

# CSE 445/598 – Assignments 1&2 Additional Tutorials

## Table of Contents

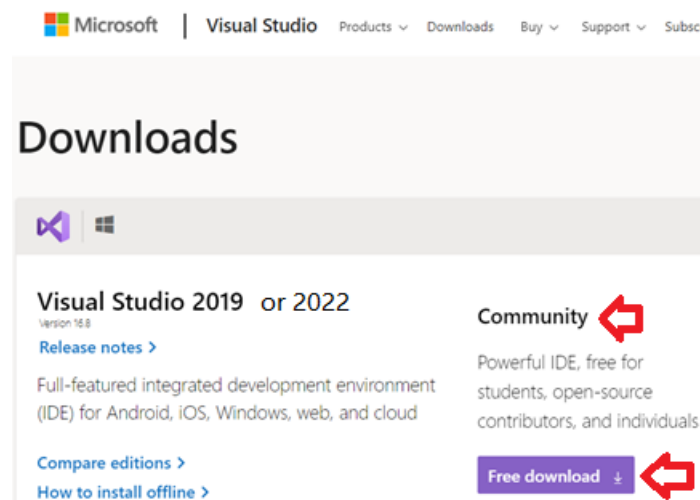
Section I. Downloading Visual Studio .....	1
Section II. Creating a WCF Web Service.....	2
Section III. Creating a Windows Forms Application to Consumer the Service .....	7
Section IV. Creating a Web Application to Consume the Service .....	16
FAQ.....	19

Brief Tutorials are given in the Project 1 document.

This document is based on textbook Chapter 3 and Appendix A. This document is from the textbook and it provides additional details. It is assumed that you have installed Visual Studio Community 2019. Visual Studio Community 2022 should work too.

## Section I. Downloading Visual Studio

You can download free version of Visual Studio Community 2019 or Visual Studio Community 2022 directly from Microsoft site: <https://www.visualstudio.com/downloads/>



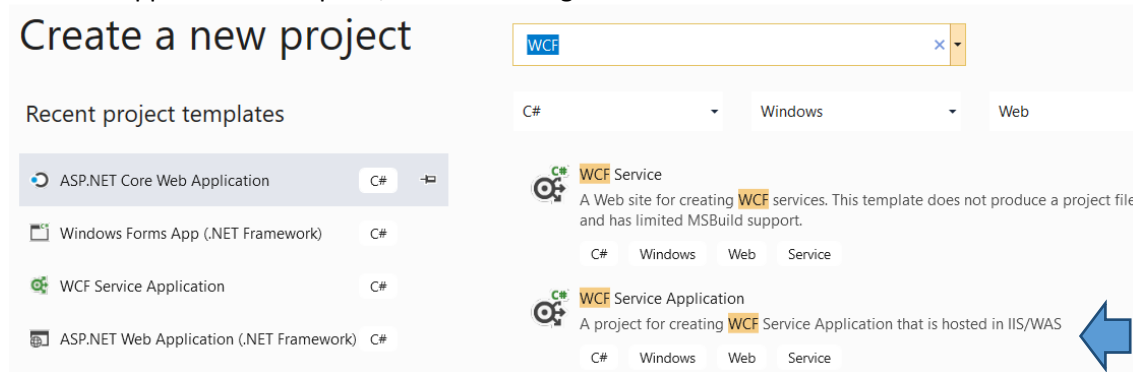
If you have other paid versions, such as Visual Studio Professional or Enterprise editions, they should work fine too.

When you download, make sure you select all Web development components, which will be used in this course. If you have already downloaded Visual Studio, you need to run Visual Studio **Installer**. Search “Installer” in your Windows search bar. Run Visual Studio Installer and modify your Visual Studio to include needed components.

## Section II. Creating a WCF Web Service

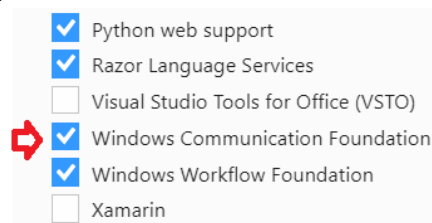
### Step 1: Start Windows Communication Foundation Project

Start Visual Studio 2019 and choose “Create a new project”. You can search “WCF” and then choose “WCF Service Application” template, as shown in Figure 1.



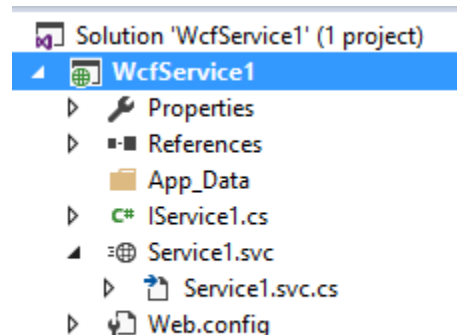
**Figure 1.** Creating a new project and choose WCF Service Application template

If you do not find this template, you need to start Visual Studio **Installer**. Search “Installer” in your Windows search bar. Run Visual Studio Installer and modify your Visual Studio adding components. When you add new components, you will **not** see Windows Communication Foundation in the Workloads tag. You need to click the “Individual components” tag and add Windows Communication Foundation, as shown in Figure 2.



**Figure 2.** Install Windows Communication Foundation through individual components

Your solution should look like this:



Step 2: Develop your service. Start by opening IService1.cs. Define the service contract here.

Original code:

```
1  using System;
2  using System.Collections.Generic;
3  using System.Linq;
4  using System.Runtime.Serialization;
5  using System.ServiceModel;
6  using System.ServiceModel.Web;
7  using System.Text;
8
9  namespace WcfService1
10 {
11     // NOTE: You can use the "Rename" command on the "Refactor" menu to ch
12     [ServiceContract]
13     public interface IService1
14     {
15
16         [OperationContract]
17         string GetData(int value);
18
19         [OperationContract]
20         CompositeType GetDataUsingDataContract(CompositeType composite);
21
22         // TODO: Add your service operations here
23     }
24
25
26     // Use a data contract as illustrated in the sample below to add compo
27     [DataContract]
28     public class CompositeType
29     {
30         bool boolValue = true;
31         string stringValue = "Hello ";
32
33         [DataMember]
34         public bool BoolValue
35         {
36             get { return boolValue; }
37             set { boolValue = value; }
38         }
39
40         [DataMember]
41         public string StringValue
42         {
43             get { return stringValue; }
44             set { stringValue = value; }
45         }
46     }
47 }
48
```

Removing unnecessary contracts and adding a new one:

```
1  using System;
2  using System.Collections.Generic;
3  using System.Linq;
4  using System.Runtime.Serialization;
5  using System.ServiceModel;
6  using System.ServiceModel.Web;
7  using System.Text;
8
9  namespace WcfService1
10 {
11     // NOTE: You can use the "Rename"
12     [ServiceContract]
13     public interface IService1
14     {
15         [OperationContract]
16         double PiValue();
17     }
18 }
19
```

Step 3: Implement your service. Open Service1.svc.cs and implement the service contract here that you defined in step 2.

Original code:

```
1  using System;
2  using System.Collections.Generic;
3  using System.Linq;
4  using System.Runtime.Serialization;
5  using System.ServiceModel;
6  using System.ServiceModel.Web;
7  using System.Text;
8
9  namespace WcfService1
10 {
11     // NOTE: You can use the "Rename" command on the "Refactor" menu to change
12     // NOTE: In order to launch WCF Test Client for testing this service, pleas
13     public class Service1 : IService1
14     {
15         public string GetData(int value)
16         {
17             return string.Format("You entered: {0}", value);
18         }
19
20         public CompositeType GetDataUsingDataContract(CompositeType composite)
21         {
22             if (composite == null)
23             {
24                 throw new ArgumentNullException("composite");
25             }
26             if (composite.BoolValue)
27             {
28                 composite.StringValue += "Suffix";
29             }
30             return composite;
31         }
32     }
33 }
34
```

Removing unnecessary implementations and implementing the PiValue contract.

```
1  using System;
2      using System.Collections.Generic;
3      using System.Linq;
4      using System.Runtime.Serialization;
5      using System.ServiceModel;
6      using System.ServiceModel.Web;
7      using System.Text;
8
9  namespace WcfService1
10 {
11     // NOTE: You can use the "Rename" co
12     // NOTE: In order to launch WCF Test
13     public class Service1 : IService1
14     {
15         public double PiValue()
16         {
17             double pi = Math.PI;
18             return pi;
19         }
20     }
21 }
22
```

Step 4: Test your service. By pressing F5 (or Debug -> Start Debugging), the WCF Test Client should start, as shown in Figure 3. Using this interface, you can test your service.

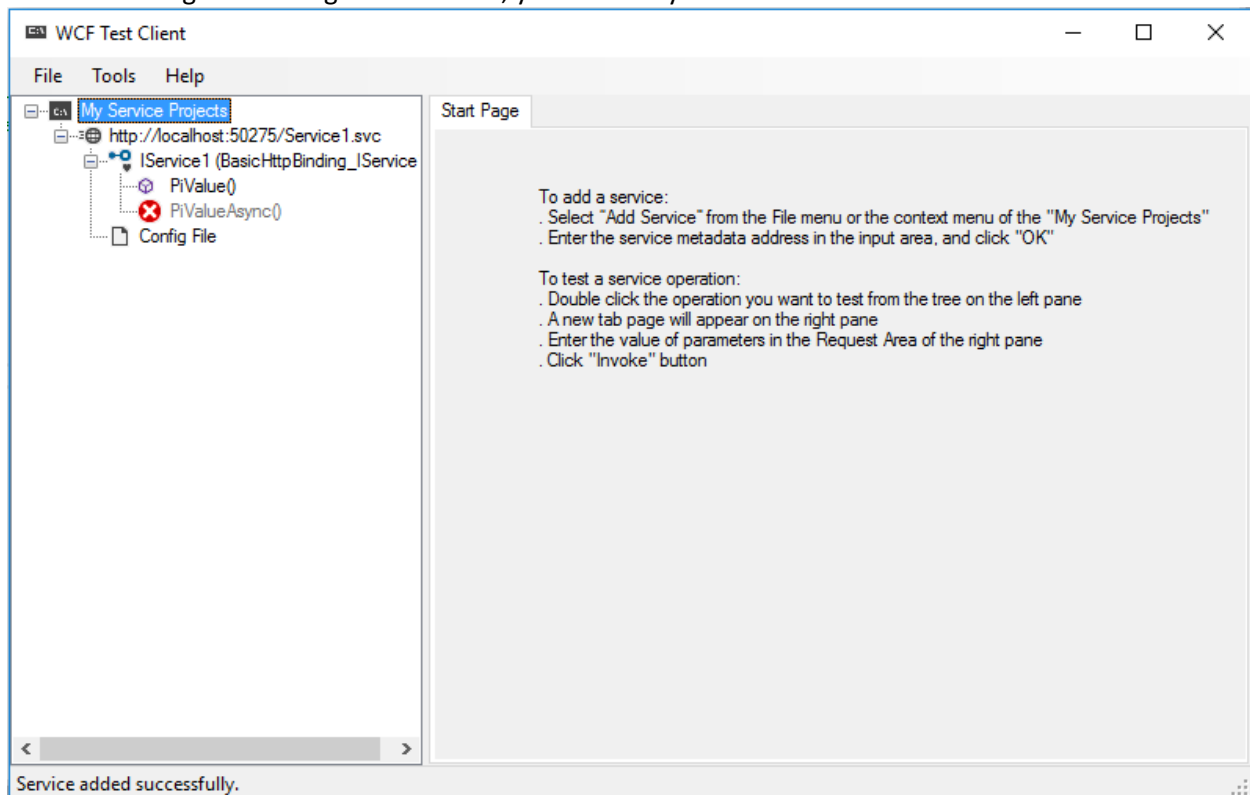


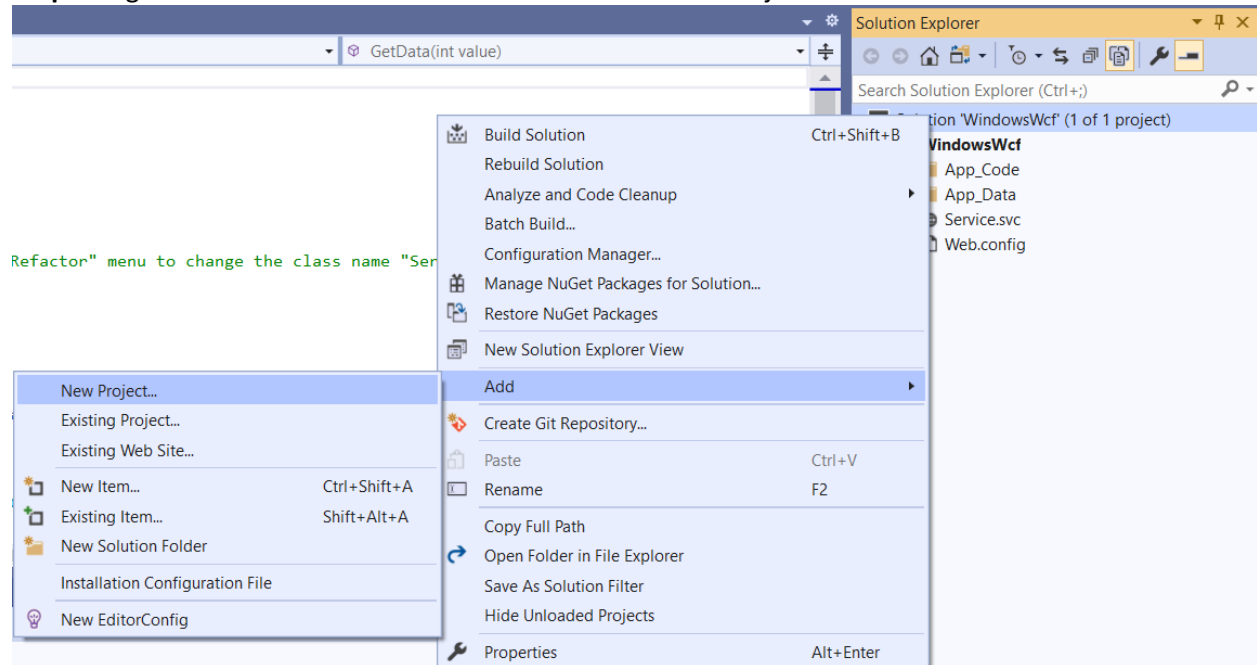
Figure 3. WCF Test Client

## Section III. Creating a Windows Forms Application to Consumer the Service

For Visual Studio 2019 or 2022, you can follow these steps.

Assuming that you have already created a solution and you want to add a Windows Forms application project into the same solution.

**Step 1.** Right click the Solution and choose to add a New → Project:



**Step 2.** A window will open as follows. Choose Windows Forms App (.Net Framework). **Do not** choose Windows Forms App (.Net Core).

# Add a new project

## Recent project templates

- WCF Service C#
- Empty Project C++
- Console App (.NET Framework) C#
- ASP.NET Web Application (.NET Framework) C#
- ASP.NET Core Web Application C#
- Windows Forms App (.NET Framework) C#

Windows

C# Windows All project types

**Windows Forms App (.NET Framework)**  
A project for creating an application with a Windows Forms (WinForms) user interface  
C# Windows Desktop

**Blank App (Universal Windows)**  
A project for a single-page Universal Windows Platform (UWP) app that has no predefined controls or layout.  
C# XAML Windows Xbox Desktop UWP

**Class Library (Universal Windows)**  
A project for creating a managed class library (.dll) for Universal Windows Platform (UWP) apps.  
C# Windows Library UWP

**Windows Forms App (.NET Core)**  
A project for creating an application with a Windows Forms (WinForms) user interface

If you use Visual Studio 2017, the corresponding steps are:

Right click your solution, and go to Add -> New Project.... Then, select the “Windows Forms App (.Net Framework)” Visual C# template.

Solution 'WcfService1' (1 project)

- WcfService1
  - Properties
  - References
  - App\_Data
  - IService1.cs
  - Service1.svc
    - Service1.svc.cs
  - Web.config

Build Solution Ctrl+Shift+B

Rebuild Solution

Clean Solution

Analyze

Batch Build...

Configuration Manager...

Manage NuGet Packages for Solution...

Restore NuGet Packages

Power Commands

New Solution Explorer View

Calculate Code Metrics

**Add**

Set StartUp Projects...

Add Solution to Source Control...

Compare with Unmodified...

Paste Ctrl+V

Rename

Open Folder in File Explorer

Properties Alt+Enter

**New Project...**

Existing Project...

New Web Site...

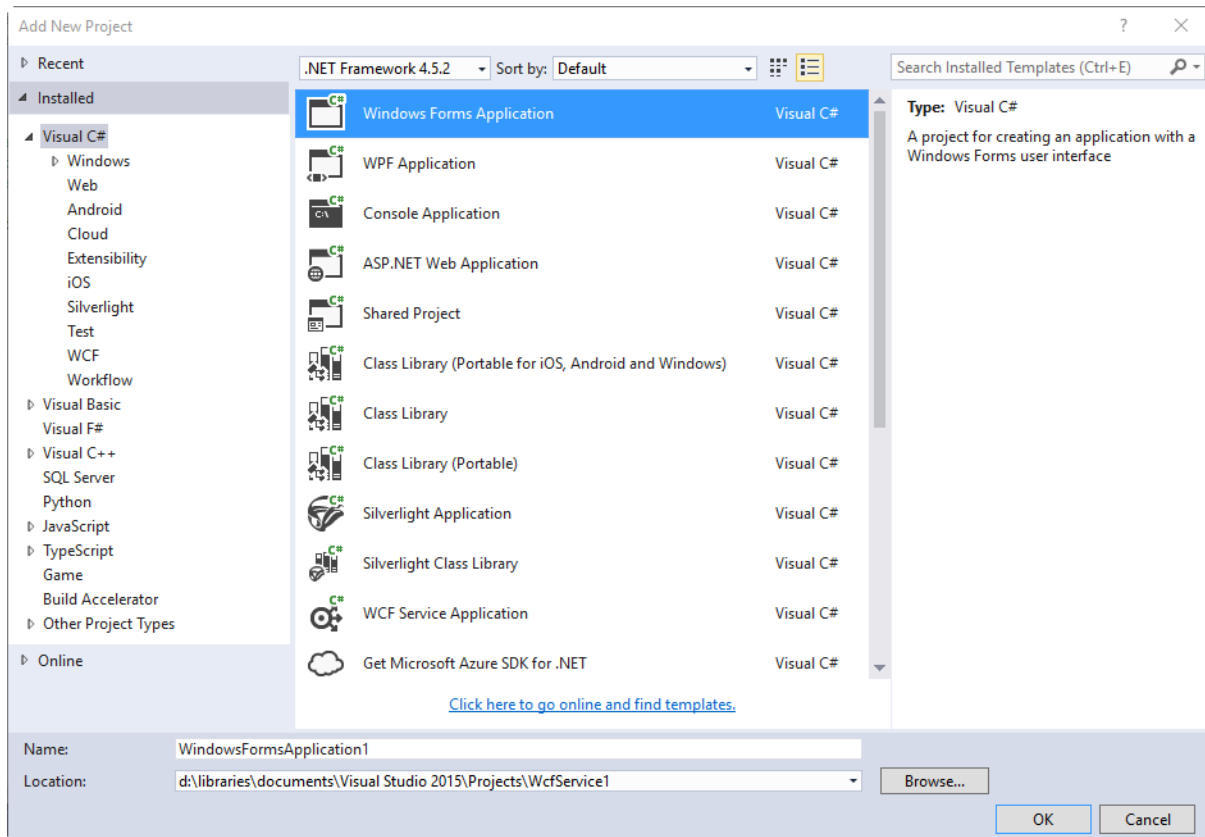
Existing Web Site...

New Item... Ctrl+Shift+A

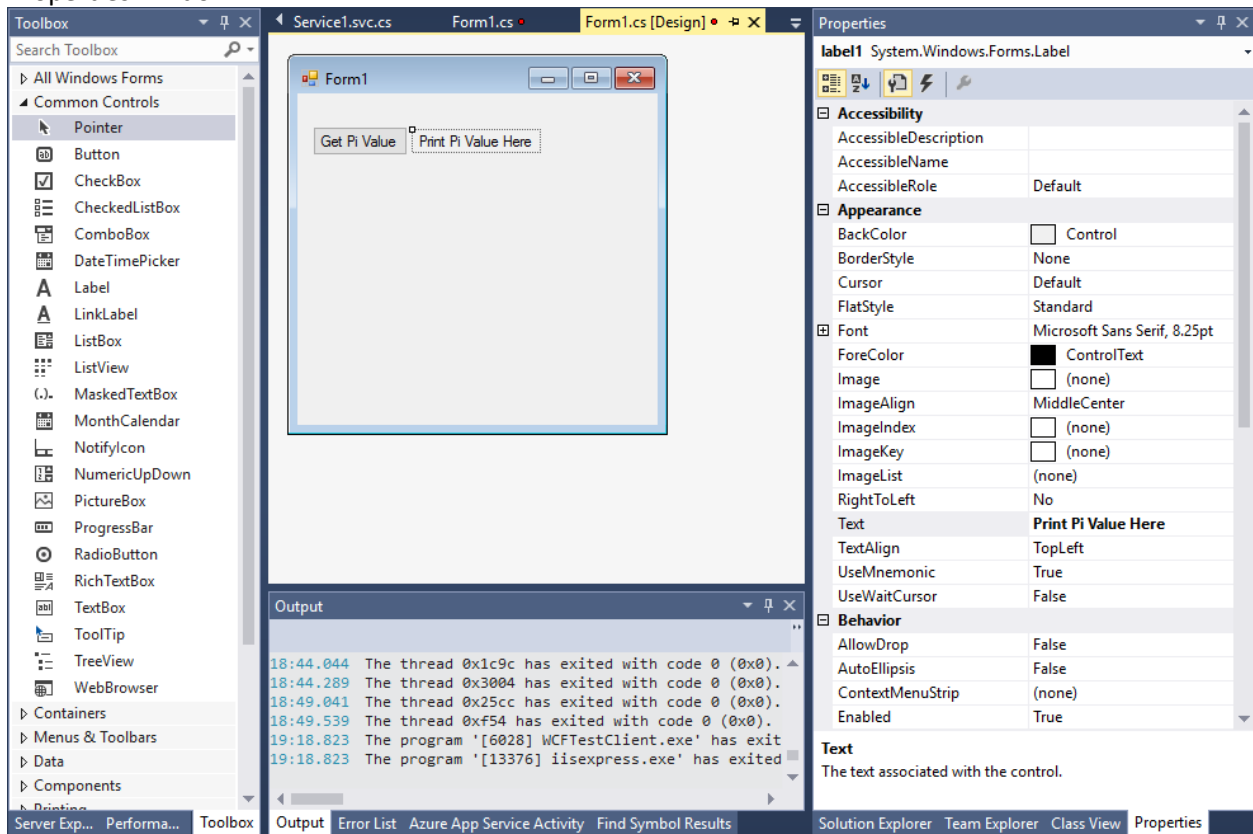
Existing Item... Shift+Alt+A

New Solution Folder

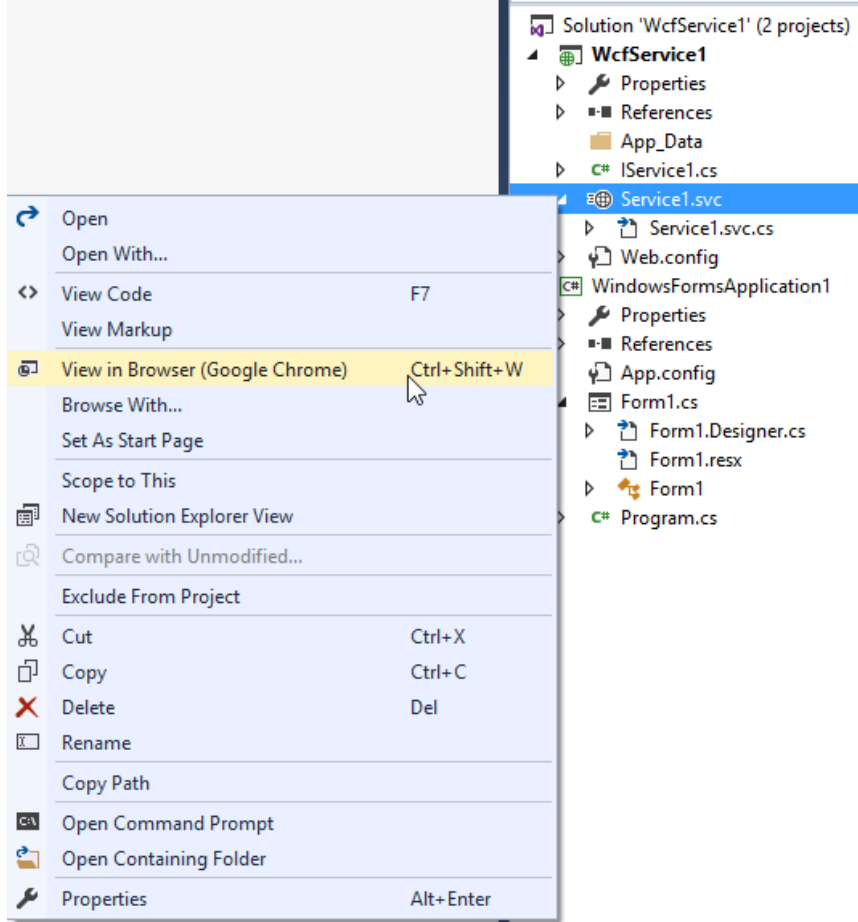




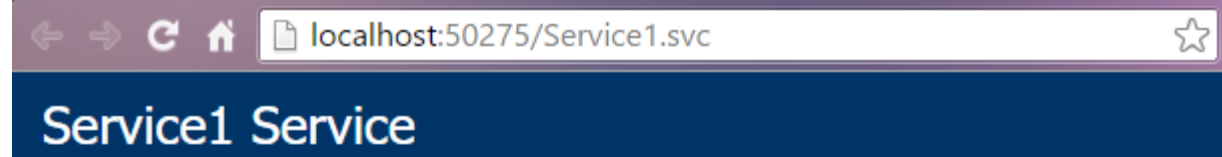
Step 3: Build your GUI using the GUI Builder by dragging components from the Toolbox to your form. You can change names and text by single clicking a component and editing the respective values in the Properties window.



Step 4: Add your service reference to your Windows Forms Application. First, right click your Service1.svc and select “View in Browser.”



You must keep your service running in a Web browser. Find the URL for your service in the Web browser. In this example, it is <http://localhost:50275/Service1.svc>.



You have created a service.

To test this service, you will need to create a client and use it to call the service. You can do this using the following syntax:

```
svcutil.exe http://localhost:50275/Service1.svc?wsdl
```

You can also access the service description as a single file:

```
http://localhost:50275/Service1.svc?singleWsdl
```

This will generate a configuration file and a code file that contains the client class. Add the two files to your application to call the Service. For example:

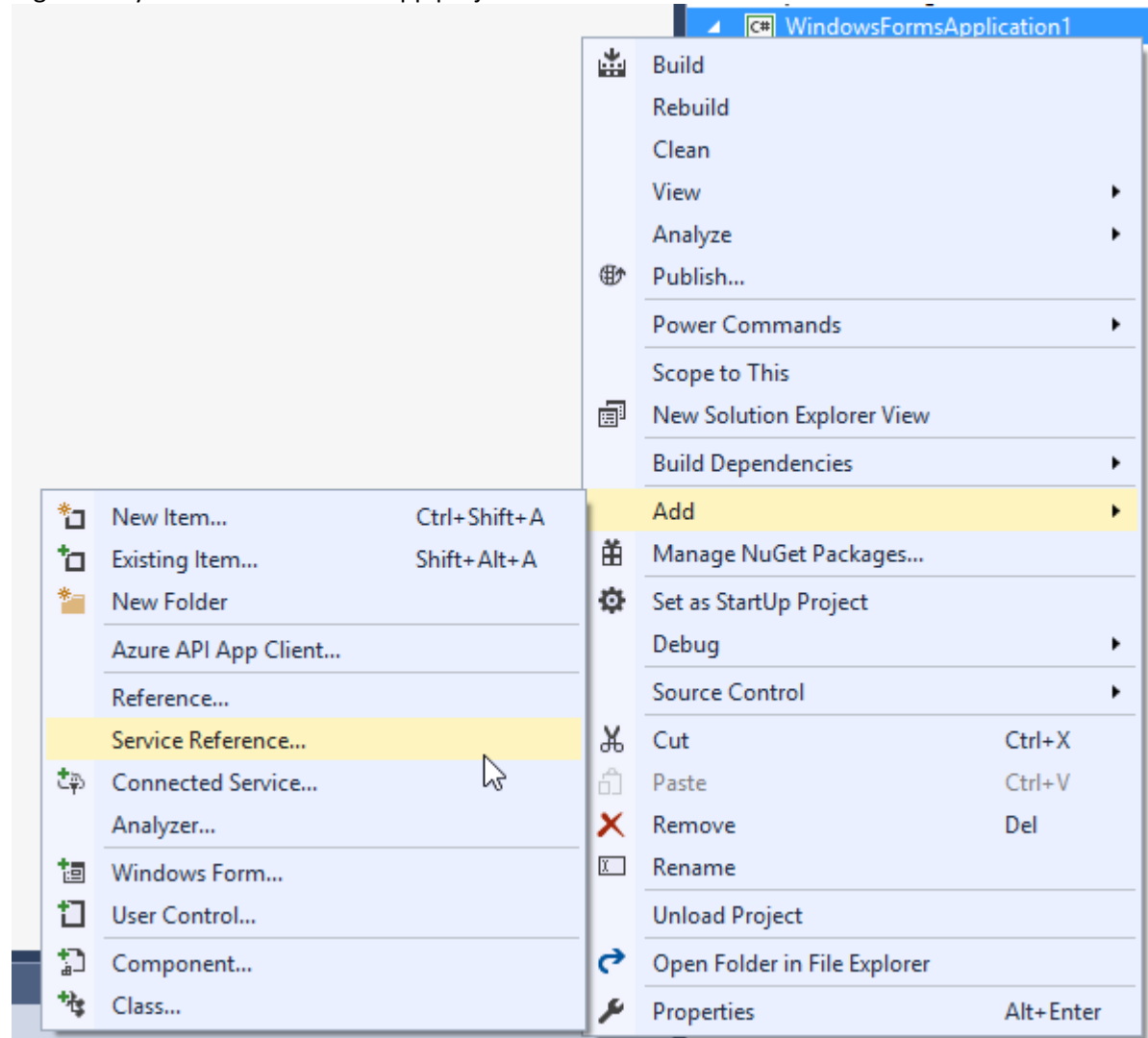
**C#**

```
class Test
{
    static void Main()
    {
        Service1Client client = new Service1Client();

        // Use the 'client' variable to call operations on the service.

        // Always close the client.
        client.Close();
    }
}
```

Right click your Windows Forms App project and select Add -> Service Reference....



Paste your URL in the Address box, and press "Go."

Add Service Reference?×


To see a list of available services on a specific server, enter a service URL and click Go. To browse for available services, click Discover.

Address:

▼GoDiscover▼

Services:

Operations:

▶  Service1

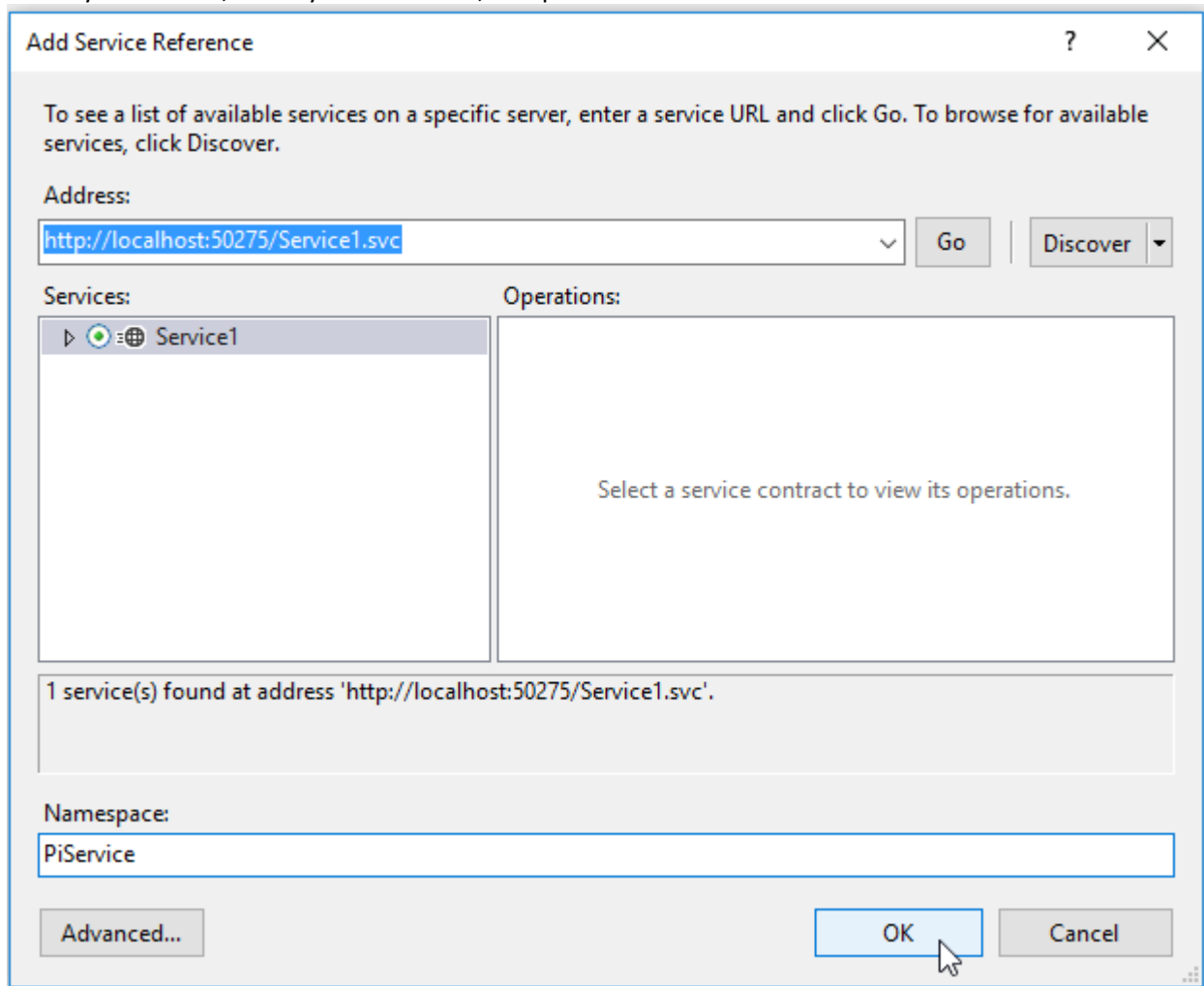
Select a service contract to view its operations.

1 service(s) found at address 'http://localhost:50275/Service1.svc'.

Namespace:

Advanced...OKCancel

Select your service, name your reference, and press “OK.”

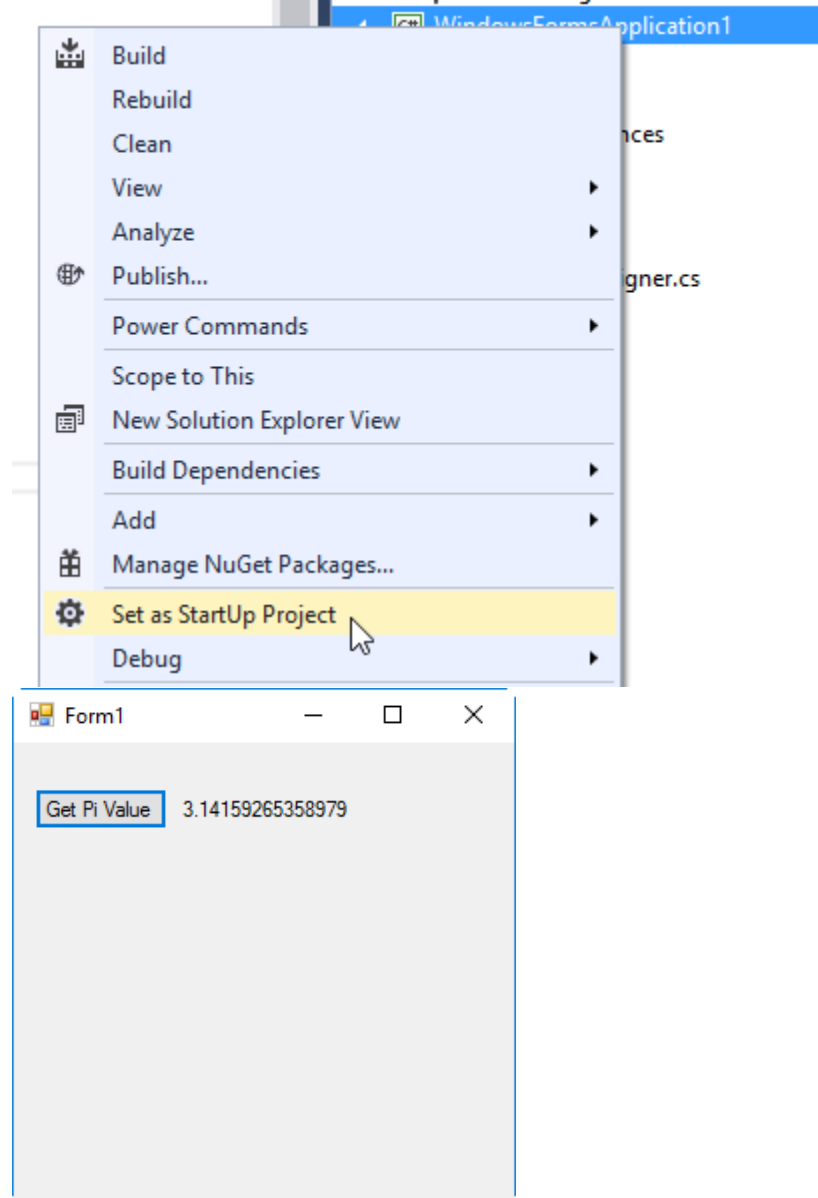


The image shows the 'Add Service Reference' dialog box in Visual Studio. At the top, there is a title bar with a question mark and a close button. Below the title bar, a text box explains: 'To see a list of available services on a specific server, enter a service URL and click Go. To browse for available services, click Discover.' The 'Address:' section contains a text box with 'http://localhost:50275/Service1.svc', a 'Go' button, and a 'Discover' button with a dropdown arrow. Below this, there are two panes: 'Services:' on the left and 'Operations:' on the right. The 'Services:' pane shows a tree view with 'Service1' selected. The 'Operations:' pane is empty and contains the text 'Select a service contract to view its operations.' Below the panes, a status bar says '1 service(s) found at address 'http://localhost:50275/Service1.svc'.' The 'Namespace:' section has a text box containing 'PiService'. At the bottom, there are three buttons: 'Advanced...', 'OK', and 'Cancel'. A mouse cursor is pointing at the 'OK' button.

Step 4: Add a button listener and call your service. To add a button listener, double click your button in the GUI Builder. Then, add the following code to call your service (inside your button handler). This method will be generated in your code behind file (Form1.cs for Windows Forms Applications and Default.aspx.cs for Web Site Applications, when the button is in Form1.cs [Design] and Default.aspx, respectively).

```
private void button1_Click(object sender, EventArgs e)
{
    PiService.Service1Client piService = new PiService.Service1Client();
    double piValue = piService.PiValue();
    piValueLabel1.Text = piValue.ToString();
}
```

Step 5: Test your application. First, make sure your service is running, by following the above step to view your service in the browser. Then, right click your Windows Forms Application, set it as the startup project, and run your code (F5 or Debug -> Start Debugging).

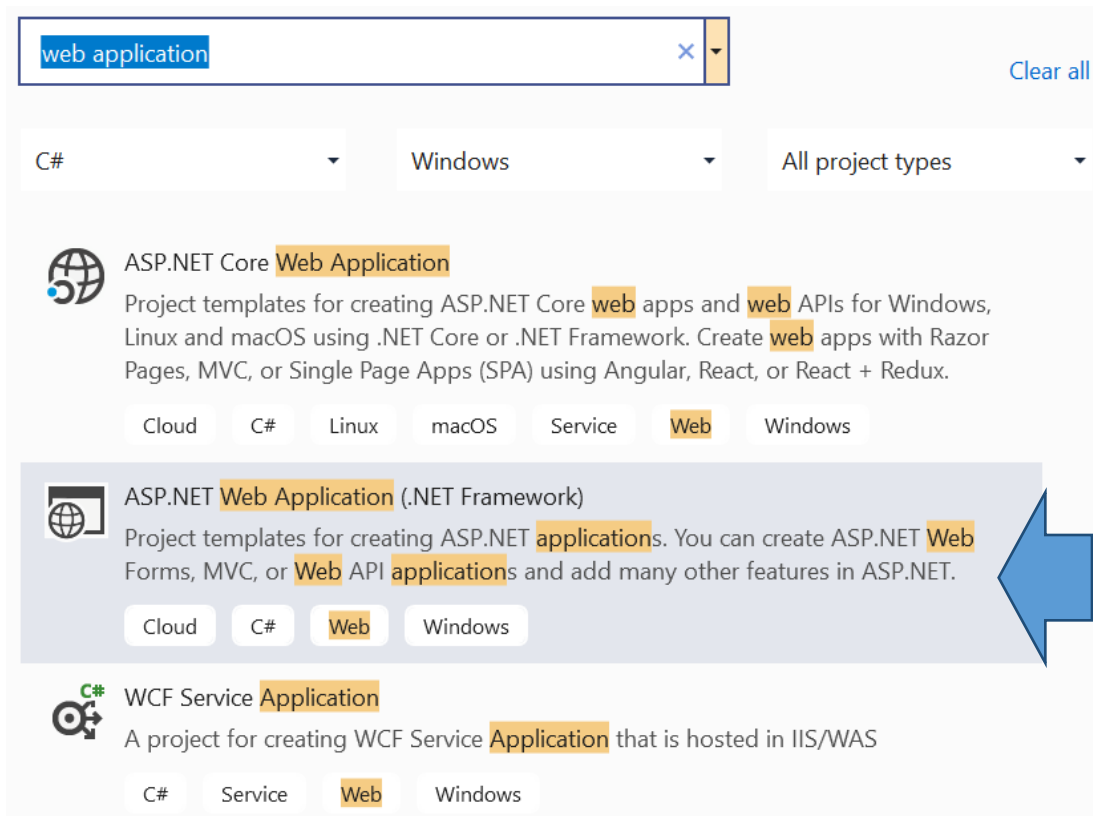


## Section IV. Creating a Web Application to Consume the Service

The steps to create, build, and run a simple Web application are nearly identical to those of a Windows Forms Application.

Step 1. The first difference is in adding a new project. You search web application and then chose ASP .Net Web Application (.Net Framework).

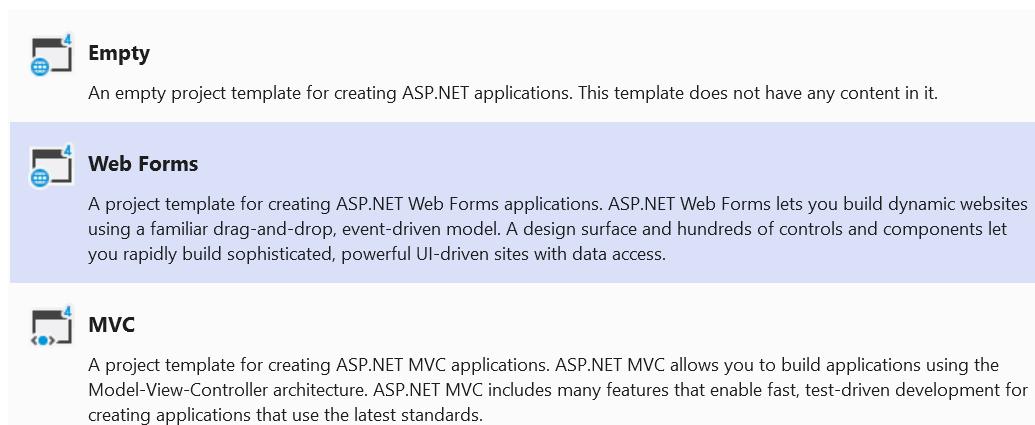




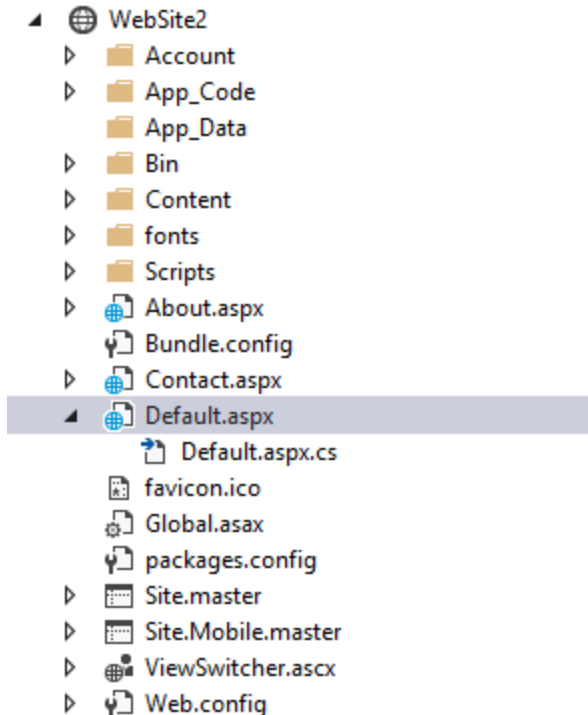
Change the location to your own location. Do not use the default location!

Step 2. Choose “Empty” or “Web Forms” from this new window. Make sure to uncheck the “Host in the cloud” checkbox if you choose this latter method.

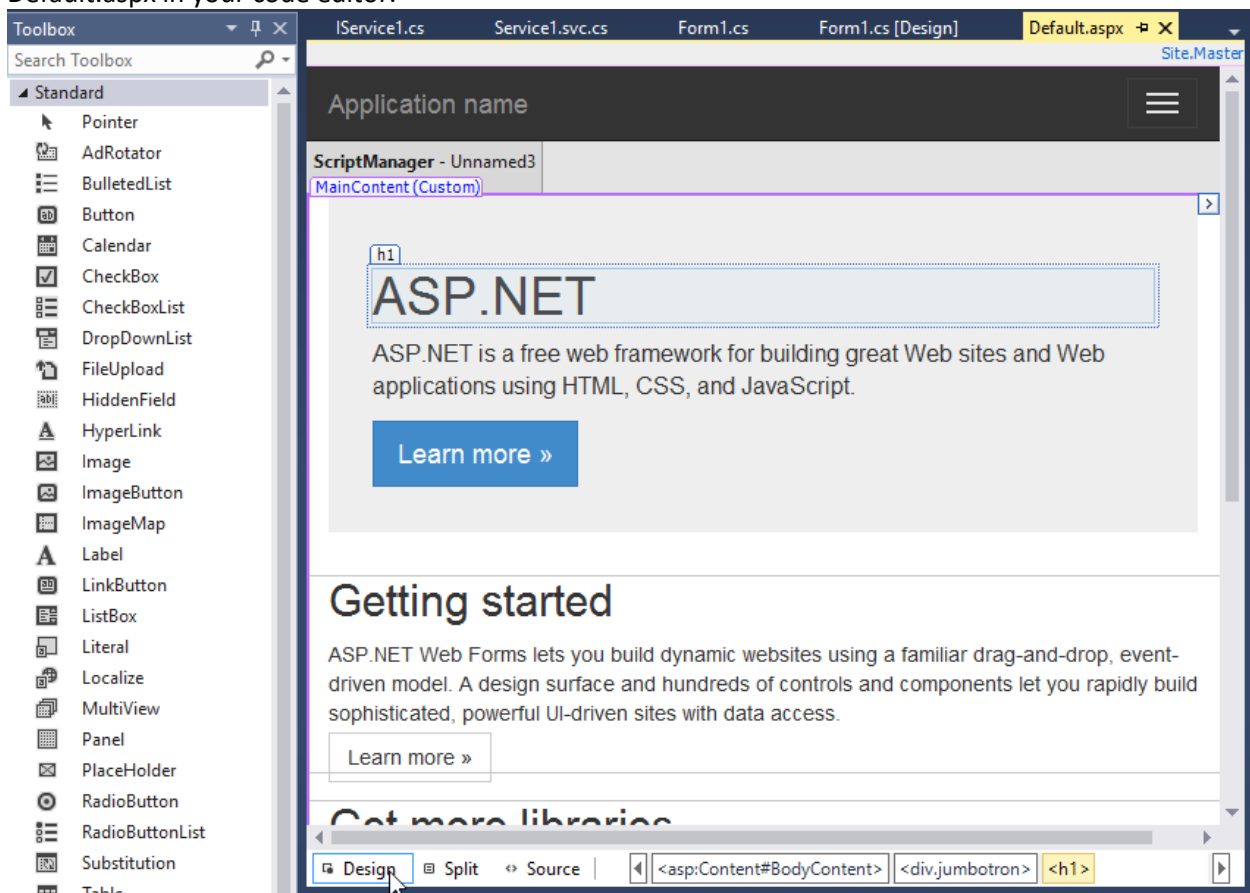
## Create a new ASP.NET Web Application



This tutorial assumes you have added a new Web Forms from the New Web Site... menu. Your project should look like this:



Default.aspx is the equivalent of Form1.cs [Design], and Default.aspx.cs is the code behind (Form1.cs). You can open the drag and drop editor for Default.aspx by selecting the Design tab after opening up Default.aspx in your code editor.



To quickly remove the unnecessary content, you can open up the Source tab. The initial code will look something like this:

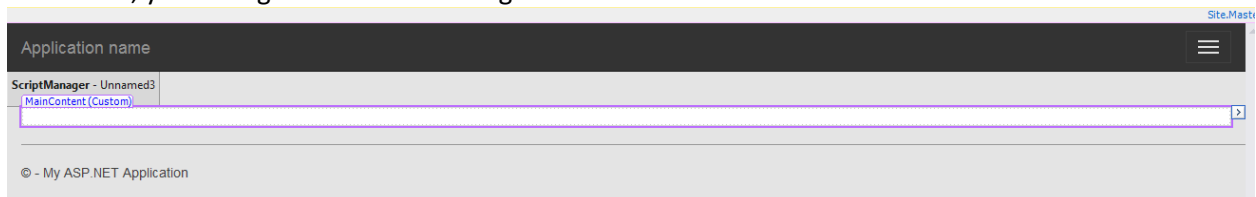
```
1 <%@ Page Title="Home Page" Language="C#" MasterPageFile="~/Site.Master" AutoEventWireup="true" CodeFile="Default.aspx.cs" Inherits="_Default" %>
2
3 <asp:Content ID="BodyContent" ContentPlaceHolderID="MainContent" runat="server">
4
5     <div class="jumbotron">
6         <h1>ASP.NET</h1>
7         <p class="lead">ASP.NET is a free web framework for building great Web sites and Web applications using HTML, CSS, and JavaScript.</p>
8         <p><a href="http://www.asp.net" class="btn btn-primary btn-lg">Learn more &raquo;</a></p>
9     </div>
10
11     <div class="row">
12         <div class="col-md-4">
13             <h2>Getting started</h2>
14             <p>
15                 ASP.NET Web Forms lets you build dynamic websites using a familiar drag-and-drop, event-driven model.
16                 A design surface and hundreds of controls and components let you rapidly build sophisticated, powerful UI-driven sites with data access.
17             </p>
18             <p>
19                 <a class="btn btn-default" href="http://go.microsoft.com/fwlink/?LinkId=301948">Learn more &raquo;</a>
20             </p>
21         </div>
22         <div class="col-md-4">
23             <h2>Get more libraries</h2>
24             <p>
25                 NuGet is a free Visual Studio extension that makes it easy to add, remove, and update libraries and tools in Visual Studio projects.
26             </p>
27             <p>
28                 <a class="btn btn-default" href="http://go.microsoft.com/fwlink/?LinkId=301949">Learn more &raquo;</a>
29             </p>
30         </div>
31         <div class="col-md-4">
32             <h2>Web Hosting</h2>
33             <p>
34                 You can easily find a web hosting company that offers the right mix of features and price for your applications.
35             </p>
36             <p>
37                 <a class="btn btn-default" href="http://go.microsoft.com/fwlink/?LinkId=301950">Learn more &raquo;</a>
38             </p>
39         </div>
40     </div>
41 </asp:Content>
```

By removing all the code inside (but not including) the

`<asp:Content ID="BodyContent" ContentPlaceHolderID="MainContent" runat="server">`  
tags, your code should look like this:

```
1 <%@ Page Title="Home Page" Language="C#" MasterPageFile="~/Site.Master" AutoEventWireup="true" CodeFile="Default.aspx.cs" Inherits="_Default" %>
2
3 <asp:Content ID="BodyContent" ContentPlaceHolderID="MainContent" runat="server">
4
5 </asp:Content>
6
```

As a result, your Design view should change to look like this:



From this point, you can build your GUI using the GUI Editor as in the Windows Forms Application example, add your service reference in the same way, create a button listener in the same way, and even run your code in the same way (except this project will open in a browser, not on your desktop).

## FAQ

Q: What version of Visual Studio/Windows is this?

A: I am running Visual Studio 2019 and 2022 **Community** Edition on Windows 10. The Community version is free for everyone. These steps should work in most versions of prior to 2019 and 2022 versions.

Q: Does it matter how I make my Web application? Should I use Web Application template or Web Site Application template?

A: No, there should be no difference between the above templates for the purposes of this class, as long as Default.aspx and Default.aspx.cs are created or manually added (in the case of the Empty Web Site). Note. You should use .Net framework template, not use .Net Core template. The latter is for creating cross-platform applications, which is more complex.

Q: Why are there red crosses in my WCF Test Client in Figure 3?

A: Those red crosses mark asynchronous services, which cannot be tested from the test client. They do not suggest an error with your code.

Q: How do I use the image verifier service in a Windows Forms Application?

A: There are several main differences between the book example (a Web Site Application) and implementing the image verifier in a Windows Forms Application (desktop application). First, you may need to use the "Picture Box" control instead of an Image control. Next, you do not need to use the "Session" array. You can use normal variables instead.

To grab a verifier string (copied from the textbook, appendix 3):

```
MyImageService.ServiceClient fromService = new MyImageService.ServiceClient();
```

```
string myStr = fromService.GetVerifierString(userLength);
```

where fromService is an instance of the service reference for the image verifier service (available in the neptune repository).

To set the image in a PictureBox (suppose the PictureBox is named myPictureBox):

```
Stream myStream = fromService.GetImage(myStr);
```

```
myPictureBox.Image = Image.FromStream(myStream);
```