How to: Migrate Extensibility Projects to Visual Studio 2017

原文<https://docs.microsoft.com/zh-cn/visualstudio/extensibility/how-to-migrate-extensibility-projects-to-visual-studio-2017>

本文内容

[Install Visual Studio 2017 with required workloads](https://docs.microsoft.com/zh-cn/visualstudio/extensibility/how-to-migrate-extensibility-projects-to-visual-studio-2017" \l "install-visual-studio-2017-with-required-workloads)

[Open VSIX Solution in Visual Studio 2017](https://docs.microsoft.com/zh-cn/visualstudio/extensibility/how-to-migrate-extensibility-projects-to-visual-studio-2017" \l "open-vsix-solution-in-visual-studio-2017)

[Update the Microsoft.VSSDK.BuildTools NuGet package](https://docs.microsoft.com/zh-cn/visualstudio/extensibility/how-to-migrate-extensibility-projects-to-visual-studio-2017" \l "update-the-microsoftvssdkbuildtools-nuget-package)

[Make changes to the VSIX extension manifest](https://docs.microsoft.com/zh-cn/visualstudio/extensibility/how-to-migrate-extensibility-projects-to-visual-studio-2017" \l "make-changes-to-the-vsix-extension-manifest)

[If migrating from Preview 4 or Preview 5](https://docs.microsoft.com/zh-cn/visualstudio/extensibility/how-to-migrate-extensibility-projects-to-visual-studio-2017" \l "if-migrating-from-preview-4-or-preview-5)

[Update Debug settings for project](https://docs.microsoft.com/zh-cn/visualstudio/extensibility/how-to-migrate-extensibility-projects-to-visual-studio-2017" \l "update-debug-settings-for-project)

[Check that the extension builds correctly (as a VSIX v3)](https://docs.microsoft.com/zh-cn/visualstudio/extensibility/how-to-migrate-extensibility-projects-to-visual-studio-2017" \l "check-that-the-extension-builds-correctly-as-a-vsix-v3)

[Check when all required prerequisites are installed](https://docs.microsoft.com/zh-cn/visualstudio/extensibility/how-to-migrate-extensibility-projects-to-visual-studio-2017" \l "check-when-all-required-prerequisites-are-installed)

[Check when missing the required prerequisites](https://docs.microsoft.com/zh-cn/visualstudio/extensibility/how-to-migrate-extensibility-projects-to-visual-studio-2017" \l "check-when-missing-the-required-prerequisites)

[Deciding on Components](https://docs.microsoft.com/zh-cn/visualstudio/extensibility/how-to-migrate-extensibility-projects-to-visual-studio-2017" \l "deciding-on-components)

[Finding Component IDs](https://docs.microsoft.com/zh-cn/visualstudio/extensibility/how-to-migrate-extensibility-projects-to-visual-studio-2017" \l "finding-component-ids)

This document explains how to upgrade extensibility projects to Visual Studio 2017. In addition to describing how to update the project files, it also describes how to upgrade from extension manifest version 2 (VSIX v2) to the new version 3 VSIX manifest format (VSIX v3).

Install Visual Studio 2017 with required workloads

Make sure your installation includes the following workloads:

**1.** .NET desktop development

**2.** Visual Studio extension development

Open VSIX Solution in Visual Studio 2017

All VSIX projects will require a major version one-way upgrade to Visual Studio 2017.

The project file (for example \*.csproj) will be updated:

**1.** MinimumVisualStudioVersion - now set to 15.0

**2.** OldToolsVersion (if previously exists) - now set to 14.0

Update the Microsoft.VSSDK.BuildTools NuGet package

Note: If your solution does not reference the Microsoft.VSSDK.BuildTools NuGet package, you can skip this step.

In order to build your extension in the new VSIX v3 (version 3) format, your solution will need to be built with the new VSSDK Build Tools. This will be installed with Visual Studio 2017, but your VSIX v2 extension might be holding a reference to an older version via NuGet. If so, you will need to manually install an update of the Microsoft.VSSDK.BuildTools NuGet package for your solution.

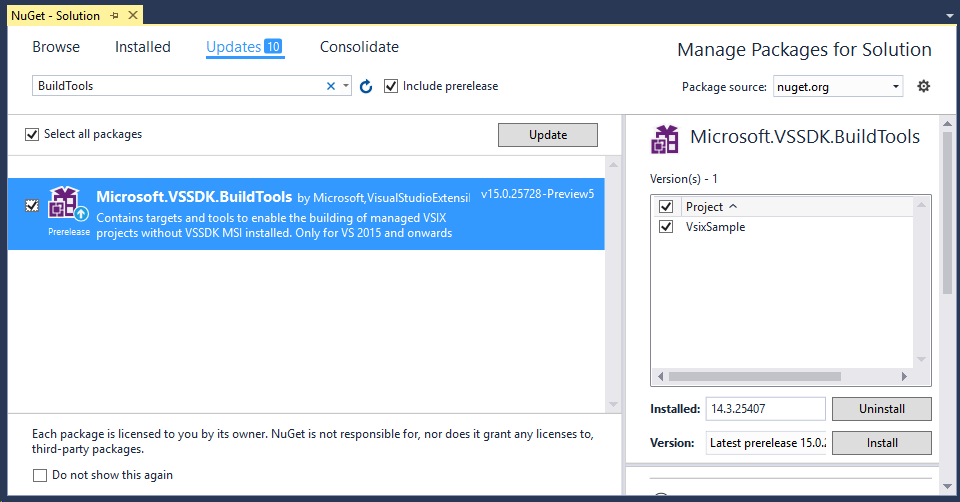
**1.** To update the NuGet references to Microsoft.VSSDK.BuildTools:

**2.** Right-click on the Solution and choose **Manage NuGet Packages for Solution...**

**3.** Navigate to the **Updates** tab.

**4.** Select Microsoft.VSSDK.BuildTools (latest version).

**5.** Press **Update**.



Make changes to the VSIX extension manifest

To ensure that the user's installation of Visual Studio has all the assemblies required to run the extension, specify all the prerequisite components or packages in the extension manifest file. When a user attempts to install the extension, the VSIXInstaller will check to see if all the prerequisites are installed. If some are missing, the user will be prompted to install the missing components as part of the extension installation process.

Note: At a minimum, all extensions should specify the Visual Studio core editor component as a prerequisite.

Edit the extension manifest file (usually called source.extension.vsixmanifest).

Ensure InstallationTarget includes 15.0.

Add required installation prerequisites (as shown in the example below).

We recommend you specify only Component IDs for installation prerequisites.

See the section at the end of this document for [instructions on identifying Component IDs](https://docs.microsoft.com/zh-cn/visualstudio/extensibility/how-to-migrate-extensibility-projects-to-visual-studio-2017" \l "finding-component-ids).

Example:

<PackageManifest>

...

<Prerequisites>

<Prerequisite Id="Microsoft.VisualStudio.Component.CoreEditor" Version="[15.0,16.0)" />

<Prerequisite Id="Microsoft.VisualStudio.Component.DiagnosticTools" Version="[15.0.25814.0,16.0)" />

<Prerequisite Id="Microsoft.VisualStudio.Shell.12.0" Version="[12.0]" />

</Prerequisites>

...

</PackageManifest>

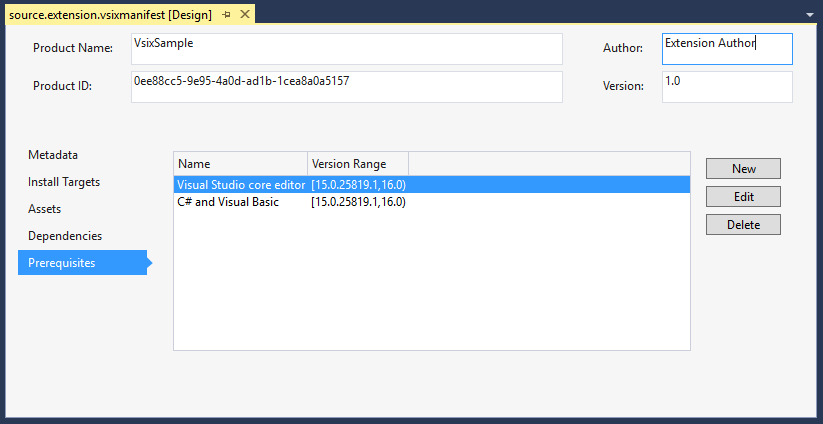
**Option: Use the designer to make changes to the VSIX extension manifest**

Instead of directly editing the manifest XML, you can use the new **Prerequisites** tab in the Manifest Designer to select the prerequisites and the XML will be updated for you.

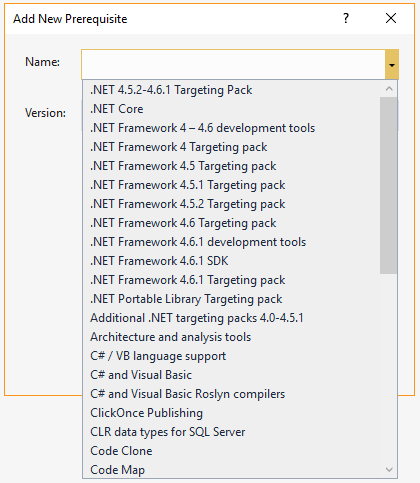
Note: The Manifest Designer will only allow you to select Components (not Workloads or Packages) that are installed on the current Visual Studio instance. If you need to add a prerequisite for a workload, package, or component that is not currently installed, edit the manifest XML directly.

**1.** Open source.extension.vsixmanifest [Design] file.

**2.** Select **Prerequisites** tab and press **New** button.



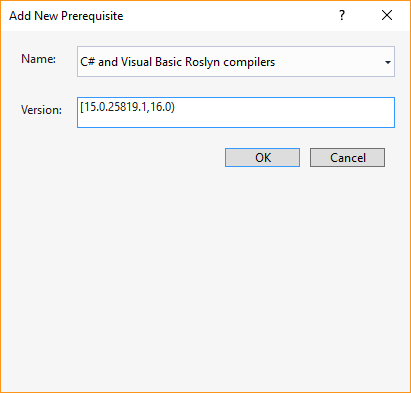
**3.** The **Add New Prerequisite** window will open.



**4.** Click on the dropdown for **Name** and select the desired prerequisite.

**5.** Update the version if required.

Note: The version field will be pre-populated with the version of the currently installed component, with a range spanning up to (but not including) the next major version of the component.

****

**6.** Press OK.

If migrating from Preview 4 or Preview 5

**1.** Replace SetupDependencies with Prerequisites and move the elements out of the Installer element. Prerequisites now sits directly inside the PackageManifest element.

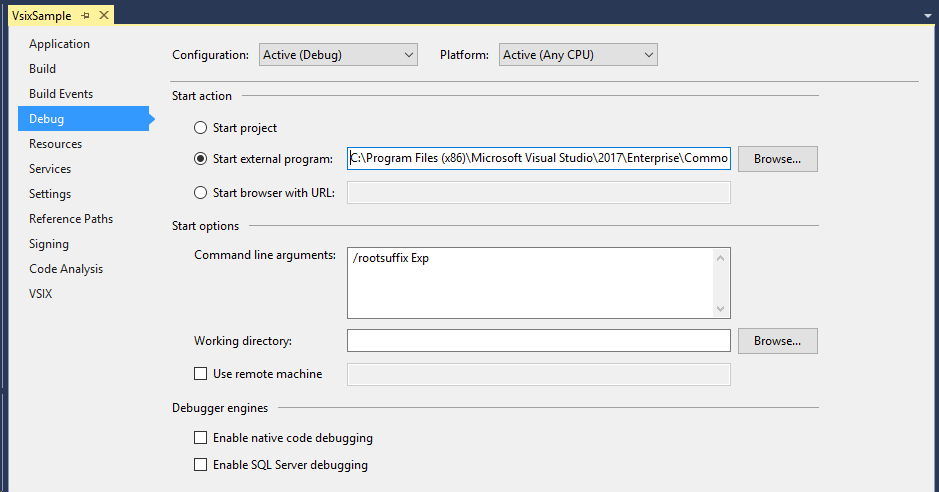
**2.** [Optional] Remove the GenerateVsixV3 element. (This was required in Preview 5 only.) The GenerateVsixV3 element will be ignored in versions beyond Preview 5.

Update Debug settings for project

If you wish to debug your extension in an experimental instance of Visual Studio, make sure that the project settings for **Debug** > **Start action** has the **Start external program:** value set to the devenv.exe file of your Visual Studio 2017 installation.

It might look like:

C:\Program Files (x86)\Microsoft Visual Studio\2017\Enterprise\Common7\IDE\devenv.exe



Note: The Debug Start Action is typically stored in the .csproj.user file. This file is usually included in the .gitignore file and, hence, is not normally saved with other project files when committed to source control. As such, if you have pulled your solution fresh from source control it is likely the project will have no values set for Start Action. New VSIX projects created with Visual Studio 2017 will have a .csproj.user file created with defaults pointing to the current Visual Studio install directory. However if you are migrating a VSIX v2 extension, it is likely that the .csproj.user file will contain references to the previous Visual Studio version's install directory. Setting the value for **Debug > Start action** will allow the correct Visual Studio experimental instance to launch when you try to debug your extension.

Check that the extension builds correctly (as a VSIX v3)

**1.** Build the VSIX project.

**2.** Unzip the generated VSIX.

**1.** By default, the VSIX file lives inside bin/Debug or bin/Release as [YourCustomExtension].vsix.

**2.** Rename .vsix to .zip to easily view the contents.

**3.** Check for the existence of three files:

**1.** extension.vsixmanifest

**2.** manifest.json

**3.** catalog.json

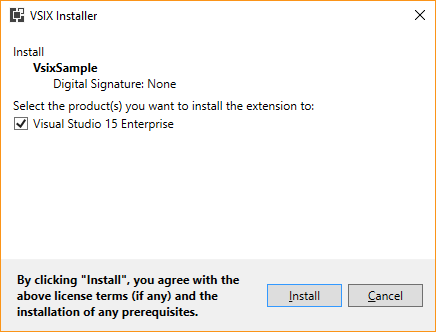
Check when all required prerequisites are installed

Test that the VSIX installs successfully on a machine with all required prerequisites installed.

Note: Before installing any extension, please shut down all instances of Visual Studio.

Attempt to install the extension:

**1.** On Visual Studio 2017



**2.** Optional: Check on previous versions of Visual Studio.

**1.** Proves backward compatibility.

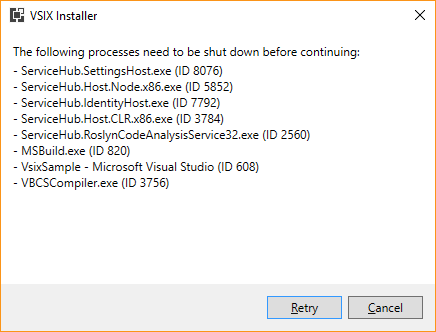
**2.** Should work for Visual Studio 2012, Visual Studio 2013, Visual Studio 2015.

**3.** Optional: Check that VSIX Installer Version Checker offers a choice of versions.

**1.** Includes previous versions of Visual Studio (if installed).

**2.** Includes Visual Studio 2017.

If Visual Studio was recently opened, you might see a dialog box like this:



Wait for the processes to shut down, or manually end the tasks. You can find the processes by the listed name, or with the PID listed in parenthesis.

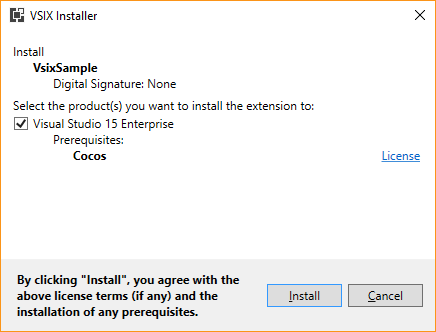
Note: These processes will not automatically shut down while an instance of Visual Studio is running. Ensure that you've shut down all instances of Visual Studio on the machine - including those from other users, then continue to retry.

Check when missing the required prerequisites

**1.** Attempt to install the extension on a machine with Visual Studio 2017 that DOES NOT CONTAIN all the components defined in the Prerequisites (above).

**2.** Check that the installation identifies the missing component/s and lists them as a prerequisite in the VSIXInstaller.

**3.** Note: Elevation will be required if any prerequisites need to be installed with the extension.



Deciding on Components

When looking up your dependencies, you will find that one dependency could map to multiple components. To determine which dependencies you should specify as your prerequisite, we suggest that you choose a component that has a functionality similar to your extension and to also consider your users and what type of components would they most likely have installed or wouldn't mind installing. We also suggest building your extensions in a way where the required prerequisites satisfy only the minimum that will allow your extension to run and for additional features have them be dormant if certain components are not detected.

|  |  |  |
| --- | --- | --- |
| Extension Type | Display Name | Id |
| Editor | Visual Studio core editor | Microsoft.VisualStudio.Component.CoreEditor |
| Roslyn | C# and Visual Basic | Microsoft.VisualStudio.Component.Roslyn.LanguageServices |
| WPF | Managed Desktop Workload Core | Microsoft.VisualStudio.Component.ManagedDesktop.Core |
| Debugger | Just-In-Time debugger | Microsoft.VisualStudio.Component.Debugger.JustInTime |

To provide further guidance, we have identified a few common extension types and their suggested prerequisites:

Finding Component IDs

The list of Components sorted by Visual Studio product is at [Visual Studio 2017 Workload and Component IDs](https://aka.ms/vs2017componentIDs). Use these Component IDs for your Prerequisite IDs in your manifest.

If you are unsure which Component contains a specific binary, download the [Component -> Binary mapping spreadsheet](https://aka.ms/vs2017componentid-binaries).

**vs2017-ComponentBinaryMapping.xlsx**

There are four columns in the Excel sheet: **Component Name, ComponentId, Version**, and **Binary / File Names**. You can use the filters to search and find specific components and binaries.

For all your references, first determine which ones are in the core editor (Microsoft.VisualStudio.Component.CoreEditor) component. At minimum, we require the core editor component to be specified as a prerequisite for all extensions. Of the references that are left that are not in core editor, add filters in the **Binaries / Files Names** section to find components that have any of the subset of those references.

Examples:

If you have a debugger extension and know that your project has a reference to VSDebugEng.dll and VSDebug.dll, click on the filter button in the **Binaries / Files Names** header. Search for "VSDebugEng.dll" and select OK. Next click on the filter button in the **Binaries / Files Names** header again and search for "VSDebug.dll". Select the checkbox "Add current selection to filter" and select OK. Now look through the **Component Name** to find a component that is most related to your extension type. In this example, you would chose the Just-In-Time debugger and add it to your vsixmanifest.

If you know that your project deals with debugger elements, you can search on "debugger" in the filter search box to see what components contain debugger in its name.