ETLIPSE REVIT ADDIN WPF VISUAL STUDIO TEMPLATE – INSTRUCTIONS

Hello, world! These are some instructions and steps you should follow in order to use our Visual Studio template for a Revit add-in application with multiple commands in a modeless dialog.
(download it here)

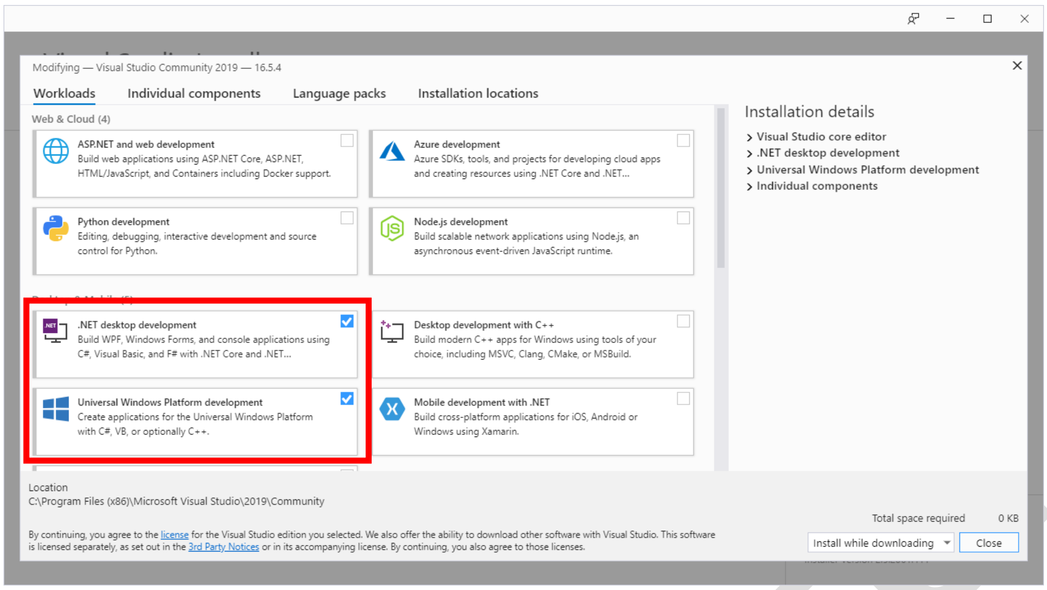
It is important to mention that this is a work based and developed after studies on the *Revit SDK* Windows Forms-based example project named *ModelessForm_ExternalEvent*. So, we came up with a solution that adapts the logic in the example to a simple and useful WPF application project that can be used as a template.

The code in this template is commented with a lot of explanation about each class, method and interaction used in the project. Below you have the instructions and some troubleshooting tips at the end of this document. Let's go!

General Instructions

1. The first thing you'll need, of course, is a copy of **Autodesk Revit 2019** or **2020** installed on your computer. Although we have not tested our template for other versions, if this is your case, you can see the steps in the last section of this document, *Troubleshooting*.
2. The second thing you need is **Microsoft Visual Studio** installed on your computer. Our template is for this specific program. If you don't have a copy installed already, you can easily get its current free edition, *Visual Studio Community 2019*, here, so you can perform everything you need to create your Revit add-ins.
3. Keep in mind that the installation process of Visual Studio can be lengthy. But you should have no problem with the dialogs and creating or using a Microsoft account. The only important decision you must pay attention to is about the packages you will want to install. To be able to use this template, you just have to check the *.NET desktop development* and the *Universal Windows Platform development* workloads (as shown in the image below). For now, this will suffice and you can finish the installation, but remember you can always run the Visual Studio Installer again in case you need to install other packages for your future applications.

eTLipse - Computação, Engenharia e Responsabilidade Social Ltda.
www.etlipse.com | info@etlipse.com
Developed by Joel Diniz and Edson Andrade


Modifying — Visual Studio Community 2019 — 16.5.4

Workloads | Individual components | Language packs | Installation locations

Web & Cloud (4)

ASP.NET and web development
Build web applications using ASP.NET Core, ASP.NET, HTML/JavaScript, and Containers including Docker support.

Azure development
Azure SDKs, tools, and projects for developing cloud apps and creating resources using .NET Core and .NET...

Python development
Editing, debugging, interactive development and source control for Python.

Node.js development
Build scalable network applications using Node.js, an asynchronous event-driven JavaScript runtime.

.NET desktop development ✓
Build WPF, Windows Forms, and console applications using C#, Visual Basic, and F# with .NET Core and .NET...

Desktop development with C++
Build modern C++ apps for Windows using tools of your choice, including MSVC, Clang, CMake, or MSBuild.

Universal Windows Platform development ✓
Create applications for the Universal Windows Platform with C#, VB, or optionally C++.

Mobile development with .NET
Build cross-platform applications for iOS, Android or Windows using Xamarin.

Installation details
- Visual Studio core editor
- .NET desktop development
- Universal Windows Platform development
- Individual components

Location
C:\Program Files (x86)\Microsoft Visual Studio\2019\Community

By continuing, you agree to the license for the Visual Studio edition you selected. We also offer the ability to download other software with Visual Studio. This software is licensed separately, as set out in the 3rd Party Notices or in its accompanying license. By continuing, you also agree to those licenses.

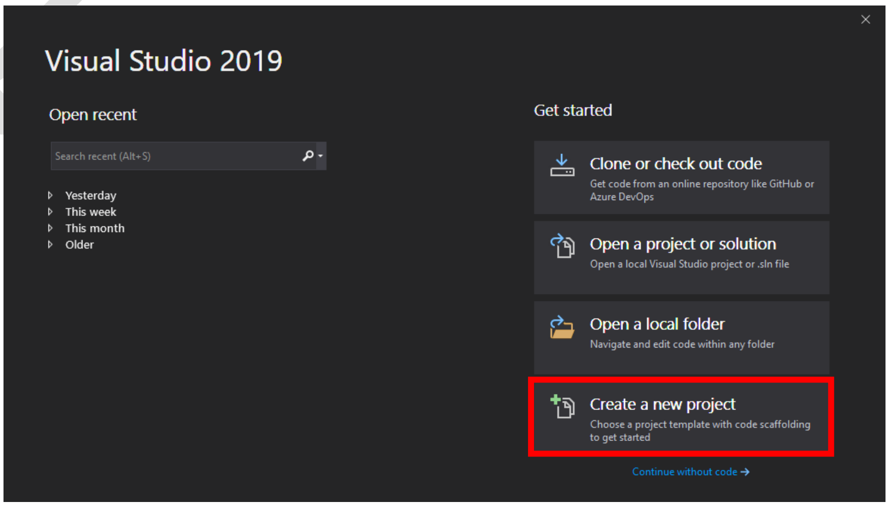
Total space required 0 KB

4. With Visual Studio installed, download our template file here. The .zip files are named after each version of Revit the template was tested for. So, for example, if you have Revit 2019 on your machine, you should download the *TL Revit 2019 WPF Addin.zip* file. Remember that in case you have other version than Revit 2019 or 2020 installed, you can see the last procedure in the *Troubleshooting* section at the end of this document.
5. Now it's time to store the template in the right place. Copy the .zip file you just downloaded to the following folder:

```
C:\Users\<User>\Documents\Visual Studio ****\Templates\ProjectTemplates\
```

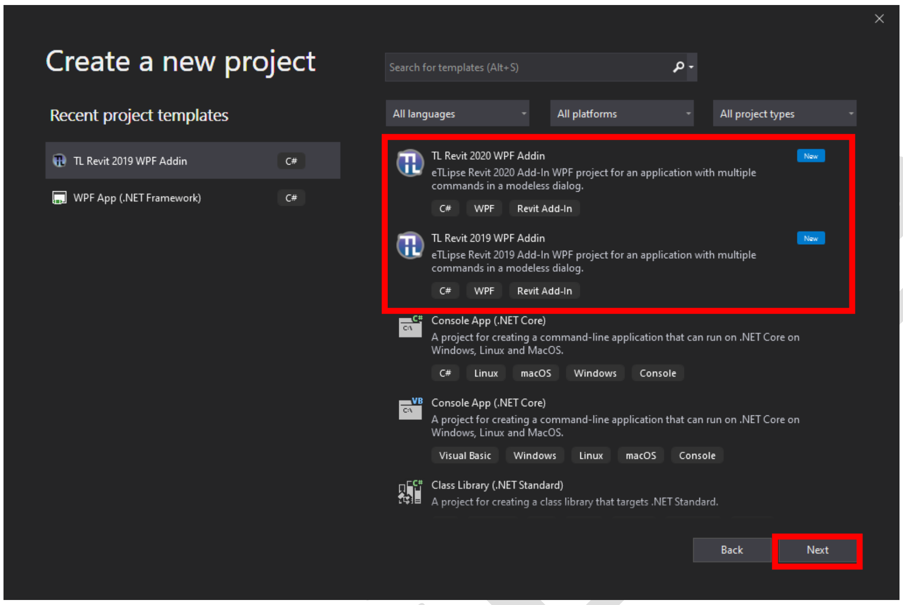
Where "<User>" refers to the user folder in Windows and "****" to the version of Visual Studio you have ("2019", for example).

6. Launch Visual Studio and click the *Create a new project* button (image below).



Visual Studio 2019
Open recent
Search recent (Alt+S)
Yesterday
This week
This month
Older
Get started
Clone or check out code
Get code from an online repository like GitHub or Azure DevOps
Open a project or solution
Open a local Visual Studio project or .sln file
Open a local folder
Navigate and edit code within any folder
Create a new project
Choose a project template with code scaffolding to get started
Continue without code →

7. Now, if you copied the .zip file to the right place, you should see the option to select the *TL Revit WPF Addin* template, according to your Revit version, and start your project (image below). Select this option and click *Next*.


Create a new project

Recent project templates

TL Revit 2019 WPF Addin — C#

WPF App (.NET Framework) — C#

Search for templates (Alt+S)

All languages | All platforms | All project types

TL Revit 2020 WPF Addin — New
eTLipse Revit 2020 Add-In WPF project for an application with multiple commands in a modeless dialog.
C# | WPF | Revit Add-In

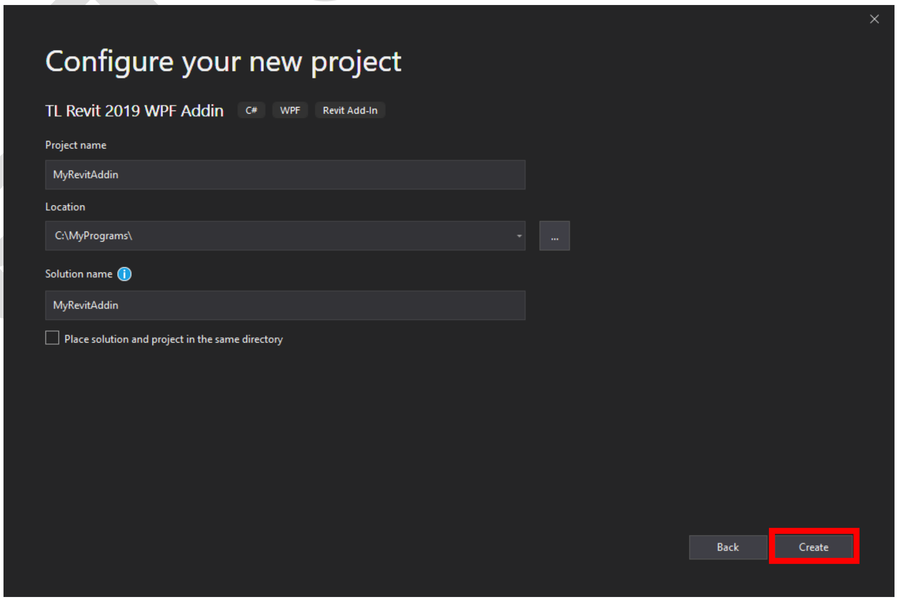
TL Revit 2019 WPF Addin — New
eTLipse Revit 2019 Add-In WPF project for an application with multiple commands in a modeless dialog.
C# | WPF | Revit Add-In

Console App (.NET Core)
A project for creating a command-line application that can run on .NET Core on Windows, Linux and MacOS.
C# | Linux | macOS | Windows | Console

Console App (.NET Core)
A project for creating a command-line application that can run on .NET Core on Windows, Linux and MacOS.
Visual Basic | Windows | Linux | macOS | Console

Class Library (.NET Standard)
A project for creating a class library that targets .NET Standard.

8. Now you are asked to name your project (and your solution), as well as set the location for it. In the image below you see just an example. Choose any names and folder you prefer. After you're done, click *Create* and your project is set.


Configure your new project

TL Revit 2019 WPF Addin — C# | WPF | Revit Add-In

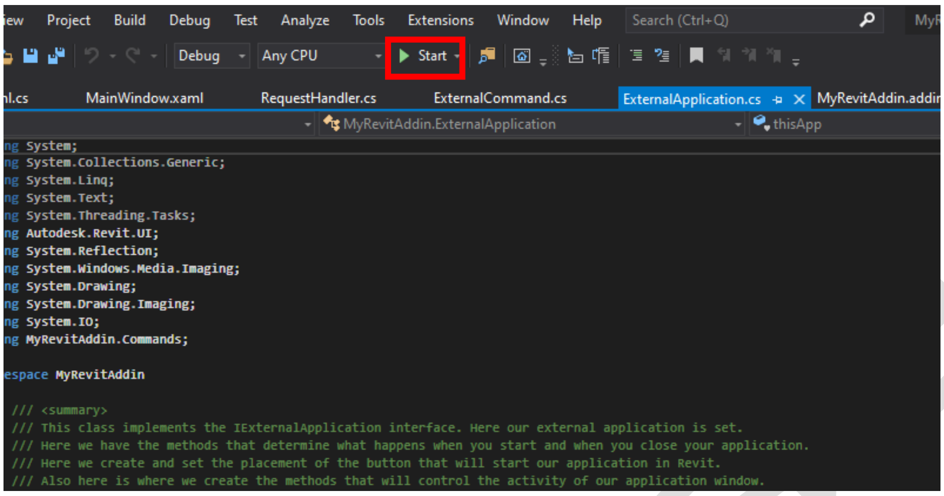
Project name: MyRevitAddin

Location: C:\MyPrograms\

Solution name: MyRevitAddin

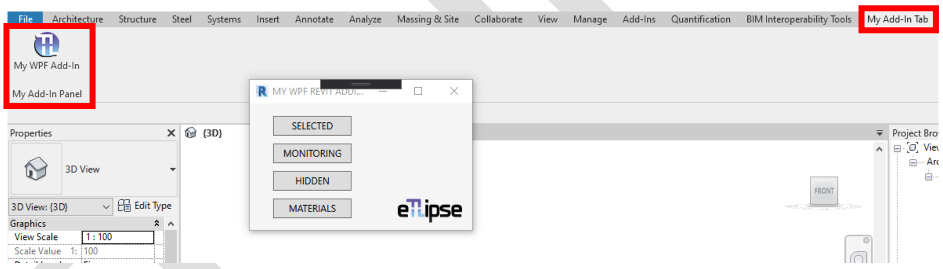
☐ Place solution and project in the same directory

9. Before interacting with any of the project items in the template, we recommend that you simply click on the *Start* button (image below) to build and run the add-in.

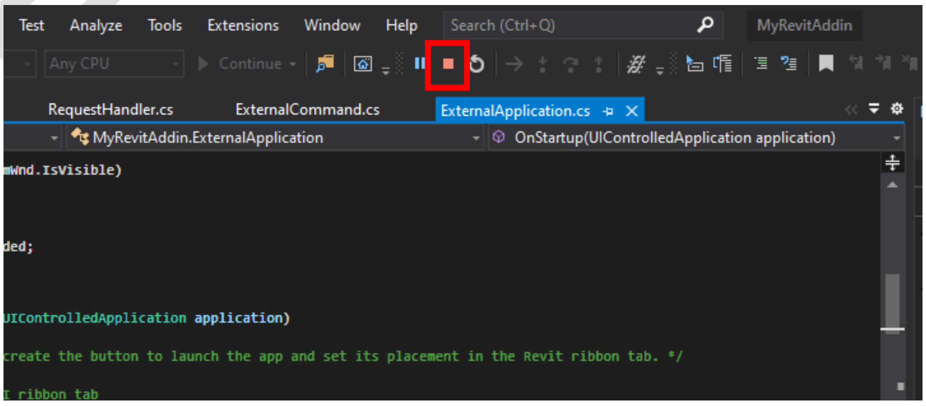


```
ng System;
ng System.Collections.Generic;
ng System.Linq;
ng System.Text;
ng System.Threading.Tasks;
ng Autodesk.Revit.UI;
ng System.Reflection;
ng System.Windows.Media.Imaging;
ng System.Drawing;
ng System.Drawing.Imaging;
ng System.IO;
ng MyRevitAddin.Commands;

espace MyRevitAddin

 /// <summary>
 /// This class implements the IExternalApplication interface. Here our external application is set.
 /// Here we have the methods that determine what happens when you start and when you close your application.
 /// Here we create and set the placement of the button that will start our application in Revit.
 /// Also here is where we create the methods that will control the activity of our application window.
```

10. An instance of your Revit application should initialize. Open any Revit project and then execute the add-in by accessing the ribbon tab *My Add-In Tab*, and then clicking on the *My WPF Add-In* button in the *My Add-In Panel* (image below).



11. Now, play around with the commands (buttons) in the add-in user interface. Once you are done, go back to Visual Studio and click on the *Stop Debugging* button (image below).

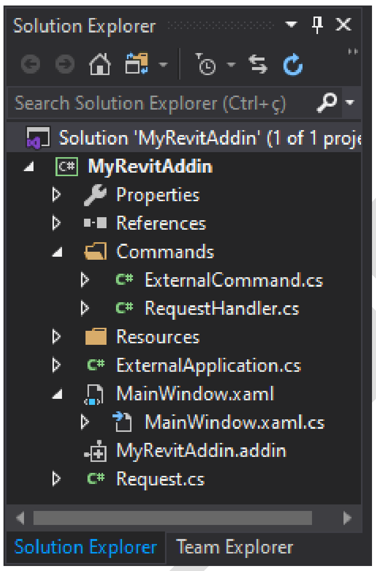


```
mWnd.IsVisible)


ded;


UIControlledApplication application)

create the button to launch the app and set its placement in the Revit ribbon tab. */

I ribbon tab
```

If you encountered any error or warning message during the steps above, you can head straight to the *Troubleshooting section*, at the end of this document.

Project Items, References and Properties

Your project is open in Visual Studio and you have the following structure in your Solution Explorer:



Properties: here you have very important settings of your project. You define what platform and application will be used in debugs, what location your output files are going to be copied to, as well as essential project information, like assembly and namespace names.

References: this is where you will find and add all references like libraries, external solutions and frameworks that you'll need in your project. Two specific references are required to use the Revit API items, methods and interactions in this template: the *RevitAPI.dll* and the *RevitAPIUI.dll*. They are already added in the project, but if you encounter any warning or error regarding these two libraries, see the *Troubleshooting* section at the end of this document.

ExternalApplication.cs: *C#* code. This class implements the *IExternalApplication interface*. Here your external application is set. Here you have the methods that determine what happens when you start and when you close your application. Here you create and set the placement of the *button* that will start your application in Revit. Also here is where you create the methods that will control the activity of your application *window*.

ExternalCommand.cs: *C#* code. This class implements the *IExternalCommand interface* and contains the command to execute the application, which will be called by the *button* created in the application class.

RequestHandler.cs: *C#* code. This is the class that will implement the *IExternalEventHandler interface* to handle all commands started by user action in the *MainWindow* (modeless dialog) as requests listed in an enumeration used by the *Request* class. Also, here you will define all the methods that will build the application functionality using the Revit API.

Request.cs: C# code. This is the class that will take the user command by the *RequestId enum* and make the request once the *RequestHandler* identifies it while the external event is being raised.

MainWindow.xaml: *WPF* code. This is the window containing the buttons that will convert user actions into requests for Revit.

MainWindow.xaml.cs: *C#* code. This is the code behind of the *MainWindow* control, which presents the application main user interface and we want to work as a modeless dialog. Here you set methods that should be called by other classes to control the window activity. Here you also set the interactions between the user input through the window controls and the classes that build the commands that Revit should receive as external event requests.

MyRevitAddin.addin: *xml* code. This is the *manifest* file with reference to the application class and the name of the output *dll* of your add-in. Revit needs this to run your application. By default in our template this will be named after your project name.

With this initial knowledge it's time for you to shine! We recommend, for beginners, going to the *RequestHandler* class and edit the methods you see there in the *METHODS TO EXECUTE THROUGH DELEGATE* section with some code of your own. Practice your skills and knowledge of the Revit API through these multiple commands. Notice that every time you click on *Start* in Visual Studio, before running the Revit application, it will copy a *.pdb*, your *manifest (.addin)* and a *.dll* files of your application in the Revit *Addins* user folder in your machine. So, you should delete these files prior to every time you need to run a new test to see your solution in action. The folder where you normally can find them is this:

```
C:\Users\<User>\AppData\Roaming\Autodesk\Revit\Addins\****\
```

Where "<User>" refers to the user folder in Windows and "****" to the version of Revit you have ("2019", for example).

These output file copies have their path set in the *Properties* of your project in Visual Studio. If you go there, you'll see in the *Build Events* section, a line in a text box for *Post-build event command line*. This is the line that send the copies of the files in your Debug to the Revit *Addins* user folder.

Basically, the safest workflow is this: you code> you delete old files in the addins folder> you run the solution> you test it in Revit>you stop the debugging> (repeat).

Once you are done testing the core functionalities of your app, you can explore the *MainWindow* control and other interactions in the template to fully develop your own add-in. We hope this template will be helpful for many people.

External Links and other References

These are some extra material that may be useful as well (click the title of each item to access/download it):

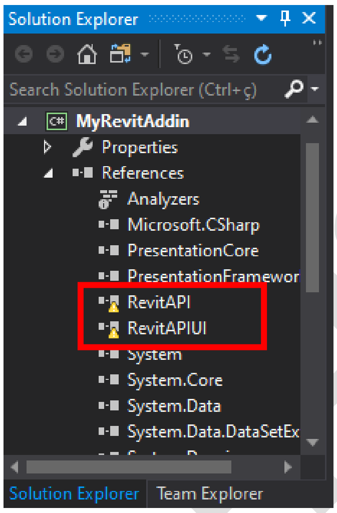
- If you want to make your add-in files management less repetitive, take a look at the *Add-In Manager*.
- Our guide *Basics of Revit Add-In Programming*, a separate section of our general guide (below), focusing on the basics of the Revit API for add-ins.
- Our guide *Use of Programming in BIM Methodology*, where we work the basics of the Revit API for add-ins and *Dynamo* workflows (currently PORTUGUESE language only).
- Our *TL SOLIDS Interaction Tool* Revit add-in, useful for solids operations in Revit (Join, Cut, Switch Join Order) and checks (intersections, joined elements) with a nice set of visualization and selection tools.
- If you look for a non-WPF-based useful Revit add-in template, we recommend this great *BIM² Revit Add-In Template* from *James Simpson*.
- Also, if you look for good publications and code references to study the Revit API for add-ins, we recommend *The Building Coder* – *Jeremy Tammik*'s blog and probably the most popular and renowned reference for Revit API knowledge – and also the *Torsion Tools* GitHub repository, by *Sean Page*, with a lot useful code being shared.

Troubleshooting

The tips you can find in this section may be of help to solve some of the problems you may encounter while using any Revit add-in template for Visual Studio.

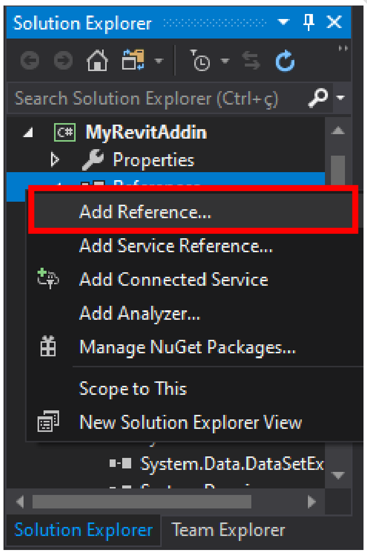
Revit References must be found

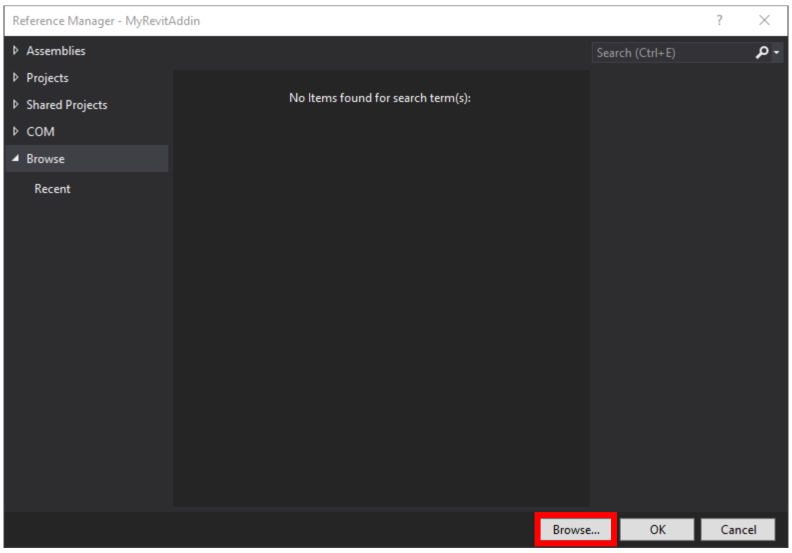
In some cases, you may encounter the following problem in the *References* of your project:



This happens when your project cannot find the respective Revit libraries. How do you fix this?

1. Right click on *References* and then *Add Reference*. Then a dialog opens, you can just click *Browse* (images below).





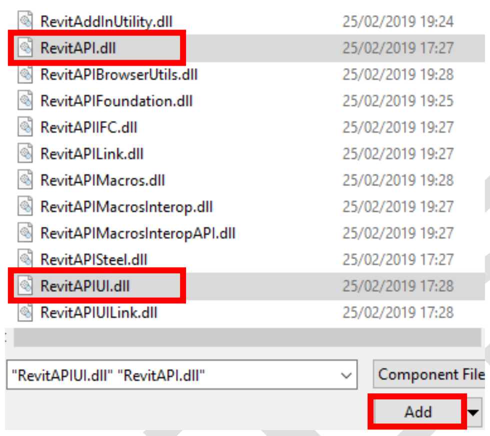
No items found for search term(s):

2. A new dialog appears. Go to the following folder (or the folder where your Revit copy is installed):

```
C:\Program Files\Autodesk\Revit ****\
```

Where "****" refers to the version of Revit you have ("2019", for example).

3. Now, select both the *RevitAPI.dll* and the *RevitAPIUI.dll* files (you can use the *ctrl* key to help with this task) and click *Add* (image below), then *OK* and you're done!

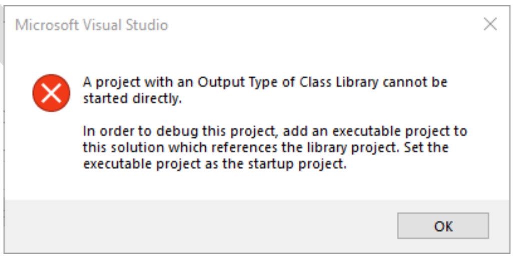


RevitAddInUtility.dll	25/02/2019 19:24
RevitAPI.dll	25/02/2019 17:27
RevitAPIBrowserUtils.dll	25/02/2019 19:28
RevitAPIFoundation.dll	25/02/2019 19:25
RevitAPIIFC.dll	25/02/2019 19:27
RevitAPILink.dll	25/02/2019 19:27
RevitAPIMacros.dll	25/02/2019 19:28
RevitAPIMacrosInterop.dll	25/02/2019 19:27
RevitAPIMacrosInteropAPI.dll	25/02/2019 19:27
RevitAPISteel.dll	25/02/2019 17:27
RevitAPIUI.dll	25/02/2019 17:28
RevitAPIUILink.dll	25/02/2019 17:28

"RevitAPIUI.dll" "RevitAPI.dll"

Revit must be set as the Start application for Debugging

Everything is alright with the code, but you clicked *Start* in Visual Studio and, instead of getting the Revit splash screen, you only got this:

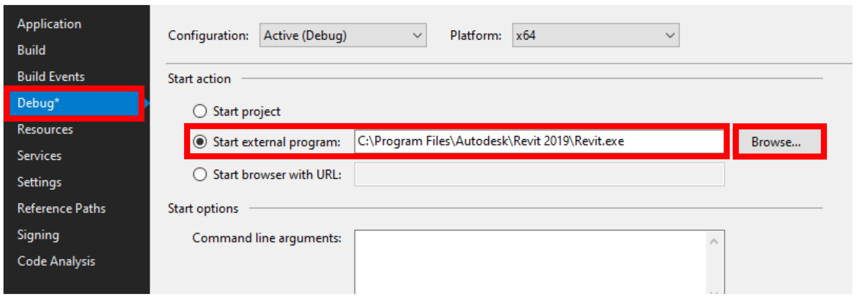


Microsoft Visual Studio

A project with an Output Type of Class Library cannot be started directly.

In order to debug this project, add an executable project to this solution which references the library project. Set the executable project as the startup project.

This happens because Revit is not indicated as the external program that will start at the debugging of your application. To fix this, simply access the *Properties* of your project, go to the *Debug* section on the left, check *Start external program* and browse for the *Revit.exe* file in the file location displayed in the image below:



Configuration: Active (Debug) Platform: x64

Start action

Start project

Start external program: C:\Program Files\Autodesk\Revit 2019\Revit.exe Browse...

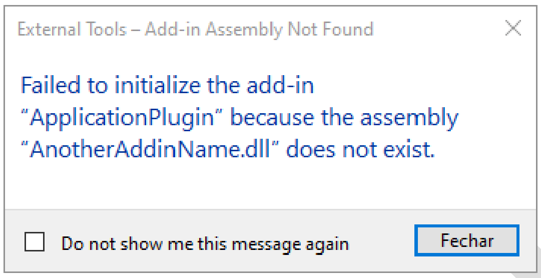
Start browser with URL:

Start options

Command line arguments:

The assembly and the application class must be correctly referenced

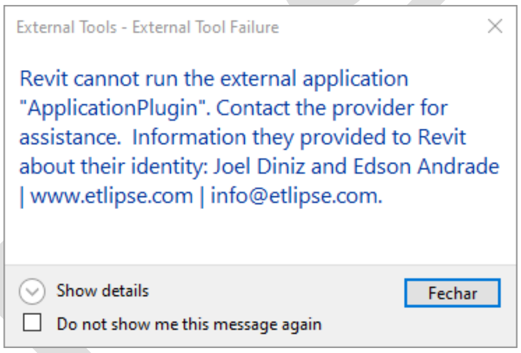
Having some of the following dialogs showing when trying to test your application?



External Tools – Add-in Assembly Not Found

Failed to initialize the add-in "ApplicationPlugin" because the assembly "AnotherAddinName.dll" does not exist.

Do not show me this message again Fechar



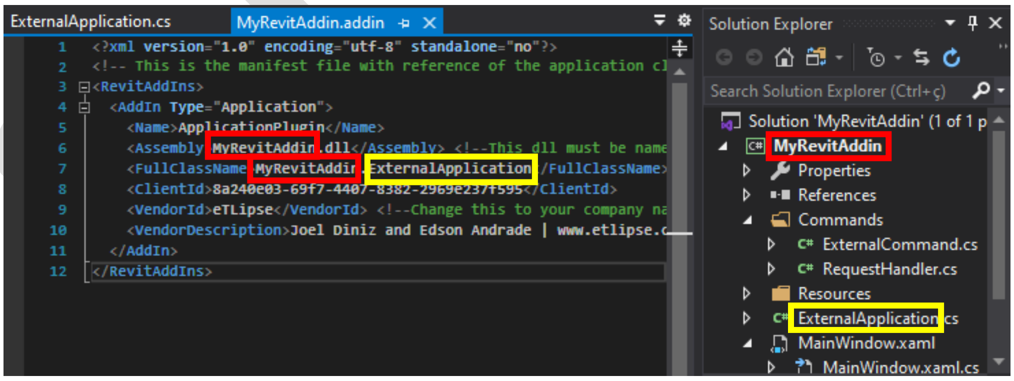
External Tools - External Tool Failure

Revit cannot run the external application "ApplicationPlugin". Contact the provider for assistance. Information they provided to Revit about their identity: Joel Diniz and Edson Andrade | www.etlipse.com | info@etlipse.com.

Show details Fechar

Do not show me this message again

This happens when the manifest file fails to point to the right files due to incorrect reference. To simplify things and avoid problems, we recommend that you name your *dll* after your project name and double-check if the namespace of your external application class is correctly written in the manifest file (see image below).



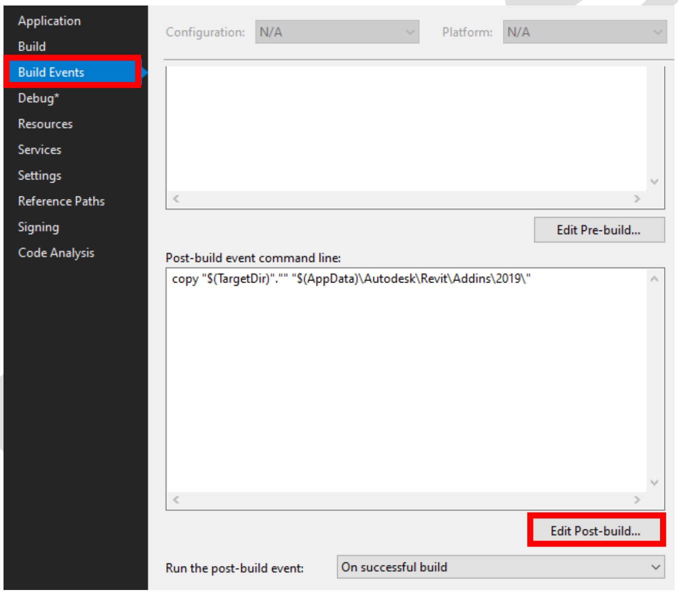
ExternalApplication.cs MyRevitAddin.addin

```
<?xml version="1.0" encoding="utf-8" standalone="no"?>
<!-- This is the manifest file with reference of the application cl
<RevitAddIns>
  <AddIn Type="Application">
    <Name>ApplicationPlugin</Name>
    <Assembly>MyRevitAddin.dll</Assembly> <!--This dll must be name
    <FullClassName>MyRevitAddin.ExternalApplication</FullClassName>
    <ClientId>8a240e03-69f7-4407-8382-2969e237f595</ClientId>
    <VendorId>eTLipse</VendorId> <!--Change this to your company na
    <VendorDescription>Joel Diniz and Edson Andrade | www.etlipse.c
  </AddIn>
</RevitAddIns>
```

You can use this template for other versions of Revit

Finally, if you have any Revit version other than the 2019 and 2020 ones, you can still create a project in Visual Studio from this template and do some changes to be able to use it:

1. The first thing you must do is perform the troubleshooting procedure *Revit References must be found* that is described above. This will retrieve the right *dlls* from your Revit main folder.
2. Then, you should perform the troubleshooting procedure *Revit must be set as the Start application for Debugging* that is also described above. This will make Visual Studio start your version of Revit when debugging.
3. The last steps consist of changing the Post-Build directory for your files. Still in *Properties*, go to the *Build Events* section and click on the *Edit Post-build...* button (image below).

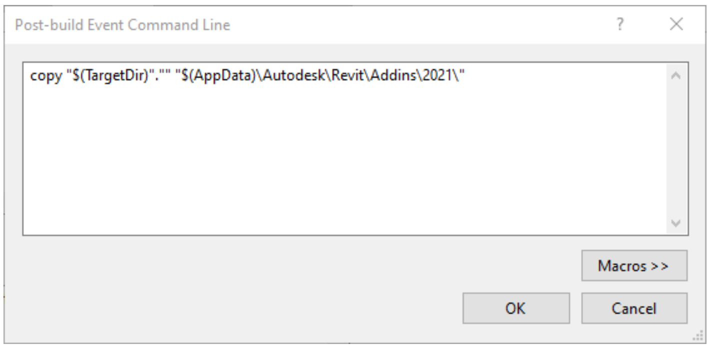


Post-build event command line:

```
copy "$(TargetDir)"."" "$(AppData)\Autodesk\Revit\Addins\2019\"
```

Run the post-build event: On successful build

4. Now the command line opens and you just need to replace the "2019" or "2020" you see in the line for the number of your Revit version ("2021", like in the example below).



```
copy "$(TargetDir)"."" "$(AppData)\Autodesk\Revit\Addins\2021\"
```