**Lab 2 – AutoCAD .NET Wizard and User Input**

**The AutoCAD Managed Application Wizard**

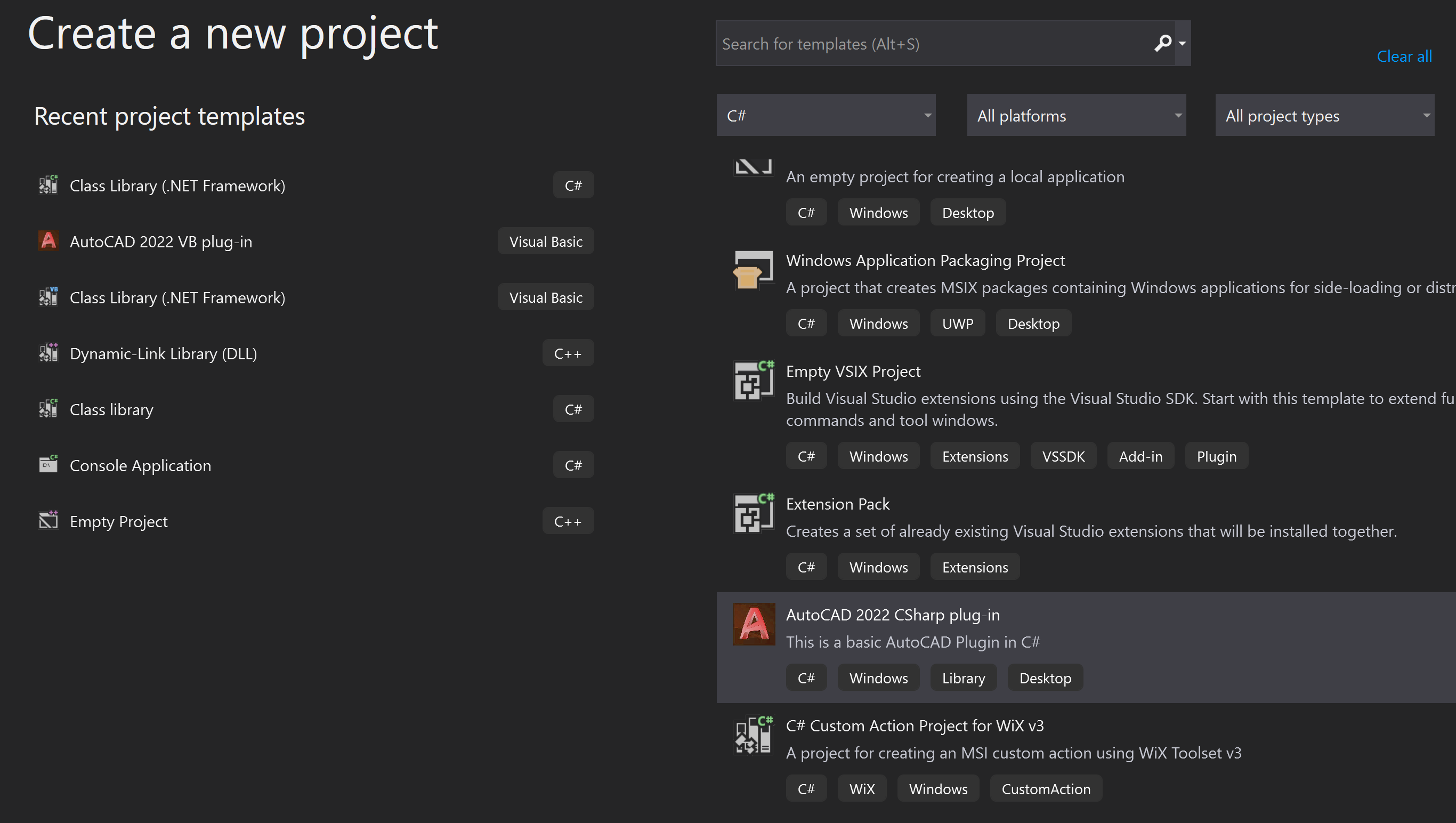
In this lab you will use the AutoCAD Managed C# Application Wizard to create a .NET project. You will then create a command that will prompt the user to create either a circle or a block. (In Lab3 you will actually be adding the entities to the drawing).

In the first lab we used a Class Library template and had to manually reference acdbmgd.dll and acmgd.dll. In this Lab we will use the AutoCAD Managed C# Application Wizard to create the .NET project which will do this for us. You will need to install the AutoCAD .NET wizard before beginning this lab. You can download the installer for the wizard from this location:

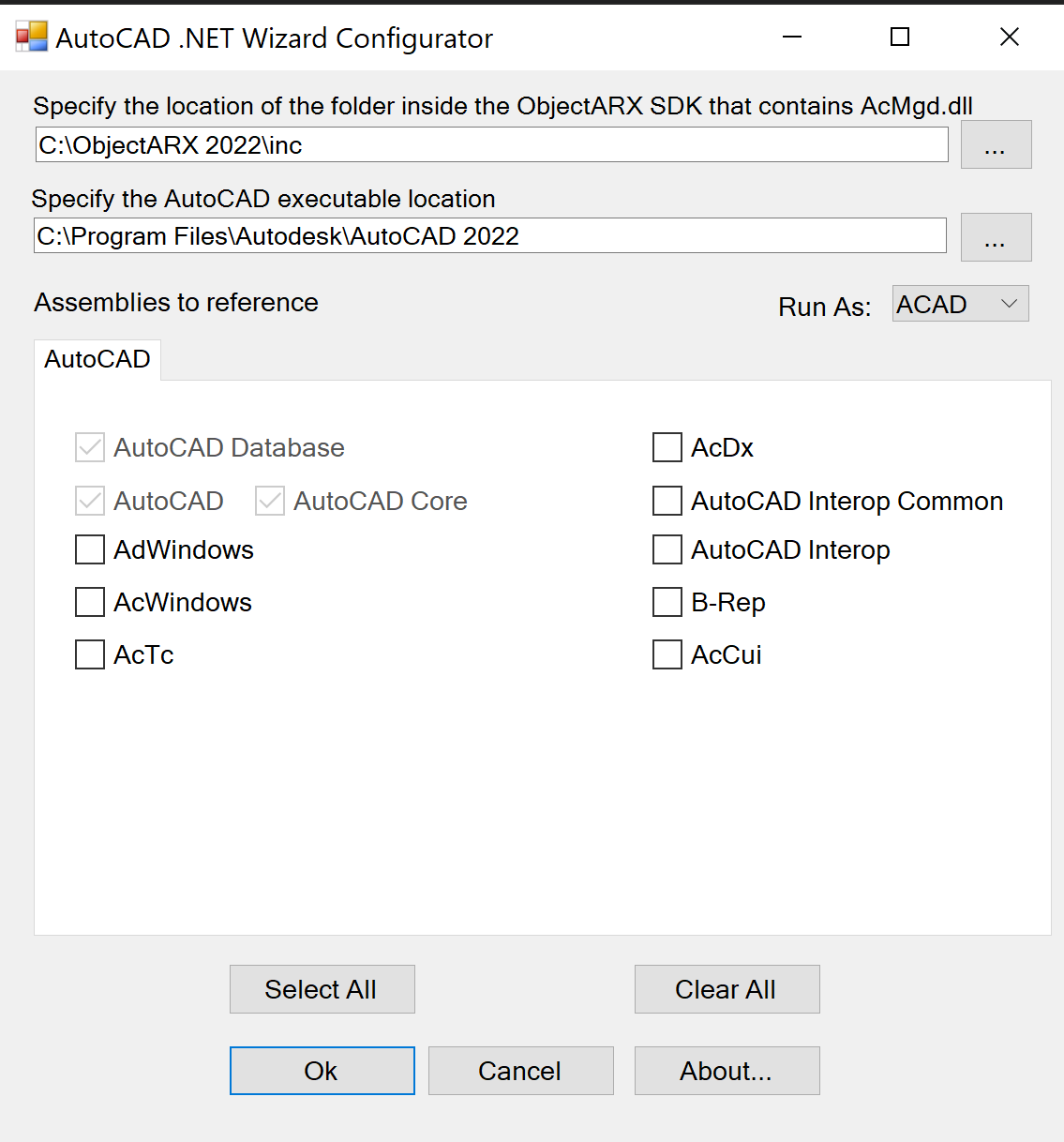
<https://www.autodesk.com/developer-network/platform-technologies/autocad>

**Create a new project using the AutoCAD Managed C# Application Wizard**

* Launch Visual Studio and then select File> New> Project.
* In the New Project dialog select Visual C# Projects for the Project Type.
* Select Autodesk and then “AutoCAD 2022 CSharp plug-in”.



* Make the name “Lab2” and set the location where you want the project to be created. Select ok.
* The “AutoCAD .NET Wizard Configurator will then appear.
* If working with AutoCAD 2022, specify the location of ObjectARX 2022



* Browse to the location of the ObjectARX SDK\inc directory.
* Click ok to create the project.
* Also, you can use NuGet package for 2022 (<https://www.nuget.org/profiles/Autodesk>)

Take a look at the project that the Wizard created. In the properties for the project notice that acdbmgd and acmgd have been referenced automatically. Also Two cs modules were created myCommands.cs and myPlugin.cs. Explore the code that the wizard has added to these two modules. myPluggin.cs has a class named MyPlugin that Implements IExtensionApplication. This allows the application to run code when it the dll is loaded in AutoCAD. (we will use this in Lab8 to add a context menu) In myCommands.cs you will find several commands have been added and are ready to be completed with your custom functionality.

**Get input from the user using classes from the EditorInput namespace.**

The steps below will add a command. When this command is run, the user will be prompted to have code create a circle or block. You can copy these lab steps into the myCommands.cs (inside the MyCommands class) created by the wizard. Or you can open the existing project Lab2 that already has these steps.

// Start of Lab2

// 1. Add a command named addAnEnt. Use the CommandMethod attribute and a

// Public void function.

// Note: put the closing curley brace after step 21.

// 2. Declare an Editor variable named ed. Instantiate it using the Editor property

// of the Application.DocumentManager.MdiActiveDocument.Editor

// 3. Declare a PromptKeywordOptions variable and instantiate it by creating

// a new PromptKeywordOptions. Use a string similar to the following for the

// messageAndKeywords string.

// "Which entity do you want to create? [Circle/Block] : ", "Circle Block"

// 4. Declare a PromptResult. Use the GetKeywords method of the Editor variable

// created in step 1. Pass in the PromptKeywordOptions created in step 2. Instantiate

// the PromptResult by making it equal to the return value of the GetKeywords method.

// 5. Add an if statement that tests the Status of the PromptResult created in step 4.

// Use the PromptStatus enum for the test. (see if it is equal to PromptStatus.OK)

// Note: Move the closing curly brace after step 21.

// (After the following instructions)

// 6. PromptStatus was ok. Now use a switch statement. For the switch argument

// use the StringResult property of the PromptResult variable used above

// Note: Move the closing curly brace after step 21.

// (Above the closing curly brace for the if statement in step 5)

// 7. Use "Circle" for the case. (if the StringResult is "Circle") Below

// we will use "Block" for the case. (jump ahead to step 15 to add the break

// to resolve the "Control cannot fall through... message")

// 8. We want to ask the user for the center of the circle. Declare

// a PromptPointOptions variable and instatiate it by making it equal

// to a new PromptPointOptions. Use "Pick Center Point : " for message parameter

// 9. Declare a PromptPointResult variable. Use the GetPoint method of

// the Editor created in step 2. (Pass in the PromptPointOptions created

// in step 8). Instantiate the PromptPointResult by making it equal to the

// return of the GetPoint method.

// 10. Add an if statement that tests the Status of the PromptPointResult

// created in step 9. Use the PromptStatus enum for the test. (make sure it is OK)

// Note: Move the closing curly brace right before step 15.

// 11. Now we want to ask the user for the radius of the circle. Declare

// a PromptDistanceOptions variable. Instatiate it by making it equal

// to a new PromptDistanceOptions. Use "Pick Radius : " for the message parameter.

// 12. We want to use the point selected in step 9 as the

// base point for the GetDistance call coming up. To do this use

// the BasePoint property of the PromptDistanceOptions variable created

// in the previous step. Make the BasePoint equal to the Value property

// of the PromptPointResult created in step 9.

// 13. We need to tell the input mechanism to actually use the basepoint.

// Do this by setting the UseBasePoint property of the

// PromptDistanceOptions created in step 11 to True.

// 14. Get the radius for the circle. Declare a PromptDoubleResult variable.

// Instantiate it using the GetDistance method of the Editor variable created

// in step 2. Pass in the PromptDistanceOptions created in step 11 and

// modified in the previous steps.

// 15. Add break to mark the end of the code for the "Circle" case.

// 16. Add the Case for the "Block" (jump ahead to step 20 to add the break

// to resolve the "Control cannot fall through... message")

// 17. Now we want to ask the user for the name of the block. Delcare

// a PromptStringOptions varable and instatiate it by creating a new

// PromptStringOptions. Use "Enter name of the Block to create : " for

// the message parameter.

// 18. No spaces are allowed in a blockname so disable it. Do this by setting

// the AllowSpaces property of the PromptStringOptions created in step 15

// to false.

// 19. Get the name the user entered. Declare a PromptResult variable

// and instantiate it using the GetString method of the Editor object

// created in step 2. Pass in the PromptStringOptions created in step 17.

// 20. Add break to mark the end of the code for the "Block" case.

// 21. Build the project. Place a break point. Use the NETLOAD command

// and run the AddAnEnt command. Step through the code and fix any errors.

// Remember to run the command and test the code for both circle and block.

// End of Lab2