Lesson 4: Debugging your code.

Learn how to use the Visual Studio debugger to step through your code line-by-line to follow the program execution; and to watch the values of variables as they are changed by your code.

It's rare that you will write a program that runs perfectly the first time. Even if you haven't made any mistakes that prevent Visual Studio from building your plug-in, it's quite likely that you made a mistake in your program logic that causes it to produce different results from the ones you expected. Errors in your code are called bugs, and the process of stepping through your code examining the program execution flow and variable values is called debugging.

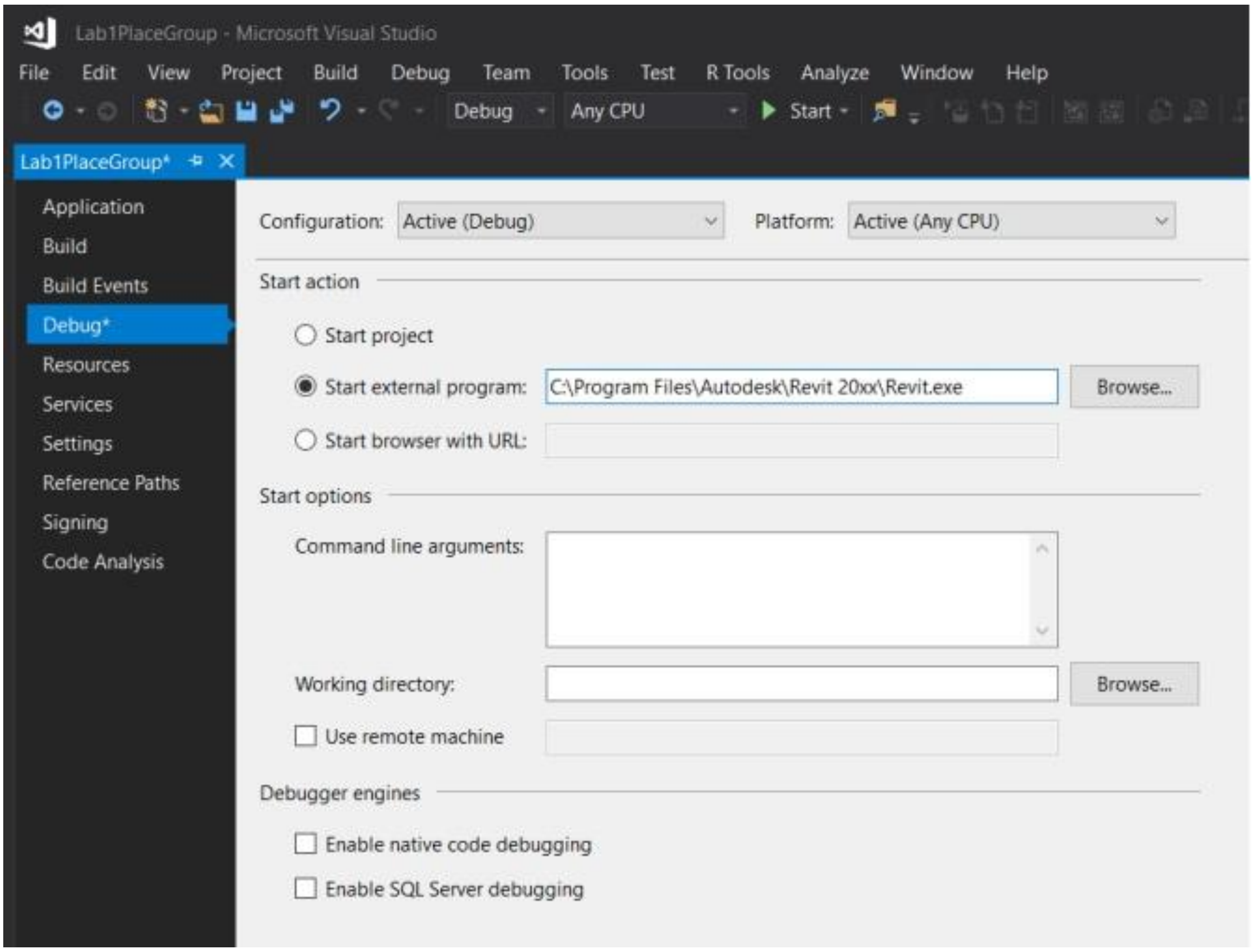
To debug your plug-in code, you're going to have to launch Revit from your Visual Studio debugger and load your plug-in into Revit.

In order to be able to debug using **Visual Studio**, we have to make some manual edits to the project files by implementing the following steps:

1. In the Solution Explorer window, right Click Lab1PlaceGroup and select Properties.
2. In the Properties window, Go to Debug option and choose "Start external program".
3. Browse for "Revit.exe" file and normally the file will be present in this path(**C:\Program Files\Autodesk\Revit 20xx\Revit.exe**).

(Edit the file path for Revit.exe if you've installed it in a non-default location).

Thus, the Properties settings allow you to set the executable to launch when you start the debugger:



Configuration: Active (Debug) Platform: Active (Any CPU)

Start action

○ Start project
● Start external program: C:\Program Files\Autodesk\Revit 20xx\Revit.exe
○ Start browser with URL:

Start options

Command line arguments:

Working directory:

☐ Use remote machine

Debugger engines

☐ Enable native code debugging
☐ Enable SQL Server debugging

Launching the debugger

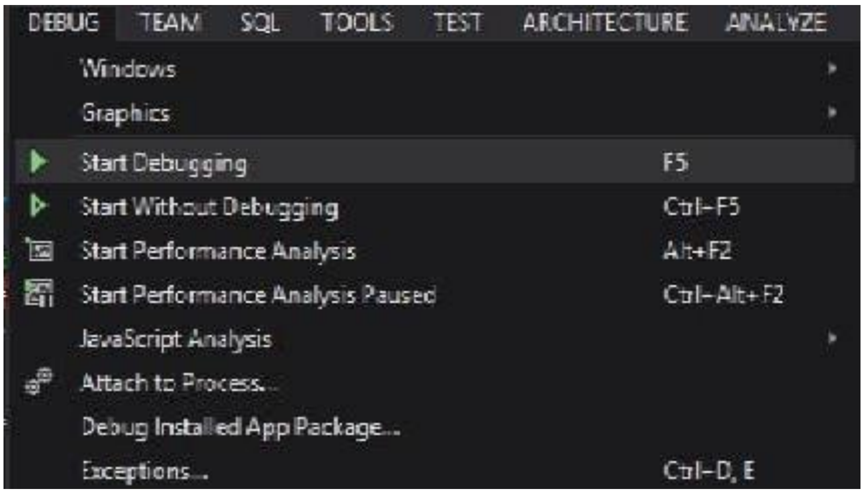
If you closed the Visual Studio, launch it again now and open the project you saved in Lesson 1 (called the Lab1PlaceGroup).

Before you launch the debugger, please ensure that the AddIn manifest file for this lab has been created and exists at the location described in the *Writing an AddIn Manifest* section in Lesson 1: The Basic Plug-in. Since you will now be debugging the plug-in code, Visual Studio has created a 'debug' version of your .NET plug-in DLL. And so for Revit to read this 'debug' DLL instead of the 'release' DLL that was created in Lesson 1, we need to change the path of the DLL in the manifest to the following (highlighted in bold) –

```
<?xml version="1.0" encoding="utf-8"?>
<RevitAddIns>
  <AddIn Type="Command">
       <Name>Lab1PlaceGroup</Name>
       <FullClassName>Lab1PlaceGroup.Class1</FullClassName>
       <Text>Lab1PlaceGroup</Text>
```

```
        <Description>Places the Group at Particular Point</Description>
        <VisibilityMode>AlwaysVisible</VisibilityMode>

<Assembly>C:\test\Lab1PlaceGroup\Lab1PlaceGroup\bin\Debug\Lab1placeGrou
p.dll</Assembly>
        <AddInId>502fe383-2648-4e98-adf8-5e6047f9dc34</AddInId>
     <VendorId>ADSK</VendorId>
     <VendorDescription>Autodesk, Inc,
www.autodesk.com</VendorDescription>
   </AddIn>
</RevitAddIns>
```

To start your debugging session, simply open the Debug menu and select the Start Debugging option, or just hit F5. This will launch Revit from your debugger. The flavor of Revit that gets launched will depend on the path you have provided in the tag in the .csproj file that we edited manually at the beginning of this lesson.


DEBUG
TEAM
SQL
TOOLS
TEST
ARCHITECTURE
ANALYZE
Windows
Graphics
Start Debugging F5
Start Without Debugging Ctrl+F5
Start Performance Analysis Alt+F2
Start Performance Analysis Paused Ctrl+Alt+F2
JavaScript Analysis
Attach to Process...
Debug Installed App Package...
Exceptions... Ctrl+D, E

You have to launch Revit in this way so that your debugger can hook into the Revit process to monitor the execution of the code you compiled into your plug-in DLL. Because we've placed the AddIn manifest in the correct location, Revit will automatically load our plug-in.

Remember that the DLL you loaded in Lesson 1 was in the bin\Release *folder. When you are building a 'final' version of your plug-in DLL that you want to give to your users and customers, Visual Studio will build a* **release** *version. Visual Studio makes various optimizations to the compiled code in a release build so that it will run faster and take up less memory. When you want to debug your code, Visual Studio will create a* **debug** *version of your plug-in DLL. The debug version isn't optimized for speed/memory, and also includes additional information that the debugger uses to tell you about what is happening when the code runs.*

Please read more on this in this article: How to: Attach to a Running Process (https://learn.microsoft.com/en-in/previousversions/visualstudio/visual-studio-2015/debugger/attach-to-running-processes-with-the-visual-studio-debugger?view=vs-2015&redirectedfrom=MSDN)

Now, open the Hotel.rvt Project file.

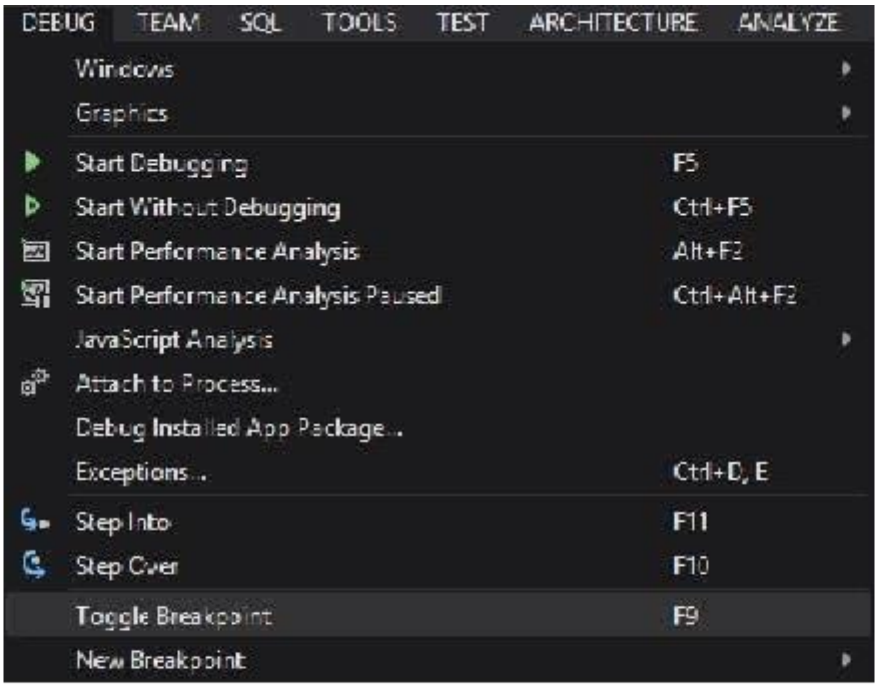
Breakpoints

Your plug-in DLL is now ready to debug. But before you run your **Lab1PlaceGroup** command, you have to tell the debugger you want it to stop when it is executing your code. You do this using a **breakpoint**.

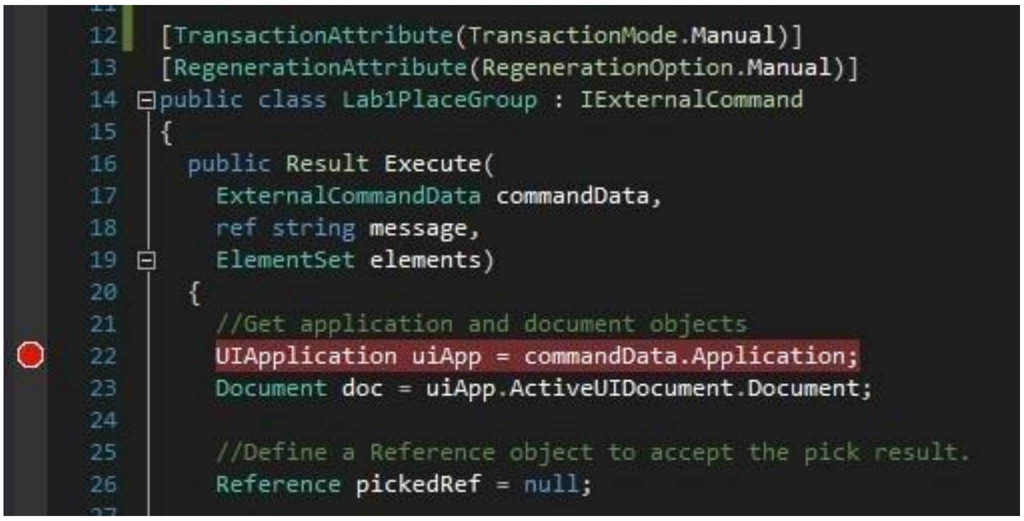
In Visual Studio, double-click on Class1.cs in the Solution Explorer to display your code and click anywhere in the line:

```
UIApplication uiapp = commandData.Application;
```

Then select **Toggle Breakpoint** from the **Debug** menu (or hit **F9**).



That line will now be highlighted in red and have a red circle in the margin next to it to indicate that you've set a breakpoint for this line:



```
[TransactionAttribute(TransactionMode.Manual)]
[RegenerationAttribute(RegenerationOption.Manual)]
public class Lab1PlaceGroup : IExternalCommand
{
  public Result Execute(
    ExternalCommandData commandData,
    ref string message,
    ElementSet elements)
  {
    //Get application and document objects
    UIApplication uiApp = commandData.Application;
    Document doc = uiApp.ActiveUIDocument.Document;

    //Define a Reference object to accept the pick result.
    Reference pickedRef = null;
```

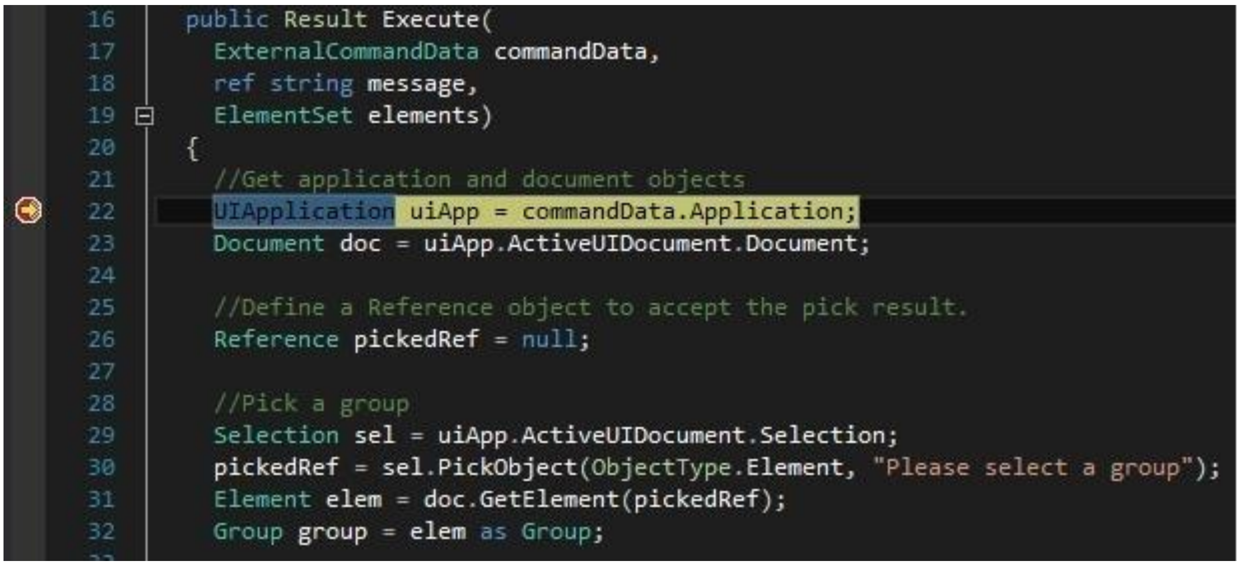
Set a breakpoint for the **PickPoint** function in the same way:

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

When Revit calls these methods in your code, the debugger will stop at these lines and wait for you to tell it what to do.

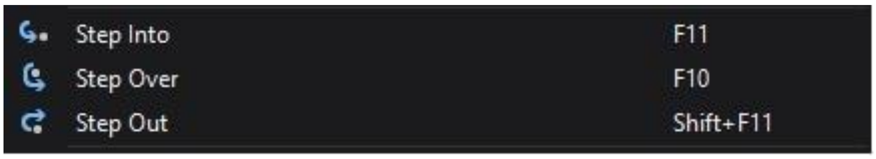
Stepping through your code

Now it’s time to invoke your command. Inside Revit on the **Add-Ins** ribbon tab, click the **External Tools** drop-down list, then click **Lab1PlaceGroup**. This will start your plug-in in Revit and Visual Studio should take control and become your foreground application. (If it doesn’t, click on its icon in your Windows taskbar to activate it.) The debugger is now stopped waiting for you with the line of code it’s about to execute highlighted in yellow and with a little yellow arrow in the margin next to it.

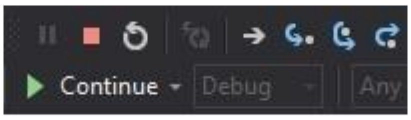


```
  public Result Execute(
    ExternalCommandData commandData,
    ref string message,
    ElementSet elements)
  {
    //Get application and document objects
    UIApplication uiApp = commandData.Application;
    Document doc = uiApp.ActiveUIDocument.Document;

    //Define a Reference object to accept the pick result.
    Reference pickedRef = null;

    //Pick a group
    Selection sel = uiApp.ActiveUIDocument.Selection;
    pickedRef = sel.PickObject(ObjectType.Element, "Please select a group");
    Element elem = doc.GetElement(pickedRef);
    Group group = elem as Group;
```

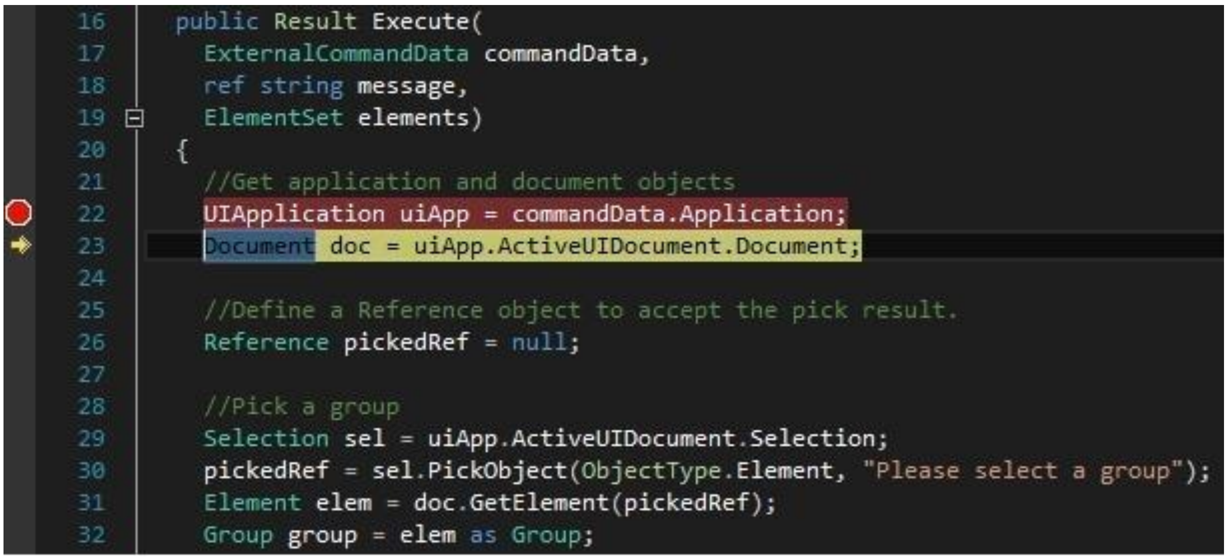
Now you're ready to step through your code. The Visual Studio debug menu gives you three ways to step through your code: **Step Into**; **Step Over**; and **Step Out**. You'll mostly be using *Step Over* – this executes the next line of code (the line highlighted in yellow) in the debugger, and then moves to the next line. If the line of code to be executed is a method call, then Step Over executes the entire method. If you also want to execute the called method a line at a time, you can use *Step Into*; and you can use *Step Out* to move back up (out of the method) to the code you were originally debugging.



As well as in the *Debug* menu, you should also see *Step Into*, *Step Over* and *Step Out* icons on a toolbar, and each also has a corresponding hot key (**F11**, **F10**, and **Shift+F11**).

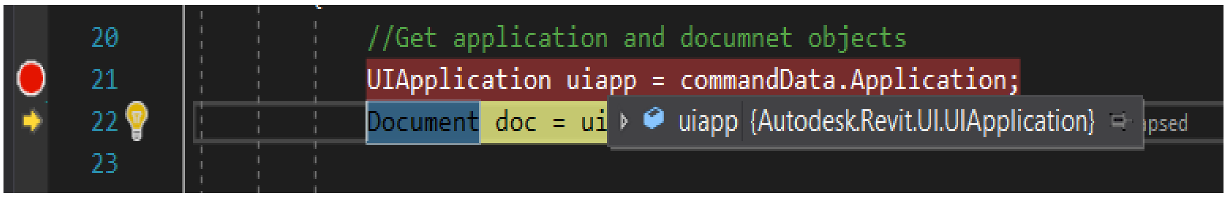


Click on the *Step Over* icon on the toolbar now. The debugger moves to the next line of code (it ignores comments).



```
public Result Execute(
  ExternalCommandData commandData,
  ref string message,
  ElementSet elements)
{
  //Get application and document objects
  UIApplication uiApp = commandData.Application;
  Document doc = uiApp.ActiveUIDocument.Document;

  //Define a Reference object to accept the pick result.
  Reference pickedRef = null;

  //Pick a group
  Selection sel = uiApp.ActiveUIDocument.Selection;
  pickedRef = sel.PickObject(ObjectType.Element, "Please select a group");
  Element elem = doc.GetElement(pickedRef);
  Group group = elem as Group;
```

You can now hover the mouse over the text **uiApp** and a tooltip is displayed showing the value of the variable.

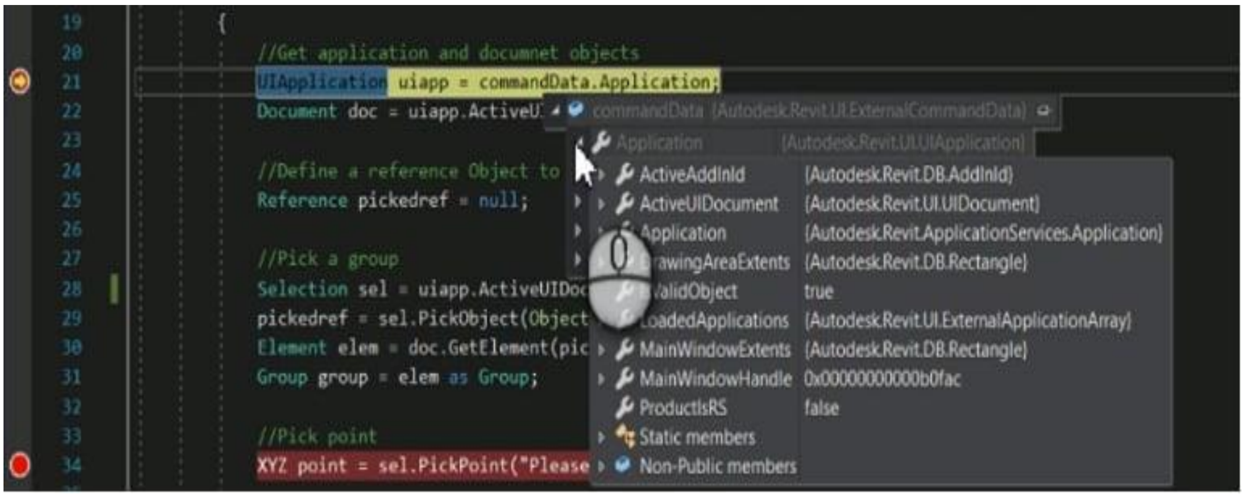


```
//Get application and documnet objects
UIApplication uiapp = commandData.Application;
Document doc = ui
```

uiapp {Autodesk.Revit.UI.UIApplication}

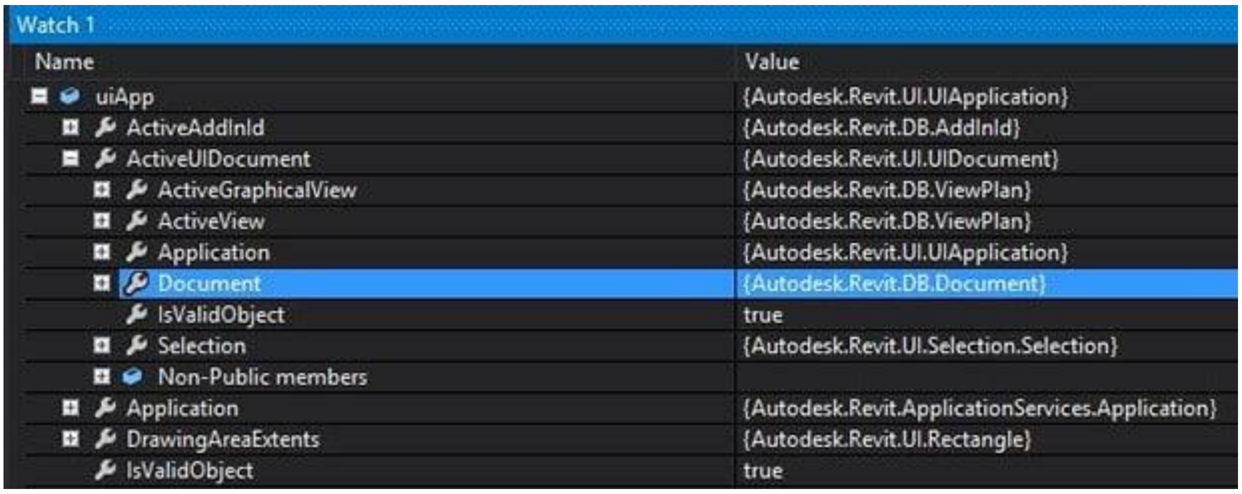
You can click on the sign to drill down to the various properties and check their values.

To display variable values, you can also right click on the variable in your code and select **Add Watch**. This displays the variable and its value in the Watch window at the bottom of your IDE.



```
{
    //Get application and documnet objects
    UIApplication uiapp = commandData.Application;
    Document doc = uiapp.ActiveU

    //Define a reference Object to
    Reference pickedref = null;

    //Pick a group
    Selection sel = uiapp.ActiveUIDoc
    pickedref = sel.PickObject(Object
    Element elem = doc.GetElement(pic
    Group group = elem as Group;

    //Pick point
    XYZ point = sel.PickPoint("Please
```

commandData	{Autodesk.Revit.UI.ExternalCommandData}
Application	{Autodesk.Revit.UI.UIApplication}
ActiveAddInId	{Autodesk.Revit.DB.AddInId}
ActiveUIDocument	{Autodesk.Revit.UI.UIDocument}
Application	{Autodesk.Revit.ApplicationServices.Application}
DrawingAreaExtents	{Autodesk.Revit.DB.Rectangle}
IsValidObject	true
LoadedApplications	{Autodesk.Revit.UI.ExternalApplicationArray}
MainWindowExtents	{Autodesk.Revit.DB.Rectangle}
MainWindowHandle	0x0000000000b0fac
ProductIsRS	false
Static members	
Non-Public members	

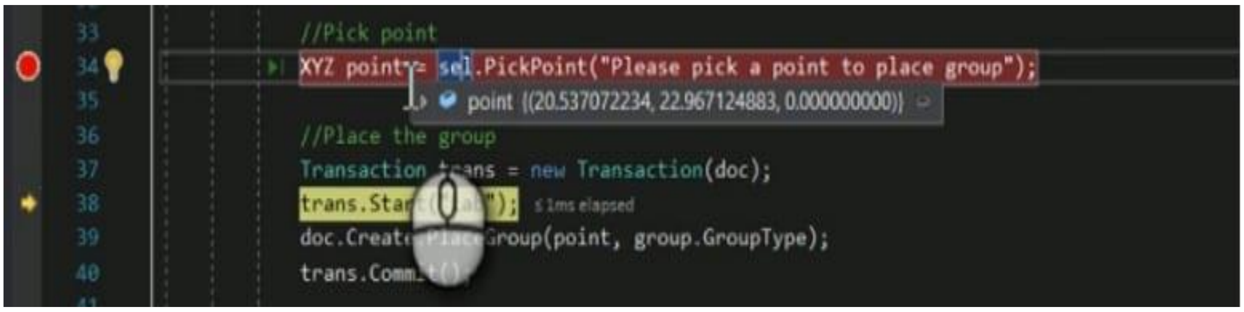


Watch 1

Name	Value
uiApp	{Autodesk.Revit.UI.UIApplication}
ActiveAddInId	{Autodesk.Revit.DB.AddInId}
ActiveUIDocument	{Autodesk.Revit.UI.UIDocument}
ActiveGraphicalView	{Autodesk.Revit.DB.ViewPlan}
ActiveView	{Autodesk.Revit.DB.ViewPlan}
Application	{Autodesk.Revit.UI.UIApplication}
Document	{Autodesk.Revit.DB.Document}
IsValidObject	true
Selection	{Autodesk.Revit.UI.Selection.Selection}
Non-Public members	
Application	{Autodesk.Revit.ApplicationServices.Application}
DrawingAreaExtents	{Autodesk.Revit.UI.Rectangle}
IsValidObject	true

Keep stepping through the code, hovering over variable values and properties to see how they change as the code executes. If at any point you want to stop line-by-line debugging, just hit *F5* to **Continue** to the next breakpoint you had set – which in our case was set to the **sel.PickPoint()** line.

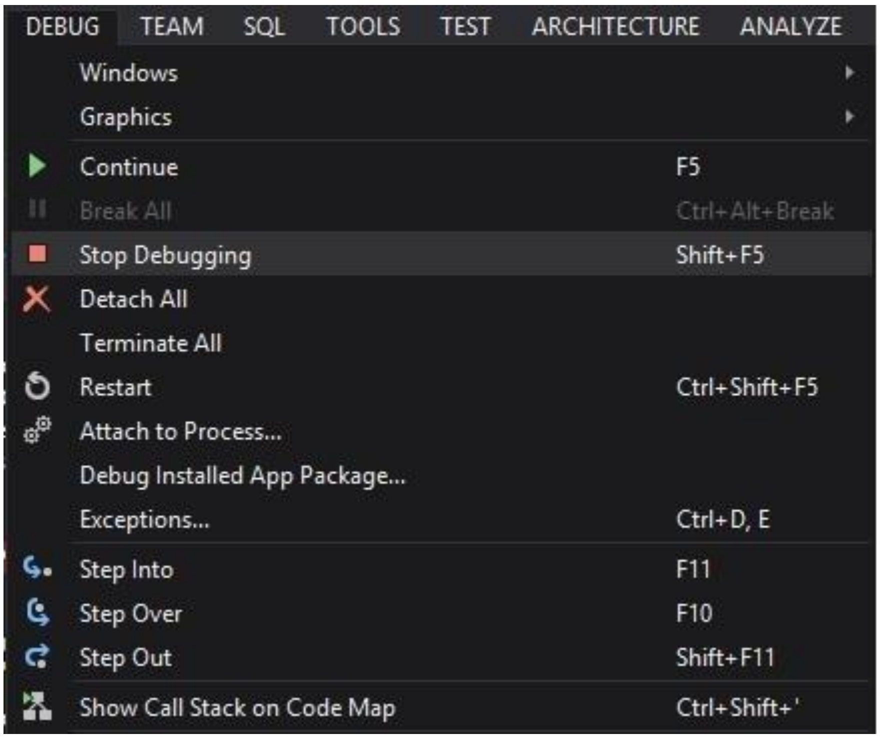
Stepping through this line by clicking *F10*, makes Revit the foreground application and prompts you to select the point where you want the Furniture group to be copied to. Let us select the approximate center of the adjacent room now. As soon as the point is selected, Visual Studio now becomes the foreground application with the next line of code highlighted in yellow with the debugger waiting for us to either check the value of the selected point or continue line by line debugging or just jump to the next breakpoint by clicking *F5.* At this point, we can check the XYZ value of the selected point by just hovering the mouse over the point variable (which is the container/variable we created to store the target point for the furniture group to be copied to).



```
//Pick point
XYZ point = sel.PickPoint("Please pick a point to place group");
    point {(20.537072234, 22.967124883, 0.000000000)}

//Place the group
Transaction trans = new Transaction(doc);
trans.Start("Lab");   ≤1ms elapsed
doc.Create.PlaceGroup(point, group.GroupType);
trans.Commit();
```

If you hit *F5* now, you will find that the debugger steps through all the remaining code, and makes Revit the foreground application again with the copy operation successfully completed.

Once you've finished experimenting, select **Stop Debugging** from the Debug menu to close Revit and end your debugging session.



Congratulations! You now know how to debug a project in Visual Studio.

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