

CSE 445 – Assignment 1 Tutorial

Creating a 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 3.4.

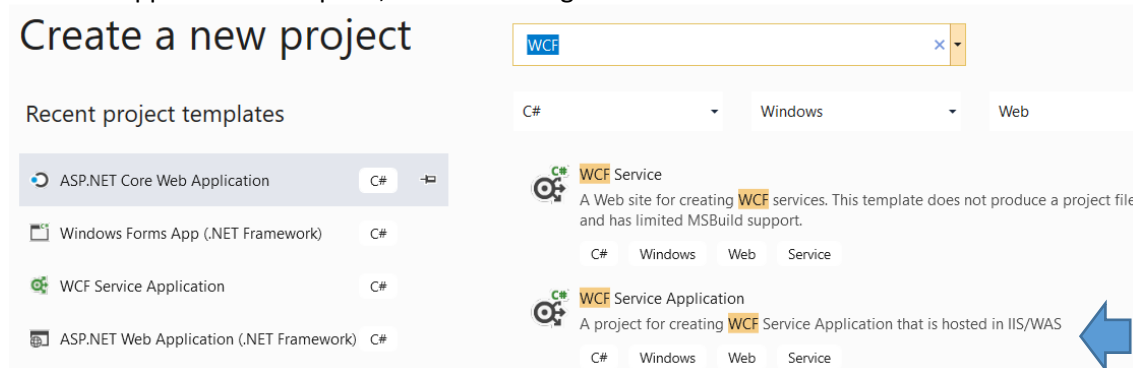


Figure 3.4. Creating a new project and choose WCF Service Application template

If you do not find this template, you need to start Visual Studio Installer. 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 3.5.

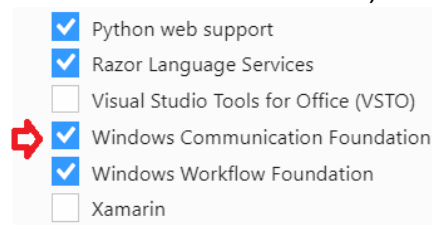
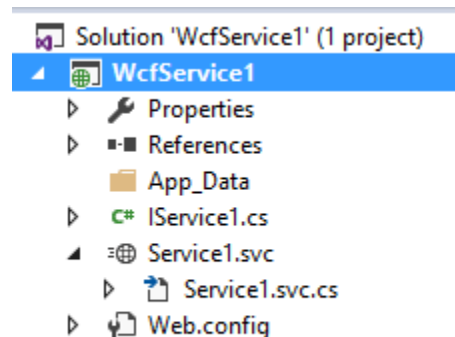


Figure 3.5. Install Windows Communication Foundation through individual components

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 up Service1.svc.cs and implement the service contract here that you defined in step 2.

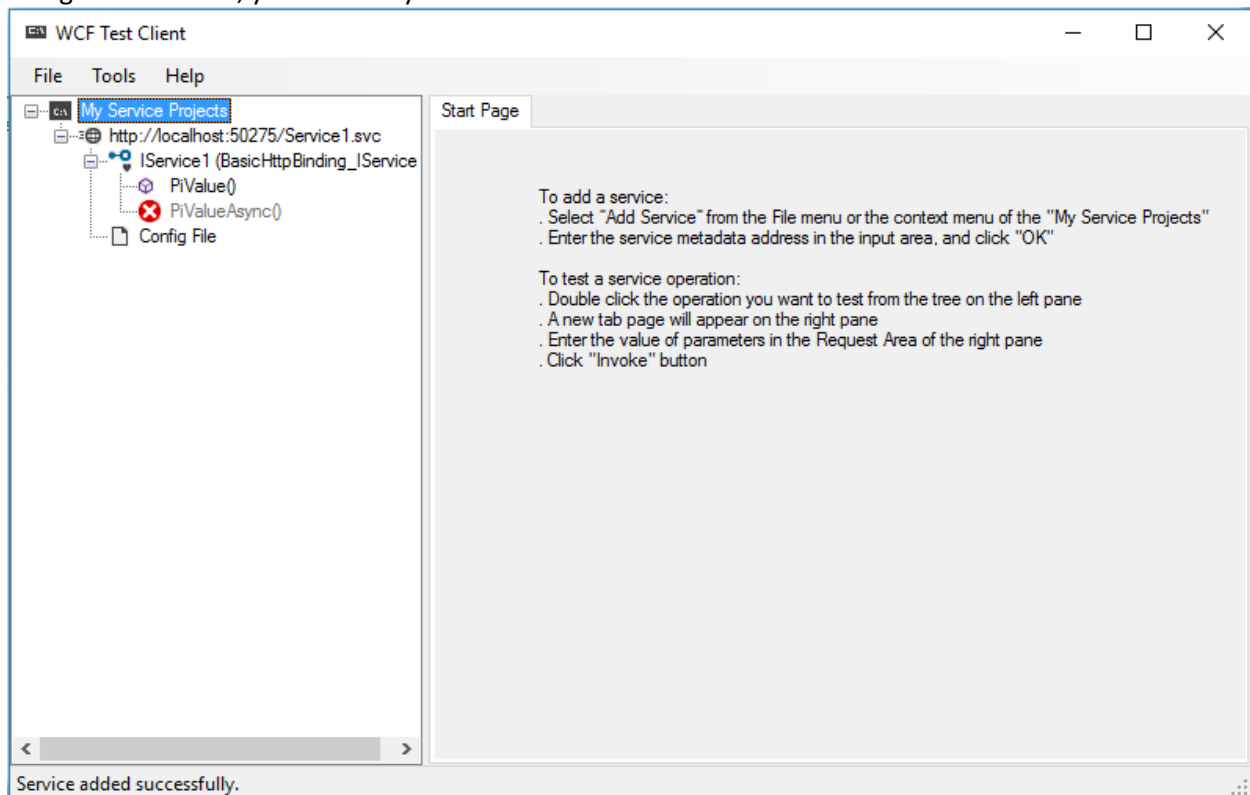
Original code:

```
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.

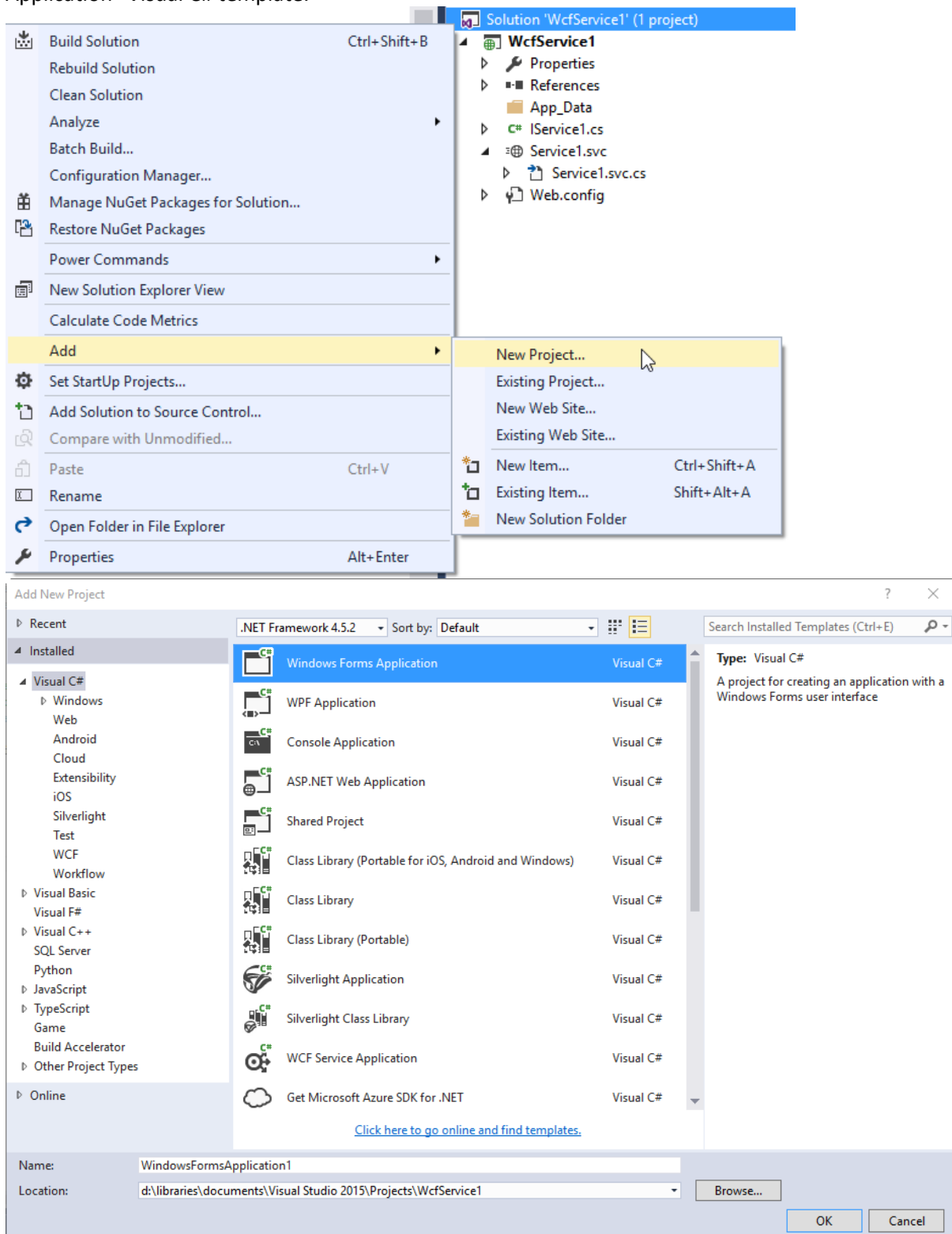
```
1  using System;
2      using System.Collections.Generic;
3      using System.Linq;
4      using System.Runtime.Serialization;
5      using System.ServiceModel;
6      using System.ServiceModel.Web;
7      using System.Text;
8
9  namespace WcfService1
10 {
11     // NOTE: You can use the "Rename" 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. Using this interface, you can test your service.

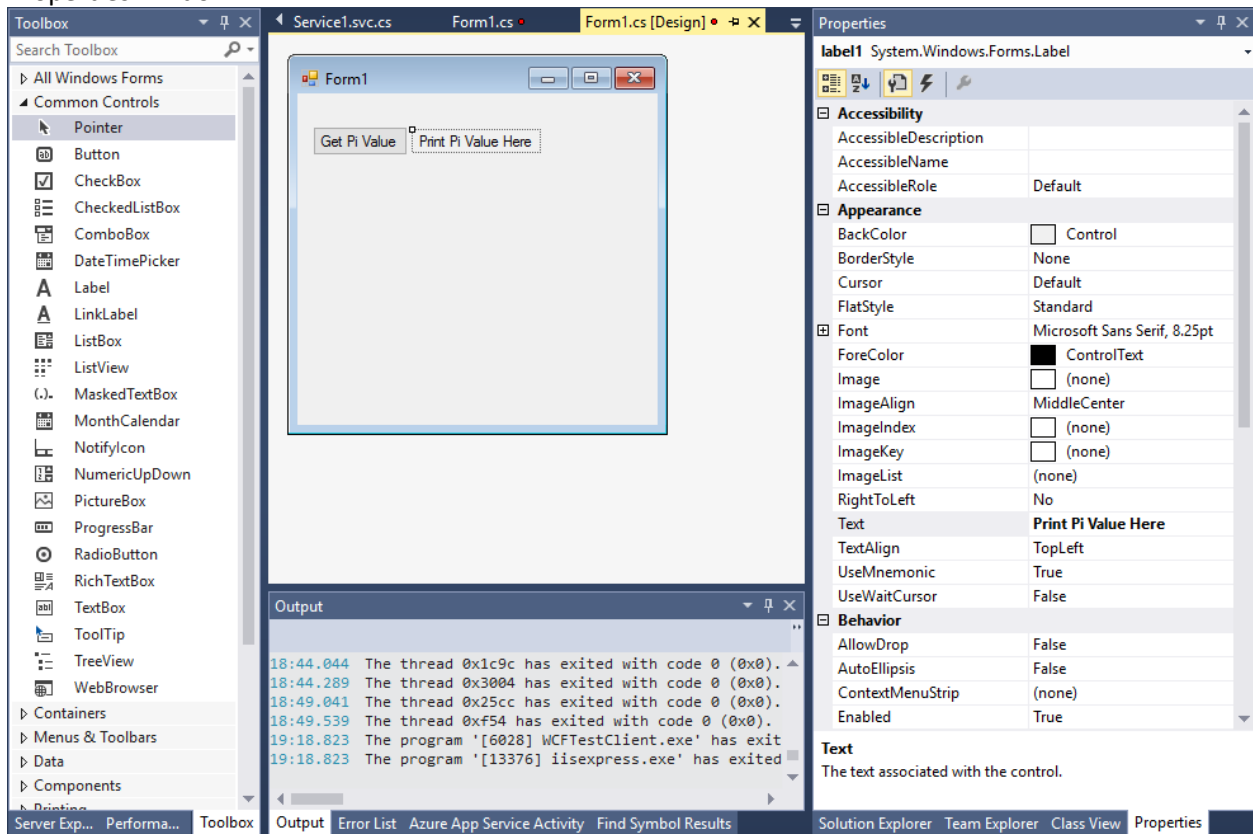


Creating a Windows Forms Application to Consumer the Service

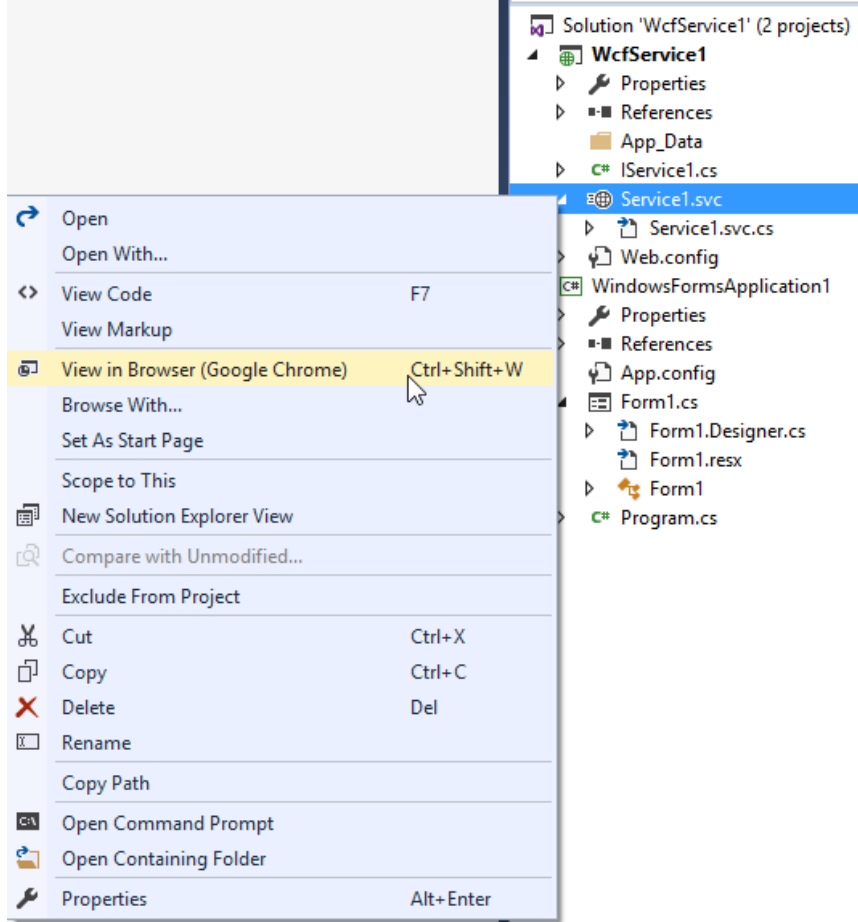
Step 1: Right click your solution, and go to Add -> New Project.... Then, select the “Windows Forms Application” Visual C# template.



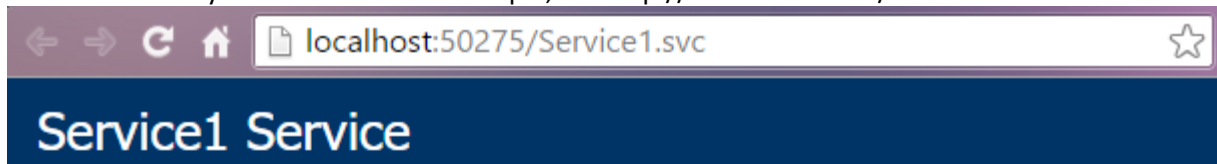
Step 2: Build your GUI using the GUI Builder by dragging components from the Toolbox to your form. You can change names and text by single clicking a component and editing the respective values in the Properties window.



Step 3: Add your service reference to your Windows Forms Application. First, right click your Service1.svc and select “View in Browser.”



Find the URL for your service. 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 project to call the Service. For example:

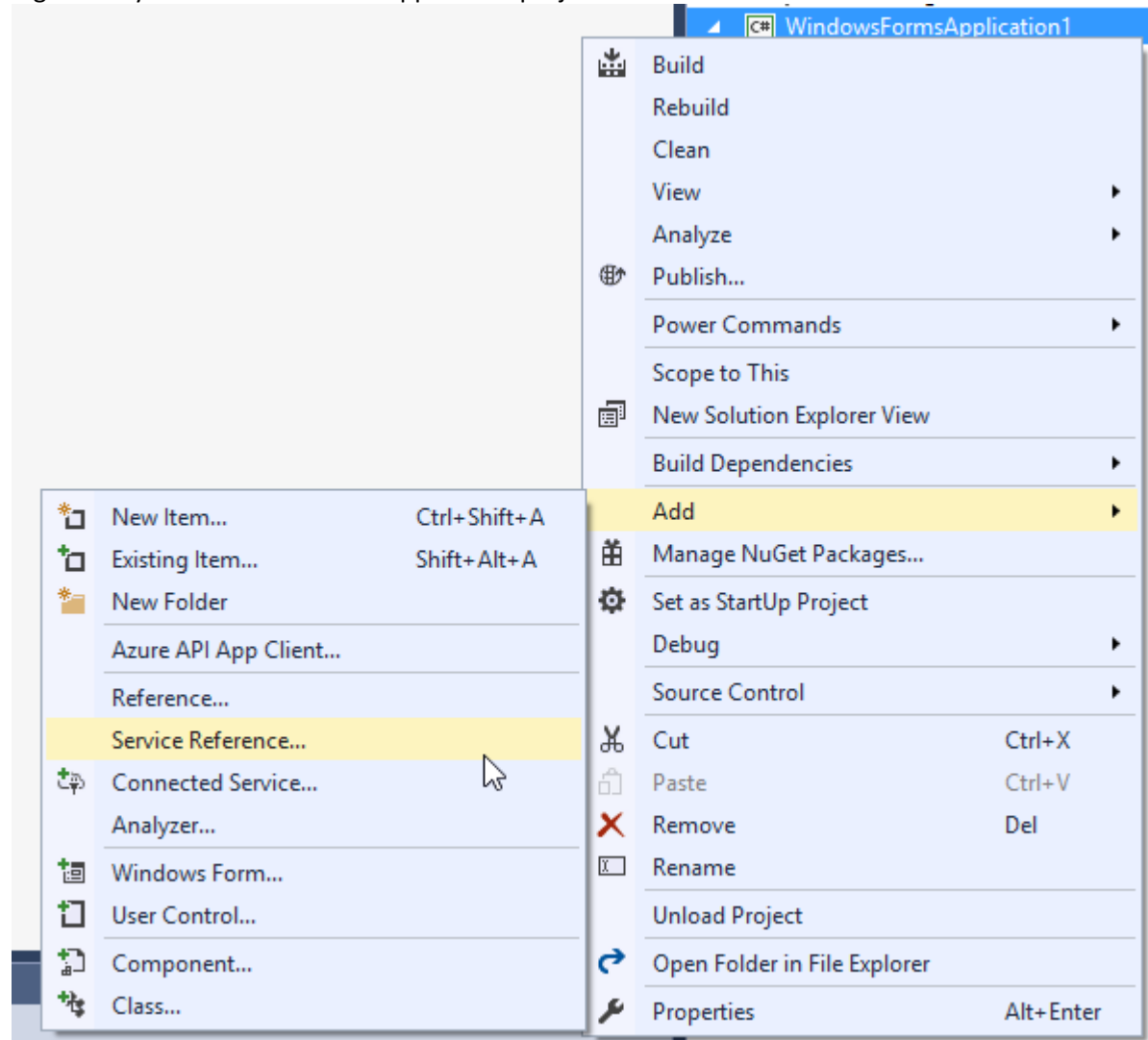
C#

```
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 Application 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:

▶  Service1

Operations:

Select a service contract to view its operations.

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

Namespace:

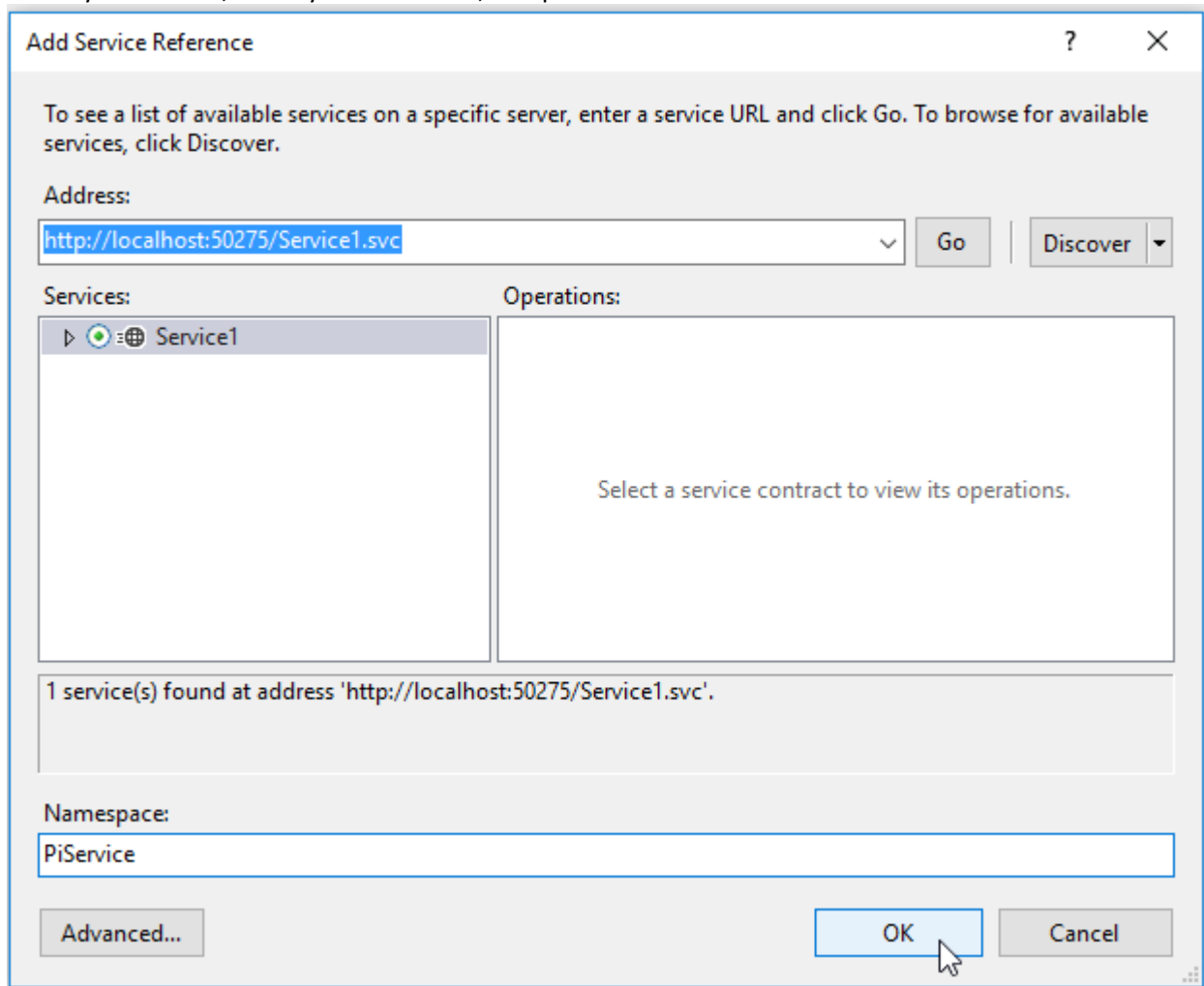
ServiceReference1

Advanced...

OK

Cancel

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 contains instructions: '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.' Below this, there is an 'Address:' label followed by a text box containing 'http://localhost:50275/Service1.svc'. To the right of the text box are 'Go' and 'Discover' buttons. Below the 'Address' section, there are two panes: 'Services:' and 'Operations:'. The 'Services:' pane contains a tree view with a single item 'Service1'. The 'Operations:' pane is empty and contains the text 'Select a service contract to view its operations.' Below these panes, a status bar indicates '1 service(s) found at address 'http://localhost:50275/Service1.svc'.' At the bottom, there is a 'Namespace:' label followed by a text box containing 'PiService'. At the very bottom, there are three buttons: 'Advanced...', 'OK', and 'Cancel'. A mouse cursor is pointing at the 'OK' button.

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:

http://localhost:50275/Service1.svc

Go Discover

Services:

Service1

Operations:

Select a service contract to view its operations.

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

Namespace:

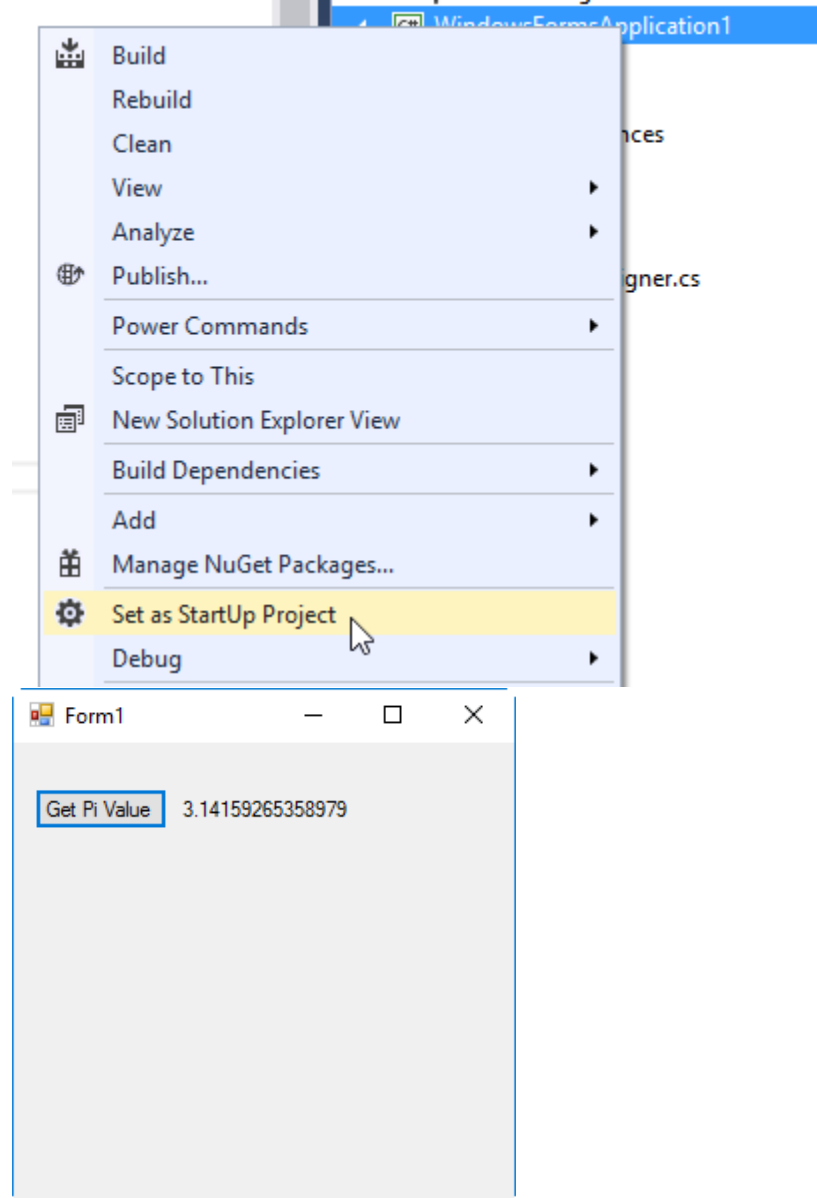
PiService

Advanced... OK Cancel

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();
}
```

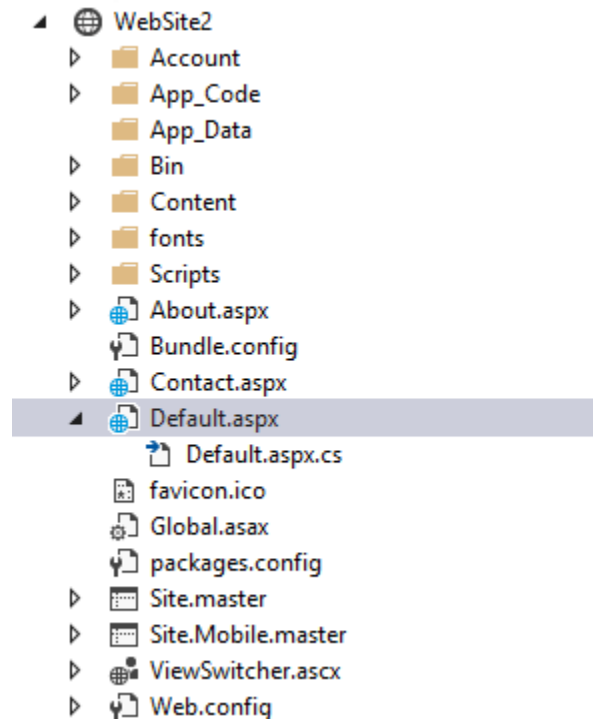
Step 5: Test your application. First, make sure your service is running, by following the above step to view your service in the browser. Then, right click your Windows Forms Application, set it as the startup project, and run your code (F5 or Debug -> Start Debugging).



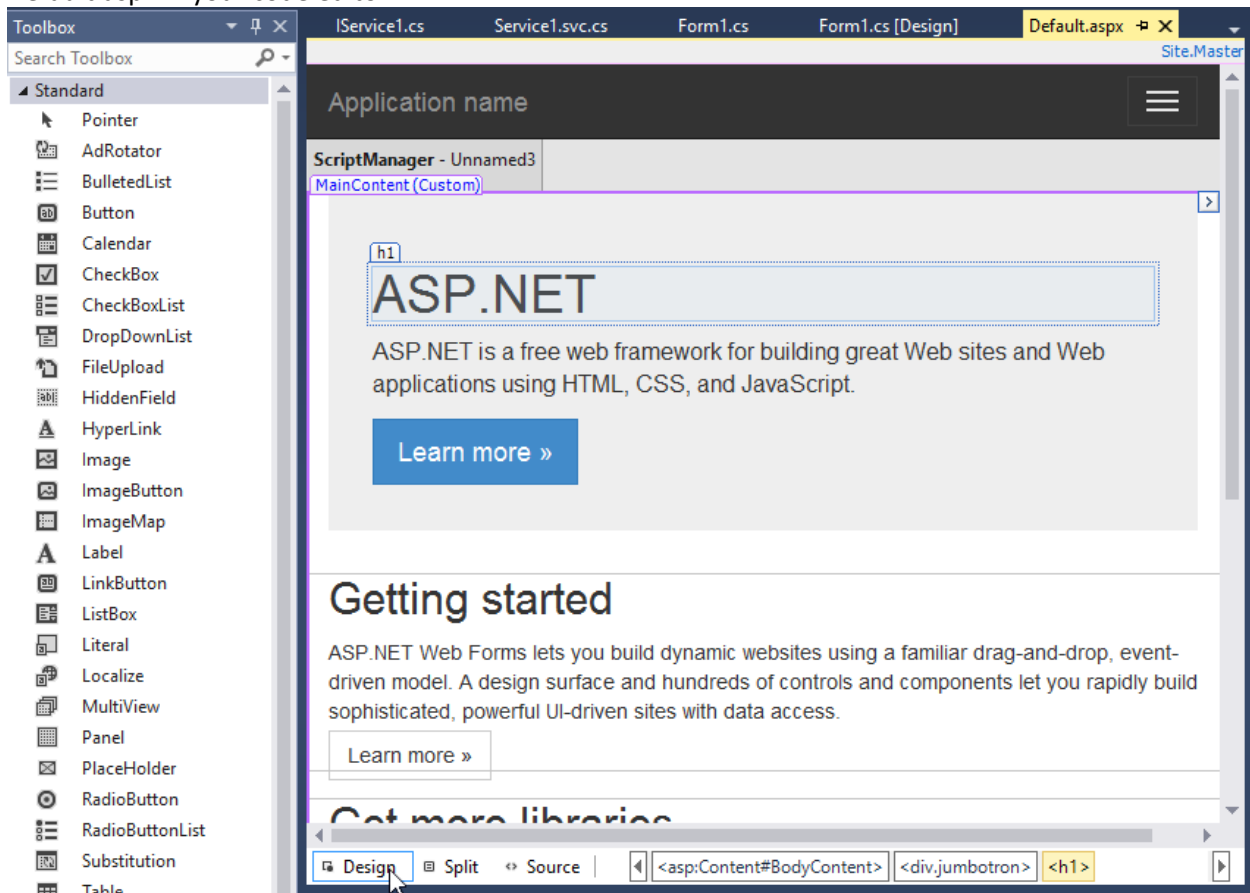
Creating a Web Site Application to Consume the Service

The steps to create, build, and run a simple Web site application are nearly identical to those of a Windows Forms Application. The first difference is in adding a new project. You have several choices. You may choose Add->New Web Site..., and select the "ASP.NET Web Forms Site" Visual C# template or the "ASP.NET Empty Web Site" template (after which you need to manually add a Web Form). You may also choose Add->New Project..., select "ASP.NET Web Application," and then 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.

This tutorial assumes you followed the first method, adding a new Web Forms Site 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:

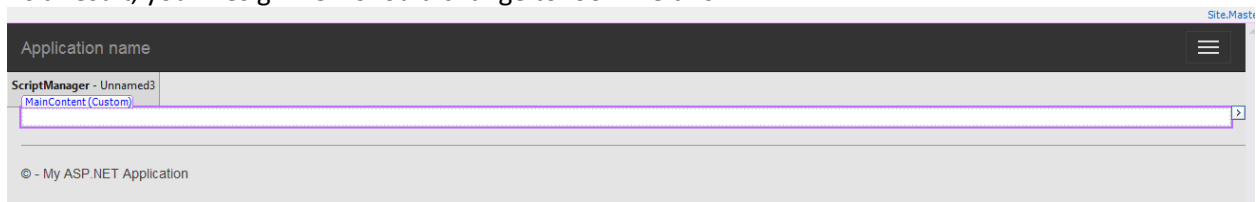
```
1 <%@ Page Title="Home Page" Language="C#" MasterPageFile="~/Site.Master" AutoEventWireup="true" CodeFile="Default.aspx.cs" Inherits="_Default" %>
2
3 <asp:Content ID="BodyContent" ContentPlaceHolderID="MainContent" runat="server">
4
5     <div class="jumbotron">
6         <h1>ASP.NET</h1>
7         <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 2015 Community Edition on Windows 10. These steps should work in most versions of Visual Studio 2012, 2013, and 2015. These steps should work on Windows 7, 8, 8.1, and 10.

Q: Does it matter how I make my Web Site Application?

A: No, there should be no difference between the above methods 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).

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

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 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);
```