In-class Laboratory Exercise 4 (L04)

Part I – Objectives and Laboratory Materials

Objectives:

The objectives of this laboratory are to:

- Introduce the client-server computation model;
- Introduce writing a web service client; and
- Introduce the role of middleware, including SOAP and XML, in this context.

After completing the assignment, you should be able to:

- Characterize design requirements of typical client-server applications; and
- Design a simple web service client using C#.

Hardware to be used in this laboratory assignment:

- Notebook computer with IEEE 802.11b card
- iPAQ handheld computer with IEEE 802.11b card and cradle
- Intel 802.11b access point (GTA will setup one access point for the entire class)

Software to be used in this lab assignment:

- Microsoft Visual Studio .NET 2003 (VS), .NET Compact Framework (.NETcf) on the notebook computer

Overview:

Web services resemble the remote procedure call (RPC) paradigm. They are a way for a client to call methods that are located on a service host somewhere else on a network. However, the web service concept differs from the RPC concept in several ways. First, web services use standard protocols to exchange information. HTTP, SOAP, and XML are examples of such standard protocols. HTTP is used as the application-layer transport mechanism to move data between the two nodes. SOAP is used to enclose XML and to provide a context in which the XML can be interpreted. Finally, XML describes and carries the data from one node to another. This method aims to reduce development time since protocols do not have to created from the ground up, be safer since the components used are established and tested, and to be more robust since SOAP and XML can take care of describing data without the developer needing to pay attention to this detail. All of this leads to happiness, we hope, in the land of distributed processing.

That all sounds good, but why do we need web services? One reason is that mobile devices have little computational power relative to desktop devices. Why process something locally and spend five minutes wasting battery power, when you could send the data off to a larger system that can do it in five seconds (or less) without any drain on your battery? Another reason is to obtain information quickly and easily for use in an application. Web services offer a standard way to exchange information between applications. Mobile applications like stock quote systems, on-line purchasing, and inventory control often need to gather or send information to larger systems. Web services offer an open and standard way to accomplish such tasks. In this in-class laboratory session, we will work with describing and implementing web services for use in just such as scenario.

Part II – Pre-laboratory Assignment

This portion of the assignment should be completed *prior* to the in-class lab session.

Reading Assignment:

- Understand basic web service concepts by reading the Web Services tutorial posted on the class web site.
- Review the document "An XML Overview Towards Understanding SOAP" available at http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dnwebsrv/html/xmloverchap2.asp
- Optionally, if web services interest you, review the material at http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dnanchor/html/webservicesanchor.asp

Tasks:

- Think about how web services could be used with your pizza pricing application developed in Week 3. What services could be offered by adding a server back-end to your pizzeria? (You will need to document at least two such services in your submitted report.)

Part III – In-class Laboratory Assignment

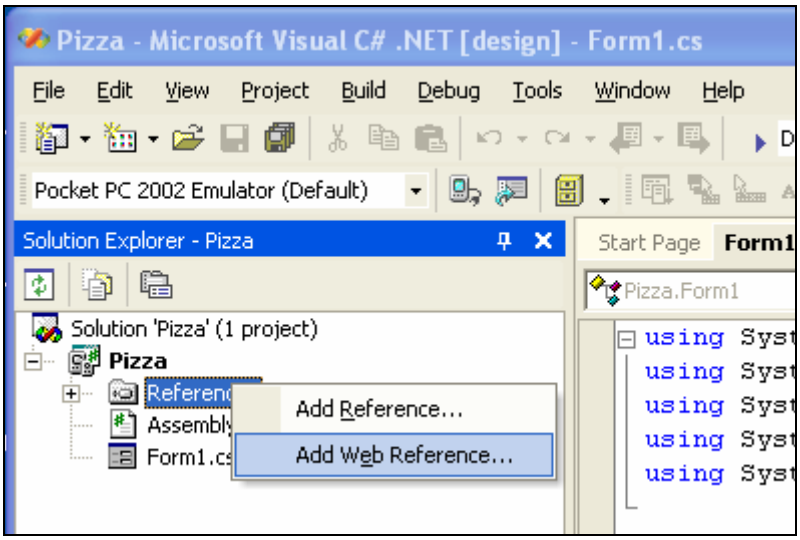
Part A: Order up!

After implementing the previous week's assignment and teaching your UVA graduate employees how to use it, you find there are still issues concerning the ordering process. Frustrated, you search for another idea. You remember hearing something about web services in your mobile systems class and decide to investigate. Alas, you find an answer. You decide to implement the ability to place an order using the iPAQ. Now the employees can simply hand the iPAQ to the table and the order can be placed when the patrons are ready.

In speaking to the instructors for your course, they agree to help you. They agree to design the server back-end of the service, but need you to design the client-end that runs on the iPAQ. Fortunately, you already have a portion of the code written from the Week 3 assignment!

Part B: The Client

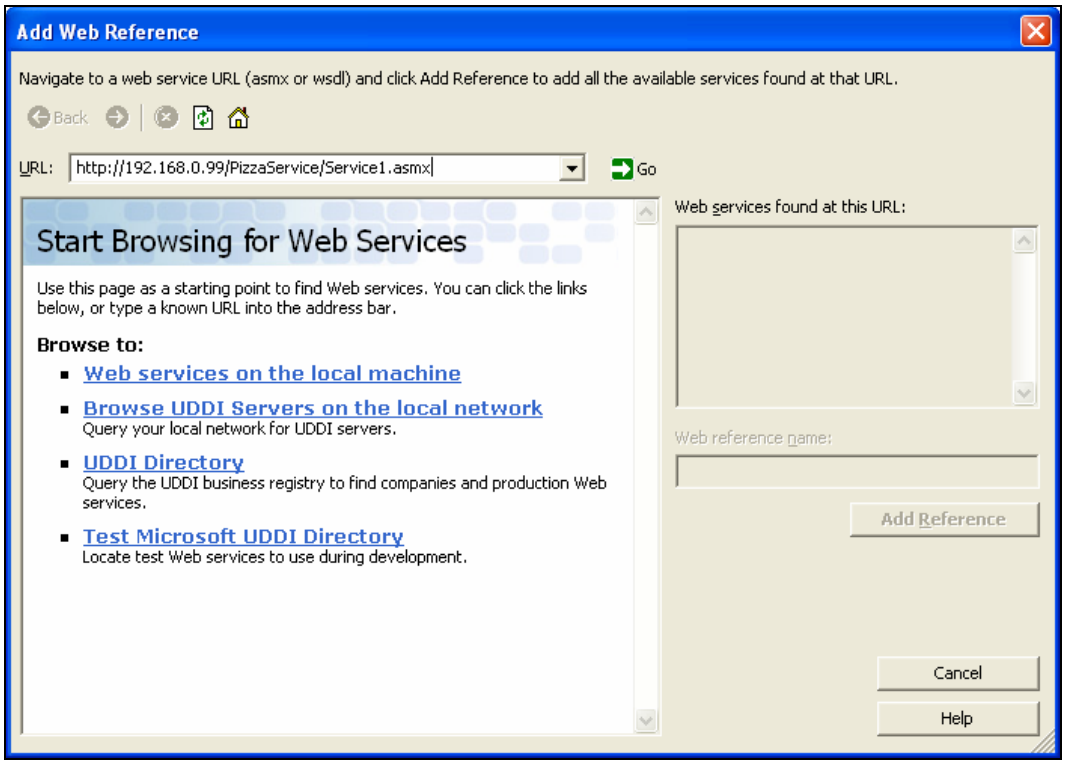
There are two pieces to a web service client that you need to make remote method calls. The first is a *web reference*. The web reference tells the application where to find the web service description; this is essentially a URL. When you create your client you have to add a web reference. This is done in the "Solution Explorer" box of Visual Studio. Right click on the *References* tab and choose *Web Reference*, as illustrated below.



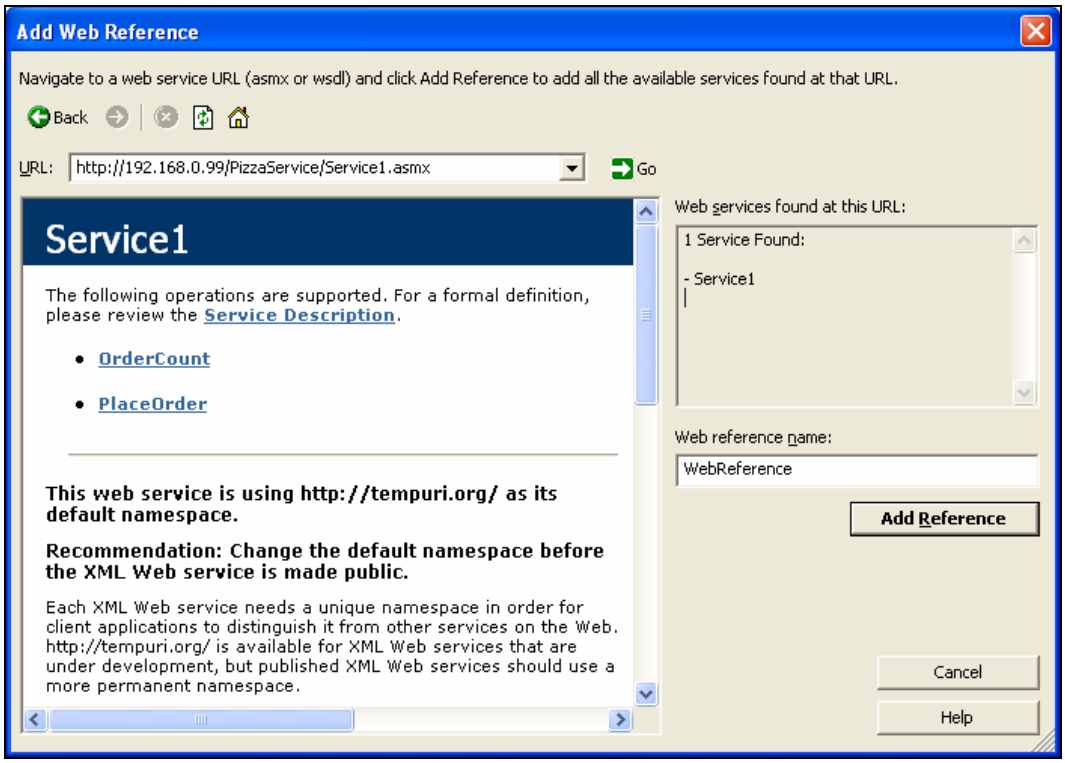
As illustrated below, you will put the URL of the web service in the URL section. The GTA will give you the URL for the web service. For this exercise, we will use the following URL as the web service URL.

http://192.168.0.99/PizzaService/Service1.asmx

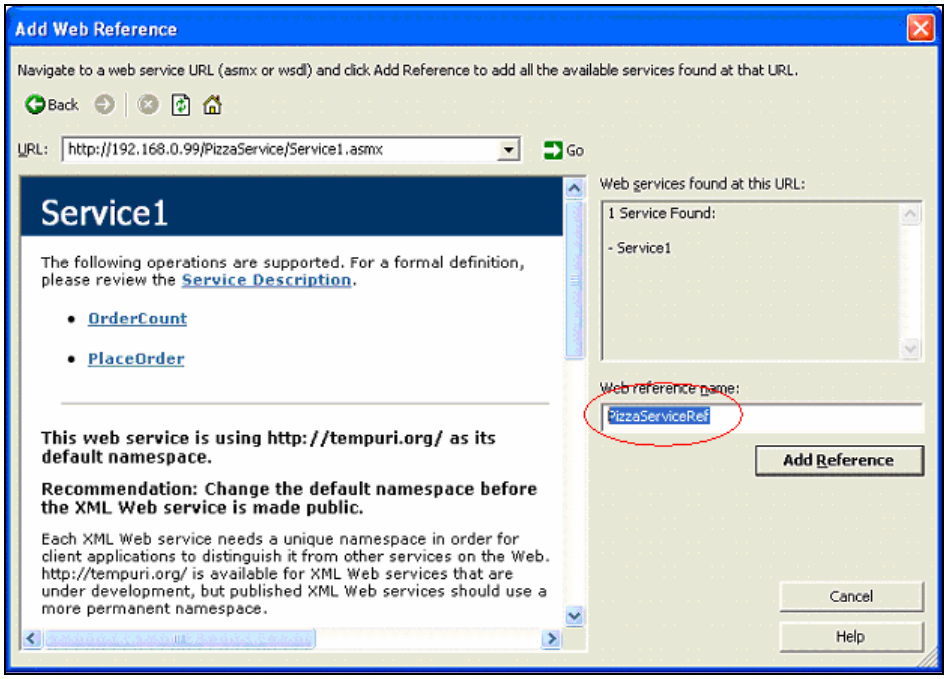
Click "Go" and VS will locate the web service and download its description file.



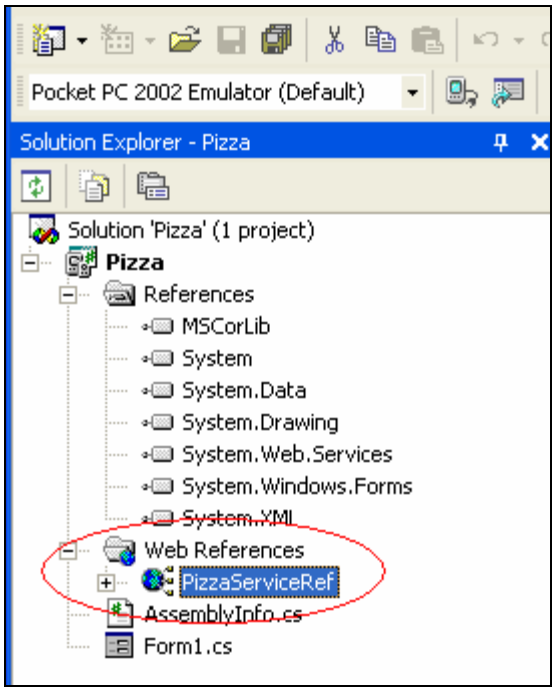
In this case, our web service has two operations available, OrderCount and PlaceOrder, as illustrated below.



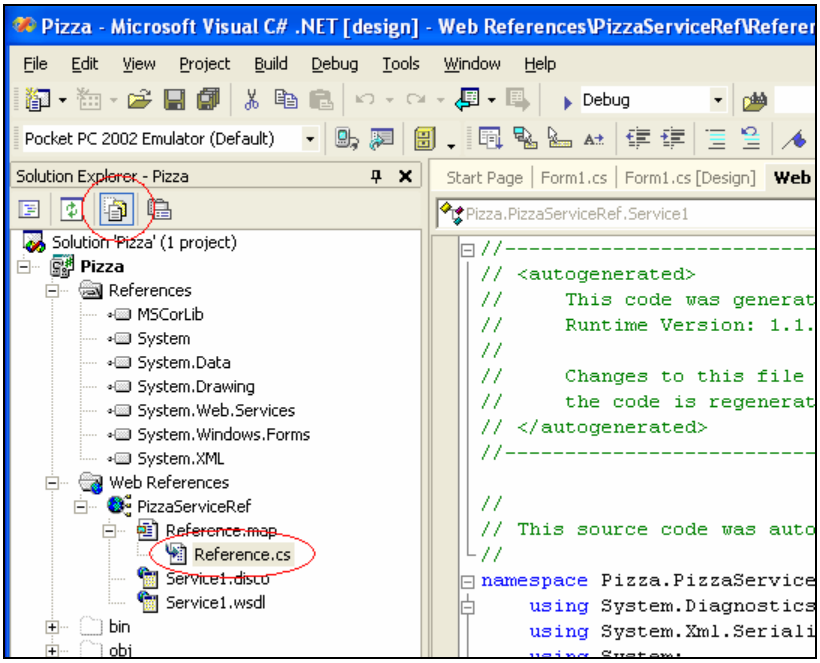
Change the name of the web reference and click "Add Reference," as shown below.



This will place a web reference into the project, as indicated below.



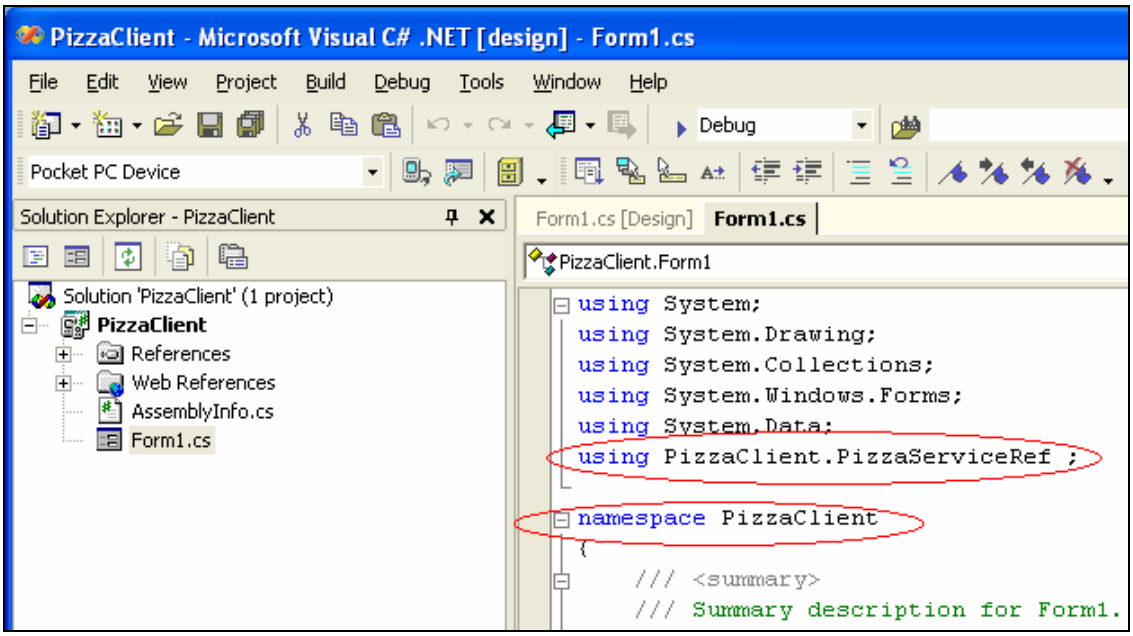
Let us go back a couple steps and talk more about web references first. For an application to make a remote call, a *proxy service* must be present. The proxy service handles taking the parameters and *serializing* them, so they may be sent to the service. The proxy service also handles extracting the returned data from the web service and passing it back to the local calling method. Visual Studio helps us out here and creates the proxy service in the background for us; thank goodness. To view this code, first, click the "Show all files" button in the solution explorer (see figure below). Open the "Reference.cs" file by double-clicking it. Scroll through this file and take a look at the automatically-generated code.



You may return to your other code now.

Not only can we make calls to remote methods, but we can use objects defined on the remote service as well. This means we do not have to define classes that we might be passing around in two places. If you think about it, this is extremely handy and safe. You may not pass object methods across nodes. This makes sense, because you might have a different hardware platform on the other end which means that your local code might not run. Object data attributes may travel freely though.

With the web reference in place, you may make calls to any *web methods* that are defined. You will learn about writing web methods in the at-home assignment associated with this in-class laboratory. The web reference is found in the object, *PizzaClient.PizzaServiceRef*. *PizzaClient* is the namespace that has been defined for your project; this can be found at the top of your client application. *PizzaServiceRef* is our web reference that we defined when we added it. This can be found in the "Solution Explorer" box in the "Web References" section. Instead of writing out the whole path, you can make use of the *using* keyword and make your life easier. See the "using" line below as an example; substituting in the appropriate class names.



For this lab, you will be using a *pizzaOrder* object as well as calling a method that are both defined on the remote service. The *pizzaOrder* class will be used to place an order using the web service. You will fill in all the information fields and complete the *PlaceOrder* ( ) method, sending it the *pizzaOrder* object as the sole parameter.

The *pizzaOrder* object is defined in the following way:

```
public class pizzaOrder
{
      public int size ;       // 0-small 1-med  2-large
      public int crust ;      // 0-thin  1-orig 2-sicilian
      public int serverID ;   // id of server
      public bool pepperoni ;
      public bool sausage ;
      public bool ham ;
      public bool peppers ;
      public bool onions ;
      public bool pineapple ;

}
```

All values are fairly self explanatory. The Booleans for toppings are set to true if the topping is to be included and false if not. The *serverID* attribute is your group number.

The *PlaceOrder* ( ) method is defined as:

```
public string PlaceOrder ( pizzaOrder order )
```

If successful, it will return the order string written to file by the service. If unsuccessful, it will return "busy…".

To call the PlaceOrder method, you must instantiate the class specified in the web reference, *Service1*. Once you have instantiated the class, you may call the remote web methods just as you would a local object method.

```
Service1 ws = new Service1 ( ) ;
// instantiate class from web reference
…
// calling the method
temp = ws.PlaceOrder ( order ) ;
```

Now that you have the tools for the job, you may begin. Design a user interface (UI) using pieces from the previous week's assignment to construct an application that allows employees to order using the iPAQ as well as displaying the price. The GTA's notebook will offer the web service of taking orders. You will need to add a button to perform the "order" function and add a small screen area for the return value to be displayed.

Part C: The Setup and Procedure

1. The GTA will have an IEEE 802.11b access point setup using WEP. The GTA will tell you the SSID and WEP key. Setup the iPAQ and the notebook computer to connect to the GTA's access point using IEEE 802.11b. Give the notebook computer an IP address of 192.168.1.*m*, where *m* is your group number plus 100. For example, group 32 would use IP address 192.168.1.132 for their notebook computer. Give the iPAQ an address of 192.168.1.*n*, where *n* is your group number plus 200. So, group 32 would use IP address 192.168.1.232 for their iPAQ. Each device should have a network mask of 255.255.255.0. Setup WEP using the key given by the GTA. Check the connections by using the notebook to ping both the GTA's notebook and your iPAQ.
2. Modify your Pizza application from the previous week's assignment to support the additional functionality specified in this assignment. **Do not overwrite the previous week's code!** You should just start this assignment using a copy of it.
3. Use the return value of the service to debug your code.
4. Use Ethereal to capture one of your order sessions. (Use filtering to make sure you are seeing just your session.) Use the "Follow TCP" stream attribute of Ethereal to view the entire session and save it to a file. The trace will be analyzed in the associated at-home assignment.