Lab 1

Description

In this lab you will create your first WCF server and client applications. This lab will be broken into several sections with each new section building upon the last. You will only create one solution, but you will add a project to that solution.

Part 1 – Creating a Basic WCF Service

Start Visual Studio 2013

Open Visual Studio 2013 using administrative privileges. You can do this by right-clicking on the Visual Studio shortcut and select "Run as administrator". Failure to run with elevated privileges may cause certain features of WCF to fail. The title bar of Visual Studio should look like this:

Create a new WCF Service Application

From the menu, select File ➔ New ➔ Project.

Navigate to Templates ➔ Visual C# ➔ WCF.

Select "WCF Service Application".

Change the Name of the project to "Lab1Service".

Change the Location to whatever best suits your development environment.

Ensure that "Create directory for solution" is checked as we will be adding an additional project in later steps.

Click OK.

Test the Project

Visual Studio should have created a project called "Lab1Service" with the following default files:

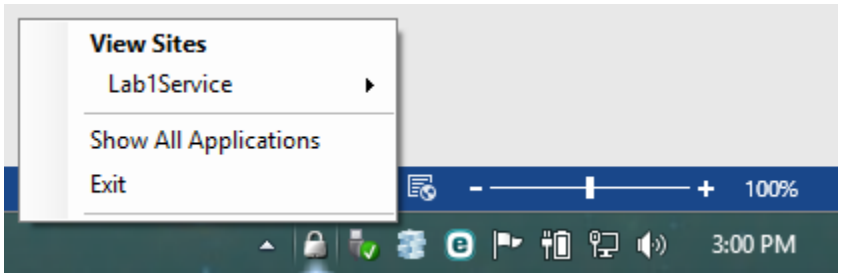
- IService1.cs
- Service1.svc
- Web.config

Make sure you have the Service1.svc file selected in Solution Explorer and select from the menu Debug ➔ Start Debugging, or simply hit F5. If you don't have Service1.svc selected, then running the project may just pop open a browser window.

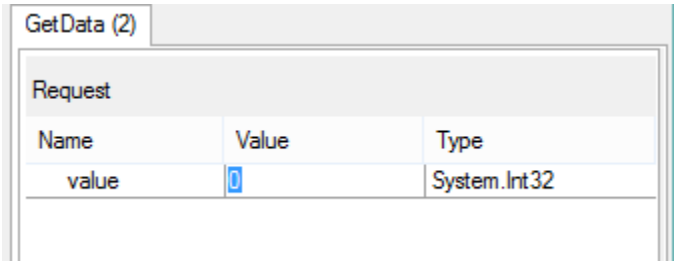
Two things should happen:

1. The "WCF Test Client" should popup showing the IService1 service running over http on a pre-defined port (in my example, port 49492, but yours will likely differ). This is a simple tester used to ensure your service can be communicated with and is not intended for production.

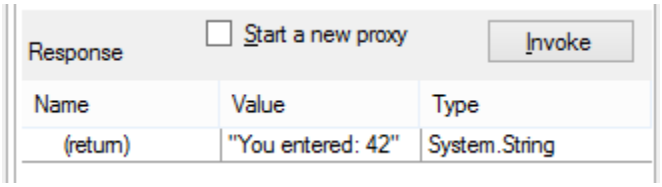
2. "IIS Express" should launch, but your only indication will likely be a new icon in the "Notification Area":



Next, in the "WCF Test Client", select the "GetData()" method under My Service Projects ➔ http://localhost:<porrt#>/Service1.svc ➔ IService1 (BasicHttpBinding_IService1). Double-click on GetData() to bring up the test client tab. Make sure "Formatted" is selected at the bottom of the "GetData" tab. Then, select the "0" under Value in the Request portion of the window.



Change the value to something else, like 42 and click "Invoke". You may get a Security Warning. If you do click OK to dismiss it. In the Response portion of the window, you should see the following:



Now, click on the "XML" tab at the bottom of the screen. There you will see the SOAP Request and Response envelopes.

You can also experiment with the GetDataUsingDataContract() method. When you are done, close the "WCF Test Client" window which will also stop the project debugging.

Removing Service1 and Creating Our Own Service

Now that we have successfully tested a simple WCF service it is time to make it more useful. IService1 is a poor name for a production service, so we are going to simply remove it and its implementing class "Service1.svc". In Solution Explorer, right-click on IService1.cs and select Delete. Confirm any popup dialog by clicking OK. Do the same for "Service1.svc". We now have a blank slate to work with. We could have renamed the files and classes but that is actually harder than adding a new service from scratch.

In the menu, select Project ➔ Add New Item… then navigate to Visual C# ➔ Web in the "Add New Item" window. In the template list, find "WCF Service" and select it. Change the name to "MathService.svc" and click "Add". Your project will now contain an IMathService interface (Service Contract) and a MathService class that

implements the interface. The interface and the class will both contain a DoWork method which we will now change.

Open the IMathService.cs file to view the interface/Service Contract. Change the DoWork method signature to look like this:

```
[OperationContract]
double Add(double value1, double value2);
```

Next, add three more operation contracts that will represent the other three common arithmetic operations: Subtract, Multiply and Divide.

Save the file and open MathService.svc.cs by double clicking on MathService.svc in Solution Explorer.

Delete the DoWork method from the MathService class.

Then, right-click on the IMathService interface name to the right of the ":" in the class definition. Select Implement Interface ➔ Implement Interface. All four methods you defined in the interface should now be present in your MathService class with a body that throws a NotImplementedException. Change the Add method to the following:

```
public double Add(double value1, double value2)
{
        return value1 + value2;
}
```

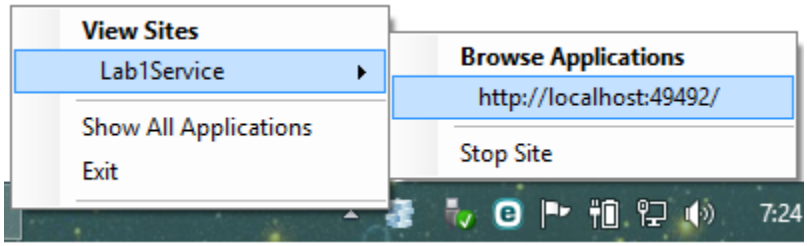
Next, fix the other three methods so they do the correct operation based on the method's name.

Finally, run the project as you did at the beginning of the lab to ensure your four methods work as expected.

When you done DO NOT stop the client! I want to show you the Metadata Exchange feature of WCF…

Viewing the Service Metadata

With the service still running, right-click on the IIS Express icon in the Notification Area. Select View Sites ➔ Lab1Service ➔ Browse Application ➔ http://localhost:XXXXX/. (XXXXX is the port number assigned by Visual Studio).



This should pop open your default browser and show you a list of files at the root directory of your web application. Select "MathService.svc" from the list. This will display the WSDL information page with links to the actual WSDL file (click one) and the instructions for creating clients in C# and VB.

When you are done poking around, close your browser and stop debugging.

Part 2 – Creating a Basic WCF Client

Add a Console Client Project to the Solution

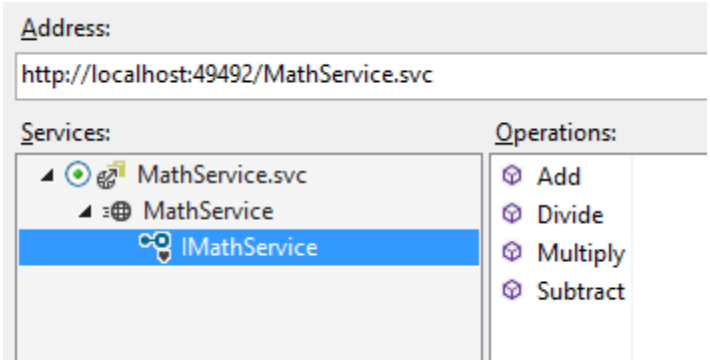
Now that we have a functioning service it is time to create a client. In the menu, select: File ➔ Add ➔ New Project…

Next, in the template tree on the left side of the "Add New Project" dialog, navigate to: Visual C# ➔ Windows Desktop, then select "Console Application" from the list. Name the project "TestClient" and click OK.

Add a Service Reference to the Console Tester

To connect the client console app to the service we need to add a service reference to the client. Right-click on the client project name in Solution Explorer and select Add ➔ Service Reference… Note: if your app is still running this choice will not be available.

In the Add Service Reference dialog, click "Discover". It should populate the Address field with something like this: "http://localhost:XXXXX/MathService.svc". In the Services tree, expand MathService.svc. If your service is not running, IIS Express should spin up and display the information seen in the WSDL file.



Change the Namespace to "MathServiceRef" and click OK. The service reference should now be visible in the project.

Calling the Service

In the console's Program.cs file add the following using directive:

```
using TestClient.MathServiceRef;
```

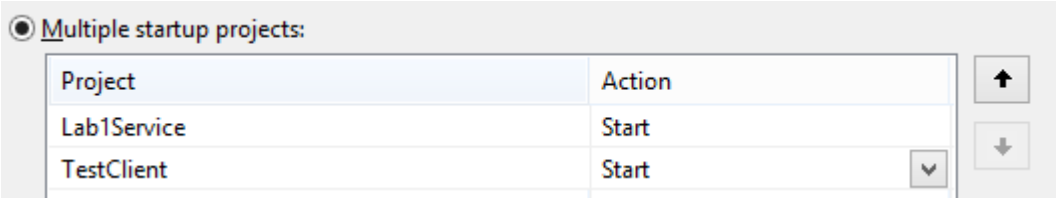
In Main(), add the following code to test the Add method:

```
MathServiceClient proxy = new MathServiceClient();
double result = proxy.Add(12.5, 2.3);
Console.WriteLine(result);
```

Configuring the Solution

Now that we have two projects in our solution we must configure the solution to run both projects in a specific order when F5 is hit. In Solution Explorer, right-click on the "Solution 'Lab1Service' (2 projects)" node and select Properties.

In the left-hand pane of the Property pages dialog, select Common Properties ➔ Startup Project. Next, check the "Multiple startup projects" radio button. Change both projects' actions to "Start" and order the projects so Lab1Service is on top. It should look like this:



Click OK.

Test the Client

Now, test the client by pressing F5. If all goes according to plan, a browser window listing the web directory should pop up along with a console window. The console window should display "14.8". If your console window shuts prematurely just add a "Console.ReadLine();" statement at the end of Main() to keep the window open until you hit Enter.

Stop debugging.

Go back to Main() and add tests for the other 3 operations.

Test again.

Part 3 – Enhancing the Service

In this last part of the lab you will work a little more independently. The requirements are as follows:

- Add a method to IMathService called "CircleArea" that accepts a double parameter called "radius" and returns a double.
- Implement this method in MathService.svc.cs. The return value should be set to the standard formula for calculating a circle's area:
  - area = $\pi \times r^2$ - Note: π (PI) is defined as Math.PI in C#.
- Update the client's service reference using the "Update Service Reference" context menu of the MathServiceRef service reference.
- Add code in the client to test that new function.