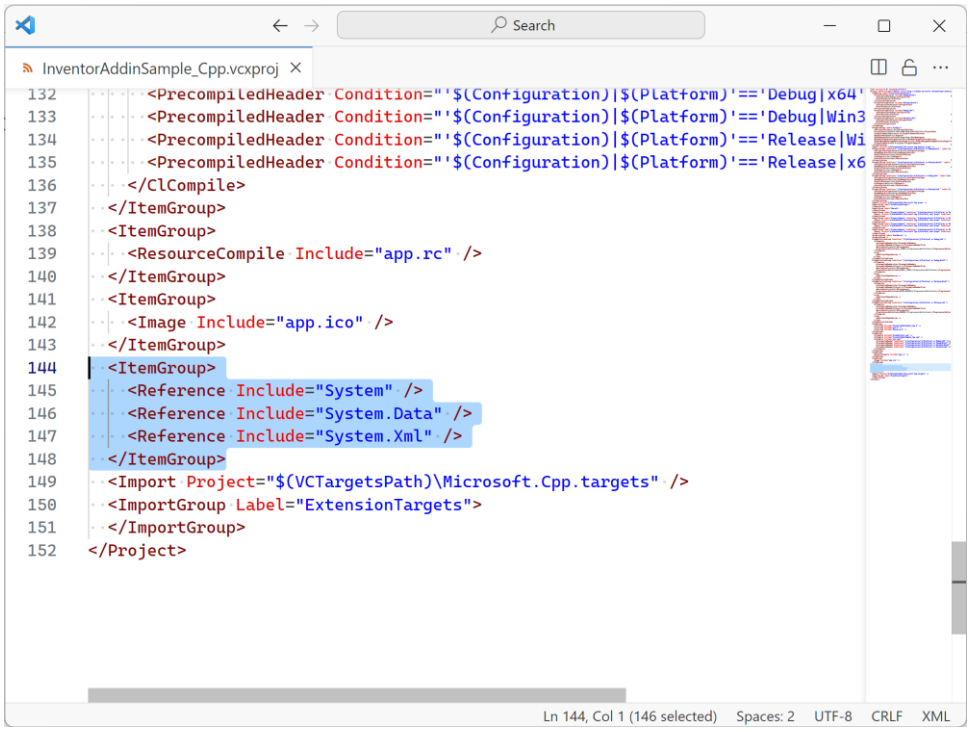
c) In the “Advanced” page, choose “.NET Core Runtim Support (/clr:netcore)” for Common Language Runtime Support. Then click “Apply” button.



d) The propert page will change. Then fill ".NET Target Framework" with "net8.0-windows" or "net8.0".



e) Open the project file, and delete system references.



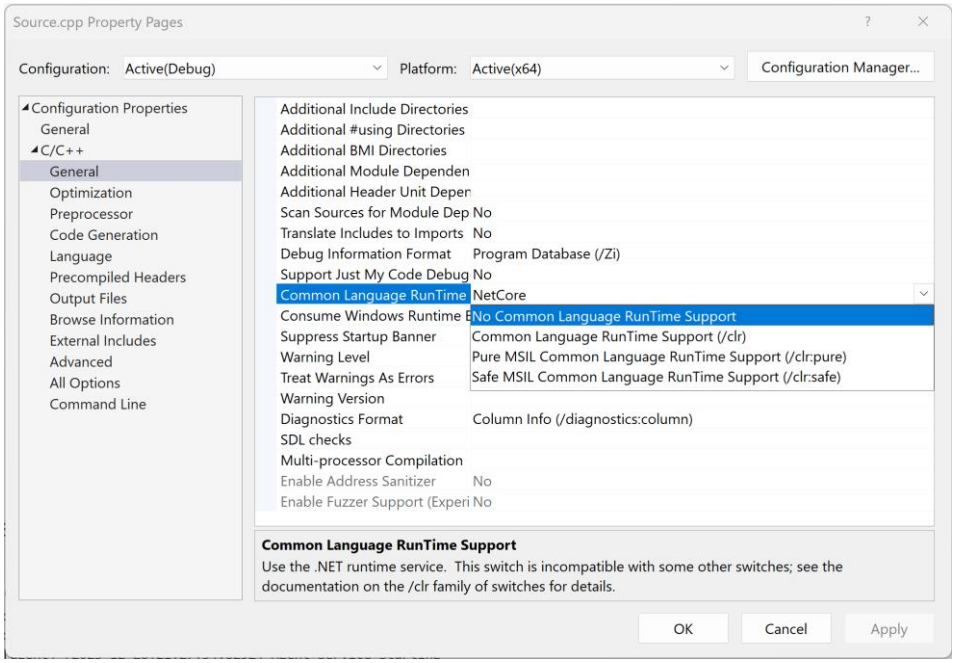
f) Add framework references to WPF/WinForms when necessary, such as:

```
 <!-- Reference all of WPF -->
<FrameworkReference Include="Microsoft.WindowsDesktop.App.WPF" />
```

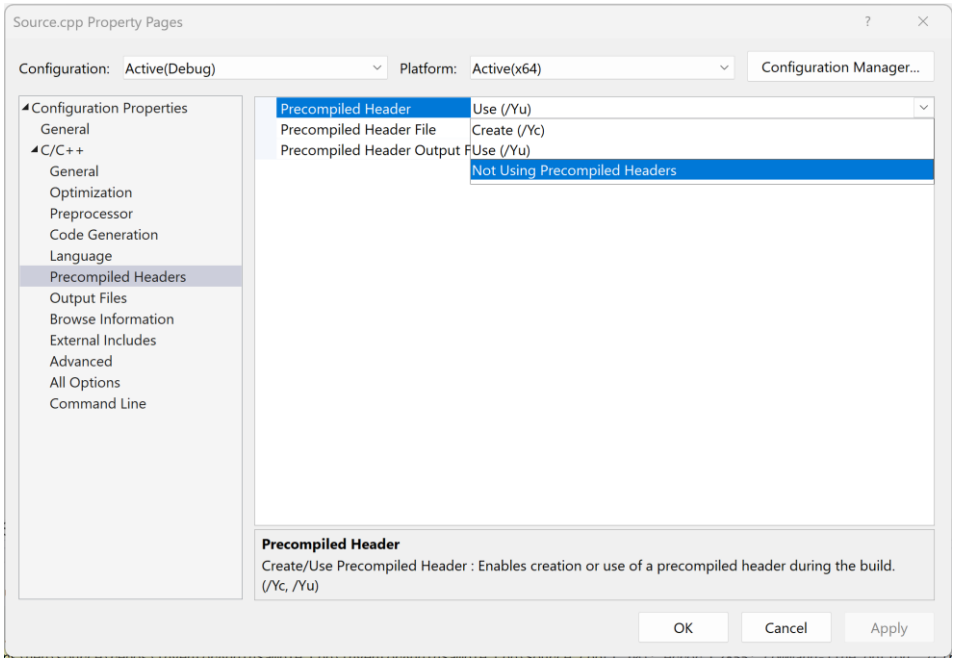
g) In the project file, check if there are any individual files has the “CompileAsManaged” property. Remove it as this property will cause build error and the file is compiled as managed by default.

```
<CompileAsManaged
Condition="'$(Configuration)|$(Platform)'=='Debug|x64'">true</CompileAsManaged>
```

h) To compile a file as native code, choose “No Common Language Runtime Support” in the “General” page.



Also choose “Not Using Precompiled Headers” in the “Precompiled Headers” page.

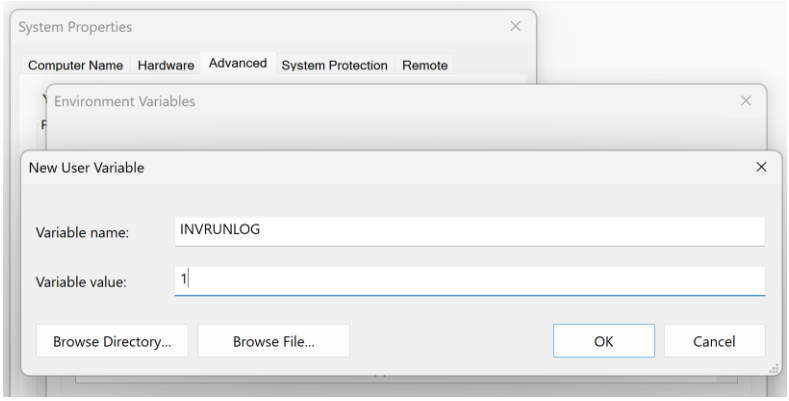


i) Re-reference Autodesk.Inventor.interop.dll if it is referenced by the project.

j) Rebuild the project and deploy the binaries.

5. Troubleshoot problems when unable to load the add-in:

- **Check whether the [libname].runtimeconfig.json file exists in the add-in folder if the library is loaded from native code.**
  This file is generated during build. It is used to config the .NET runtime environment. Copy it from the build output folder if not found.
- **For C++/CLI libraries, check whether ijwhost.dll exists in the add-in folder.**
  This file is copied from .NET SDK to the build output folder during build. It is used to load the C++/CLI library.
- **Define an environment variable "INVRUNLOG" to enable the logs for add-in load.**



  The logs will be generated to "%LocalAppData%\Autodesk\Inventor 2025\RunLogs". Search for the add-in name and look for errors recorded when loading the add-in.
- **If there are any exceptions for assemblies unable to be found, please check whether the [libname].deps.json file exists in the add-in folder.**
  Th file is also generated during build. It is used to locate the dependencies of the current assembly.
  - If it exists, then the information in the file will be used for searching the assemblies to load.
  - Otherwise, the assemblies in the current folder are used instead.
  - An add-in can create a custom AssemblyLoadContext to resolve the dependencies of itself.

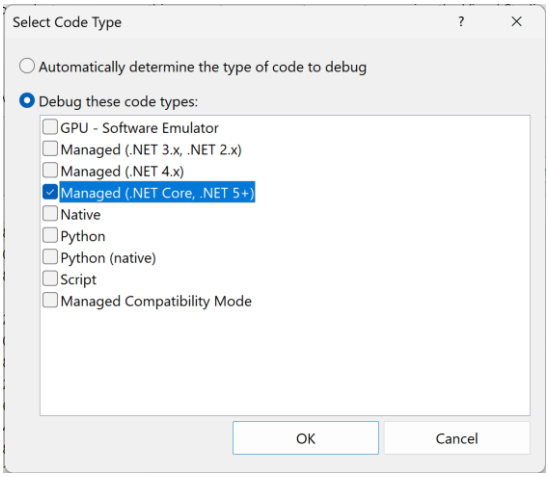
6. Upgrade tips:

- **Which target framework to choose?**
  The target framework determines the .NET API set that can be used in the project. It can be assigned in the property page as well as the project file:

```
<TargetFramework>net8.0-windows</TargetFramework>
```

  - If there are no other considerations, please use "net8.0-windows", which is to target NET 8.0 and Windows OS.
  - If the assembly will also be used in other operation systems, it can be set as "net8.0".
  - If the same assembly will still be loaded in a .NET Framework application, please set it as "netstandard2.0". This will limit the number of .NET APIs that can be used in the add-in.

- **Can .NET Framework assemblies be loaded?**
  They can be loaded by the compatibility mode in .NET 8. There may be runtime issues if the assembly calls some API that is not supported in .NET 8.
- **Dependencies are not in the output folder.**
  Define the property CopyLocalLockFileAssemblies as true in the project file.
- **Unable to reference assemblies from GAC.**
  This is not supported. Specify the assembly path explicitly or use NuGet reference instead.
- **Code can't be debugged. The breakpoint doesn't work.**
  Please make sure "Managed (.NET Core, .NET 5+)" is selected when attached the debugger. Also select "Native" if C++ code needs to be debugged.



- **Process.Start doesn't work**.
  UseShellExecute should be set explicitly:

```
System.Diagnostics.Process.Start(new ProcessStartInfo(url) { UseShellExecute = true }); // url = https://autode.sk or url = "D:\\a.txt"
```

- **Encoding.GetEncoding doesn't work.**
  Register the provider before using this API.

```
Encoding.RegisterProvider(CodePagesEncodingProvider.Instance);
```

- **Thread can't be aborted.**
  This is not supported anymore. The recommended method is to use CancellationToken.
- **Binary Formatter is obsolete.**
  It should be replaced by up-to-date technologies such as System.Text.Json.

7. More information:

a) CLI tool to upgrade project:
Visual Studio also provides CLI tool to upgrade your project using command, refer to below link:
https://learn.microsoft.com/en-us/dotnet/core/porting/upgrade-assistant-overview#upgrade-with-the-cli-tool

b) If your project targets.Net Framework lower than 4.7.2, it is recommended to firstly update it to .Net Framework 4.7.2 or higher. See:

https://learn.microsoft.com/en-us/dotnet/core/porting/premigration-needed-changes#update-net-framework-target-version

c) If your project use some older assemblies like VB6 compatibility assembly for Icon, you can refer below link to find the solution: https://adndevblog.typepad.com/manufacturing/2012/06/how-to-convert-iconbitmap-to-ipicturedisp-without-visualbasiccompatibilityvb6supporticontoipicture.html

d) For more information about how to port a C++/CLI project: https://learn.microsoft.com/en-us/dotnet/core/porting/cpp-cli

e) To understand the content of .runtimeconfig.json and .deps.json files: https://github.com/dotnet/sdk/blob/main/documentation/specs/runtime-configuration-file.md

f) To debug assembly loading problems by collecting the loading information: https://learn.microsoft.com/en-us/dotnet/core/dependency-loading/collect-details

g) Several technologies are unsupported in .NET 8, such as Application Domains and Remoting. Check the list here: https://learn.microsoft.com/en-us/dotnet/core/porting/net-framework-tech-unavailable