

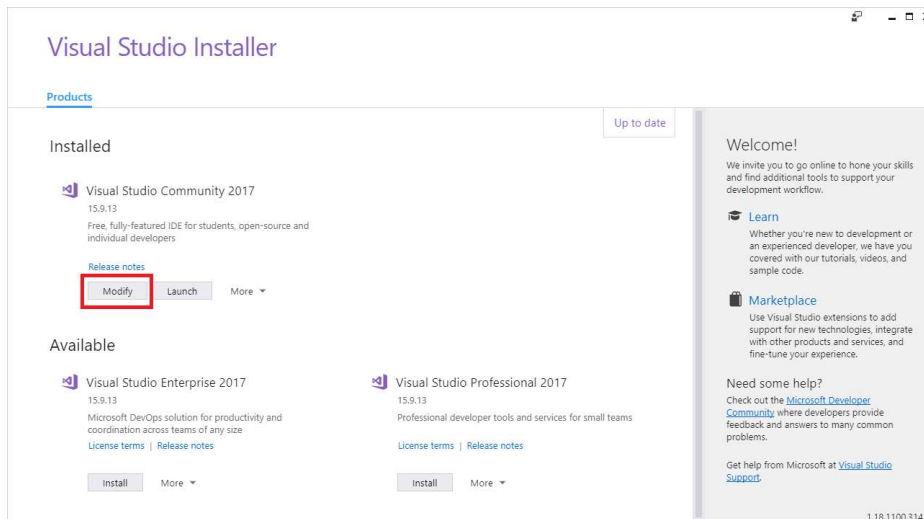
Creating Windows Install Packages

Part 1: Setup Visual Studio

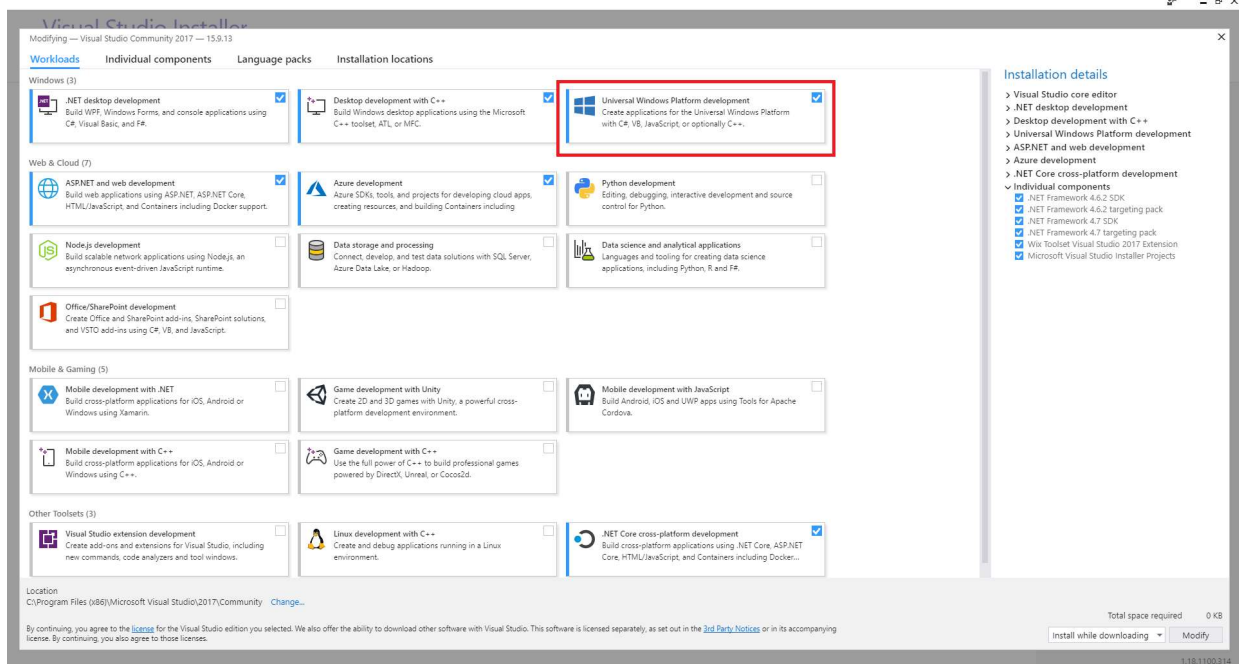
Ensure you have Visual Studio 2017 installed and the required SDKs.

To complete Part 4 requires “Universal Windows Platform Development” and the .NET Framework 4.6.2 SDK. If necessary, install it.

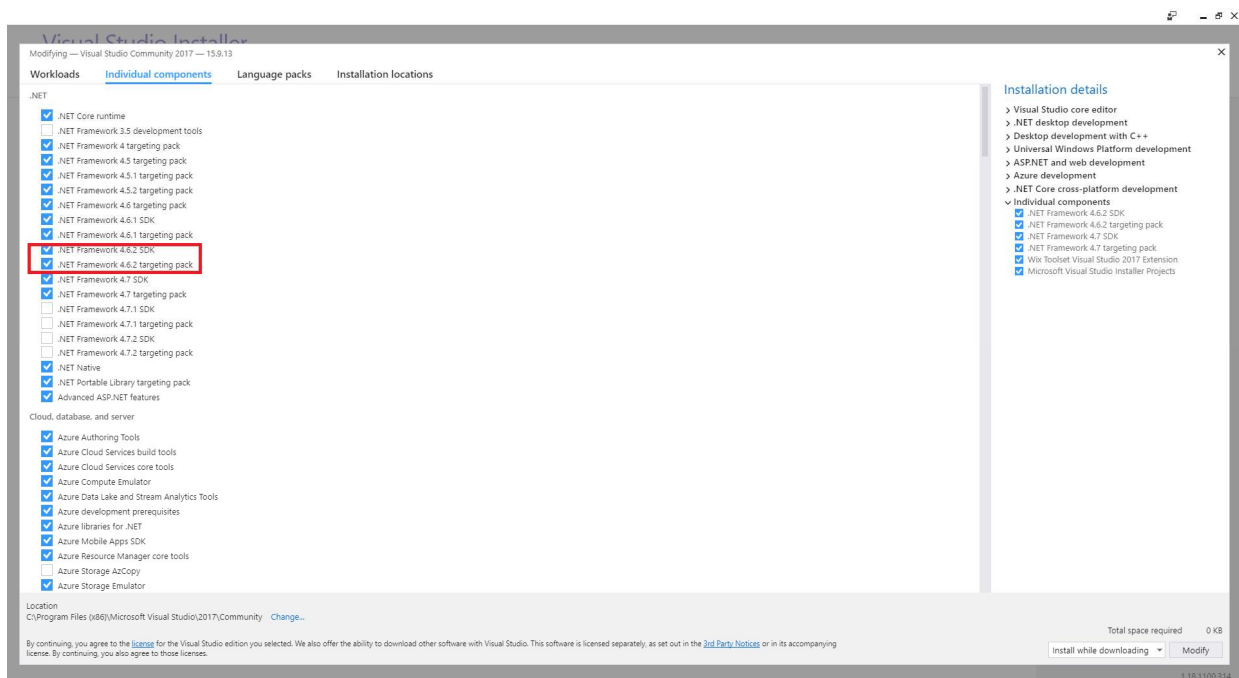
1. Open the Visual Studio Installer and click modify



2. Make sure the UWP box is ticked




3. Click the individual components tab at the top and ensure the correct .NET SDK boxes is ticked.



4. If any of the boxes weren't ticked and you have modified them. Click the modify button in the bottom right. This could take a while as it will download and install the required pieces automatically.

5. Download Wix. Head to <https://wixtoolset.org/releases/> and download WiX. Wix is a toolkit to build configurable Windows installers. Run the installer and click the install WiX Button on the GUI that pops up.



NEWSBUGS

WiX Toolset build tools includes everything you need to create installations on your development and build machines.

[DOWNLOAD WIX V3.11.1](#)

These have the latest changes, but may intro unexpected bugs. These are **not** considered t production-ready.

If you are still interested, you can find them [h](#)


WiX Toolset Visual Studio Extension this extension provides integration for the WiX Toolset into Visual Studio. You will still need to install the WiX Toolset v3.11.1 build tools.

Download the appropriate versions from the Visual Studio Marketplace:

- [WiX Toolset Visual Studio 2019 Extension](#)
- [WiX Toolset Visual Studio 2017 Extension](#)
- [WiX Toolset Visual Studio 2015 Extension](#)
- [WiX Toolset Visual Studio 2013 Extension](#)
- [WiX Toolset Visual Studio 2012 Extension](#)

Visual Studio | Marketplace

Visual Studio > Tools > Wix Toolset Visual Studio 2017 Extension



Wix Toolset Visual Studio 2017 Extension

WiX Toolset | 281,736 installs | 1,432,617 downloads | ★★★★★ (19) | Free

Visual Studio integration for the WiX Toolset - the most powerful set of tools available to create your Windows installation experience.

[Download](#)

[Overview](#) [Rating & Review](#)

Get the **WiX Toolset build tools** and then install this extension to get build integration with WiX in Visual Studio 2017.

To report bugs or feature requests please go to <http://wixtoolset.org/issues/>.

The WiX Toolset lets developers create installers for Windows Installer, the Windows installation engine.

- The core of WiX is a set of build tools that build Windows installer packages using the same build concepts

Categories

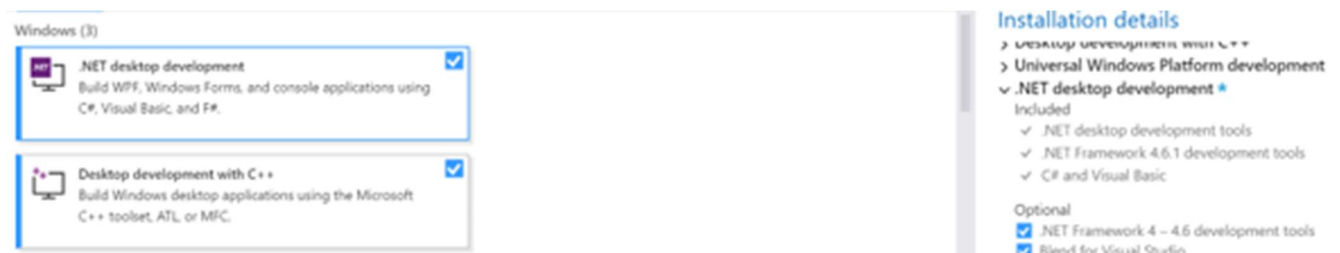
Tools Build Coding Setup & Deployment

Tags

MSI Setup Windows Installer Windows Installer XML WiX



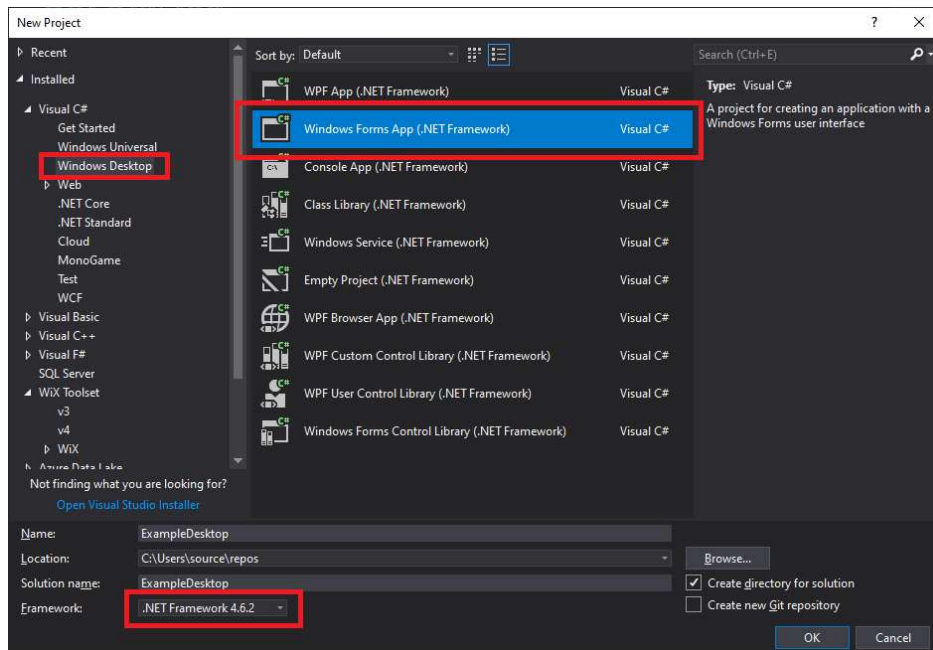
After installing Wix you will have to go back into the Visual Studio installer to add the Wix Extension:



Part 2: Create a project to package

1. Open Visual Studio and create a new project with “File” --> “New” --> “Project” and in the new window click “Visual C#” then “Windows Desktop” on the sidebar. And select Windows Forms App (.NET Framework). Down the bottom ensure the framework is .NET Framework 4.6.2 or newer. Finally click “Ok” to create the new solution.

Note: The requirement of C# and .Net Framework version is only a requirement for Part 4. Part 3 building a traditional installer using WiX can package up other types of applications. Including c++ console applications for example.



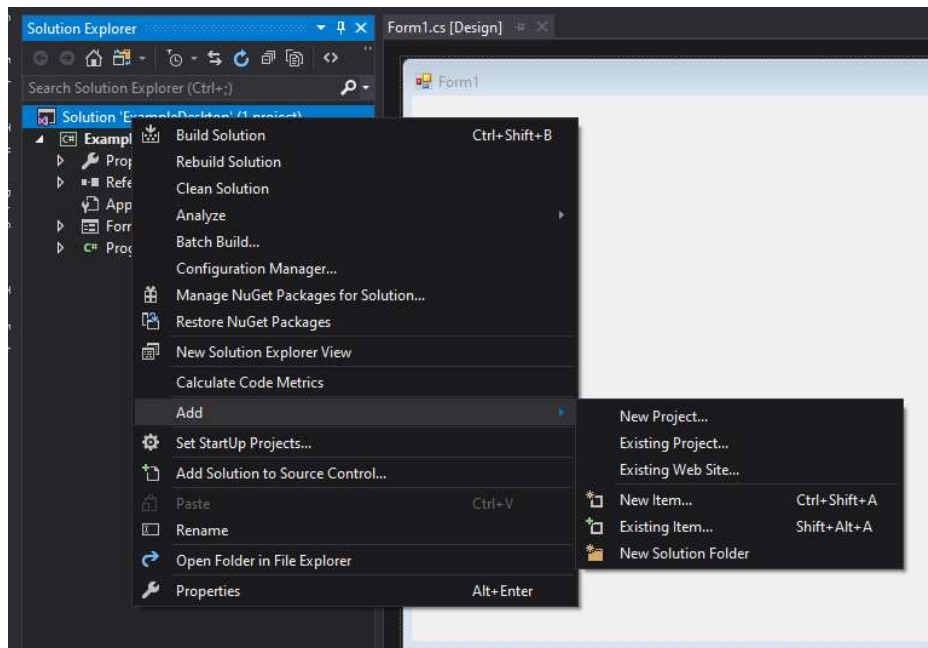
2. Test the project template works by clicking start. Wait a bit and if a window pops up that looks like the form1.cs then it has built successfully.

If you're familiar with C# and Winforms then feel free to customize the Window. But don't spend too much time on it!

Part 3 – WiX

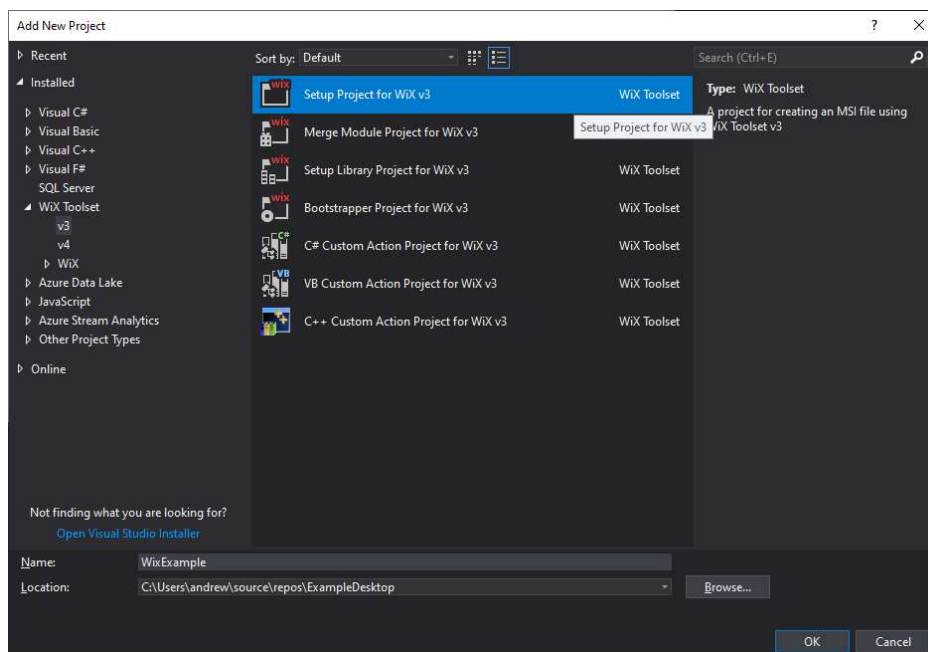
WiX is the Windows Installer XML Toolset. It's a toolset that allows the description of a .msi installer file with XML. It can be used to create an installer for most applications. In part 3 we'll make a hello world installer.

1. Right click on the Solution in the Solution Explorer and go "Add" --> "New Project"



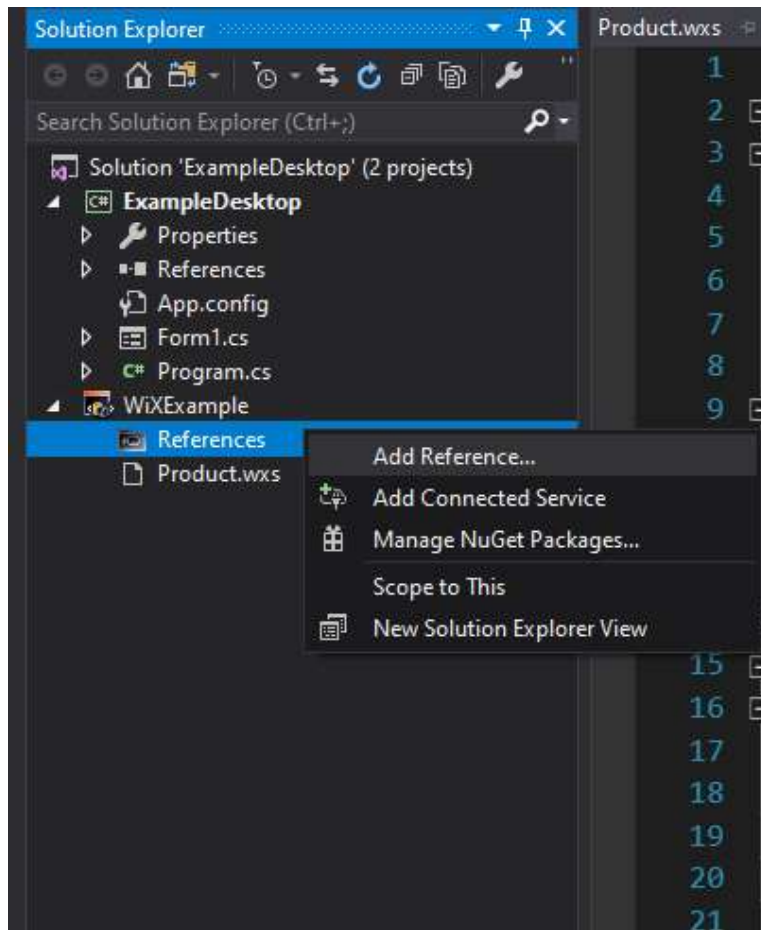
2. In the new window click “WiX Toolset” and “v3” in the sidebar. Then select “Setup Project for WiX v3”

Name the installer project and click “OK”

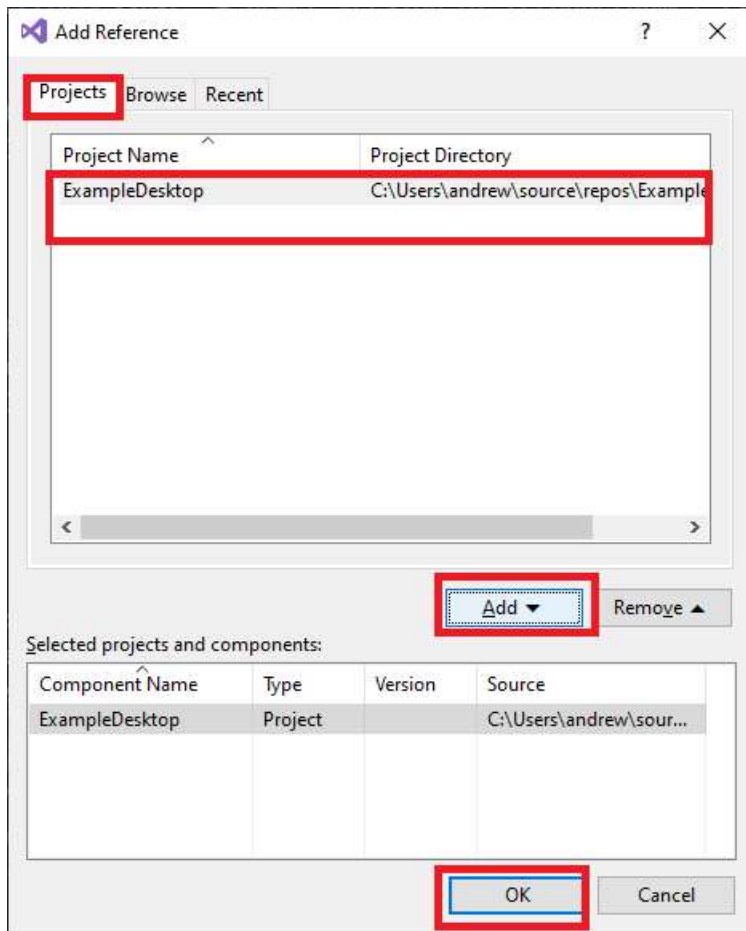


This will add a WiX project to the solution.

3. Create a reference to the application we want to package in the WiX project. Do this by right clicking references and selecting add reference.



4. In the new window click the projects tab. Click the project we want to include, then click add. When that's done click "OK"



5. In the newly created Product.wxs file find the <ComponentGroup> tag.

Inside that tag we need to create a <Component> tag for everything that must be included in the installer. For our simple application we only need to create one for the executable file.

By default the executable file generated by visual studio are named after the project. e.g. My project is ExampleDesktop so the exe that is built is called ExampleDesktop.exe

The format for the tag is:

```
<Component Id="Filename.exe" Guid="aa8986dc-8c10-4148-998d-4caa2f91dcb5" >  
  <File Id="Filename.exe" Name="Filename.exe" Source="c:/Path/To/Filename.exe"></File>  
</Component>
```

We can replace the Filename.exe with ExampleDesktop.exe:


```
<Component Id="ExampleDesktop.exe" Guid="aa8986dc-8c10-4148-998d-4caa2f91dcb5" >
  <File Id="ExampleDesktop.exe" Name="ExampleDesktop.exe" Source="c:/Path/To/Filename.exe"> </File>
</Component>
```

The Source property describes where the file to be included is. Because this can change based on the projects settings we can future proof it by using the preprocessor to generate it for us.

```
Source="$$(var.ExampleDesktop.TargetDir)ExampleDesktop.exe"
```

When the compiler runs it will evaluate var.ExampleDesktop.TargetDir to the target directory of the ExampleDesktop project.

Finally, we need to create a GUID ('Globally Unique Identifier', outside of the Microsoft world it is often referred to as a UUID 'Universally Unique Identifier'). This can be done via online generators such as <https://www.guidgenerator.com/> or within Visual Studio itself by going

"Tools" --> "Create GUID" and copying to the clipboard. You will have to fix the formatting, the easiest would be creating one in "Registry Format" as you just need to remove the { } at the beginning and end.

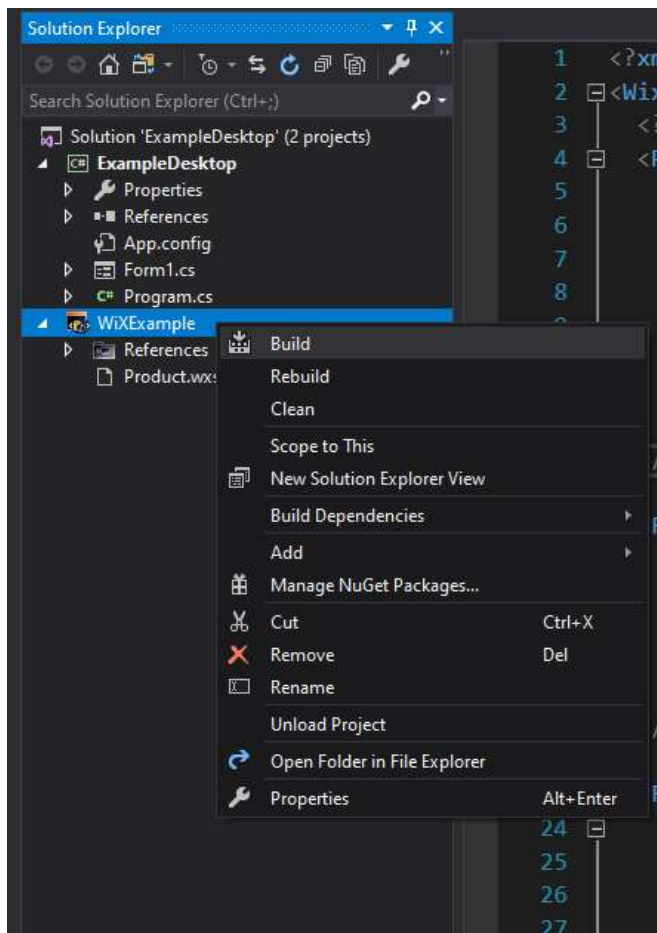
Place that into the GUID section.

7. Final touches in the Product.wxs file.

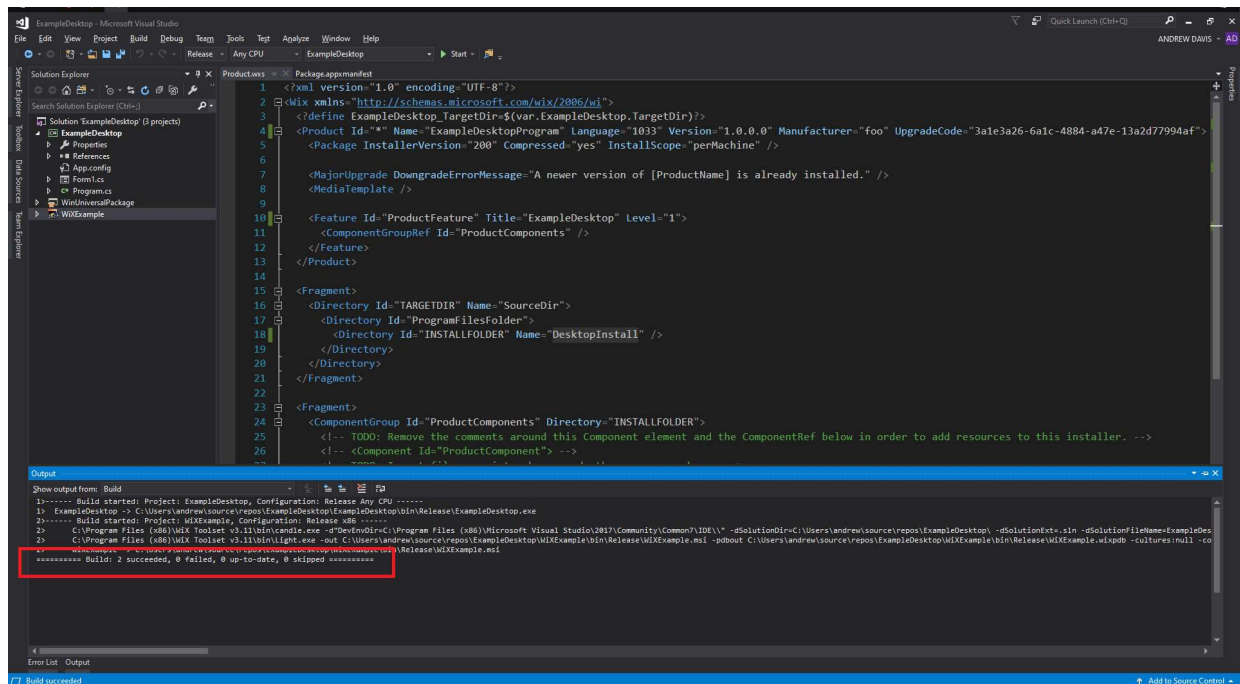
- In the <Product> tag:
 - put your name as the manufacturer e.g. Manufacturer="ICT Student"
 - Set the name to something appropriate. This is what will be displayed in Windows "add remove programs" settings.
- Find the <Directory> tag with the Id="INSTALLFOLDER". Set the name to something appropriate, this will be the name of the directory created in Program files. e.g. if Name="My Example Installer" it will install the program to "c:/Program Files/My Example Installer"
- By default each file will be placed in a .cab file that the installer will use. All files can be bundled into the msi installer by editing the <MediaTemplate/> tag to <MediaTemplate EmbedCab="yes" />

8. Ready to build.

Right click on the WiXExample project in the solution explorer and click “Build”



If all went well it should have built successfully



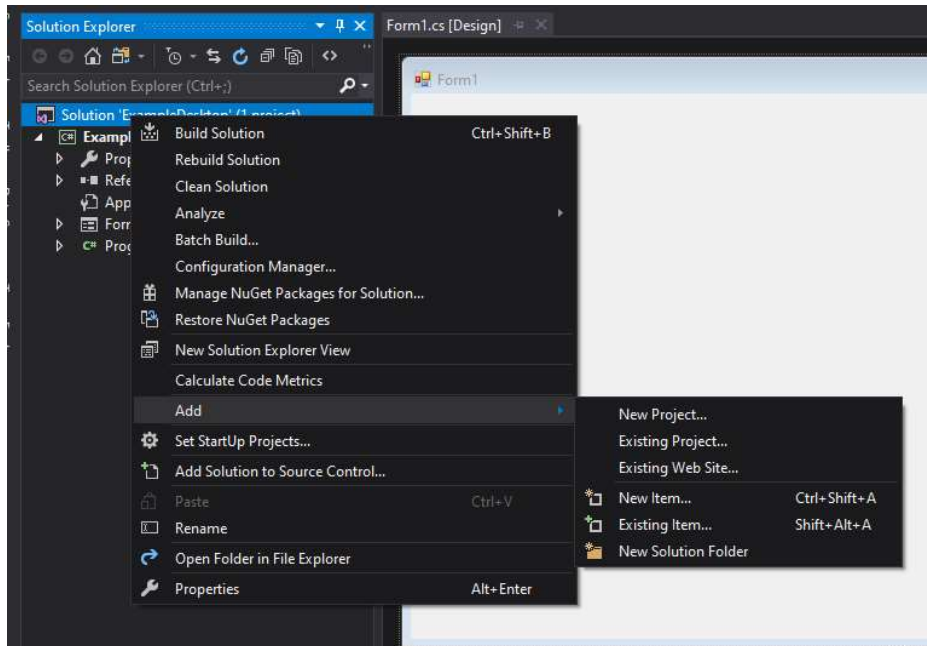
9. Test your installer. Navigate to the directory of the Visual Studio Solution. You can do this by right clicking at the top of the solution explorer and selecting “Open Folder in File Explorer”

Navigate to the WiX project (in the example it’s WixExample) then WixExample/bin/release and you’ll find 3 files including the .msi file which is the installer. Double click on the installer to install your application.

Part 4 – Universal Windows App Packaging

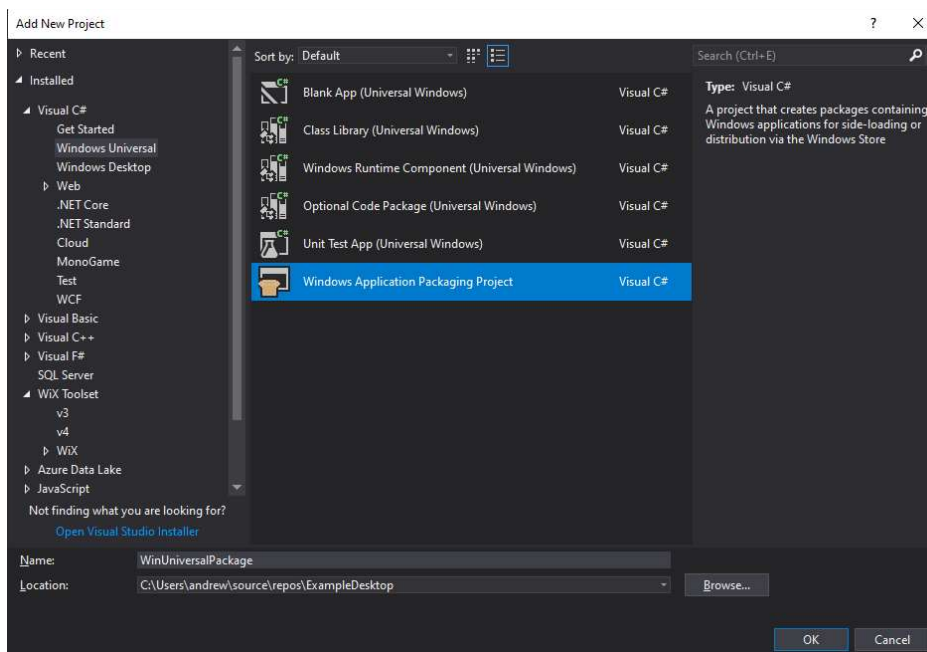
In addition to traditional installers Microsoft also now has the Windows store. These installers are quite different and far more restrictive than traditional installers. The process is quite straightforward and sideloading of the packages (ie not using the store directly) is supported.

1. Right click on the Solution in the Solution Explorer and go “Add” --> “New Project”



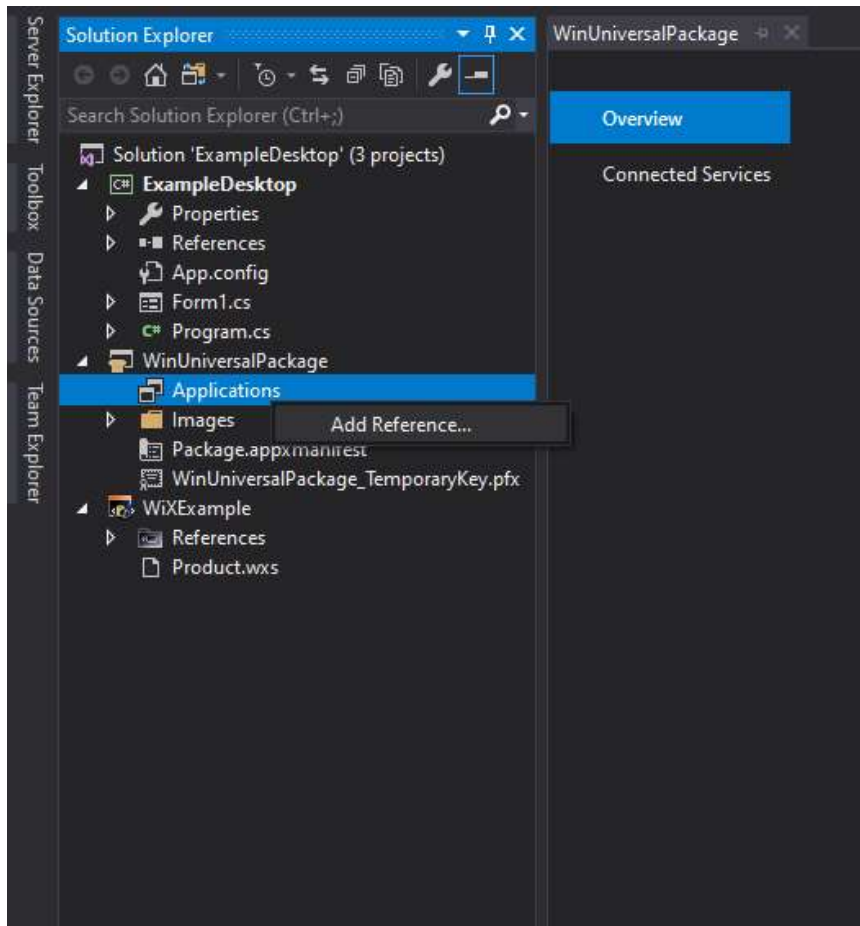
2. In the new window click “Visual C#” and “Windows Universal” in the sidebar. Then select “Windows Application Packaging Project”

Name the packaging project and click “OK”

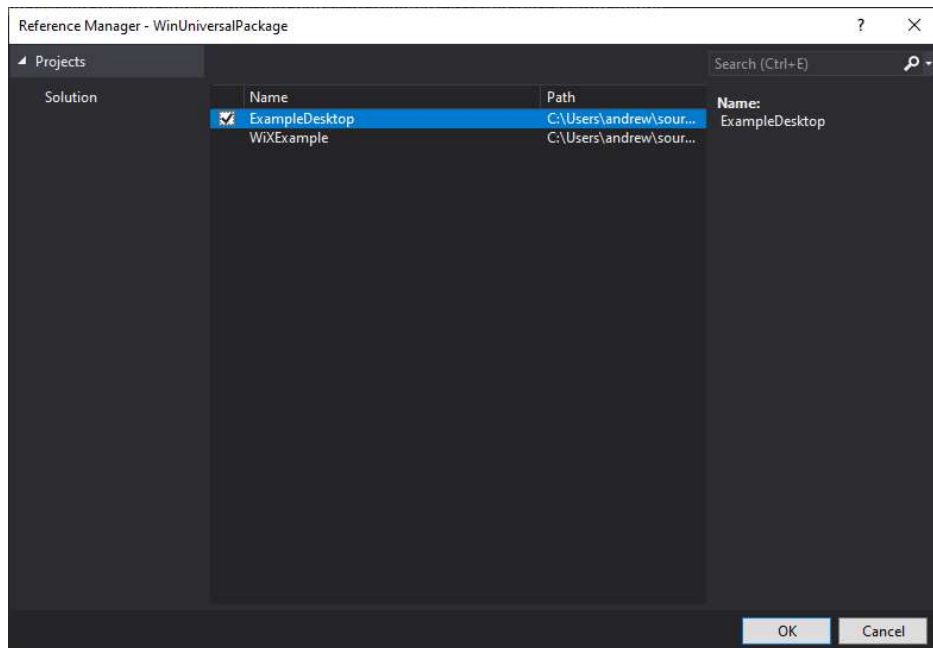


A dialog box will appear for your target version of Windows and .NET. Just accept the defaults for this project.

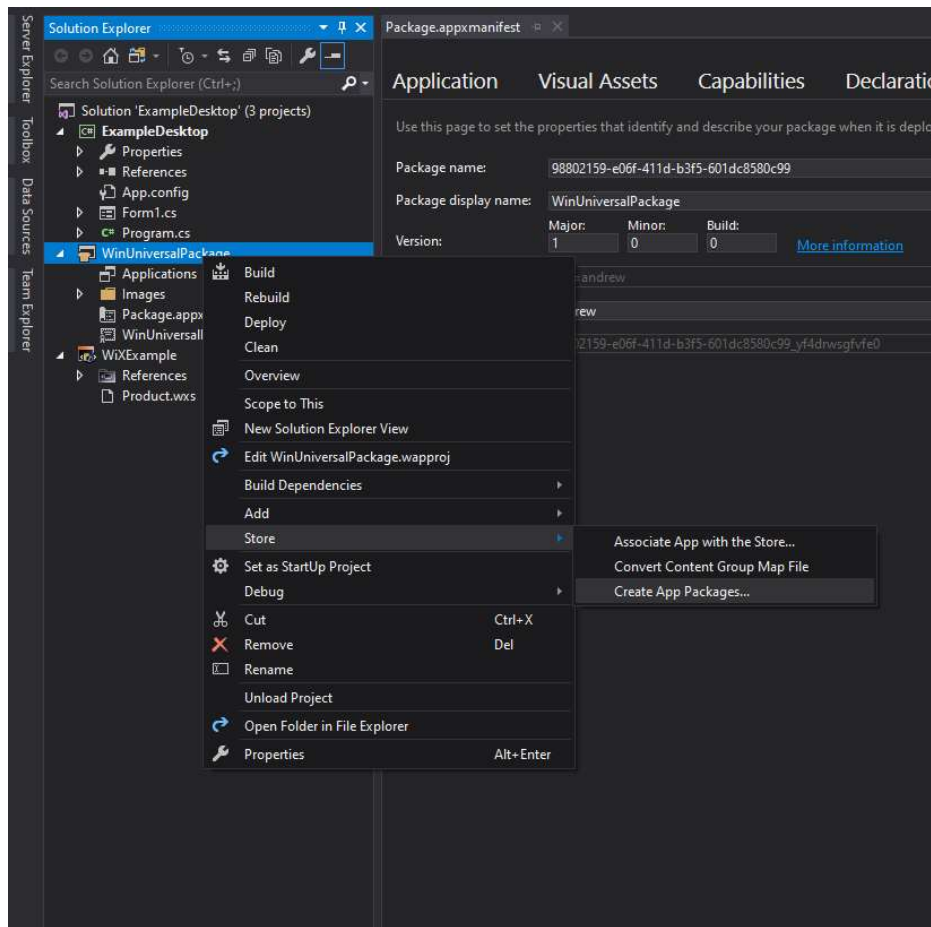
3. Add a reference to the project to build by right clicking on “applications” in the solution explorer in our new Universal package project and clicking add reference.



Then tick the project you want to install



4. Right click on the project and go "Store" --> "Create App Packages Wizard"
- This will open a Wizard to walk through creating the package.



5. Select the “I want to create packages for sideloading” then click “Next”
This allows us to create a package that can be distributed freely, versus uploading to the Windows store and
6. Here is where we can set a version number for our package and select an architecture for our output. If we leave it as neutral it will be compatible with all platforms UWP is. Finally click “Create”

Create App Packages

Select and Configure Packages

Output location:
 C:\Users\andrew\source\repos\ExampleDesktop\WinUniversalPackage\AppPackages\

Version:
 1 . 0 . 0 . 0

☒ Automatically increment
[More information](#)

Generate app bundle:
 Always
[What does an app bundle mean?](#)

Select the packages to create and the solution configuration mappings:

Architecture	Solution Configuration
<input checked="" type="checkbox"/> Neutral	Release (Any CPU)
<input type="checkbox"/> x86	Release (x86)
<input type="checkbox"/> x64	Release (x64)
<input type="checkbox"/> ARM	None

☒ Include full PDB symbol files, if any, to enable crash analytics for the app. [Learn More](#)

Previous Create Cancel

A build will start and if successful a new window will open saying “Package Creation Completed” along with options to have your app certified. We’re going to skip that. Instead click on the link under “Output Location” to open explorer.

7. Explorer will have a group of files. The key files is the .appxbundle file. If you double click on that it will bring up a dialog box to install your application. But wait! It won’t work just yet. All UWP packages are signed by a certificate and can’t be installed (including sideloading) unless the target machine trusts the certificate.

You can configure your certificate in the Package.appxmanifest file in the package project under the “Packaging” tab. By default Visual Studio creates a certificate for us and includes it in the output of our package directory.

In the output location double click on the .cer file, this is the certificate. Click “install certificate” and choose “Local Machine”

Choose the “Place all certificates in the following store” then “Browse” to “Trusted Root Certification Authorities” and click “OK”, “Next” and “Finish”

Now you’re ready to install your application.

8. Double click on the .appxbundle file and follow the prompts. Your application should install.