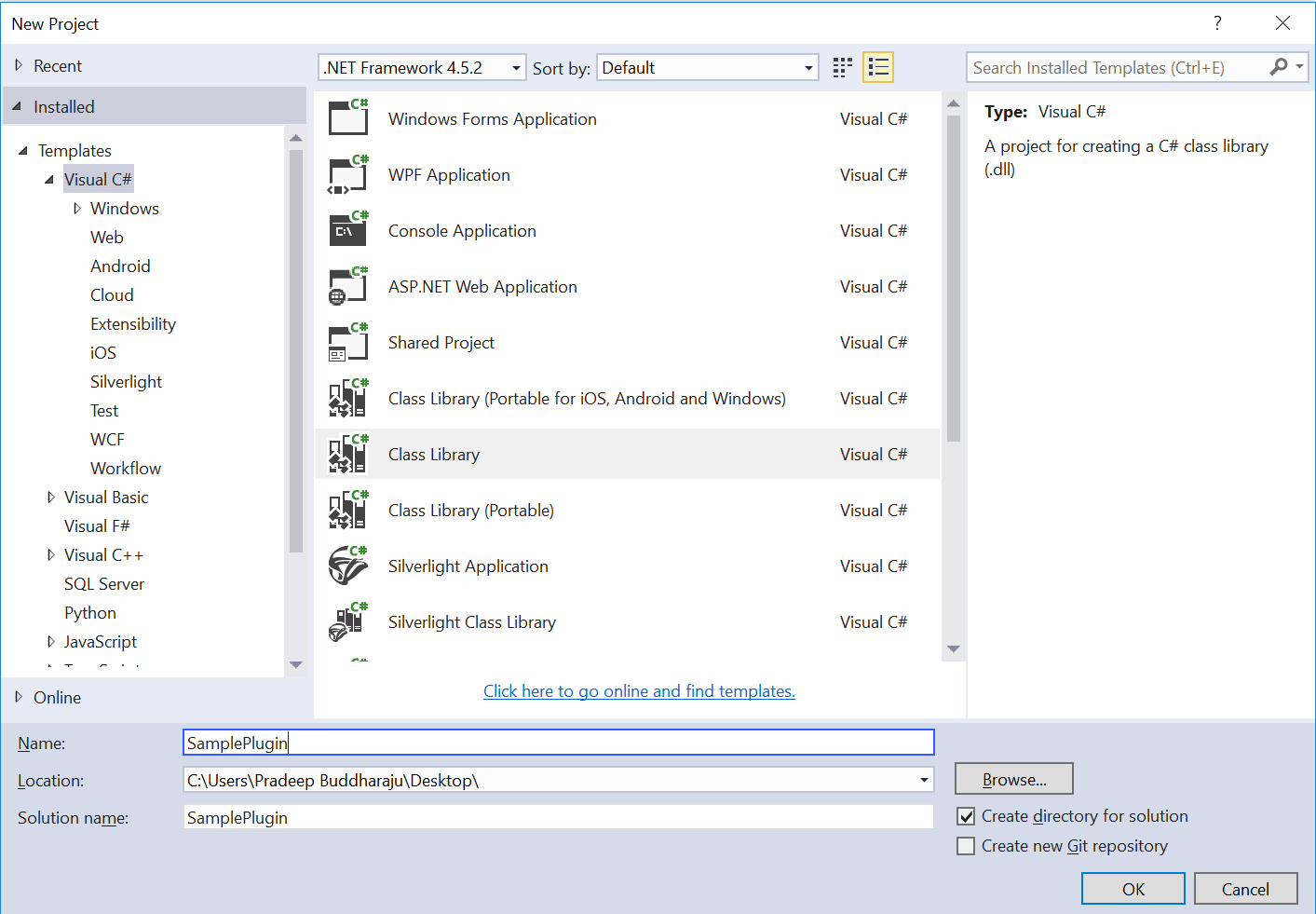
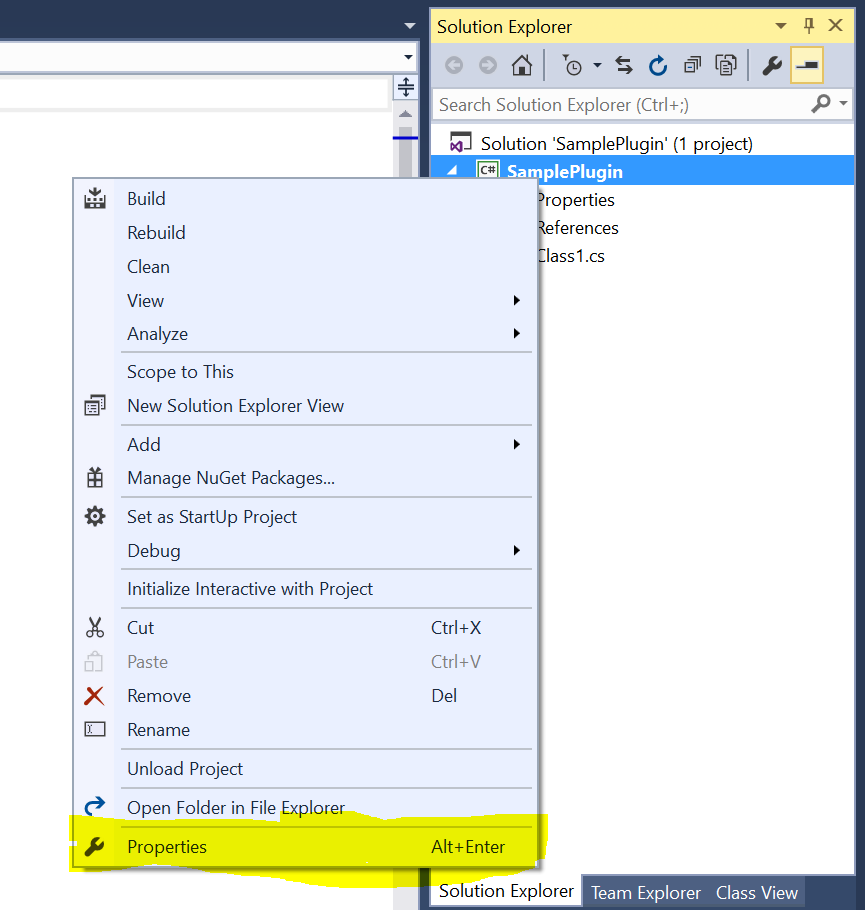
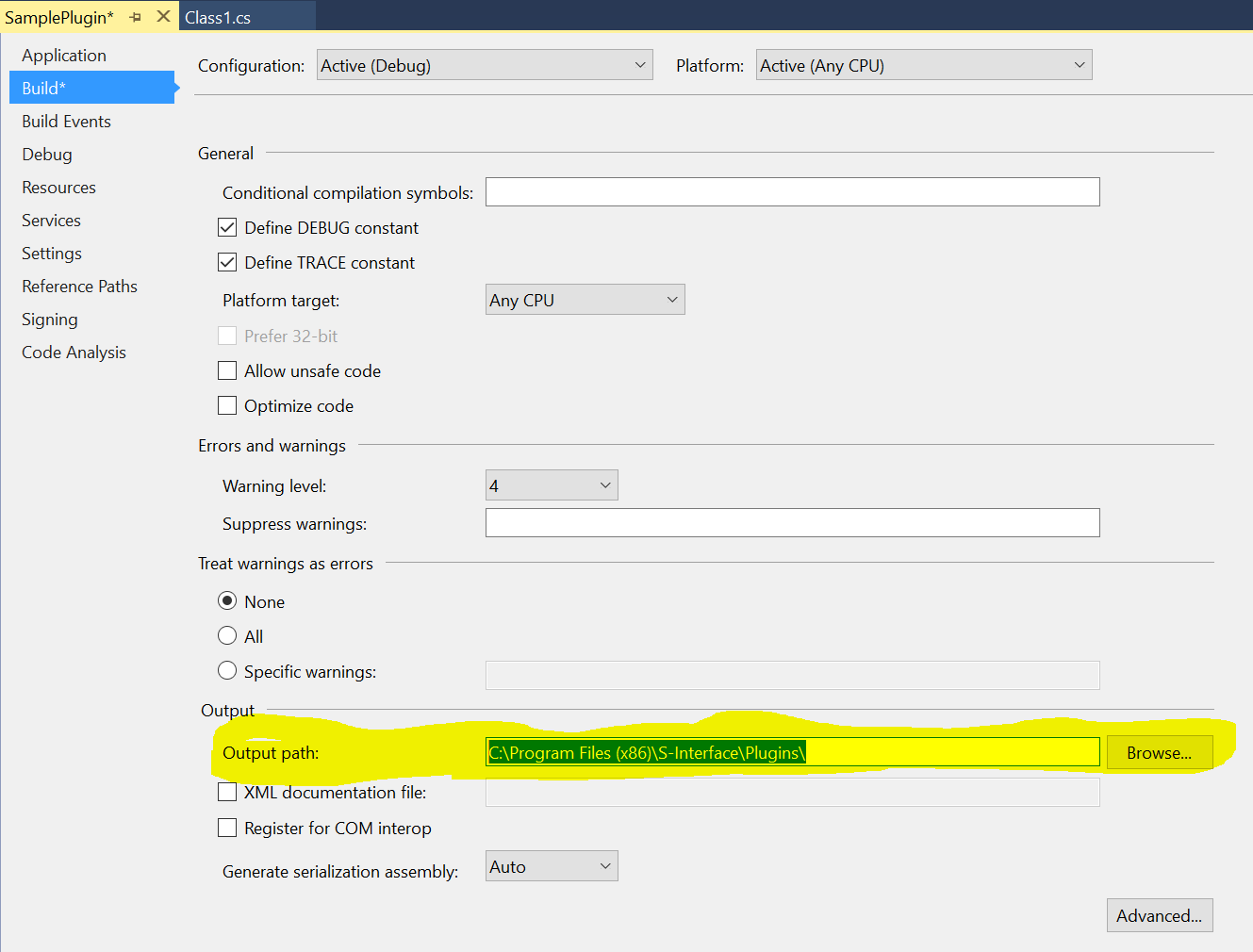
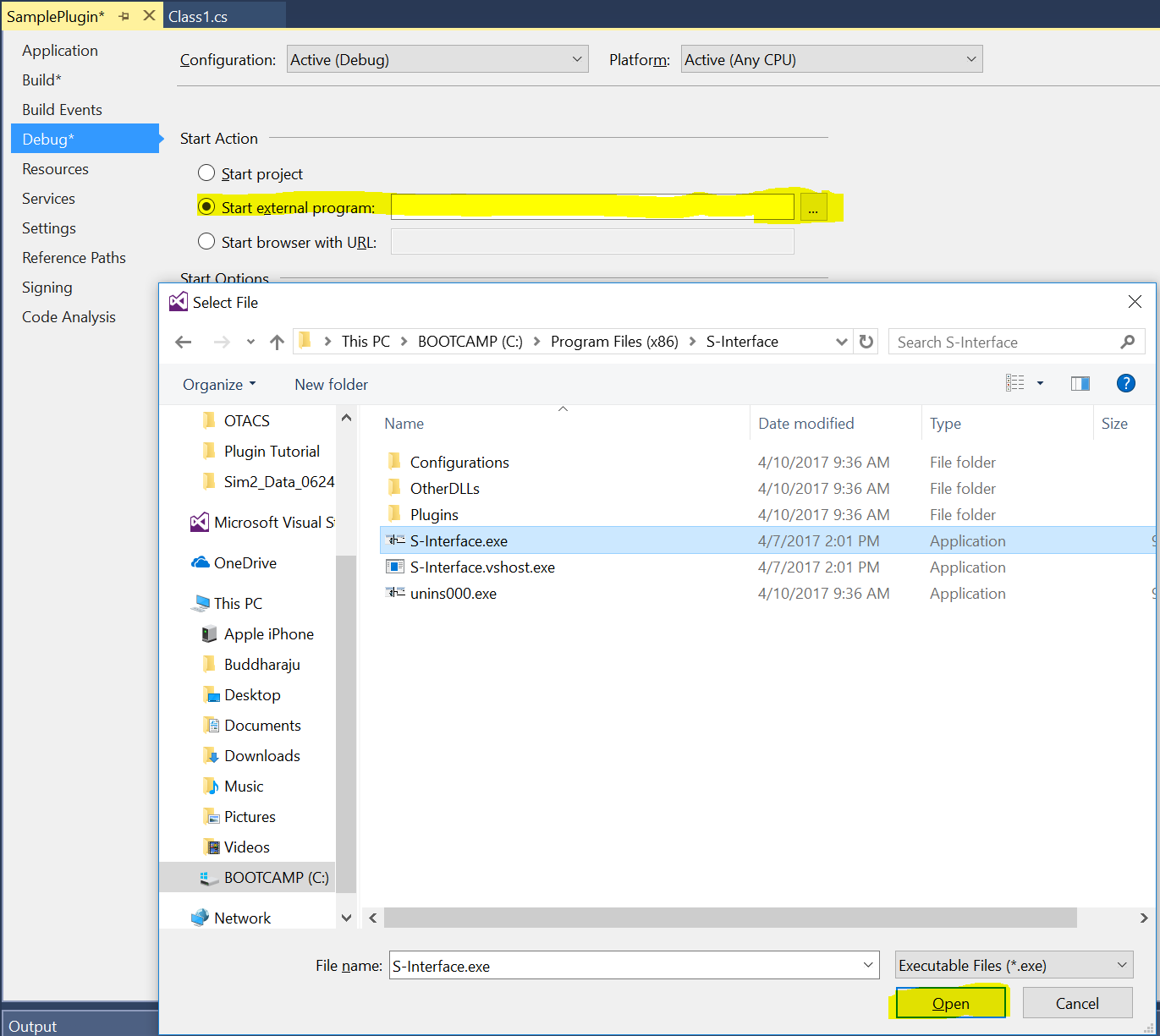
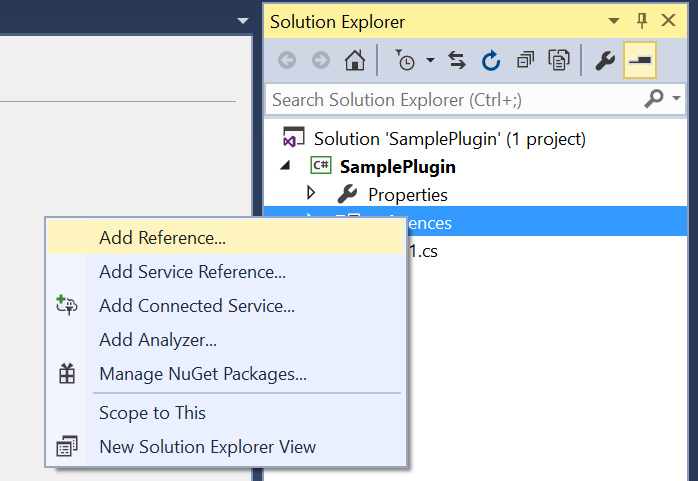
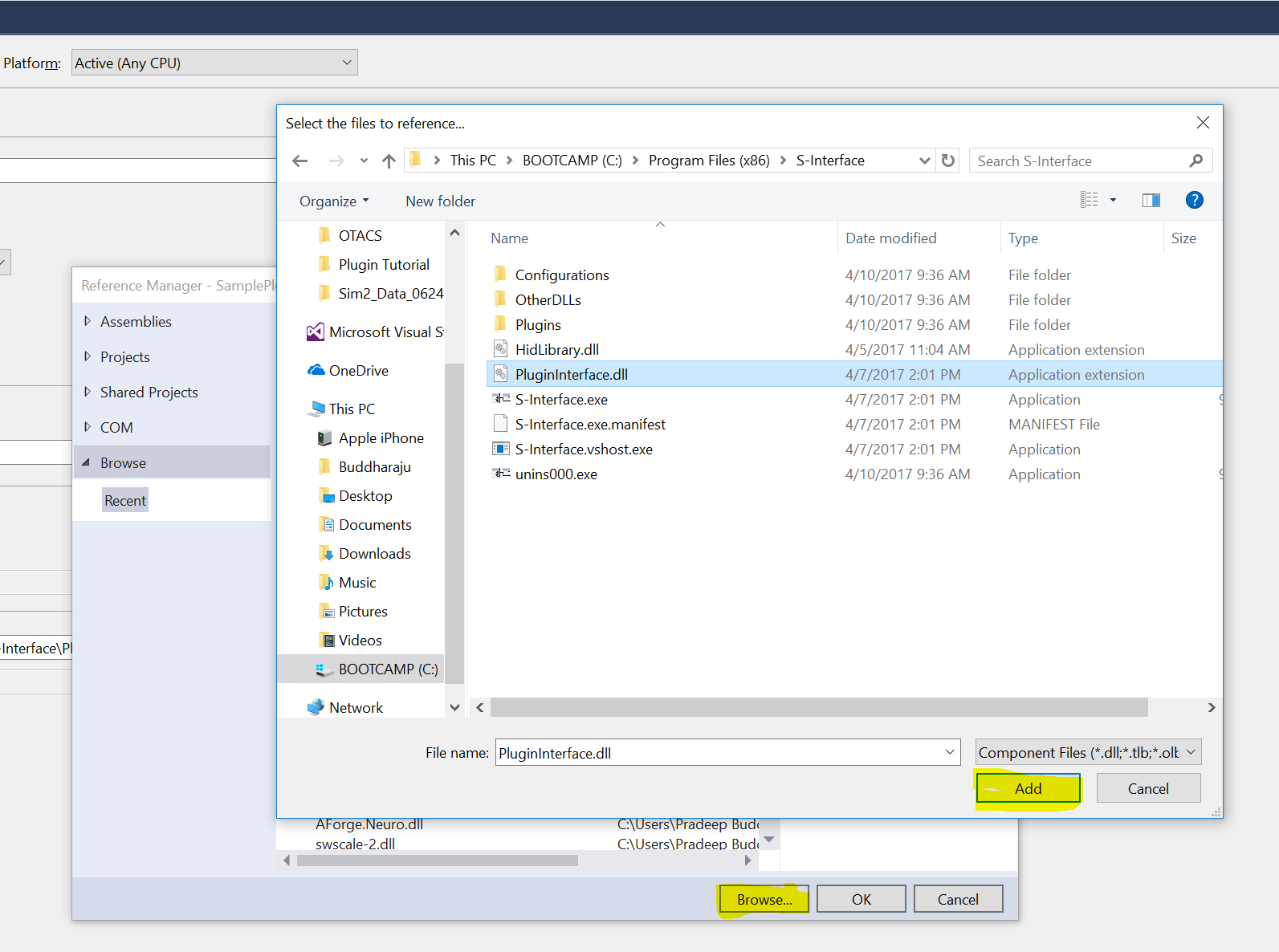
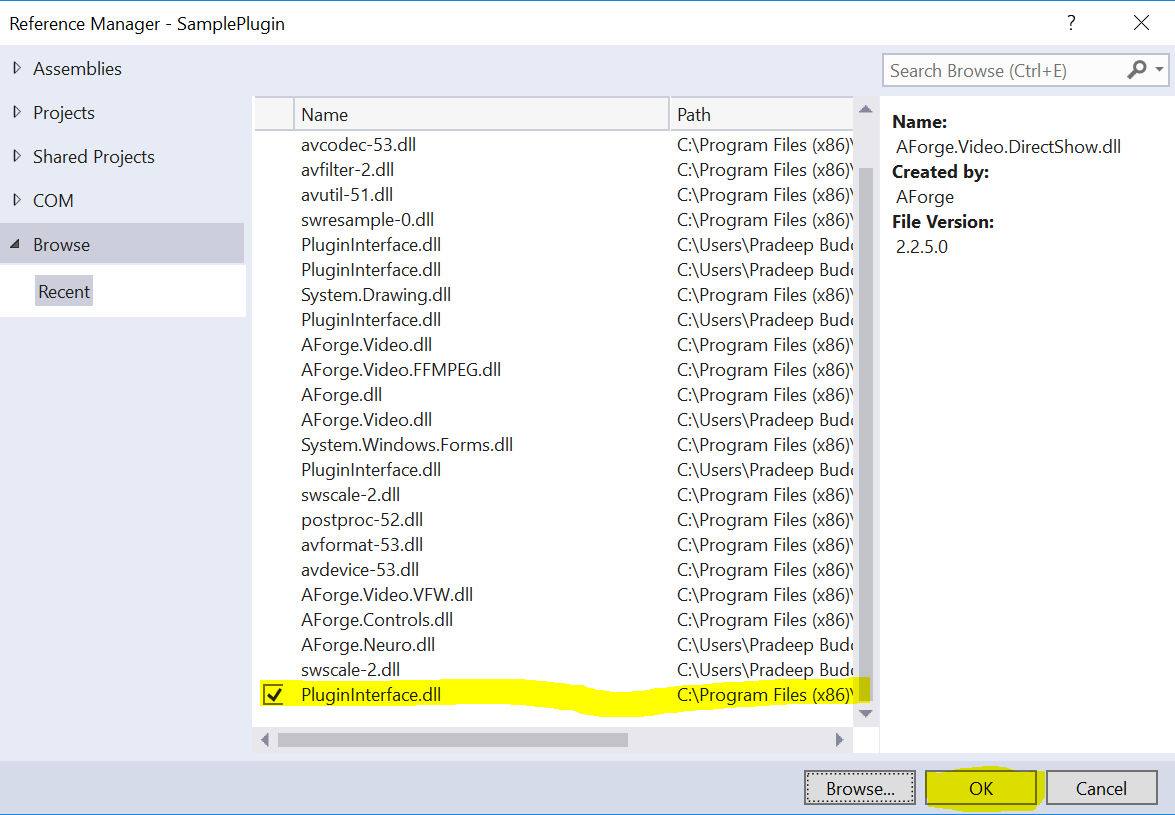
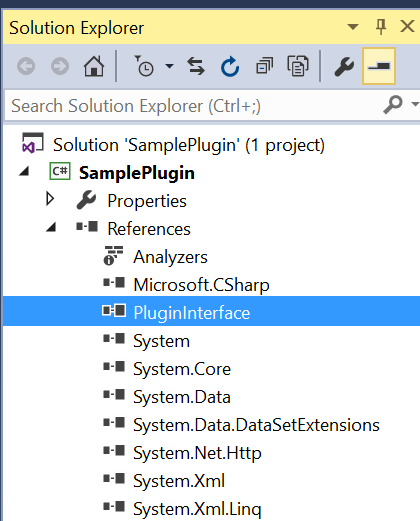
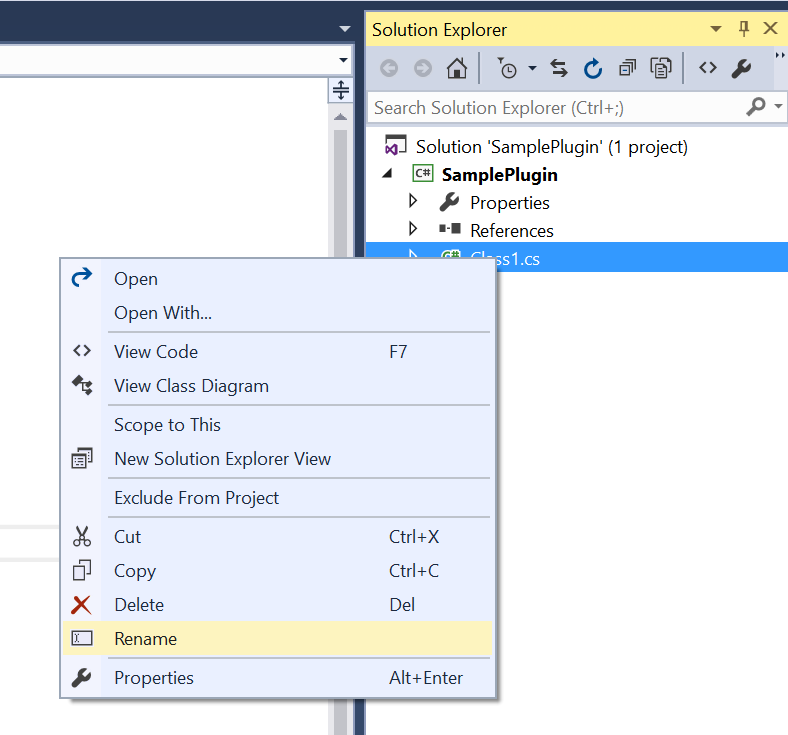
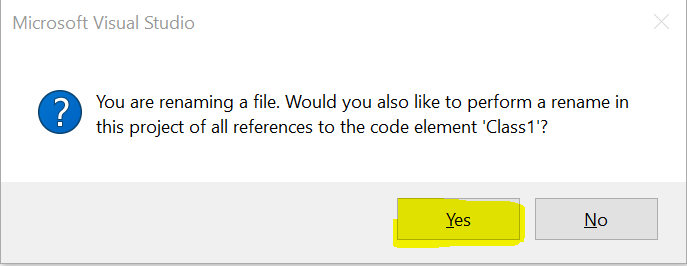
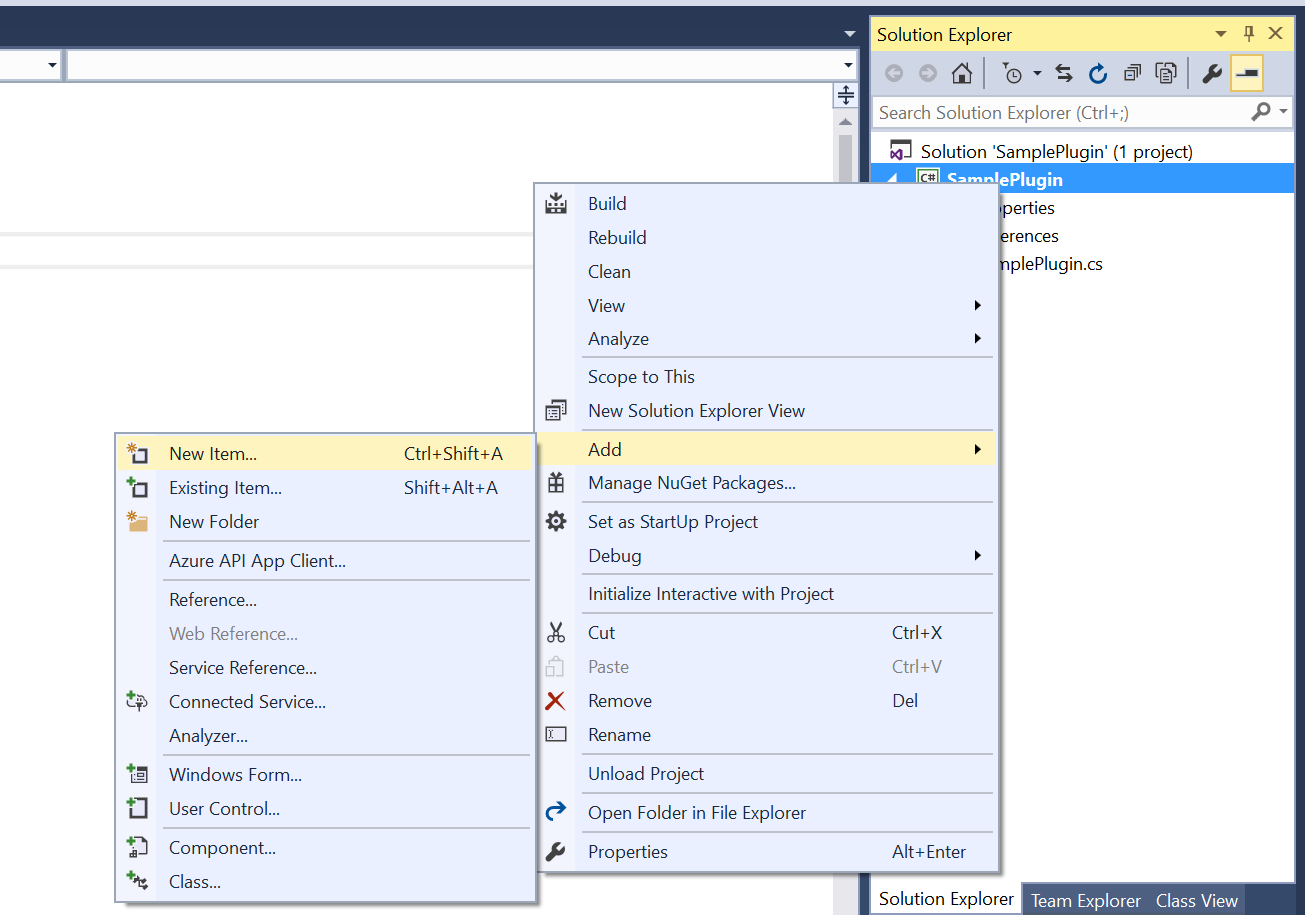
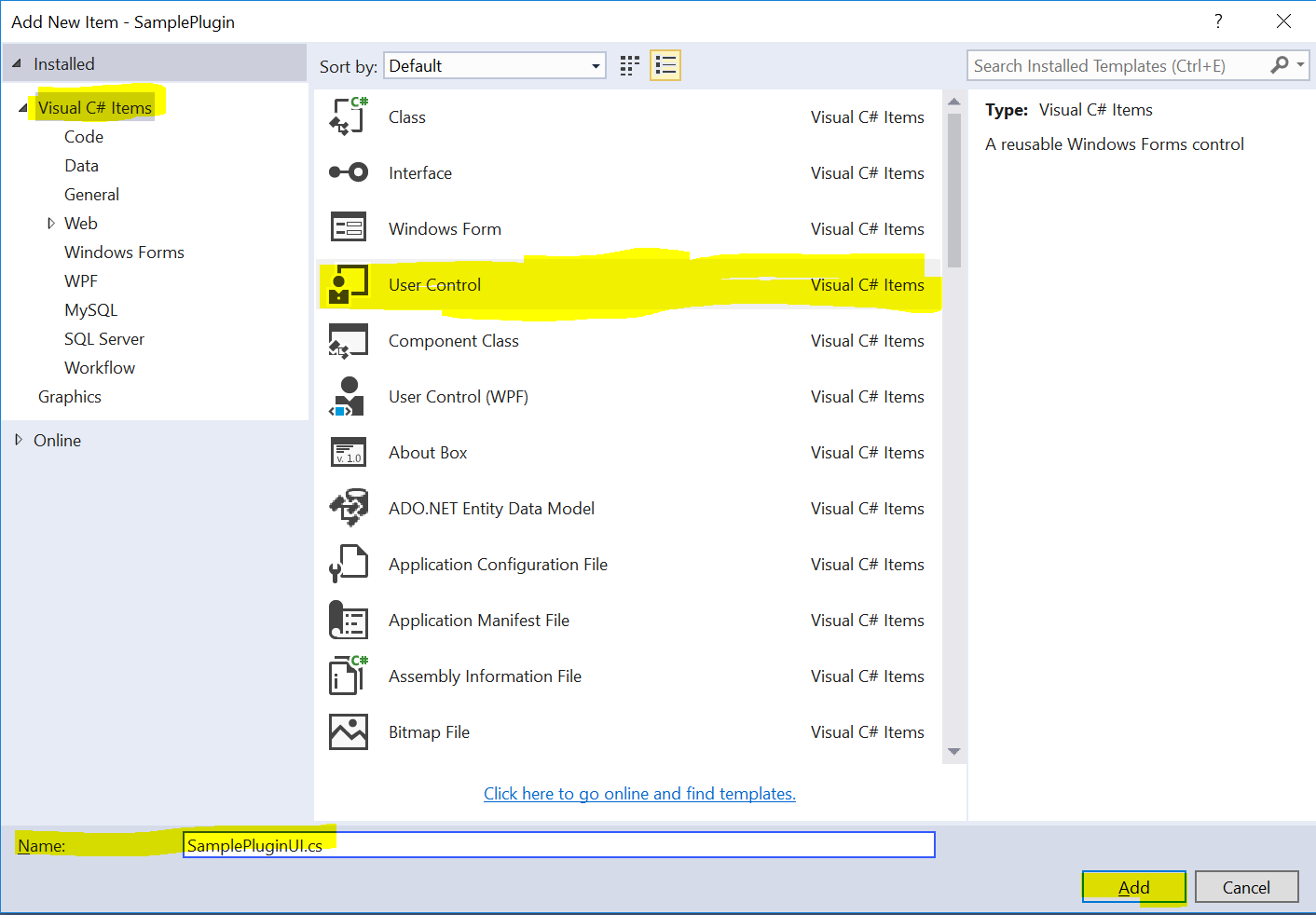
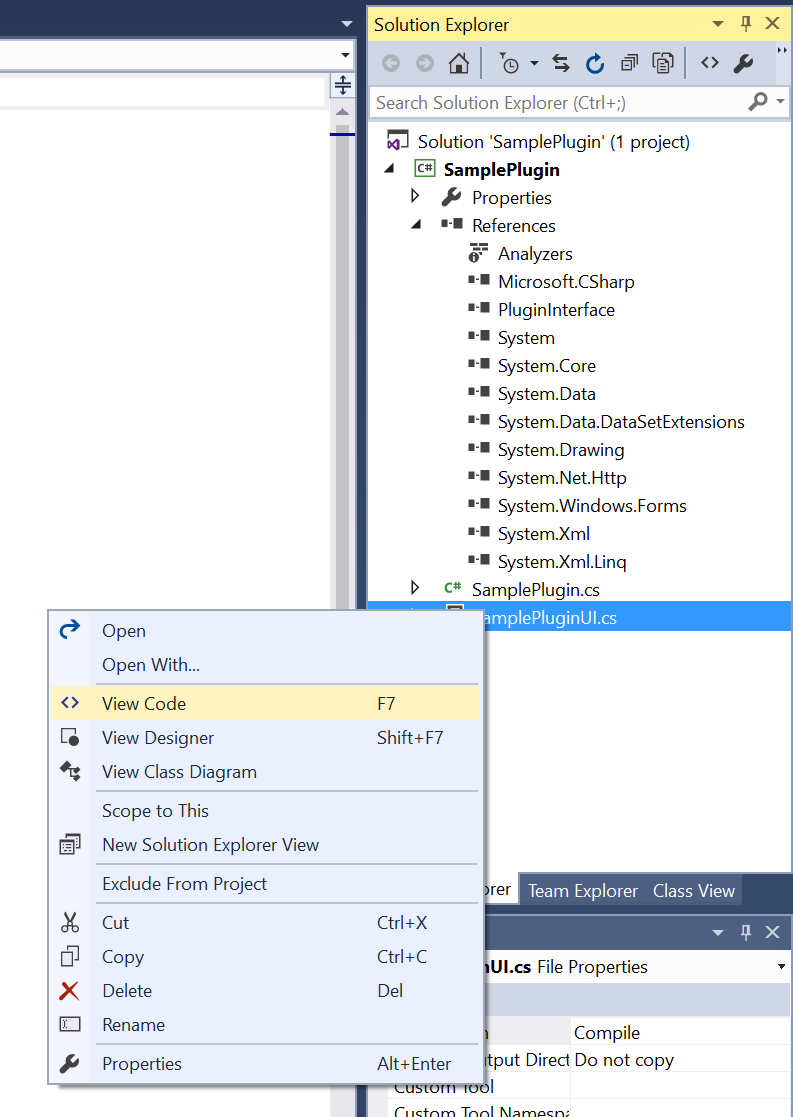
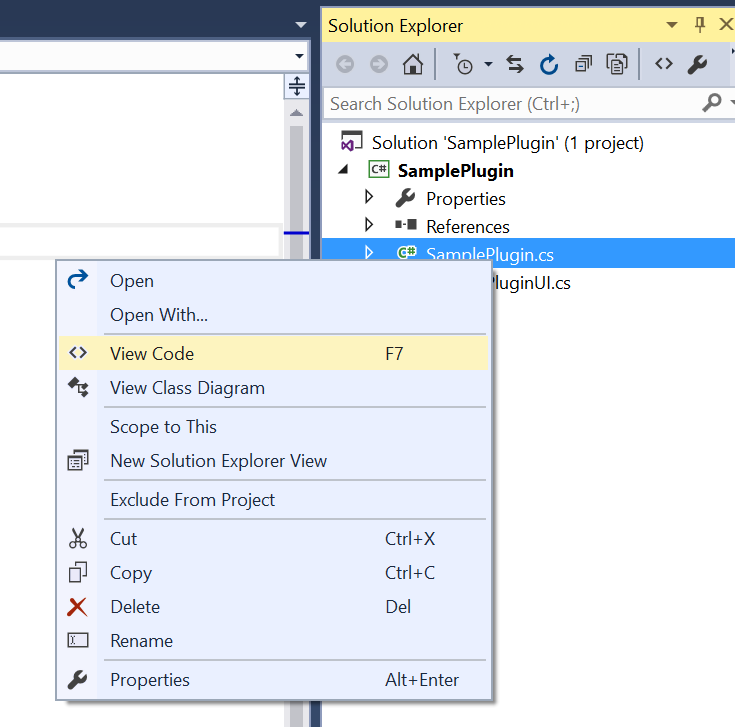
1. Create a New Project of type Class Library
   * From the ***Visual Studio*** menu, select ***File*** 🡪 ***New*** 🡪 ***Project***
   * From the window that pops up, expand the ***Templates*** group, select ***Visual C#***, and on the right, select ***Class Library***.
   * Enter the appropriate ***Name*** for your plugin, choose the ***Location*** where you would like to save the project, and click ***OK***.



1. Change the Project Properties
   * In the ***Solution Explorer***, right click on your project name, select ***Properties***  
      
   * Select ***Build*** tab on the left, and change the ***Output Path*** to ***C:\Program Files (x86)\S-Interface\Plugins\*** 
   * Select ***Debug*** tab on the left, and from the ***Start Action*** group select ***Start external program***. Click on the browse button on the right, and navigate to the path ***C:\Program Files (x86)\S-Interface\*** and select ***S-interface.exe*** file, and click ***Open***



1. Add a reference to ***PluginInterface.dll***
   * In the ***Solution Explorer***, right click on ***References***, select ***Add Reference…*** 
   * In the Reference Manager window, click ***Browse***. Navigate to the folder ***C:\Program Files (x86)\S-Interface\***, select ***PluginInterface.dll***, and click ***Add*** 
   * In the Reference Manager window, make sure that the ***PluginInterface.dll*** is selected, and click ***OK***.   
      
   * You should now see ***PluginInterface*** in the list of your project ***References*** 
2. Rename the Class
   * Right click on ***Class1.cs***, select ***Rename***, and enter your plugin name  
      
   * In the warning window, click ***YES***  
     
3. Create a User Control for the plugin
   * Right click on the project name, select ***Add***, and select ***New Item…***  
      
   * In the ***Add New Item*** window, make sure ***Visual C# Items*** is selected on the left, and then select ***User Control*** on the right. Enter an appropriate ***Name*** to the user control, and click ***Add*** 
4. Update the User Control class
   * Right click on ***SamplePluginUI.cs*** class, and select ***View Code***  
      
   * In ***SamplePlugin.cs***, add the following member variable:  
     SamplePlugin plugin;
   * In ***SamplePlugin.cs***, add the following constructor overload method:  
     public SamplePluginUI(SamplePlugin setPlugin)  
     {  
      InitializeComponent();  
      plugin = setPlugin;  
     }
5. Update the plugin class
   * Right click on ***SamplePlugin.cs*** class, and select ***View Code*** 
   * In ***SamplePlugin.cs*** class, add the following namespace declarations:

using PluginInterface;

using System.Collections;

* + In ***SamplePlugin.cs*** class, modify the ***SamplePlugin.cs*** class declaration to inherit from ***IPlugin*** interface:

public class SamplePlugin: IPlugin

* + Suppose your plugin need two input pins (one ***Critical*** pin of type ***string*** named ***sampleInputPin1*** and one ***Optional*** pin of type ***integer*** named ***sampleInputPin2***) and one output pin (of type ***integer*** named ***sampleOutputPin***). In ***SamplePlugin.cs*** class, add the following member variables:

string myName = "SamplePlugin";

string myDescription = "This is a sample plugin";

string myAuthor = "Pradeep Buddharaju";

string myVersion = "1.0.0";

IPluginHost myHost = null;

int myID = -1;

System.Windows.Forms.UserControl myMainInterface;

ArrayList inPins = new ArrayList();

ArrayList outPins = new ArrayList();

IPin sampleInputPin1 = null;

IPin sampleInputPin2 = null;

IPin sampleOutputPin = null;

public string Description

{

get { return myDescription; }

}

public string Author

{

get { return myAuthor; }

}

public IPluginHost Host

{

get { return myHost; }

set { myHost = value; }

}

public int MyID

{

get { return myID; }

set { myID = value; }

}

public string Name

{

get { return myName; }

}

public System.Windows.Forms.UserControl MainInterface

{

get { return myMainInterface; }

}

public string Version

{

get { return myVersion; }

}

public ArrayList InputPins

{

get { return inPins; }

}

public ArrayList OutputPins

{

get { return outPins; }

}

* + In ***SamplePlugin.cs*** class, add the following required methods:

public void Initialize()

{

//This is the first Function called by the host...

sampleInputPin1 = Host.LoadOrCreatePin("Input1", PinCategory.Critical, new Type[] { typeof(StringData) });

inPins.Add(sampleInputPin1);

sampleInputPin2 = Host.LoadOrCreatePin("Input2", PinCategory.Optional, new Type[] { typeof(IntegerData) });

inPins.Add(sampleInputPin2);

sampleOutputPin = Host.LoadOrCreatePin("Output1", PinCategory.Optional, new Type[] { typeof(IntegerData) });

outPins.Add(sampleOutputPin);

myMainInterface = new SamplePluginUI(this);

}

public void Dispose()

{

//Put any cleanup code in here for when the program is stopped

}

public void Process(IPin pin, IPinData input)

{

//Put process code here

// this.Host.SignalCriticalProcessingIsFinished(inImage, this);

if (pin == sampleInputPin1)

{

if (input != null)

{

//code to process the input pin...

//Suppose you are ready to send output on output pin

this.Host.SendData(sampleOutputPin, new IntegerData(0), this);

//Since sampleInputPin1 is a critical pin, signal that its processing is done

this.Host.SignalCriticalProcessingIsFinished(sampleInputPin1, this);

}

}

// this.Host.SignalCriticalProcessingIsFinished(inImage, this);

if (pin == sampleInputPin2)

{

if (input != null)

{

//code to process the input pin...

//Note that sampleInputPin2 is a optional pin, so no need to signal that its processing is done

}

}

}

1. ***Build*** the solution and click on ***Start*** to run the executable. S-Interface UI will be loaded, and if you choose your plugin in the ***Create Configuration*** dropdown, you will see your plugin graph with the corresponding input and output pins.   
    