My First Revit Plug-in Overview >> Lesson 1 >> Lesson 2 >> Lesson 3>> Lesson 4 >> Lesson 5 >> Lesson 6 >> Lesson 7 >> Lesson 8

Lesson 2: Programming Overview

In the previous lesson, you saw how you can increase productivity in Autodesk Revit by implementing a plug-in built from a small amount of C# code.

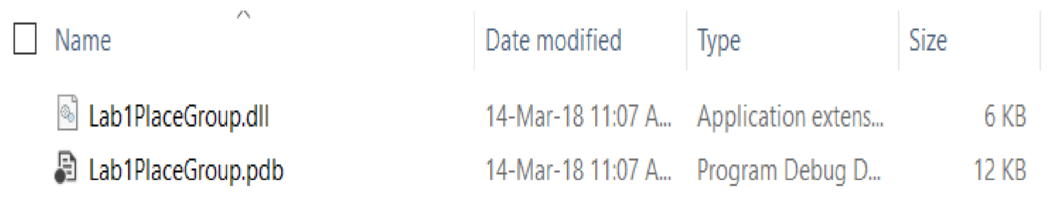
You have heard the term .NET from Lesson 1 with reference to programming with Revit. .NET is a technology that enables communication between software, if you are interested in learning more, you will find information in the Additional Topics section.

You will now look more closely at what happened when you built and executed the code in the previous lesson.

What does it mean to "build" code?

The code you typed into Visual Studio in Lesson 1 was a set of human-readable instructions (source code) that needed to be converted into code that could be understood and executed by the computer. The "build" you performed did just that: it packaged up the resulting executable code into a DLL (Dynamic-Link Library) that can be loaded into Autodesk Revit.

The following screenshot shows the output in DLL form along with the associated program debug database (which provides additional information when troubleshooting the DLL), once you have built the solution from Lesson 1 using Visual Studio. The path to which the DLL gets compiled is specified in the Visual Studio project settings and is set, by default, to the bin sub-folder of the Visual Studio project folder.



Name	Date modified	Type	Size
Lab1PlaceGroup.dll	14-Mar-18 11:07 A...	Application extens...	6 KB
Lab1PlaceGroup.pdb	14-Mar-18 11:07 A...	Program Debug D...	12 KB

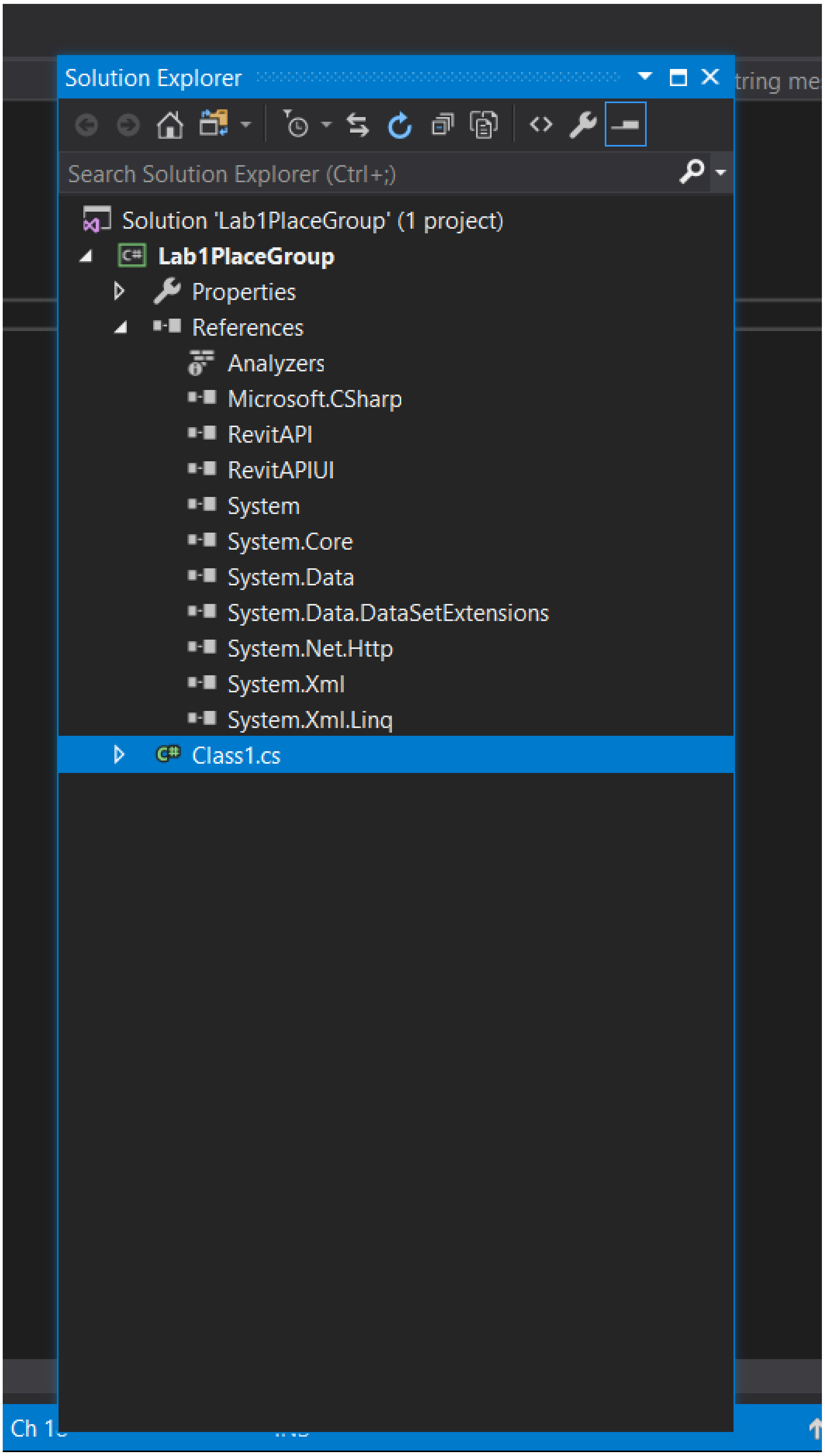
Choosing a Programming Language and Development Tool

Just as humans use different languages to communicate, you have various language options available to you when creating a Revit plug-in: for the purposes of this guide we have chosen C#, a strong general-purpose programming language that is popular with Revit developers.

There are a number of tools available for developing C# code. They range from open source tools such as SharpDevelop to Microsoft's flagship, professional development environment, Visual Studio Community. In your case you will be using, a free version of Visual Studio focused on building C# applications.

Visual Studio is an Integrated Development Environment (IDE) because it is composed of various tools, menus and toolbars which ease the creation and management of your code.

The project system in Visual Studio comprises of Solution and Project files as well as Project Items, the individual files belonging to projects. A solution is a container for one or more projects. Each project can in turn be considered a container for project items – such as source files, icons, etc. – most of which get compiled into the resultant executable file (EXE or DLL). Visual Studio provides a Solution Explorer that organizes and displays the contents of the loaded solution in a tree-view format:



Solution Explorer

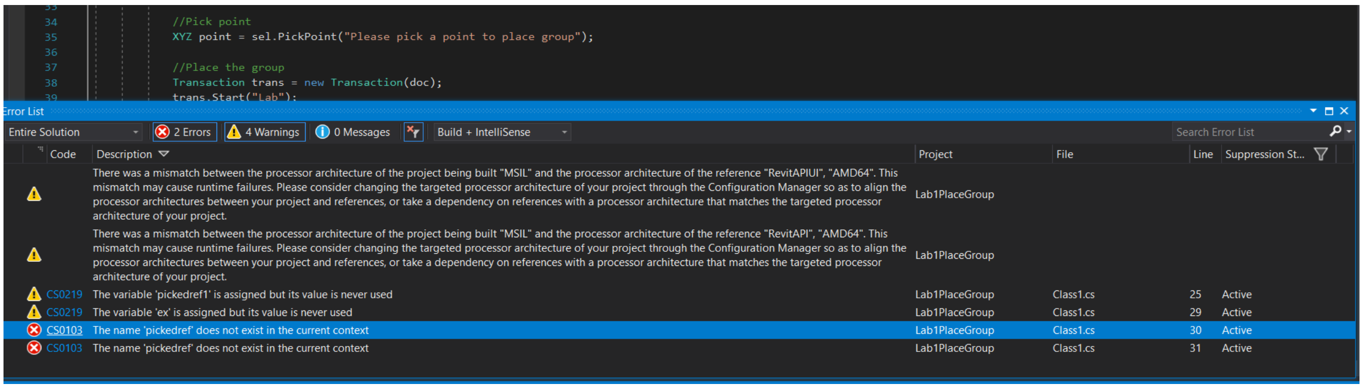
Search Solution Explorer (Ctrl+;)

Solution 'Lab1PlaceGroup' (1 project)
- Lab1PlaceGroup
  - Properties
  - References
    - Analyzers
    - Microsoft.CSharp
    - RevitAPI
    - RevitAPIUI
    - System
    - System.Core
    - System.Data
    - System.Data.DataSetExtensions
    - System.Net.Http
    - System.Xml
    - System.Xml.Linq
  - Class1.cs

The Visual Studio interface also contains a text editor and interface designer. These are displayed in the main window depending on the type of file being edited. The text editor is where you will enter the C# code for your Revit plug-in. This editor provides advanced features such as IntelliSense and collapsible code sections along with the more classic text-editing features such as bookmarks and the display of line numbers.

IntelliSense is an extremely valuable feature of the Visual Studio family that greatly improves programmer productivity: it automatically provides suggestions for the code being written based on the objects available and the letters that are being typed.

Clearly one of the key features of Visual Studio is its ability to build C# code into an executable file. During the build process, the language compiler performs various checks and analyses on the code. One such check is to ensure the code conforms to the syntactical rules of the C# language. The compiler also performs various other checks, such as whether a variable has been appropriately defined or not. Detected errors are reported via the Error List window, typically found at the bottom of the main window.



```
//Pick point
XYZ point = sel.PickPoint("Please pick a point to place group");

//Place the group
Transaction trans = new Transaction(doc);
trans.Start("Lab");
```

Code	Description	Project	File	Line	Suppression St...
	There was a mismatch between the processor architecture of the project being built "MSIL" and the processor architecture of the reference "RevitAPIUI", "AMD64". This mismatch may cause runtime failures. Please consider changing the targeted processor architecture of your project through the Configuration Manager so as to align the processor architectures between your project and references, or take a dependency on references with a processor architecture that matches the targeted processor architecture of your project.	Lab1PlaceGroup			
	There was a mismatch between the processor architecture of the project being built "MSIL" and the processor architecture of the reference "RevitAPI", "AMD64". This mismatch may cause runtime failures. Please consider changing the targeted processor architecture of your project through the Configuration Manager so as to align the processor architectures between your project and references, or take a dependency on references with a processor architecture that matches the targeted processor architecture of your project.	Lab1PlaceGroup			
CS0219	The variable 'pickedref1' is assigned but its value is never used	Lab1PlaceGroup	Class1.cs	25	Active
CS0219	The variable 'ex' is assigned but its value is never used	Lab1PlaceGroup	Class1.cs	29	Active
CS0103	The name 'pickedref' does not exist in the current context	Lab1PlaceGroup	Class1.cs	30	Active
CS0103	The name 'pickedref' does not exist in the current context	Lab1PlaceGroup	Class1.cs	31	Active

Reviewing your use of Visual Studio

In this section, you will review the steps performed using Visual Studio from the previous lesson. However, we will put them in the context of what you have just learned about programming in general and building your code.

1. In the first step, you simply launched Visual Studio.
2. You then created a new C# project of type Class Library.

Since the development language used for this guide is C#, you are working with Visual Studio, and therefore you see **Visual C#** under **Installed Templates** portion of the **New Project** dialog.

In the middle section of this dialog, you saw various types of applications that can be created; you selected the template according to the type of application you wish to create.

For plug-ins to load into Revit, they need to be Class Library assemblies (DLLs). It’s for this reason, in the second step, that you selected the **Class Library** template. The name you entered is used to identify the project within the solution.

3. Your blank project was created, containing a few standard project references to core .NET components along with a blank C# class file. It’s this file that gets displayed in the text editor window.

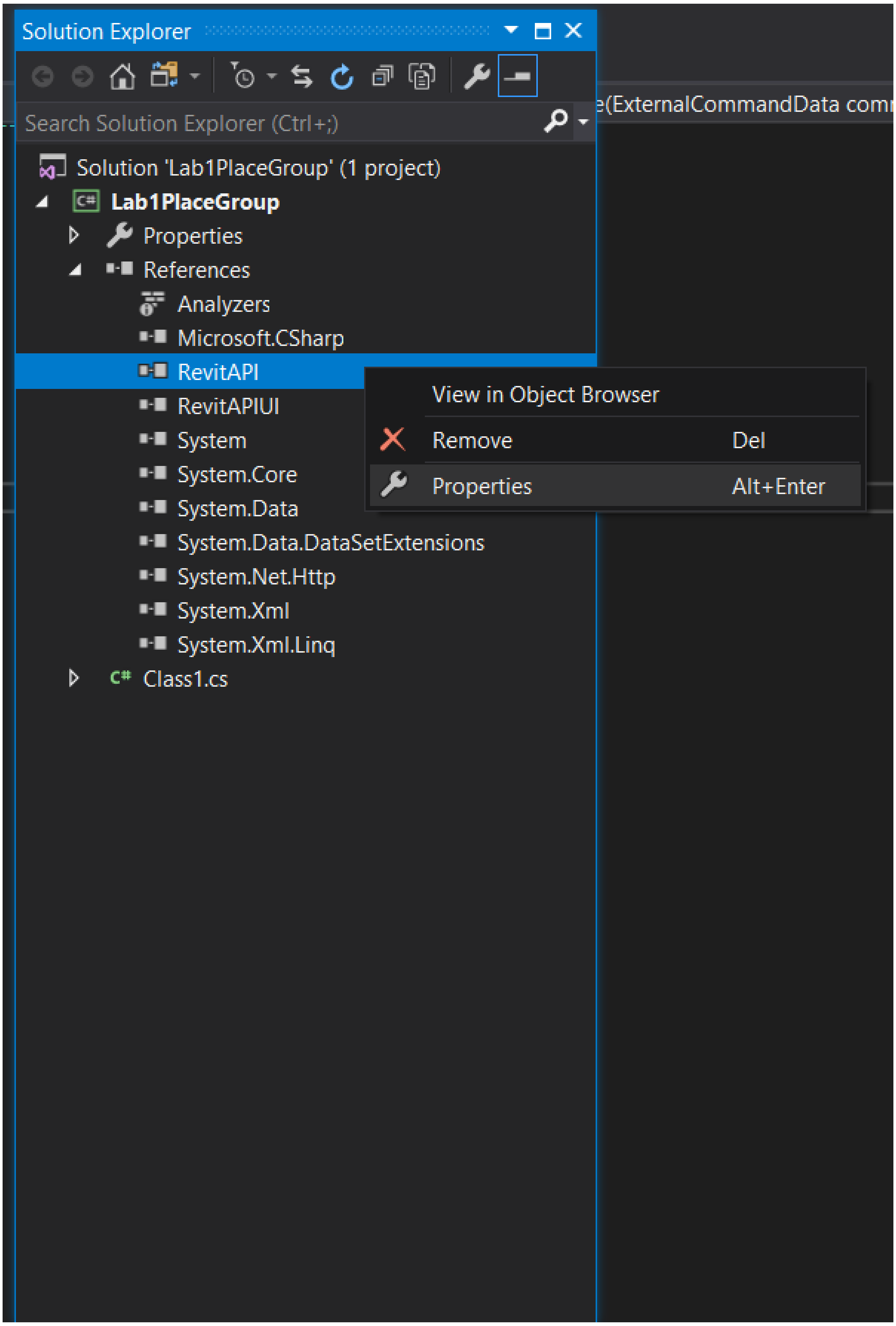
4. Saving the solution created physical files representing the contents of your class library project on the computer’s hard drive, allowing you to open and edit it at another time in the future.

5. This blank project, as created by Visual Studio, did not automatically make use of the Revit API. For it to do so, you added project references to the interface DLLs in Revit describing its *RevitAPI.dll* and *RevitAPIUI.dll*.

6. When using the Revit API, it is usual to add project references to the two separate interface DLLs making up the API: one deals with core product functionality, the other with the product’s user interface. You must link your project to these files to be able to work with Revit API.

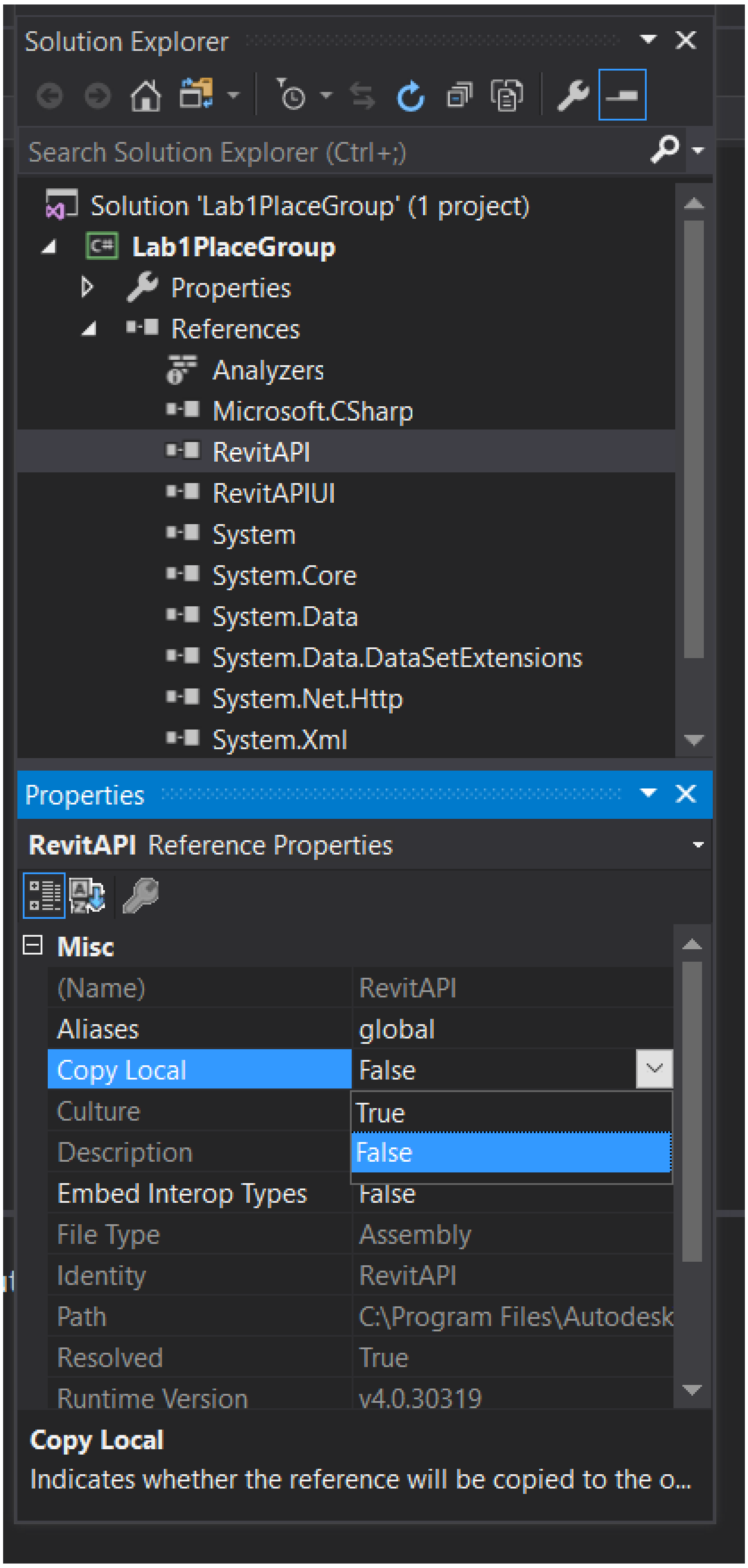
- *RevitAPI.dll* contains the APIs to access the Revit application, documents, elements, parameters, etc.
- *RevitAPIUI.dll* contains the APIs related to manipulation and customization of the Revit user interface, including commands, selections and dialogs.

Having added your project references, it’s important that you set one of their properties appropriately.


Solution Explorer
Search Solution Explorer (Ctrl+;)
e(ExternalCommandData comm
Solution 'Lab1PlaceGroup' (1 project)
Lab1PlaceGroup
Properties
References
Analyzers
Microsoft.CSharp
RevitAPI
RevitAPIUI
System
System.Core
System.Data
System.Data.DataSetExtensions
System.Net.Http
System.Xml
System.Xml.Linq
Class1.cs
View in Object Browser
Remove
Del
Properties
Alt+Enter

By default, Visual Studio adds project references with its **Copy Local** property set to **True**. This means that the referenced DLLs will get copied to the project's output folder when it is built. In your case you wanted to change this setting to **False**, to make sure the DLLs did not get copied along with your assembly DLL.

To change this setting for the DLL, you selected the Revit API DLL (located under the **References** folder in the **Solution Explorer** on the right side of the Visual Studio interface) which populated the DLL reference's properties in the **Properties** panel below. In this properties panel, you saw a property called **Copy Local**. You clicked on it to change the value from *True* to *False* using the drop down list. Then you repeated these steps for the other DLL as well.


Solution Explorer

Search Solution Explorer (Ctrl+;)

```
Solution 'Lab1PlaceGroup' (1 project)
◢ Lab1PlaceGroup
    ▷ Properties
    ◢ References
          Analyzers
          Microsoft.CSharp
          RevitAPI
          RevitAPIUI
          System
          System.Core
          System.Data
          System.Data.DataSetExtensions
          System.Net.Http
          System.Xml
```

Properties

RevitAPI Reference Properties

Misc

(Name)	RevitAPI
Aliases	global
Copy Local	False
Culture	True
Description	False
Embed Interop Types	False
File Type	Assembly
Identity	RevitAPI
Path	C:\Program Files\Autodesk
Resolved	True
Runtime Version	v4.0.30319

Copy Local
Indicates whether the reference will be copied to the o...

The reason for doing this is two fold: firstly, it’s unnecessary to copy these files – they consume disk space and take time to copy – but, more importantly, the Common Language Runtime CLR can get confused about which copy of each DLL needs to be loaded if they exist in multiple places. Making sure the DLLs do not get copied with the project output means the CLR will correctly find the ones in the Revit folder.

7. Next you added C# code using the Revit API into your project. In other words providing Revit with instructions on how to perform the functionality of copying a user-selected group from one place to another.

While developing code, it’s a good idea to build the solution from time to time, to check whether errors have been introduced in the code. The code does not necessarily have to be complete or functional when building the solution. This approach can help avoid potentially lengthy troubleshooting once the code is complete, and has the side benefit of automatically saving any edited source files before the build starts.

To build a solution inside Visual Studio, select **Build Solution** from the **Build** pull-down menu.

If the build process was successful, you would see a **Build Succeeded** status in the bottom left corner of the Visual Studio interface.

A quick recap: in this lesson you took a brief look at what happens when you build a project, as well as some background information on C# and Visual Studio Community. You reviewed the steps you had taken in the previous lesson to build your basic Revit plug-in, putting it in the context of your learning about programming.

Additional Topics

What is .NET?

The .NET Framework is a software framework that sits on top of the Microsoft® Windows® operating system* and provides the underlying platform, libraries and services for all .NET applications. The services generally include memory management, garbage collection, common type system, class libraries, etc.

** Subsets of .NET are also available on other operating systems, whether via the open source Mono project or via Microsoft® Silverlight®, but these are not topics for this guide: you will focus solely on the use of .NET in the context of Microsoft Windows.*

What does the .NET Framework Contain?

The framework contains two main components:

1. **Common Language Runtime (CLR)**– This is the agent (or execution engine) in the .NET Framework responsible for managing the execution of code. Which is why code written to target this runtime is also known as managed code. All managed code runs under the supervision of the CLR, but what does this mean? The CLR manages code by providing core services such as memory management (which includes automatically releasing the computer's memory for reuse on other tasks when it is no longer needed), error (or exception) handling, managing the use of multiple threads of execution and ensuring rules around the use of different types of object are adhered to. The CLR is really the foundation of the .NET Framework.
2. **.NET Framework Class Library** (https://learn.microsoft.com/en-us/previous-versions/ms229335(v=vs.100)) – As the name suggests, this is a library or collection of object types that can be used from your own code when developing .NET applications. These .NET applications are targeted for Windows (whether command-prompt based or with a graphical user interface), the web or mobile devices. This library is available to all languages using the .NET Framework.

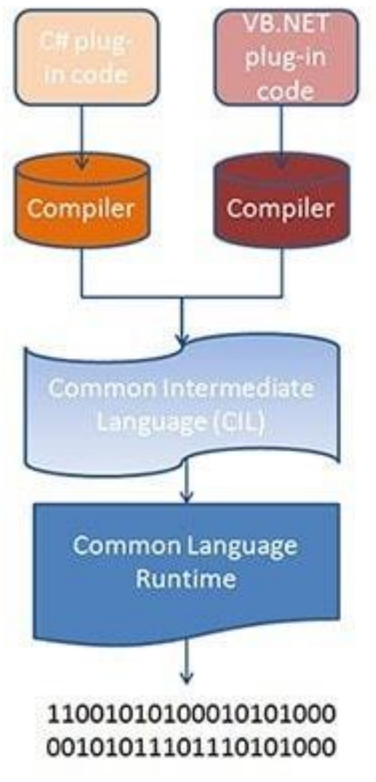
As mentioned above, the CLR improves code robustness by making sure the executing code conforms to a common type system (CTS). The CTS ensures that all .NET (or managed) code – irrespective of the language – uses a similar set of object types and can work together in the same environment. It is this feature that makes it possible for you to write applications in the development language of your choice and yet make use of components/code written by programmers using other .NET languages.

Building Executables

When you built your code into a DLL, it is compiled to Common Intermediate Language (CIL – also known as MSIL) code using a language-specific compiler. CIL is a CPU-independent set of instructions that can be executed by the CLR on Windows operating systems. CIL is typically portable across 32- and 64-bit systems and even – to some extent – to non-Windows operating systems. The CIL code generated from your C# source code is then packaged into a .NET assembly. Such an assembly is a library of CIL code stored in Portable Executable (PE) format (which contains both the CIL and its associated metadata). Assemblies can either be process assemblies (EXEs) or library assemblies (DLLs).

During the course of this guide, Revit plug-ins are compiled into library assembly files (DLLs) which are then loaded and executed from within Revit’s memory space.

Running Executables



During execution of the .NET assembly, CIL (residing in the assembly) is passed through the CLR’s just-in-time (JIT) compiler to generate native (or machine) code. JIT compilation of the CIL to native code occurs when the application is executed. As not all of the code is required during execution, the JIT compiler only converts the CIL when it is needed, thus saving time and memory. It also stores any generated code in memory, making it available for subsequent use without the need to recompile.

In the last step of this process, the native code gets executed by the computer’s processor.

If you would like more details on the process of building .NET applications, please refer to the MSDN Library (https://learn.microsoft.com/en-us/dotnet/standard/managed-execution-process?redirectedfrom=MSDN)

My First Revit Plug-in Overview >> Lesson 1 >> Lesson 2 >> Lesson 3>> Lesson 4 >> Lesson 5 >> Lesson 6 >> Lesson 7 >> Lesson 8