

Hands-On Lab

What’s new with WCF in .NET 4?

* 1. Lab version: 1.0.0
  2. Last updated: 2/16/2011
  3. 

# Contents

[Overview 4](#_Toc280957302)

[Setup 7](#_Toc280957303)

[Task 1 – Installing the Web Platform, Visual Studio and Windows Server AppFabric 7](#_Toc280957304)

[Exercise 1: Service without Configuration 7](#_Toc280957305)

[Task 1 – Creating the Web Site 7](#_Toc280957306)

[Task 2 – Creating the WCF Service 9](#_Toc280957307)

[Exercise 1: Verification 12](#_Toc280957308)

[Exercise 2: Service Behaviors 13](#_Toc280957309)

[Task 0 – Opening the Solution 14](#_Toc280957310)

[Task 1 – Enabling Behaviors 14](#_Toc280957311)

[Exercise 2: Verification 14](#_Toc280957312)

[Exercise 3: Default Binding 17](#_Toc280957313)

[Task 0 – Opening the Solution 17](#_Toc280957314)

[Task 1 – Changing the Default Binding 18](#_Toc280957315)

[Exercise 3: Verification 18](#_Toc280957316)

[Exercise 4: Service Discovery 19](#_Toc280957317)

[Task 0 – Opening the Solution 19](#_Toc280957318)

[Task 1 – Configuring Service Discovery for the DiscoveryChat Application 19](#_Toc280957319)

[Task 2 – Enabling ad hoc Discovery 22](#_Toc280957320)

[Exercise 4: Verification 26](#_Toc280957321)

[Exercise 5: Metadata Extensions 28](#_Toc280957322)

[Task 0 – Opening the Solution 29](#_Toc280957323)

[Task 1 – Adding an EndpointDiscoveryBehavior with Extensions 29](#_Toc280957324)

[Exercise 5: Verification 32](#_Toc280957325)

[Exercise 6: Discovery Announcements 33](#_Toc280957326)

[Task 0 – Opening the Solution 33](#_Toc280957327)

[Task 1 – Enabling Announcement Endpoint 34](#_Toc280957328)

[Exercise 6: Verification 38](#_Toc280957329)

[Exercise 7: Discovery Proxy 40](#_Toc280957330)

[Task 0 – Opening the Solution 41](#_Toc280957331)

[Task 1 – Creating a DiscoveryProxy 41](#_Toc280957332)

[Task 2 – Modifying the DiscoveryChat Application to use Managed Discovery 58](#_Toc280957333)

[Exercise 7: Verification 61](#_Toc280957334)

[Exercise 8: Protocol Bridging 63](#_Toc280957335)

[Task 0 – Opening the Solution 64](#_Toc280957336)

[Task 1 – Trying the Calculator 64](#_Toc280957337)

[Task 2 – Adding the Router Service 65](#_Toc280957338)

[Task 3 – Modifying the Client to Enable Routed HTTP 67](#_Toc280957339)

[Exercise 8: Verification 68](#_Toc280957340)

[Exercise 9: Content Based Routing 70](#_Toc280957341)

[Task 0 – Opening the Solution 71](#_Toc280957342)

[Task 1 – Modifying the CalculatorClient to Add a Custom Header 71](#_Toc280957343)

[Task 2 – Adding a New Entry to the Filter Table 72](#_Toc280957344)

[Exercise 9: Verification 74](#_Toc280957345)

[Exercise 10: Backup Lists 75](#_Toc280957346)

[Task 0 – Opening the Solution 75](#_Toc280957347)

[Task 1 – Adding a Bad Endpoint 75](#_Toc280957348)

[Task 2 – Observing the Routing Fail 76](#_Toc280957349)

[Task 3 – Enabling Backup Lists 77](#_Toc280957350)

[Exercise 10: Verification 78](#_Toc280957351)

[Exercise 11: Deploying the Web Application 78](#_Toc280957352)

[Task 0 – Opening the Solution 79](#_Toc280957353)

[Task 1 – Creating the Deployment Package 79](#_Toc280957354)

[Task 2 – Deploying the Package 80](#_Toc280957355)

[Task 3 – Changing the Client Application Configuration 82](#_Toc280957356)

[Exercise 11: Verification 83](#_Toc280957357)

[Summary 84](#_Toc280957358)

Overview

* 1. Windows Communication Foundation (WCF) in .NET 4 includes three new features areas:
  + Simplified Configuration
  + Service Discovery
  + Service Routing
  1. In this lab, you will get a chance to try out these new features and learn how they work. You will also experience the improved hosting and deployment capabilities of the new Windows Server AppFabric.
  2.  **Do I have to do all the exercises?**
  3. This lab is long. To do all of it will probably take close to 2 hours. The lab is designed so that each exercise is independent of the others. Just open the solution found under the Begin folder for the exercise. For example, if you wanted to learn how to use Service Discovery you would start with Exercise 4 and open the solution from the **Source\Ex4-ServiceDiscover\Begin** folder

# Objectives

* 1. In this Hands-On Lab, you will learn:
  + How simplifications to configuration make WCF easier to use
  + How to use Service Discovery to locate services
  + How to use Service Routing
  + How to Deploy and Monitor services using AppFabric

# System Requirements

* 1. You must have the following items to complete this lab:
  + Microsoft Visual Studio 2010
  + Microsoft .NET Framework 4
  + Windows Server AppFabric - [Download](http://msdn.microsoft.com/en-us/windowsserver/ee695849.aspx)

# Setup

* 1. All the requisites for this lab are verified using the Dependency Checker. To make sure that everything is correctly configured, follow these steps:
  2. Run the **Configuration Wizard** for the Training Kit if you have not done it previously. To do this, browse to **Source\Setup** folder of this lab, and double-click the **Dependencies.dep** file. Install any pre-requisites that are missing (rescanning if necessary) and complete the wizard.
     1. **Note:** The Configuration Wizard is used for checking dependencies and setting up the environment. If the Configuration Wizard is not installed on your machine, you must install it running the DependencyChecker.msi file located on the %VS2010TKInstallationFolder%\Assets\DependencyChecker folder (e.g. by default the Training Kit is installed under C:\VS2010TrainingKit).
     2. For convenience, much of the code you will be managing along this hands-on lab is available as Visual Studio code snippets that are automatically installed by the DependencyChecker

# Exercises

* 1. This Hands-On Lab comprises the following exercises.
  2. **Simplified Configuration**
  3. Simplified Configuration
  4. Service Behavior
  5. Protocol Mapping
  6. **Service Discovery**
  7. Service Discovery
  8. Metadata Extensions
  9. Discovery Announcements
  10. Discovery Proxy
  11. **Routing**
  12. Service Routing
  13. Content Based Routing
  14. Backup Lists
  15. **AppFabric**
  16. Deploying the Web Application

# Starting Materials

* 1. This Hands-On Lab includes the following starting materials.
  + **Visual Studio solutions.** Depending on the exercise you will find Visual Studio solutions for C# and Visual Basic that you can use as starting point for the exercises.
  1.  **What if I get stuck?**
  2. The source code that accompanies this hands-on lab includes an end folder where you can find a Visual Studio solution with the code that you would obtain if you complete the steps in each exercise. You can use this solution as a guide if you need additional help working through the exercises.

# Next Step

* 1. Setup

Setup

Task 1 – Installing the Web Platform, Visual Studio and Windows Server AppFabric

* 1. Download the [Microsoft Web Platform installer](http://www.microsoft.com/web/)
  2. Install the recommended Web Platform options plus the following
     1. ASP.NET
     2. .NET Extensibility
     3. Web Deployment Tool 1.1
  3. Download and Install Windows Server AppFabric
     1. Check only the Worker Feature and Accept all the defaults

# Next Step

* 1. Exercise 1: Service without Configuration

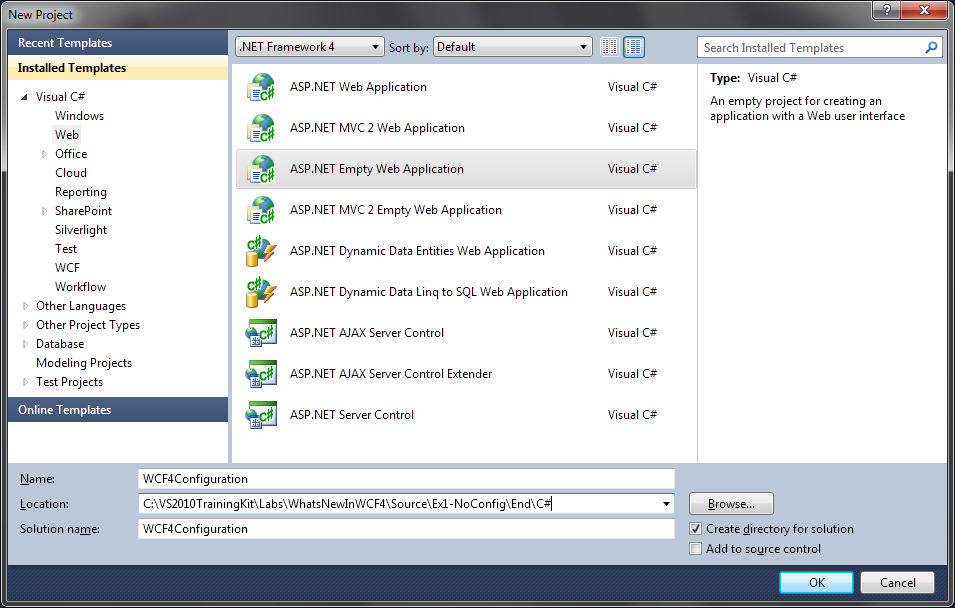
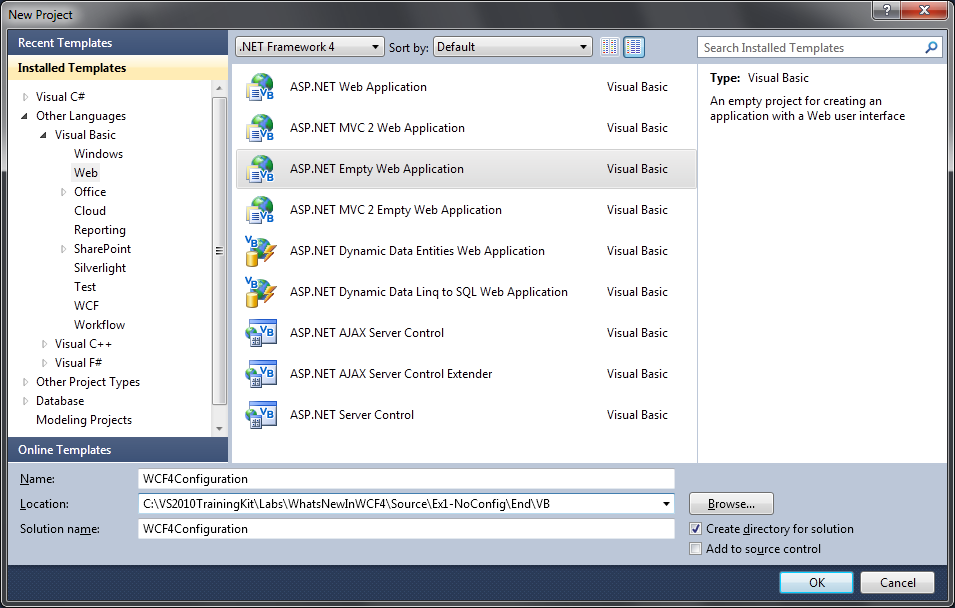
Exercise 1: Service without Configuration

* 1. In WCF 4 you can create a service without any configuration at all. In this exercise, you will create a working WCF service with no configuration.

Task 1 – Creating the Web Site

* 1. Start **Microsoft Visual Studio 2010** from **Start | All Programs | Microsoft Visual Studio 2010**.
  2. Create a new Empty ASP.NET Web Application.

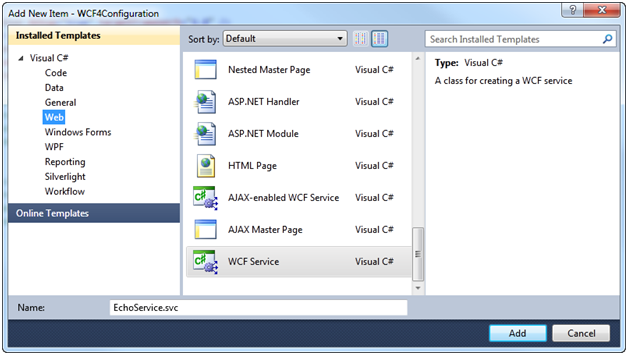
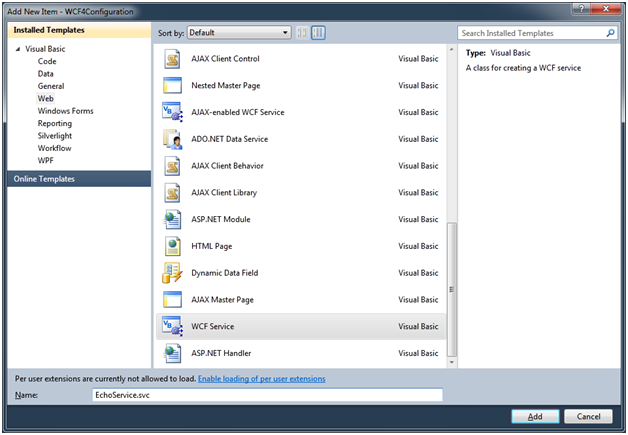
|  |  |
| --- | --- |
| Setting | Value |
| Language | **Visual C# or Visual Basic** |
| Target Framework | **.NET Framework 4** |
| Installed Templates | **Web** |
| Template | **ASP.NET Empty Web Application** |
| Name | **WCF4Configuration** |
| Location | **Source\Ex1-NoConfig\Begin** |
| Solution Name | **WCF4Configuration** |

* + 1. 
    2. Figure 1
    3. Create a new Empty ASP.NET Web Application – C#
    4. 
    5. Figure 2
    6. Create a new Empty ASP.NET Web Application – Visual Basic

Task 2 – Creating the WCF Service

* 1. Right-click the **WCF4Configuration** web site, point to **Add** and select **New** **Item**.

|  |  |
| --- | --- |
| Setting | Value |
| Language | **Visual C# or Visual Basic** |
| Installed Templates | **Web** |
| Template | **WCF Service** |
| Name | **EchoService.svc** |

* + 1. 
    2. Figure 3
    3. Create a new service named EchoService.svc – C#
    4. 
    5. Figure 4
    6. Create a new service named EchoService.svc – Visual Basic
  1. Open **IEchoService** and modify it as shown in the following code.
     1. *(Code Snippet - What is new in WCF4 Lab - IEchoService Interface CSharp)*
     2. C#
     3. **[ServiceContract]**
     4. **public interface IEchoService**
     5. **{**
     6. **[OperationContract]**
     7. **string Echo(string message);**
     8. **}**
     9. *(Code Snippet - What is new in WCF4 Lab - IEchoService Interface VB)*
     10. Visual Basic
     11. **<ServiceContract()>**
     12. **Public Interface IEchoService**
     13. **<OperationContract()>**
     14. **Function Echo(ByVal message As String) As String**
     15. **End Interface**
  2. Right-click **EchoService.svc** file, and select View Code.
  3. Add the following namespace
     1. C#
     2. **using System.Globalization;**
     3. Visual Basic
     4. **Imports System.Globalization**
  4. Delete the **DoWork** method and modify **EchoService** to implement the Echo operation as shown.
     1. *(Code Snippet - What is new in WCF4 Lab - Echo method CSharp)*
     2. C#
     3. **public string Echo(string message)**
     4. **{**
     5. **if (string.IsNullOrEmpty(message))**
     6. **throw new ArgumentNullException("message");**
     7. **return string.Format(CultureInfo.InvariantCulture, "Echo: {0}", message);**
     8. **}**
     9. *(Code Snippet - What is new in WCF4 Lab - Echo method VB)*
     10. Visual Basic
     11. **Public Function Echo(ByVal message As String) As String Implements IEchoService.Echo**
     12. **If (String.IsNullOrEmpty(message)) Then**
     13. **Throw New ArgumentNullException("message")**
     14. **End If**
     15. **Return String.Format(CultureInfo.InvariantCulture, "Echo: {0}", message)**
     16. **End Function**
  5. Open Web.config and comment out the entire **<system.serviceModel>** section by selecting it and pressing **CTRL+K,CTRL+C**

# Next Step

Exercise 1: Verification

## Exercise 1: Verification

* 1. Right-click **EchoService.svc,** and select View in Browser, to open the service in a new browser instance.
  2. You should see a service with metadata publishing disabled
     1. 
     2. Figure 5
     3. The service has metadata publishing disabled
     4.  **Why is metadata publishing disabled? Is this service even working? What binding is it using?**
     5. Yes, this is a working service. Without any configuration, it is using all the default settings, which means that metadata publishing is disabled. Since the service is hosted in an ASP.NET Web Site the service has an address that begins with **http**. WCF4 maps bindings to schemes. The default binding for **http** addresses is **basicHttpBinding**.

# Next Step

* 1. Exercise 2: Service Behaviors

Exercise 2: Service Behaviors

* 1. In WCF Service Behaviors, modify the way a service works. Without any configuration your service does not have metadata (WSDL) enabled so you cannot test it or add a service reference to it.
  2. Previously in WCF3, when you wanted to apply service behavior, you had to create a named serviceBehavior and apply the behaviorConfiguration to a named service.
  3. In WCF4, when you create an unnamed service behavior it applies to all unnamed services. In this exercise, you will enable some service behaviors.

Task 0 – Opening the Solution

* 1. To begin this exercise you can use the solution you finished from Exercise 1. Alternatively, you can follow the following steps to begin with Exercise 2.
  2. Open the starting solution for Exercise 2 located under the **Source\Ex2-ServiceBehavior\Begin (choosing the folder that matches the language of your preference.)** Use it as the starting point for this exercise.
  3. Press **CTRL+SHIFT+B** to build the solution.

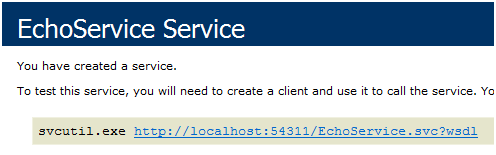
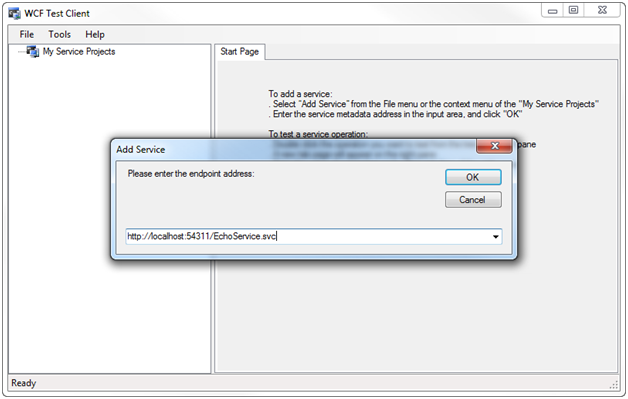
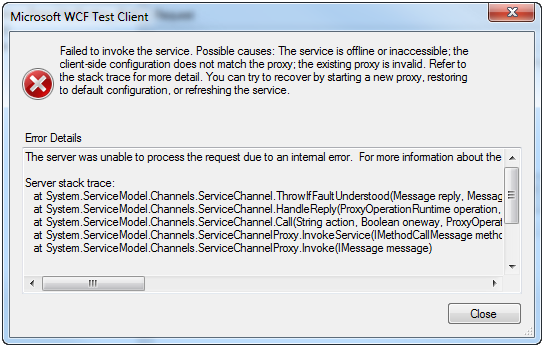
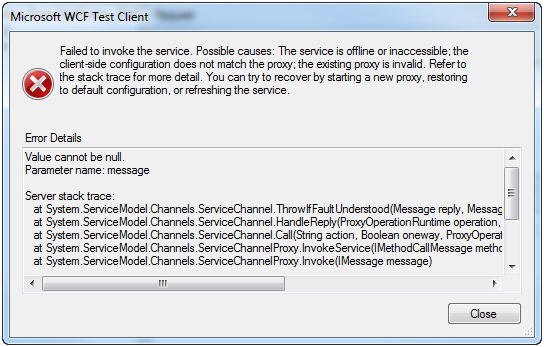
Task 1 – Enabling Behaviors

* 1. Open **Web.config**.
  2. Uncomment the **<system.serviceModel>** section. Your service will now have the serviceMetadata behavior and serviceDebug behavior enabled
     1.  **What services will this behavior apply to?**
     2. Services without named configuration get the default behaviors declared in Web.config. If you want a service to have different behaviors add a <service> tag and define a named behavior.

# Next Step

* 1. Exercise 2: Verification

## Exercise 2: Verification

* 1. Right-click **EchoService.svc**, and select View in Browser, to open the service in a new browser instance.
  2. You should now see a service page with a link to the WSDL because of the enabled **serviceMetaData** behavior.
     1. 
     2. Figure 6
     3. EchoService now has metadata enabled
  3. Copy the URL of the service from the browser address bar.
  4. Start the WCF Test Client. To do this, open a Visual Studio Command Prompt (2010), type the following command, and press Enter.
     1. Visual Studio Command Prompt (2010)
     2. **C:\>WCFTestClient**
  5. In the WCF Test Client, select **File / Add Service**, paste the URL from the browser into the endpoint address and click **OK**.
     1. 
     2. Figure 7
     3. Add the service to the WCF Test Client
  6. After the service has been loaded, double-click the **Echo()** method, set the value of the message parameter to *Test*, and click **Invoke**. The service will echo back **“Echo: Test”**.
  7. Invoke the service again – clear the value **Test**, and click **Invoke** with an empty value.
     1. 
     2. Figure 8
     3. The server reports an internal error
  8. When debugging your service it can be helpful to see the exception details. Switch back to Visual Studio 2010, open the Web.config file and change the serviceDebug behavior to set the **includeExceptionDetailInFaults** attribute to **true**.
     1. Web.config
     2. **<serviceDebug includeExceptionDetailInFaults="true" />**
  9. Save **Web.config** this will cause the app domain to recycle.
  10. Switch back to the WCF Test Client.
  11. Invoke the service again with an empty value – check the **Start a new proxy** checkbox, and click **Invoke**.
  12. This time you will see the exception message in the WCF Test Client because the serviceDebug behavior.
      1. 
      2. Figure 9
      3. The serviceDebug behavior including exception details

# Next Step

* 1. Exercise 3: Default Binding

Exercise 3: Default Binding

* 1. In the previous exercise, you were able to create and deployed a WCF Workflow Service Application that uses a default service configuration. A default endpoint for HTTP using ***basicHttpBinding*** was added to the service when this one was opened. In this exercise, you will change that default endpoint to use the *wsHttpBinding*.

Task 0 – Opening the Solution

* 1. To begin this exercise, you can use the solution you finished from Exercise 2. Alternatively, you can follow the following steps to begin with Exercise 3.
  2. Open the starting solution for Exercise 3 located under the **Source\Ex3-ProtocolMapping\Begin (choosing the folder that matches the language of your preference.)** Use it as the starting point for this exercise.
  3. Press **CTRL+SHIFT+B** to build the solution.

Task 1 – Changing the Default Binding

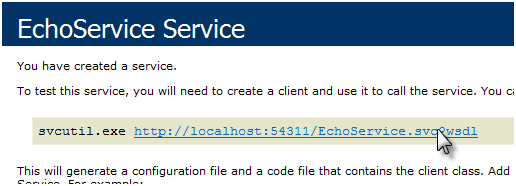
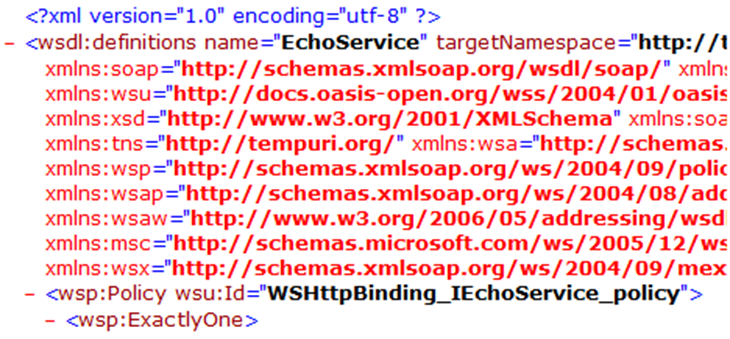
* 1. In this task, you will modify the protocol mapping for the default service configuration to use wsHttpBinding for addresses that use the http scheme.
  2. Open **Web.config**.
  3. Add the **protocolMapping** section inside the **system.serviceModel** node.
     1. *(Code Snippet - What is new in WCF4 Lab – protocolMapping XML)*
     2. Web.config
     3. <system.serviceModel>
     4. **<protocolMapping>**
     5. **<add scheme="http" binding="wsHttpBinding" />**

**</protocolMapping>**

# Next Step

* 1. Exercise 3: Verification

## Exercise 3: Verification

* 1. Right-click **EchoService.svc**, and select View in Browser, to open the service in a new browser instance.
  2. Click the link for the WSDL of the service.
     1. 
     2. Figure 10
     3. Click on link for the WSDL of the service
  3. You should see WSDL similar to the following.
     1. 
     2. Figure 11
     3. WSDL based on the wsHttpBinding

# Next Step

* 1. Exercise 4: Service Discovery

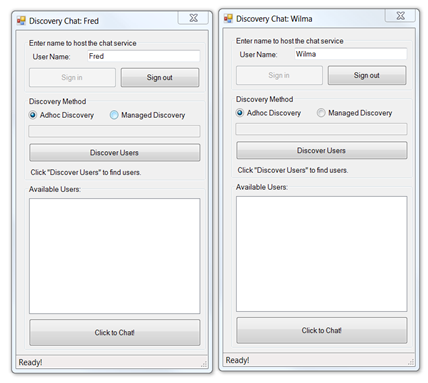
Exercise 4: Service Discovery

* 1. In this exercise, you will add an ad hoc discovery mechanism to the DiscoveryChat program using **System.ServiceModel.Discovery**, an implementation of the WS-Discovery protocol. In order for a service to be discoverable in an ad hoc manner, it needs to respond to incoming probe messages. Ad hoc discovery implies that these probe messages come in through a well-known port over UDP multicast.

Task 0 – Opening the Solution

* 1. This exercise uses a new begin solution.
  2. Open the starting solution for Exercise 4 located under the **Source\Ex4-ServiceDiscovery\Begin (choosing the folder that matches the language of your preference.)** Use it as the starting point for this exercise.
  3. Press **CTRL+SHIFT+B** to build the solution.

Task 1 – Configuring Service Discovery for the DiscoveryChat Application

* 1. **DiscoveryChat** is a chat application that automatically discovers users on the network using ad hoc or managed discovery via a proxy.
  2. 
  3. Figure 12
  4. Two simple chat windows with no discovery enabled
  5. The first thing you need to do is to enable discovery in the simple chat application.
  6. To enable discovery, you need to add a service behavior on the service that will be discoverable. Open the **App.config** configuration file from the **DiscoveryChat** project.
  7. Add a new Service Behavior named **DiscoveryBehavior** inside the ***<serviceBehaviors>*** element.
     1. *(Code Snippet – What is new in WCF4 Lab – serviceDiscovery config XML)*

App.config

* + 1. <behaviors>
    2. <serviceBehaviors>
    3. **<behavior name="DiscoveryBehavior">**
    4. **<serviceDiscovery />**
    5. **</behavior>**
    6. </serviceBehaviors>
    7. </behaviors>
  1. Modify the service description by adding the **behaviorConfiguration** attribute referencing the **DiscoveryBehavior** just created.
     1. App.config
     2. <services>
     3. <service name="Microsoft.Samples.Discovery.ChatService"
     4. **behaviorConfiguration="DiscoveryBehavior">**
     5. <endpoint
     6. address=""
     7. binding="wsHttpBinding"
     8. contract="ISimpleChatService"/>
     9. </service>
     10. </services>
  2. Next, you need to add a UDP discovery endpoint. This is where the discovery probe messages will be processed.
     1.  **Probe Messages**
     2. A Probe message is a WS-Discovery message used by a client to search for services on the network by service type. For more information about Probe messages, see section 5.2 of the [WS-Discovery Specification](http://go.microsoft.com/fwlink/?LinkId=87841).
     3. *(Code Snippet - What is new in WCF4 Lab – udpDiscoveryEndpoint XML)*
     4. App.config
     5. <services>
     6. <service name="Microsoft.Samples.Discovery.ChatService "
     7. behaviorConfiguration="DiscoveryBehavior">
     8. <endpoint
     9. address=""
     10. binding="wsHttpBinding"
     11. contract="ISimpleChatService"/>
     12. **<endpoint**
     13. **name="udpDiscoveryEpt"**
     14. **kind="udpDiscoveryEndpoint"/>**
     15. </service>
     16. </services>
     17.  **Making Services Discoverable**

Adding the DiscoveryBehavior makes your service discoverable. The UDP endpoint (specified by the attribute **kind = “udpDiscoveryEndpoint”**) is where the discovery component will listen for discovery messages.

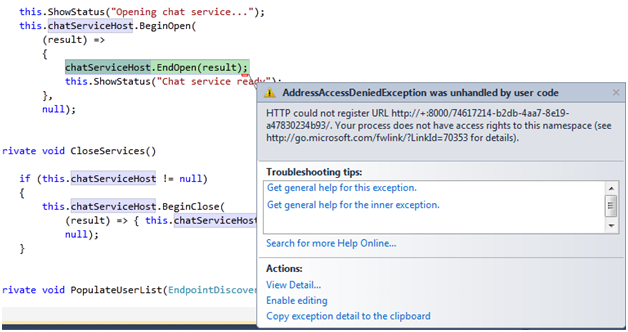
Task 2 – Enabling ad hoc Discovery

* 1. In this task, you will add code to asynchronously search for other chat users on the same subnet using service discovery. In order to utilize discovery functionality, you will need to add a reference to System.ServiceModel.Discovery to your project and add the associated namespace directive for System.ServiceModel.Discovery as well.
  2. **Note**: This has already been done for you in the Begin solution for this Exercise.
  3. Open the **SimpleChat** form in code view by clicking on it, and pressing **F7**.
  4. Locate the **AdHocDiscovery** method stub and add the following code. This code will initiate discovery looking for services that implement the **ISimpleChatService** contract and then, as services are found, it will add them to the list of available services.
     1. *(Code Snippet - What is new in WCF4 Lab – AdHocDiscovery Method CSharp)*
     2. C#
     3. private void AdHocDiscovery()
     4. {
     5. **this.discoveryClient = new DiscoveryClient(new UdpDiscoveryEndpoint());**
     6. **this.discoveryClient.FindProgressChanged +=**
     7. **new EventHandler<FindProgressChangedEventArgs>(this.OnFindProgressChanged);**
     8. **this.discoveryClient.FindCompleted +=**
     9. **new EventHandler<FindCompletedEventArgs>(this.OnFindCompleted);**
     10. **// Setup the form for discovery**
     11. **this.ShowDiscoveryInProgress(true);**
     12. **// Do async discovery**
     13. **this.discoveryClient.FindAsync(new FindCriteria(typeof(ISimpleChatService)));**
     14. }
     15. *(Code Snippet - What is new in WCF4 Lab – AdHocDiscovery Method VB)*
     16. Visual Basic
     17. Private Sub AdHocDiscovery()
     18. **Me.\_discoveryClient = New DiscoveryClient(New UdpDiscoveryEndpoint())**
     19. **AddHandler \_discoveryClient.FindProgressChanged, AddressOf OnFindProgressChanged**
     20. **AddHandler \_discoveryClient.FindCompleted, AddressOf OnFindCompleted**
     21. **' Setup the form for discovery**
     22. **Me.ShowDiscoveryInProgress(True)**
     23. **' Do async discovery**
     24. **Me.\_discoveryClient.FindAsync(New FindCriteria(GetType(ISimpleChatService)))**
     25. End Sub
     26.  **Using DiscoveryClient**
     27. The discovery process can be started either synchronous or asynchronously by using the **Find** or **FindAsync** methods respectively. Furthermore, you can also add a DynamicEndpoint to your application that searches performs discovery under the covers. In this application, you are using the **FindAsync** method because the asynchronous operation allows the chat peers to be added as they are discovered.
     28. During discovery, you can register some events handlers. For example, **FindProgressChanged** is invoked each time one of service endpoints has been discovered. If you want to be notified when the discovery process is finished, then the handler **FindCompleted** should be used.
  5. Implement the **OnFindProgressChanged** and **OnFindCompleted** handlers as shown in the following code by pasting it below the **AdHocDiscovery()** method implementation.

*(Code Snippet - What is new in WCF4 Lab – DiscoveryClient Event Handlers CSharp)*

* + 1. C#
    2. **private void OnFindProgressChanged(object sender, FindProgressChangedEventArgs e)**
    3. **{**
    4. **this.PopulateUserList(e.EndpointDiscoveryMetadata);**
    5. **}**
    6. **private void OnFindCompleted(object sender, FindCompletedEventArgs e)**
    7. **{**
    8. **if (e.Cancelled)**
    9. **{**
    10. **this.ShowStatus("Discovery cancelled");**
    11. **}**
    12. **else if (e.Error != null)**
    13. **{**
    14. **this.discoveryClient.Close();**
    15. **MessageBox.Show(**
    16. **e.Error.Message,**
    17. **this.Text,**
    18. **MessageBoxButtons.OK,**
    19. **MessageBoxIcon.Information,**
    20. **MessageBoxDefaultButton.Button1,**
    21. **(MessageBoxOptions)0);**
    22. **}**
    23. **else**
    24. **{**
    25. **if (this.discoveryClient.InnerChannel.State ==   
         CommunicationState.Opened)**
    26. **{**
    27. **this.discoveryClient.Close();**
    28. **}**
    29. **}**
    30. **this.discoveryClient = null;**
    31. **this.ShowDiscoveryInProgress(false);**
    32. **}**

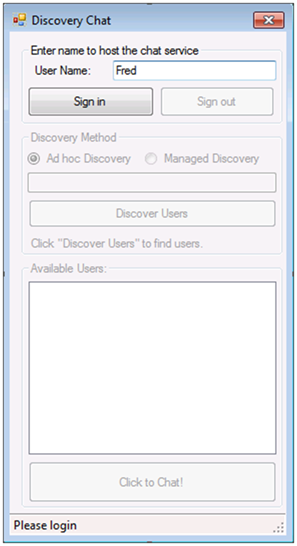
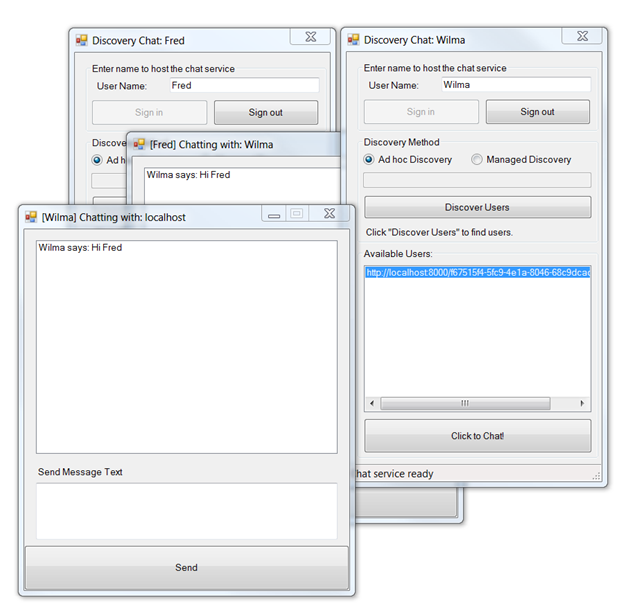
*(Code Snippet - What is new in WCF4 Lab – DiscoveryClient Event Handlers VB)*

* + 1. Visual Basic
    2. **Private Sub OnFindProgressChanged(ByVal sender As Object, ByVal e As FindProgressChangedEventArgs)**
    3. **Me.PopulateUserList(e.EndpointDiscoveryMetadata)**
    4. **End Sub**
    5. **Private Sub OnFindCompleted(ByVal sender As Object, ByVal e As FindCompletedEventArgs)**
    6. **If e.Cancelled Then**
    7. **Me.ShowStatus("Discovery cancelled")**
    8. **ElseIf e.Error IsNot Nothing Then**
    9. **Me.\_discoveryClient.Close()**
    10. **MessageBox.Show(e.Error.Message, Me.Text, MessageBoxButtons.OK, MessageBoxIcon.Information, MessageBoxDefaultButton.Button1, CType(0, MessageBoxOptions))**
    11. **Else**
    12. **If Me.\_discoveryClient.InnerChannel.State = CommunicationState.Opened Then**
    13. **Me.\_discoveryClient.Close()**
    14. **End If**
    15. **End If**
    16. **Me.\_discoveryClient = Nothing**
    17. **Me.ShowDiscoveryInProgress(False)**
    18. **End Sub**
  1. Press **CTRL+SHIFT+B** to build the solution.
     1.  **Watch Out**
     2. DiscoveryChat will open an HTTP endpoint. If you try to do this without first creating a URL ACL, you will get an AddressAccessDeniedException.
     3. 
     4. Figure 13
     5. AddressAccessDeniedException
  2. Create a URL ACL for port 8000. To do this
     1. Open a command prompt as administrator
     2. Run the SetUrlACL.cmd file from the **Source\Setup** folder as shown.
        1. CMD (Administrator)
        2. **SetURLACL.cmd 8000**
     3.  **SetURLACL.cmd**
     4. This script and the matching DelURLACL.cmd file are useful tools for managing URL ACLs.

# Next Step

* 1. Exercise 4: Verification

## Exercise 4: Verification

* 1. Press **CTRL+F5** to start an instance of the **DiscoveryChat.exe** application without debugging.
  2. Switch back to Visual Studio and press **Ctrl+F5** again to launch another instance of the **DiscoveryChat.exe** application.
  3. Switch to one of the instances of **DiscoveryChat.exe**, and setup the chat as follows:
     1. User Name: **Fred**
     2. Click **Sign In**
        1. 
        2. Figure 14
        3. Sign in as Fred
     3. **Note:** Windows Firewall may prompt you to allow the Chat client access to use the network. It is safe to allow this.
  4. Switch to the other **DiscoveryChat.exe** instance, and setup the chat as follows:
     1. Username: **Wilma**
     2. Click **Sign In**
  5. When the Wilma instance signs in, it will do ad hoc discovery immediately and locate the Fred instance.
  6. Double-click the URI in the available users list to start a chat with Fred.
     1. Send Message Text: **Hi Fred**
     2. Click **Send**
  7. When you start the chat, you will see Wilma appear in the user list on the Fred Discovery chat window.
     1. 
     2. Figure 15
     3. Two instances of DiscoveryChat application that discovered each other
  8. Close both instances of the **DiscoveryChat** application**.**

# Next Step

* 1. Exercise 5: Metadata Extensions

Exercise 5: Metadata Extensions

* 1. The chat application works OK, but you may have noticed an issue. When Wilma discovered other chat instances all you knew about them was the Uri of the service endpoint. Therefore, her chat window shows a chat with the host machine name. Fred, on the other hand, knew that the chat message came from Wilma because the chat message included her name. If you could add the name along with the discovery messages, it would be possible to display the names in the user box instead of the endpoints. In this exercise, you will learn how you can extend the metadata used in WS-Discovery to supply additional information (such as the username used for the chat session).

Task 0 – Opening the Solution

* 1. To begin this exercise you can use the solution you finished from Exercise 4. Alternatively, you can follow the following steps to begin with Exercise 5.
  2. Open the starting solution for Exercise 5 located under the **Source\Ex5-MetadataExtensions\Begin (choosing the folder that matches the language of your preference.)** Use it as the starting point for this exercise.
  3. Press **CTRL+SHIFT+B** to build the solution.

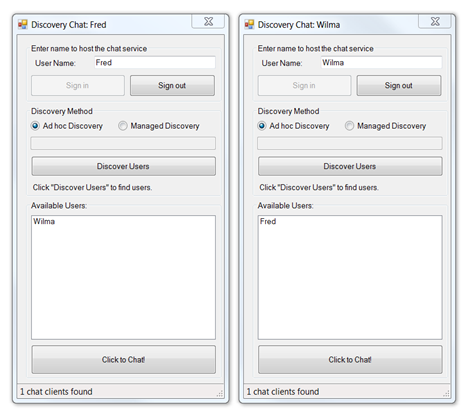
Task 1 – Adding an EndpointDiscoveryBehavior with Extensions

* 1. Open the **SimpleChat** class in code view from the **DiscoveryChat** project. You can open the code view by selecting the file, and pressing **F7**.
  2. You can add XML to the endpoint metadata when responding to a discovery probe. To do this, you will need to add namespaces directives.
     1. *(Code Snippet - What is new in WCF4 Lab – ServiceModel.Description CSharp)*
     2. C#
     3. **using System.ServiceModel.Description;**
     4. *(Code Snippet - What is new in WCF4 Lab – ServiceModel.Description VB)*
     5. Visual Basic
     6. **Imports System.ServiceModel.Description**
  3. Locate the **OpenServices** method, and modify it as shown in the following code. Here you are creating an EndpointDiscoveryBehavior instance, and as part of that behavior you are a adding an XML element, in this case the user name to the extension collection. This extension is sent along with the responses to clients looking for that endpoint. Lastly, you are applying that behavior to your service endpoint.
     1. *(Code Snippet - What is new in WCF4 Lab – OpenServices Method CSharp)*
     2. C#
     3. private void OpenServices()
     4. {
     5. // Create a singleton instance for the host
     6. ChatService chatService = new ChatService(this);
     7. chatServiceHost = new ServiceHost(chatService, this.localAddress);
     8. **// Create a discovery behavior**
     9. **EndpointDiscoveryBehavior endpointDiscoveryBehavior = new EndpointDiscoveryBehavior();**
     10. **// Add an extension element with the username**
     11. **endpointDiscoveryBehavior.Extensions.Add(**
     12. **new XElement(**
     13. **"root",**
     14. **new XElement("Name", this.userName)));**
     15. **// Find the endpoint**
     16. **ServiceEndpoint simpleEndpoint =**
     17. **this.chatServiceHost.Description.Endpoints.Find(typeof(ISimpleChatService));**
     18. **// Add your behavior to the endpoint before opening it**
     19. **simpleEndpoint.Behaviors.Add(endpointDiscoveryBehavior);**
     20. ShowStatus("Opening chat service...");
     21. chatServiceHost.BeginOpen(
     22. (result) =>
     23. {
     24. chatServiceHost.EndOpen(result);
     25. ShowStatus("Chat service ready");
     26. },
     27. null);
     28. }
     29. *(Code Snippet - What is new in WCF4 Lab – OpenServices Method VB)*
     30. Visual Basic
     31. Private Sub OpenServices()
     32. ' Create a singleton instance for the host
     33. Dim chatService As New ChatService(Me)
     34. Me.chatServiceHost = New ServiceHost(chatService, Me.\_localAddress)
     35. **' Create a discovery behavior**
     36. **Dim endpointDiscoveryBehavior = New EndpointDiscoveryBehavior()**
     37. **' Add an extension element with the username**
     38. **endpointDiscoveryBehavior.Extensions.Add(**
     39. **New XElement(**
     40. **"root",**
     41. **New XElement("Name", Me.\_userName)))**
     42. **' Find the endpoint**
     43. **Dim simpleEndpoint = Me.chatServiceHost.Description.Endpoints.Find(GetType(ISimpleChatService))**
     44. **' Add your behavior to the endpoint before opening it**
     45. **simpleEndpoint.Behaviors.Add(endpointDiscoveryBehavior)**
     46. Me.ShowStatus("Opening chat service...")
     47. Me.chatServiceHost.BeginOpen(Sub(result)
     48. chatServiceHost.EndOpen(result)
     49. Me.ShowStatus("Chat service ready")
     50. End Sub, Nothing)
     51. End Sub
  4. Now that the name is sent along with service endpoint information, you need to modify the **PopulateUserList** method to invoke the **GetPeerName** method. The **GetPeerName** method returns the first node named “Name” from the extension metadata. The **PeerUser** class will use it as the display name in the list box. To do this, replace the current call to the **AddUser** method with the following one.
     1. C#
     2. private void PopulateUserList(
     3. EndpointDiscoveryMetadata endpointDiscoveryMetadata)
     4. {
     5. if (!EndpointIsSelf(endpointDiscoveryMetadata.Address.Uri))
     6. {
     7. **this.AddUser(new PeerUser(GetPeerName(endpointDiscoveryMetadata), endpointDiscoveryMetadata.Address));**
     8. }
     9. }
     10. Visual Basic
     11. Private Sub PopulateUserList(ByVal endpointDiscoveryMetadata As EndpointDiscoveryMetadata)
     12. If Not (Me.EndpointIsSelf(endpointDiscoveryMetadata.Address.Uri)) Then
     13. **Me.AddUser(New PeerUser(GetPeerName(endpointDiscoveryMetadata), endpointDiscoveryMetadata.Address))**
     14. End If
     15. End Sub
  5. Press **CTRL+SHIFT+B** to build the solution.

# Next Step

* 1. Exercise 5: Verification

## Exercise 5: Verification

* 1. Press **CTRL+F5** to start an instance of the **DiscoveryChat** application without debugging.
  2. Switch back to Visual Studio, and press **CTRL+F5** again to launch another instance of the application**.**
  3. Switch to one of the instances and setup the chat as follows:
     1. User Name: **Fred**
     2. Click **Sign In**
     3. **Note:** Windows Firewall may prompt you to allow the Chat client access to use the network. It is safe to allow this.
  4. Switch to the other **DiscoveryChat** instance and setup the chat as follows:
     1. User Name: **Wilma**
     2. Click **Sign In**
  5. Wilma’s chat window will discover Fred’s after signing in. Switch to Fred’s chat window and click on the **Discover Users** button to find Wilma.
  6. You should see the other instance’s username appear in the **Available Users** pane. You can double-click on it to start a chat.
     1. 
     2. Figure 16
     3. Extensions allow us to send the username in discovery metadata
  7. Close both instances of the DiscoveryChatapplication.

# Next Step

* 1. Exercise 6: Discovery Announcements

Exercise 6: Discovery Announcements

1. Most chat applications notify you when other users sign in. This application can discover other users but it would be better if you were notified when other users sign in. Discovery supports this feature with announcements. Using Discovery APIs service can announce their presence on the network; this functionality is used in the chat applications to notify peers of their arrival.

Task 0 – Opening the Solution

* 1. To begin this exercise you can use the solution you finished from Exercise 5. Alternatively, you can follow the following steps to begin with Exercise 6.
  2. Open the starting solution for Exercise 6 located under the **Source\Ex6-DiscoveryAnnouncements\Begin (choosing the folder that matches the language of your preference.)** Use it as the starting point for this exercise.
  3. Press **CTRL+SHIFT+B** to build the solution.

Task 1 – Enabling Announcement Endpoint

* 1. Open the **App.config** file in the **DiscoveryChat** project.
  2. To take advantage of announcements, you will need to add an announcement endpoint. To do this, locate the **DiscoveryBehavior** behavior you added earlier in this lab and modify it as shown in the following code.
     1. *(Code Snippet - What is new in WCF4 Lab – DiscoveryBehavior udpEndpoint XML)*
     2. App.config
     3. <behavior name="DiscoveryBehavior">
     4. **<serviceDiscovery>**
     5. **<announcementEndpoints>**
     6. **<endpoint name="udpEndpointName"**
     7. **kind="udpAnnouncementEndpoint"/>**
     8. **</announcementEndpoints>**
     9. **</serviceDiscovery>**

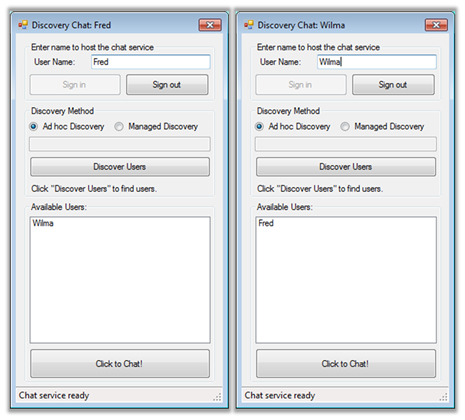
</behavior>

* + 1.  **Announcement Endpoint**
    2. Adding an announcement endpoint to the discovery service behavior creates a default announcement client for the service. This ensures that the service will send an online and offline announcement when the service is opened and closed respectively, the announcements will be sent out on the endpoint specified.
  1. Now you need to add an announcement service to your code to receive announcement messages. Open **SimpleChat.cs** (C#) or **SimpleChat.vb** (Visual Basic), and declare the following member fields; you can do that after the **discoveryClient** member declaration.
     1. *(Code Snippet - What is new in WCF4 Lab – AnnouncementService Member CSharp)*
     2. C#
     3. **private AnnouncementService announcementService;**
     4. **private ServiceHost announcementServiceHost;**
     5. *(Code Snippet - What is new in WCF4 Lab – AnnouncementService Member VB)*
     6. Visual Basic
     7. **Private announcementService As AnnouncementService**
     8. **Private announcementServiceHost As ServiceHost**
  2. Create the **OpenAnnouncementService** method as shown in the following code.
     1. *(Code Snippet - What is new in WCF4 Lab – OpenAnnouncementService Method CSharp)*
     2. C#
     3. **private void OpenAnnouncementService()**
     4. **{**
     5. **this.announcementService = new AnnouncementService();**
     6. **// Add event handlers**
     7. **this.announcementService.OnlineAnnouncementReceived +=**
     8. **new EventHandler<AnnouncementEventArgs>(this.OnOnlineAnnouncement);**
     9. **this.announcementService.OfflineAnnouncementReceived +=**
     10. **new EventHandler<AnnouncementEventArgs>(this.OnOfflineAnnouncement);**
     11. **// Create the service host with a singleton**
     12. **this.announcementServiceHost = new ServiceHost(this.announcementService);**
     13. **// Add the announcement endpoint**
     14. **this.announcementServiceHost.AddServiceEndpoint(new UdpAnnouncementEndpoint());**
     15. **// Open the host async**
     16. **this.announcementServiceHost.BeginOpen(**
     17. **(result) =>**
     18. **{**
     19. **announcementServiceHost.EndOpen(result);**
     20. **},**
     21. **null);**
     22. **}**
     23. *(Code Snippet - What is new in WCF4 Lab – OpenAnnouncementService Method VB)*
     24. Visual Basic
     25. **Private Sub OpenAnnouncementService()**
     26. **Me.announcementService = New AnnouncementService()**
     27. **' Add event handlers**
     28. **AddHandler announcementService.OnlineAnnouncementReceived, AddressOf OnOnlineAnnouncement**
     29. **AddHandler announcementService.OfflineAnnouncementReceived, AddressOf OnOfflineAnnouncement**
     30. **' Create the service host with a singleton**
     31. **Me.announcementServiceHost = New ServiceHost(Me.announcementService)**
     32. **' Add the announcement endpoint**
     33. **Me.announcementServiceHost.AddServiceEndpoint(New UdpAnnouncementEndpoint())**
     34. **' Open the host async**
     35. **Me.announcementServiceHost.BeginOpen(Sub(result) announcementServiceHost.EndOpen(result), Nothing)**
     36. **End Sub**
     37.  **Announcement Service**
     38. The self-hosted implementation of the announcement service exposes two different events you might be interested in when using Announcements: **OnlineAnnouncementReceived** and **OfflineAnnouncementReceived**. Those events are fired when Online (Hello) and Offline (Bye) announcement messages are received respectively.
  3. Now you need to implement the handlers for these announcements. When new users come online, you will add them to the list. Add the **OnOnlineAnnouncement** method as shown in the following code.
     1. *(Code Snippet - What is new in WCF4 Lab – OnOnlineAnnouncement Method CSharp)*
     2. C#
     3. **private void OnOnlineAnnouncement(object sender, AnnouncementEventArgs e)**
     4. **{**
     5. **EndpointDiscoveryMetadata metadata =**
     6. **e.EndpointDiscoveryMetadata;**
     7. **// You are looking for services that**
     8. **// implement the ISimpleChatService contract**
     9. **FindCriteria criteria =**
     10. **new FindCriteria(typeof(ISimpleChatService));**
     11. **if (criteria.IsMatch(metadata))**
     12. **{**
     13. **if (this.GetUser(metadata.Address.Uri) == null)**
     14. **{**
     15. **this.PopulateUserList(metadata);**
     16. **}**
     17. **}**
     18. **}**
     19. *(Code Snippet - What is new in WCF4 Lab – OnOnlineAnnouncement Method VB)*
     20. Visual Basic
     21. **Private Sub OnOnlineAnnouncement(ByVal sender As Object, ByVal e As AnnouncementEventArgs)**
     22. **Dim metadata As EndpointDiscoveryMetadata = e.EndpointDiscoveryMetadata**
     23. **' You are looking for services that**
     24. **' implement the ISimpleChatService contract**
     25. **Dim criteria As New FindCriteria(GetType(ISimpleChatService))**
     26. **If criteria.IsMatch(metadata) Then**
     27. **If Me.GetUser(metadata.Address.Uri) Is Nothing Then**
     28. **Me.PopulateUserList(metadata)**
     29. **End If**
     30. **End If**
     31. **End Sub**
  4. When users go offline you will remove them and close any active chat windows. Add the **OnOfflineAnnouncement** method as shown in the following code.
     1. *(Code Snippet - What is new in WCF4 Lab – OnOfflineAnnouncement Method CSharp)*
     2. C#
     3. **private void OnOfflineAnnouncement(object sender, AnnouncementEventArgs e)**
     4. **{**
     5. **EndpointDiscoveryMetadata metadata =**
     6. **e.EndpointDiscoveryMetadata;**
     7. **FindCriteria criteria =**
     8. **new FindCriteria(typeof(ISimpleChatService));**
     9. **if (criteria.IsMatch(metadata))**
     10. **{**
     11. **this.RemoveUser(metadata.Address.Uri);**
     12. **}**
     13. **}**
     14. *(Code Snippet - What is new in WCF4 Lab – OnOfflineAnnouncement Method VB)*
     15. Visual Basic
     16. **Private Sub OnOfflineAnnouncement(ByVal sender As Object, ByVal e As AnnouncementEventArgs)**
     17. **Dim metadata As EndpointDiscoveryMetadata = e.EndpointDiscoveryMetadata**
     18. **Dim criteria As New FindCriteria(GetType(ISimpleChatService))**
     19. **If criteria.IsMatch(metadata) Then**
     20. **Me.RemoveUser(metadata.Address.Uri)**
     21. **End If**
     22. **End Sub**
  5. Locate the **OpenServices** method, and add a call to **OpenAnnouncementService** serviceat the end of the method implementation.
     1. C#
     2. this.ShowStatus("Opening chat service...");
     3. this.chatServiceHost.BeginOpen(
     4. (result) =>
     5. {
     6. chatServiceHost.EndOpen(result);
     7. this.ShowStatus("Chat service ready");
     8. },
     9. null);
     10. **this.OpenAnnouncementService();**
     11. }
     12. Visual Basic
     13. Me.ShowStatus("Opening chat service...")
     14. Me.chatServiceHost.BeginOpen(Sub(result)
     15. chatServiceHost.EndOpen(result)
     16. Me.ShowStatus("Chat service ready")
     17. End Sub, Nothing)
     18. **Me.OpenAnnouncementService()**
     19. End Sub
  6. Press **CTRL+SHIFT+B** to build the solution.

# Next Step

* 1. Exercise 6: Verification

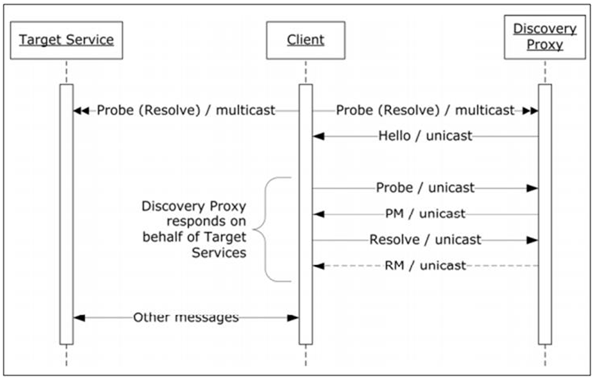
## Exercise 6: Verification

* 1. Now you will test the application to verify that the clients are able to detect the announcements.
  2. Press **CTRL+F5** to start an instance of the **DiscoveryChat** application without debugging.
  3. Switch back to Visual Studio and press **CTRL+F5** again to launch another instance of the **DiscoveryChat** application.
  4. Switch to one of the instances of **DiscoveryChat** and setup the chat as follows:
     1. User Name: **Fred**
     2. Click **Sign in**
     3. **Note:** Windows Firewall may prompt you to allow the Chat client access to use the network. It is safe to allow this.
  5. Switch to the other **DiscoveryChat** instance and setup the chat as follows:
     1. User Name: **Wilma**
     2. Click **Sign in**
  6. When Wilma signed in, Fred’s chat window should detect the online announcement and automatically add her to the list of available users.
     1. 
     2. Figure 17
     3. Users are automatically discovered
  7. Verify that the offline announcement is working by clicking the **Sign Out** button on the Wilma's chat window. This should cause Fred's window to remove Wilma from the list of available users.
  8. Try signing in and out from both applications. Each application is aware of the online / offline announcements of the other.
  9. Close both instances of the **DiscoveryChat** application.

# Next Step

* 1. Exercise 7: Discovery Proxy

Exercise 7: Discovery Proxy

* 1. All the previous exercises have relied on a well-known UDP multicast endpoint for discovery. The port and multicast address are specified by the WS-Discovery protocol documentation. The utilization of this multicast discovery is referred to as ad hoc discovery. Ad hoc discovery is limited to recognizing only services on the same subnet. Managed discovery allows you to locate services no matter where they are, as long as they are registered with a discovery proxy. In this section of the lab, you will create a discovery proxy.
  2. The following diagram shows how a Discovery Proxy responds on behalf of the target services:
  3. 
  4. Figure 18
  5. Discovery Proxy used in service discovery
  6.  **Discovery Proxy**
  7. For more information about Discovery Proxy and WS-Discovery message exchanges, see section 3 of the [WS-Discovery Specification](http://go.microsoft.com/fwlink/?LinkId=87841).

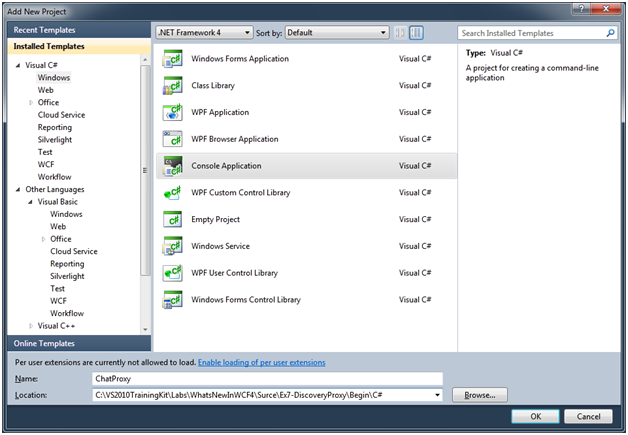
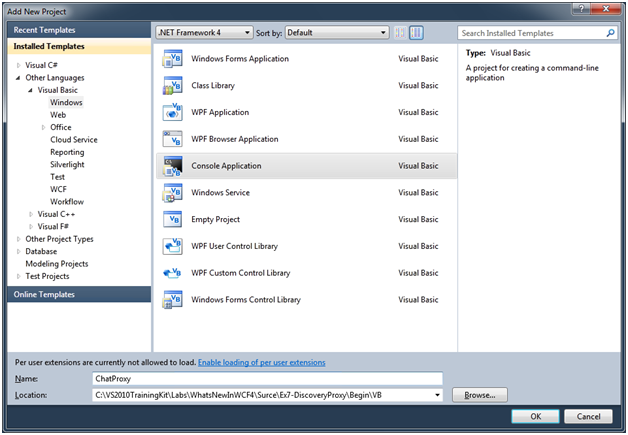
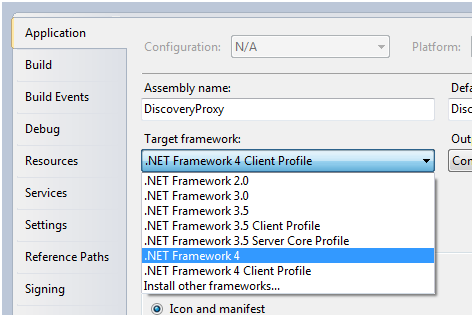
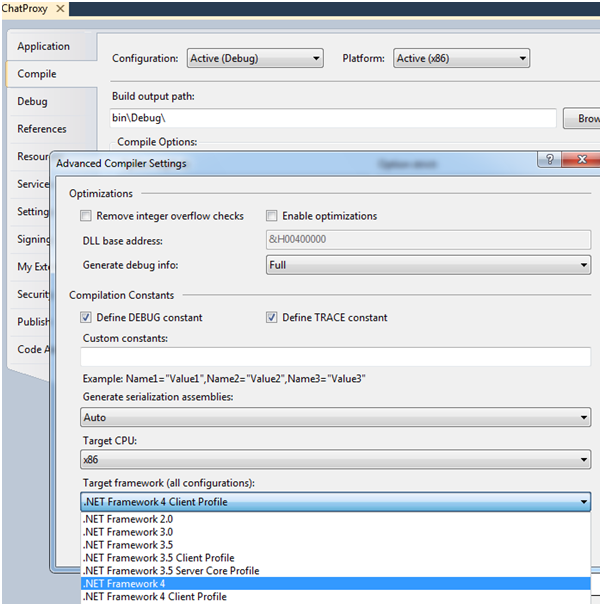
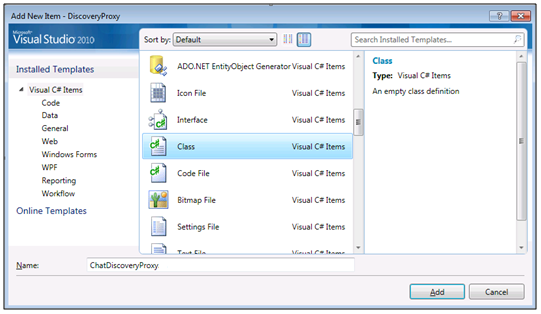
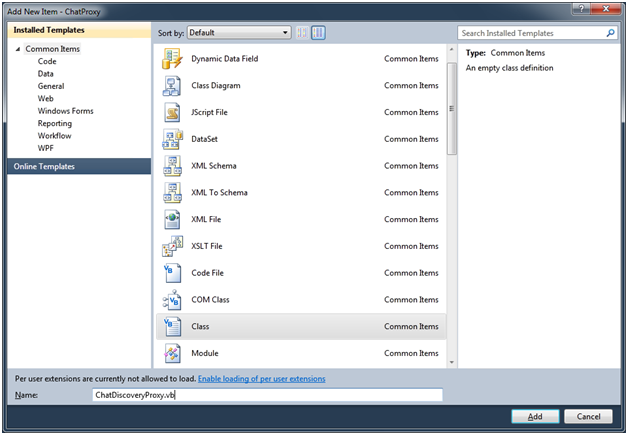
Task 0 – Opening the Solution

* 1. To begin this exercise, you can use the solution you finished from Exercise 6. Alternatively, you can follow the following steps to begin with Exercise 7.
  2. Open the starting solution for Exercise 7 located under the **Source\Ex7-DiscoveryProxy\Begin (choosing the folder that matches the language of your preference.)** Use it as the starting point for this exercise.
  3. Press **CTRL+SHIFT+B** to build the solution.

Task 1 – Creating a DiscoveryProxy

* 1. The **System.ServiceModel.Discovery** namespace includes a base class to help you in building your Discovery Proxy. To implement your proxy you will have to override a number of methods provided by the **System.ServiceModel.Discovery.*DiscoveryProxy*** class. These methods are all asynchronous to guarantee maximum scalability for the server. For this Hands-on Lab, you will focus on supporting only Announcements in the proxy. When a chat client goes online or offline the cache of metadata will be updated. Then you will respond to find requests when a client wants to query the proxy for list of chat clients.
  2. Add a new **Console Application** project to the solution. To do this, in **Solution Explorer** right-click the **Begin** solution, point to **Add** and click **New Project**. Use the following settings.

|  |  |
| --- | --- |
| Setting | Value |
| Language | **Visual C# or Visual Basic** |
| Target Framework | **.NET Framework 4** |
| Installed Templates | **Windows** |
| Template | **Console Application** |
| Name | **ChatProxy** |
| Location | **Source\Ex7-DiscoveryProxy\Begin** |

* + 1. 
    2. Figure 19
    3. Adding a new Console Application Project named “ChatProxy” (C#)
    4. 
    5. Figure 20
    6. Adding a new Console Application Project named “ChatProxy” (Visual Basic)
  1. Console applications use the .NET Framework Client Profile by default. To build a discovery proxy you need to change the target framework of the **ChatProxy** project to **.NET Framework 4**.
     1. **C#**: In **Solution Explorer**,right-click the **ChatProxy** project, and select **Properties**. In the **Application** tab, change the target framework to **.NET Framework 4**. Visual Studio will warn you that it has to unload and reload the project before continuing.
     2. **Visual Basic**: In **Solution Explorer**,right-click the **ChatProxy** project and select **Properties**. In the **Compile** tab, click **Advance Compile Options** at the bottom and change the target framework to **.NET Framework 4**. Visual Studio will warn you that it has to unload and reload the project before continuing.
     3. 
     4. Figure 21
     5. Change to the full .NET Framework 4 Profile (C#)
     6. 
     7. Figure 22
     8. Change to the full .NET Framework 4 Profile (Visual Basic)
  2. Right-click **ChatProxy** project, and click **Add Reference**. Using the **Projects** tab, add a project reference to the **DiscoveryChat** project. Repeat these steps, using the **.NET** tab to add a reference to the **System.ServiceModel** and **System.ServiceModel.Discovery** libraries.
  3. Press **CTRL+SHIFT+B** to build the solution.
  4. Your discovery proxy will need a class to implement the proxy service. Next step is to create the **ChatDiscoveryProxy** class. In **Solution Explorer**, right-click **ChatProxy** project, point to **Add**, and then click **Class**. Type **ChatDiscoveryProxy** in the **Name** box.
     1. 
     2. Figure 23
     3. Adding a new Class called “ChatDiscoveryProxy” (C#)
     4. 
     5. Figure 24
     6. Adding a new Class called “ChatDiscoveryProxy” (Visual Basic)
  5. Add the following using directives for the new class.
     1. *(Code Snippet - What is new in WCF4 Lab – ChatDiscoveryProxy Using Statements CSharp)*
     2. C#
     3. **using System.ServiceModel;**
     4. **using System.ServiceModel.Discovery;**
     5. **using System.Collections.ObjectModel;**
     6. **using Microsoft.Samples.Discovery.Contracts;**
     7. *(Code Snippet - What is new in WCF4 Lab – ChatDiscoveryProxy Using Statements VB)*
     8. Visual Basic
     9. **Imports System.ServiceModel**
     10. **Imports System.ServiceModel.Discovery**
     11. **Imports System.Collections.ObjectModel**
     12. **Imports Microsoft.Samples.Discovery.Contracts**
  6. Make your class inherit from **System.ServiceModel.Discovery.*DiscoveryProxy*** base class.
     1. C#
     2. **public class ChatDiscoveryProxy : DiscoveryProxy**
     3. Visual Basic
     4. **Public Class ChatDiscoveryProxy**
     5. **Inherits DiscoveryProxy**
  7. In this lab, your class will maintain a thread-safe in-memory cache. For this reason, you need to make your WCF service instance to be a singleton. In addition, because it is thread safe and you want to guarantee maximum scalability, you will also allow multiple concurrent calls. Modify the **ChatDiscoveryProxy** class signature, to add the following attributes.
     1. *(Code Snippet - What is new in WCF4 Lab – Proxy ServiceBehavior CSharp)*
     2. C#
     3. **[ServiceBehavior(**
     4. **InstanceContextMode = InstanceContextMode.Single,**
     5. **ConcurrencyMode = ConcurrencyMode.Multiple)]**
     6. public class ChatDiscoveryProxy : System.ServiceModel.Discovery.DiscoveryProxy
     7. *(Code Snippet - What is new in WCF4 Lab – Proxy ServiceBehavior VB)*
     8. Visual Basic
     9. **<ServiceBehavior(InstanceContextMode:=InstanceContextMode.Single, ConcurrencyMode:=ConcurrencyMode.Multiple)>**
     10. Public Class ChatDiscoveryProxy
     11. **Note:** Using an in-memory cache means that if your discovery proxy host shutdowns, all information regarding services is lost. More robust implementations would use durable storage, such as a database; this will insure that service metadata is not lost even when the service is down.
  8. You will need to create a thread safe collection class to hold instances of services you have discovered. Right-click **ChatProxy** project, point to **Add**, and then click **Class**. Type **ChatServiceCollection** in the **Name** box.
     1. **Note:** For more information see, [Collections and Synchronization (Thread Safety)](http://msdn.microsoft.com/en-us/library/573ths2x(VS.71).aspx).
  9. Add the following using directives for the new class.
     1. C#
     2. **using System.ServiceModel.Discovery;**
     3. Visual Basic
     4. **Imports System.ServiceModel.Discovery**
  10. Mark your class as **internal** (C#) **friend** (Visual Basic), and make it inherit from the **SynchronizedKeyedCollection** base class.
      1. C#
      2. **internal class ChatServiceCollection : SynchronizedKeyedCollection<Uri, EndpointDiscoveryMetadata>**
      3. Visual Basic
      4. **Friend Class ChatServiceCollection**
      5. **Inherits SynchronizedKeyedCollection(Of Uri, EndpointDiscoveryMetadata)**
  11. Implement the **GetKeyForItem** method of the **SynchronizedKeyedCollection** class as shown in following code.
      1. *(Code Snippet - What is new in WCF4 Lab – GetKeyForItem Method CSharp)*
      2. C#
      3. internal class ChatServiceCollection :
      4. SynchronizedKeyedCollection<Uri, EndpointDiscoveryMetadata>
      5. {
      6. **protected override Uri GetKeyForItem(EndpointDiscoveryMetadata item)**
      7. **{**
      8. **if (item == null)**
      9. **{**
      10. **throw new ArgumentNullException("item");**
      11. **}**
      12. **return item.Address.Uri;**
      13. **}**
      14. }
      15. *(Code Snippet - What is new in WCF4 Lab – GetKeyForItem Method VB)*
      16. Visual Basic
      17. Friend Class ChatServiceCollection
      18. Inherits SynchronizedKeyedCollection(Of Uri, EndpointDiscoveryMetadata)
      19. **Protected Overrides Function GetKeyForItem(ByVal item As System.ServiceModel.Discovery.EndpointDiscoveryMetadata) As System.Uri**
      20. **If item Is Nothing Then**
      21. **Throw New ArgumentNullException("item")**
      22. **End If**
      23. **Return item.Address.Uri**
      24. **End Function**
      25. End Class
  12. Switch back to the **ChatDiscoveryProxy** class implementation, and add a static **ChatServiceCollection** property and its backing field as shown in following code.
      1. *(Code Snippet - What is new in WCF4 Lab – ChatServiceCollection Member CSharp)*
      2. C#
      3. public class ChatDiscoveryProxy : DiscoveryProxy
      4. {
      5. **private static ChatServiceCollection cache = new ChatServiceCollection();**
      6. **internal static ChatServiceCollection Cache**
      7. **{**
      8. **get { return cache; }**
      9. **}**
      10. }
      11. *(Code Snippet - What is new in WCF4 Lab – ChatServiceCollection Member VB)*
      12. Visual Basic
      13. Public Class ChatDiscoveryProxy
      14. Inherits DiscoveryProxy
      15. **Private Shared cache\_Renamed As New ChatServiceCollection()**
      16. **Friend Shared ReadOnly Property Cache() As ChatServiceCollection**
      17. **Get**
      18. **Return cache\_Renamed**
      19. **End Get**
      20. **End Property**
      21. End Class
  13. To complete the service, you will add several helper classes, which are already developed for simplicity. To do this, right-click the **ChatProxy** project, point to **Add,** and click **Existing Item**. Then, browse to the **Source\Assets\ChatProxy** **(choosing the folder that matches the language of your preference)** folder, and add the following files:
      1. **AsyncResult.cs / AsyncResult.vb**
      2. **CompletedAsyncResult.cs / CompletedAsyncResult.vb**
      3. **FindAsyncResult.cs / FindAsyncResult.vb**
      4. **EndpointDiscoveryMetadataExtensions.cs / EndpointDiscoveryMetadataExtensions.vb**
  14. When an onlineannouncement is received, you need to determine if the service is one that you want to cache. If so, add it to your cache. Override the **OnBeginOnlineAnnouncement** method inside the **ChatDiscoveryProxy** class by adding the following code.
      1. *(Code Snippet - What is new in WCF4 Lab – OnBeginOnlineAnnouncement Method CSharp)*
      2. C#
      3. **protected override IAsyncResult OnBeginOnlineAnnouncement(DiscoveryMessageSequence messageSequence,**
      4. **EndpointDiscoveryMetadata endpointDiscoveryMetadata,**
      5. **AsyncCallback callback,**
      6. **object state)**
      7. **{**
      8. **if (endpointDiscoveryMetadata == null)**
      9. **{**
      10. **throw new ArgumentNullException("endpointDiscoveryMetadata");**
      11. **}**
      12. **// You care only about ISimpleChatService services**
      13. **FindCriteria criteria = new FindCriteria(typeof(ISimpleChatService));**
      14. **if (criteria.IsMatch(endpointDiscoveryMetadata))**
      15. **{**
      16. **endpointDiscoveryMetadata.WriteLine("Adding");**
      17. **Cache.Add(endpointDiscoveryMetadata);**
      18. **}**
      19. **return new CompletedAsyncResult(callback, state);**
      20. **}**
      21. *(Code Snippet - What is new in WCF4 Lab – OnBeginOnlineAnnouncement Method VB)*
      22. Visual Basic
      23. **Protected Overrides Function OnBeginOnlineAnnouncement(ByVal messageSequence As DiscoveryMessageSequence, ByVal endpointDiscoveryMetadata As EndpointDiscoveryMetadata, ByVal callback As AsyncCallback, ByVal state As Object) As IAsyncResult**
      24. **If endpointDiscoveryMetadata Is Nothing Then**
      25. **Throw New ArgumentNullException("endpointDiscoveryMetadata")**
      26. **End If**
      27. **' You care only about ISimpleChatService services**
      28. **Dim criteria As New FindCriteria(GetType(ISimpleChatService))**
      29. **If criteria.IsMatch(endpointDiscoveryMetadata) Then**
      30. **endpointDiscoveryMetadata.WriteLine("Adding")**
      31. **Cache.Add(endpointDiscoveryMetadata)**
      32. **End If**
      33. **Return New CompletedAsyncResult(callback, state)**
      34. **End Function**
  15. When an offline announcement message is received, you want to remove the metadata from the cache if it is there. Override the **OnBeginOfflineAnnouncement** method by adding the following code.
      1. *(Code Snippet - What is new in WCF4 Lab – OnBeginOfflineAnnouncement Method CSharp)*
      2. C#
      3. **protected override IAsyncResult OnBeginOfflineAnnouncement(DiscoveryMessageSequence messageSequence, EndpointDiscoveryMetadata endpointDiscoveryMetadata, AsyncCallback callback, object state)**
      4. **{**
      5. **try**
      6. **{**
      7. **if (endpointDiscoveryMetadata == null)**
      8. **{**
      9. **throw new ArgumentNullException("endpointDiscoveryMetadata");**
      10. **}**
      11. **// You care only about ISimpleChatService services**
      12. **FindCriteria criteria = new FindCriteria(typeof(ISimpleChatService));**
      13. **if (criteria.IsMatch(endpointDiscoveryMetadata))**
      14. **{**
      15. **endpointDiscoveryMetadata.WriteLine("Removing");**
      16. **Cache.Remove(endpointDiscoveryMetadata.Address.Uri);**
      17. **}**
      18. **}**
      19. **catch (KeyNotFoundException)**
      20. **{**
      21. **// No problem if it does not exist in the cache**
      22. **}**
      23. **return new CompletedAsyncResult(callback, state);**
      24. **}**
      25. *(Code Snippet - What is new in WCF4 Lab – OnBeginOfflineAnnouncement Method VB)*
      26. Visual Basic
      27. **Protected Overrides Function OnBeginOfflineAnnouncement(ByVal messageSequence As DiscoveryMessageSequence, ByVal endpointDiscoveryMetadata As EndpointDiscoveryMetadata, ByVal callback As AsyncCallback, ByVal state As Object) As IAsyncResult**
      28. **Try**
      29. **If endpointDiscoveryMetadata Is Nothing Then**
      30. **Throw New ArgumentNullException("endpointDiscoveryMetadata")**
      31. **End If**
      32. **' You care only about ISimpleChatService services**
      33. **Dim criteria As New FindCriteria(GetType(ISimpleChatService))**
      34. **If criteria.IsMatch(endpointDiscoveryMetadata) Then**
      35. **endpointDiscoveryMetadata.WriteLine("Removing")**
      36. **Cache.Remove(endpointDiscoveryMetadata.Address.Uri)**
      37. **End If**
      38. **Catch e1 As KeyNotFoundException**
      39. **' No problem if it does not exist in the cache**
      40. **End Try**
      41. **Return New CompletedAsyncResult(callback, state)**
      42. **End Function**
  16. Now you can override the **OnBeginFind** method, which is called when a client issues a Discovery Find request to the proxy. Here is where you can search the cache of known service endpoints and reply to the client Find request with any matching endpoints.
      1. *(Code Snippet - What is new in WCF4 Lab – OnBeginFind Method CSharp)*
      2. C#
      3. **protected override IAsyncResult OnBeginFind(FindRequestContext findRequestContext, AsyncCallback callback, object state)**
      4. **{**
      5. **if (findRequestContext == null)**
      6. **{**
      7. **throw new ArgumentNullException("findRequestContext");**
      8. **}**
      9. **Console.WriteLine(**
      10. **"Find request for contract {0}",**
      11. **findRequestContext.Criteria.ContractTypeNames.FirstOrDefault());**
      12. **// Query to find the matching endpoints**
      13. **var query = from service in Cache**
      14. **where findRequestContext.Criteria.IsMatch(service)**
      15. **select service;**
      16. **// Collection to contain the results of the query**
      17. **var matchingEndpoints = new Collection<EndpointDiscoveryMetadata>();**
      18. **// Execute the query and add the matching endpoints**
      19. **foreach (EndpointDiscoveryMetadata metadata in query)**
      20. **{**
      21. **metadata.WriteLine("\tFound");**
      22. **matchingEndpoints.Add(metadata);**
      23. **findRequestContext.AddMatchingEndpoint(metadata);**
      24. **}**
      25. **return new FindAsyncResult(matchingEndpoints, callback, state);**

1. **}**
   * 1. *(Code Snippet - What is new in WCF4 Lab – OnBeginFind Method VB)*
     2. Visual Basic
     3. **Protected Overrides Function OnBeginFind(ByVal findRequestContext As FindRequestContext, ByVal callback As AsyncCallback, ByVal state As Object) As IAsyncResult**
     4. **If findRequestContext Is Nothing Then**
     5. **Throw New ArgumentNullException("findRequestContext")**
     6. **End If**
     7. **Console.WriteLine("Find request for contract {0}", findRequestContext.Criteria.ContractTypeNames.FirstOrDefault())**
     8. **' Query to find the matching endpoints**
     9. **Dim query = From service In Cache**
     10. **Where findRequestContext.Criteria.IsMatch(service)**
     11. **Select service**
     12. **' Collection to contain the results of the query**
     13. **Dim matchingEndpoints = New Collection(Of EndpointDiscoveryMetadata)()**
     14. **' Execute the query and add the matching endpoints**
     15. **For Each metadata As EndpointDiscoveryMetadata In query**
     16. **metadata.WriteLine(Constants.vbTab & "Found")**
     17. **matchingEndpoints.Add(metadata)**
     18. **findRequestContext.AddMatchingEndpoint(metadata)**
     19. **Next metadata**
     20. **Return New FindAsyncResult(matchingEndpoints, callback, state)**
     21. **End Function**
   1. Override the **OnEndFind** method to complete the find operation.
      1. *(Code Snippet - What is new in WCF4 Lab – OnEndFind Method CSharp)*
      2. C#
      3. **protected override void OnEndFind(IAsyncResult result)**
      4. **{**
      5. **FindAsyncResult.End(result);**
      6. **}**
      7. *(Code Snippet - What is new in WCF4 Lab – OnEndFind Method VB)*
      8. Visual Basic
      9. **Protected Overrides Sub OnEndFind(ByVal result As System.IAsyncResult)**
      10. **FindAsyncResult.End(result)**
      11. **End Sub**
      12. **Note:** In this lab solution, the Discovery Proxy is implemented as a Console Application. Production environments would use a more robust hosting solution such as a Windows Service.
   2. Override the rest of the required methods declared abstract in **System.ServiceModel.Discovery.*DiscoveryProxy*** class.
      1. *(Code Snippet - What is new in WCF4 Lab – DiscoveryProxy Abstract Methods CSharp)*
      2. C#
      3. **protected override IAsyncResult OnBeginResolve(ResolveCriteria resolveCriteria, AsyncCallback callback, object state)**
      4. **{**
      5. **return new CompletedAsyncResult(callback, state);**
      6. **}**
      7. **protected override EndpointDiscoveryMetadata OnEndResolve(IAsyncResult result)**
      8. **{**
      9. **return CompletedAsyncResult<EndpointDiscoveryMetadata>.End(result);**
      10. **}**
      11. **protected override void OnEndOfflineAnnouncement(IAsyncResult result)**
      12. **{**
      13. **CompletedAsyncResult.End(result);**
      14. **}**
      15. **protected override void OnEndOnlineAnnouncement(IAsyncResult result)**
      16. **{**
      17. **CompletedAsyncResult.End(result);**
      18. **}**
      19. *(Code Snippet - What is new in WCF4 Lab – DiscoveryProxy Abstract Methods VB)*
      20. Visual Basic
      21. **Protected Overrides Function OnBeginResolve(ByVal resolveCriteria As ResolveCriteria, ByVal callback As AsyncCallback, ByVal state As Object) As IAsyncResult**
      22. **Return New CompletedAsyncResult(callback, state)**
      23. **End Function**
      24. **Protected Overrides Function OnEndResolve(ByVal result As IAsyncResult) As EndpointDiscoveryMetadata**
      25. **Return CompletedAsyncResult(Of EndpointDiscoveryMetadata).End(result)**
      26. **End Function**
      27. **Protected Overrides Sub OnEndOfflineAnnouncement(ByVal result As IAsyncResult)**
      28. **CompletedAsyncResult.End(result)**
      29. **End Sub**
      30. **Protected Overrides Sub OnEndOnlineAnnouncement(ByVal result As IAsyncResult)**
      31. **CompletedAsyncResult.End(result)**
      32. **End Sub**
   3. Now you need to modify the **Main** method to create a **ServiceHost** for your **ChatDiscoveryProxy** service. To do this, open **Program.cs (C#)** or **Module1.vb (Visual Basic)** in the **ChatProxy** project.
   4. Add the following using directives to **Program.cs (C#)** or **Module1.vb (Visual Basic)**.
      1. C#
      2. **using System.Net;**
      3. **using System.ServiceModel;**
      4. **using System.ServiceModel.Discovery;**
      5. Visual Basic
      6. **Imports System.Net**
      7. **Imports System.ServiceModel**
      8. **Imports System.ServiceModel.Discovery**
   5. For hosting the **ChatDiscoveryProxy** service, you will create the **DiscoveryEndpoint** endpoint for the service host. Add the following method inside the **Program** class (C#) or the **Module1** module (Visual Basic).
      1. **Note:** In this lab solution, you will be using TCP port 8001 for your proxy service.
      2. *(Code Snippet - What is new in WCF4 Lab – HostDiscoveryEndpoint Method CSharp)*
      3. C#
      4. **private static ServiceHost HostDiscoveryEndpoint(string hostName)**
      5. **{**
      6. **// Create a new ServiceHost with a singleton ChatDiscovery Proxy**
      7. **ServiceHost myProxyHost = new**
      8. **ServiceHost(new ChatDiscoveryProxy());**
      9. **string proxyAddress = "net.tcp://" +**
      10. **hostName + ":8001/discoveryproxy";**
      11. **// Create the discovery endpoint**
      12. **DiscoveryEndpoint discoveryEndpoint =**
      13. **new DiscoveryEndpoint(**
      14. **new NetTcpBinding(),**
      15. **new EndpointAddress(proxyAddress));**
      16. **discoveryEndpoint.IsSystemEndpoint = false;**
      17. **// Add UDP Annoucement endpoint**
      18. **myProxyHost.AddServiceEndpoint(new UdpAnnouncementEndpoint());**
      19. **// Add the discovery endpoint**
      20. **myProxyHost.AddServiceEndpoint(discoveryEndpoint);**
      21. **myProxyHost.Open();**
      22. **Console.WriteLine("Discovery Proxy {0}",**
      23. **proxyAddress);**
      24. **return myProxyHost;**
      25. **}**
      26. *(Code Snippet - What is new in WCF4 Lab – HostDiscoveryEndpoint Method VB)*
      27. Visual Basic
      28. **Private Function HostDiscoveryEndpoint(ByVal hostName As String) As ServiceHost**
      29. **' Create a new ServiceHost with a singleton ChatDiscovery Proxy**
      30. **Dim myProxyHost As New ServiceHost(New ChatDiscoveryProxy())**
      31. **Dim proxyAddress As String = "net.tcp://" & hostName & ":8001/discoveryproxy"**
      32. **' Create the discovery endpoint**
      33. **Dim discoveryEndpoint As New DiscoveryEndpoint(New NetTcpBinding(), New EndpointAddress(proxyAddress))**
      34. **discoveryEndpoint.IsSystemEndpoint = False**
      35. **' Add UDP Annoucement endpoint**
      36. **myProxyHost.AddServiceEndpoint(New UdpAnnouncementEndpoint())**
      37. **' Add the discovery endpoint**
      38. **myProxyHost.AddServiceEndpoint(discoveryEndpoint)**
      39. **myProxyHost.Open()**
      40. **Console.WriteLine("Discovery Proxy {0}", proxyAddress)**
      41. **Return myProxyHost**
      42. **End Function**
   6. Modify the **Main** method as shown in the following code, to host the **ChatDiscoveryProxy** service.
      1. *(Code Snippet - What is new in WCF4 Lab – DiscoveryProxy Main Method CSharp)*
      2. C#
      3. static void Main(string[] args)
      4. {
      5. **Console.Title = "ChatProxy Service";**
      6. **Console.WriteLine("ChatProxy Console Host");**
      7. **string hostName = Dns.GetHostName();**
      8. **using (ServiceHost proxyHost = HostDiscoveryEndpoint(hostName))**
      9. **{**
      10. **Console.WriteLine("Press <Enter> to exit");**
      11. **Console.ReadLine();**
      12. **proxyHost.Close();**
      13. **}**
      14. }
      15. *(Code Snippet - What is new in WCF4 Lab – DiscoveryProxy Main Method CSharp)*
      16. C#
      17. Sub Main()
      18. **Console.Title = "ChatProxy Service"**
      19. **Console.WriteLine("ChatProxy Console Host")**
      20. **Dim hostName As String = Dns.GetHostName()**
      21. **Using proxyHost As ServiceHost = HostDiscoveryEndpoint(hostName)**
      22. **Console.WriteLine("Press <Enter> to exit")**
      23. **Console.ReadLine()**
      24. **proxyHost.Close()**
      25. **End Using**
      26. End Sub
   7. Press **CTRL+SHIFT+B** to build the solution.

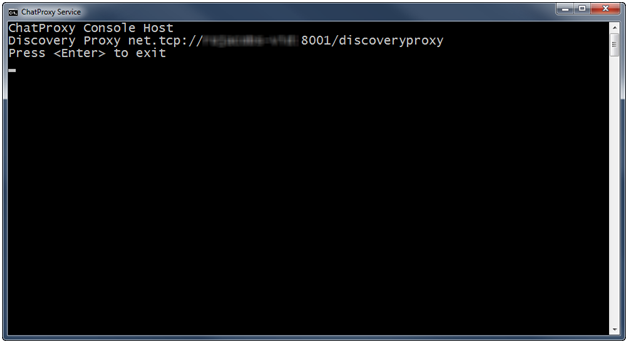
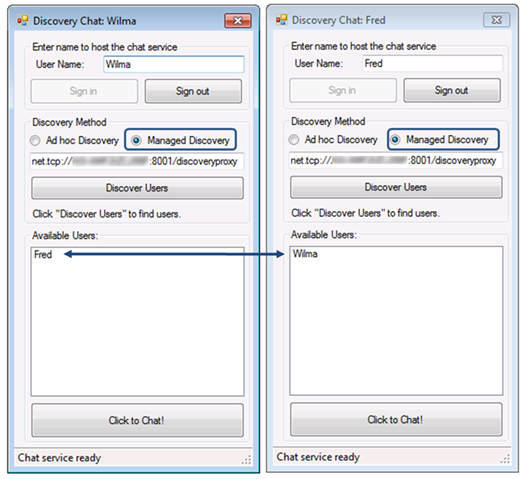
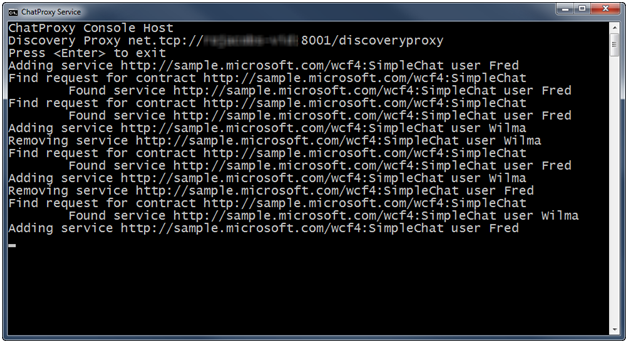
Task 2 – Modifying the DiscoveryChat Application to use Managed Discovery

* 1. Now that you have implemented a discovery proxy, you need to modify the chat application to make use of it.
  2. Open the **SimpleChat** form in code view from the **DiscoveryChat** project. You can open the code view by selecting the file and pressing **F7**.
  3. Locate the **ManagedDiscovery** method. The UI is already calling this method if the Managed Discovery radio button is checked. As you have already implemented the handlers for ad hoc discovery, the only thing pending is to add code to implement this managed discovery as shown in the following code.
     1. *(Code Snippet - What is new in WCF4 Lab – ManagedDiscovery Method CSharp)*
     2. C#
     3. private void ManagedDiscovery()
     4. {
     5. **try**
     6. **{**
     7. **// Create an endpoint for the proxy**
     8. **DiscoveryEndpoint proxyEndpoint =**
     9. **new DiscoveryEndpoint(**
     10. **new NetTcpBinding(),**
     11. **new EndpointAddress(proxyAddressText.Text));**
     12. **// Create the DiscoveryClient with a proxy endpoint**
     13. **// for managed discovery**
     14. **this.discoveryClient = new DiscoveryClient(proxyEndpoint);**
     15. **// Same handlers as ad hoc discovery**
     16. **this.discoveryClient.FindCompleted +=**
     17. **new EventHandler<FindCompletedEventArgs>(this.OnFindCompleted);**
     18. **this.discoveryClient.FindProgressChanged +=**
     19. **new EventHandler<FindProgressChangedEventArgs>(this.OnFindProgressChanged);**
     20. **// Setup the form for discovery**
     21. **this.ShowDiscoveryInProgress(true);**
     22. **this.discoveryClient.FindAsync(new FindCriteria(typeof(ISimpleChatService)));**
     23. **}**
     24. **catch (UriFormatException)**
     25. **{**
     26. **MessageBox.Show(**
     27. **Resources.InvalidUriMessage,**
     28. **this.Text,**
     29. **MessageBoxButtons.OK,**
     30. **MessageBoxIcon.Information,**
     31. **MessageBoxDefaultButton.Button1,**
     32. **(MessageBoxOptions)0);**
     33. **}**
     34. }
     35. *(Code Snippet - What is new in WCF4 Lab – ManagedDiscovery Method VB)*
     36. Visual Basic
     37. Private Sub ManagedDiscovery()
     38. **Try**
     39. **' Create an endpoint for the proxy**
     40. **Dim proxyEndpoint As New DiscoveryEndpoint(New NetTcpBinding(), New EndpointAddress(proxyAddressText.Text))**
     41. **' Create the DiscoveryClient with a proxy endpoint**
     42. **' for managed discovery**
     43. **Me.\_discoveryClient = New DiscoveryClient(proxyEndpoint)**
     44. **' Same handlers as ad hoc discovery**
     45. **AddHandler \_discoveryClient.FindCompleted, AddressOf OnFindCompleted**
     46. **AddHandler \_discoveryClient.FindProgressChanged, AddressOf OnFindProgressChanged**
     47. **' Setup the form for discovery**
     48. **Me.ShowDiscoveryInProgress(True)**
     49. **Me.\_discoveryClient.FindAsync(New FindCriteria(GetType(ISimpleChatService)))**
     50. **Catch e1 As UriFormatException**
     51. **MessageBox.Show(My.Resources.InvalidUriMessage, Me.Text, MessageBoxButtons.OK, MessageBoxIcon.Information, MessageBoxDefaultButton.Button1, CType(0, MessageBoxOptions))**
     52. **End Try**
     53. End Sub
  4. Locate the **InitializeManagedDiscovery** method, and set the managed radio button *Checked* property to **True**.
     1. C#
     2. private void InitializeManagedDiscovery()
     3. {
     4. **this.managedRadioButton.Checked = true;**
     5. }
     6. Visual Basic
     7. Private Sub InitializeManagedDiscovery()
     8. **Me.managedRadioButton.Checked = True**
     9. End Sub
  5. Press **CTRL+SHIFT+B** to build the solution.

# Next Step

* 1. Exercise 7: Verification

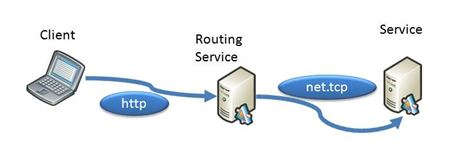
## Exercise 7: Verification

* 1. Now you will test the chat application to see that the clients are able to detect the announcements.
  2. Right-click the **ChatProxy** project, and click **Set as StartUp Project**.
  3. Press **F5** to start **ChatProxy** console in the debugger.
     1. 
     2. Figure 25
     3. ChatProxy Service running
     4. **Note:** Windows Firewall may prompt you to allow the Proxy and/or Chat client access to use the network. It is safe to allow this.
  4. Switch back to Visual Studio solution Explorer and start two instances of the **DiscoveryChat** application by right-clicking the **DiscoveryChat** project, pointing to **Debug**, and clicking **Start new instance**.
  5. Switch to one of the instances of the **DiscoveryChat** application and setup the chat as follows:
     1. User Name: **Fred**
     2. Click **Sign in**
  6. Switch to the other **DiscoveryChat** instance and setup the chat as follows:
     1. User Name: **Wilma**
     2. Click **Sign in**
  7. When Wilma signed in, her window found Fred from the managed discovery proxy. Fred's chat window should detect the online announcement from Wilma's window and automatically add her.
     1. 
     2. Figure 26
     3. Managed discovery in action
  8. Verify that the offline announcement is working by clicking the **Sign Out** button on Wilma's chat window. This should cause Fred's window to remove Wilma from the list of available users.
  9. Try signing in and out from both applications. Each application is aware of the online / offline announcements of the other.
     1. 
     2. Figure 27
     3. ChatProxy Service in action
  10. Press **SHIFT+F5 (C#)** or **CTRL+ALT+BREAK (Visual Basic)** to stop debugging.

# Next Step

* 1. Exercise 8: Protocol Bridging

Exercise 8: Protocol Bridging

* 1. WCF4 includes a new routing service found in the **System.ServiceModel.Routing** namespace. The Routing Service is designed to act as a generic, configurable SOAP intermediary. It allows you to configure Content Based Routing, set up Protocol Bridging, and handle communication errors that you encounter. The Routing Service also makes it possible for you to update your Routing Configuration while the Routing Service is running without restarting the service.
  2. 
  3. Figure 28
  4. The routing service bridging http and net.tcp
  5. Imagine that you had a client application that can only communicate using **basic http** and at the same time, you have a service that communicates using **net.tcp**. There are many ways to solve this problem. For example, with WCF you can have more than one endpoint for a service so it would be possible to expose an endpoint using **basicHttpBinding** for the client.
  6. With WCF4 **System.ServiceModel.Routing** you now have another way to solve this problem. In this exercise, you will use the routing service to bride the http and net.tcp protocol by simply configuring the routing service.

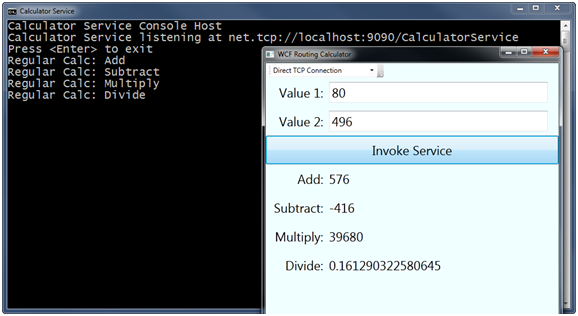
Task 0 – Opening the Solution

* 1. This exercise uses a new begin solution
  2. Open the starting solution for Exercise 8 located under the **Source\Ex8-ProtocolBridging\Begin (choosing the folder that matches the language of your preference.)** Use it as the starting point for this exercise.
  3. Press **CTRL+SHIFT+B** to build the solution.

Task 1 – Trying the Calculator

* 1. In this task, you will explore the calculator solution.
  2. The solution contains two projects a **CalculatorService** and **CalculatorClient** project.
  3. Open the app.config file from the CalculatorService project. Note the base address of the CalculatorService is using a net.tcp address.
     1. App.config
     2. <service name="CalculatorService.CalculatorService">
     3. <!-- ... -->
     4. <host>
     5. <baseAddresses>
     6. <add baseAddress="net.tcp://localhost:9090/CalculatorService" />
     7. </baseAddresses>

</host>

* 1. The **CalculatorClient** project has a Service Reference to the **CalculatorService** project. Open the app.config file for the **CalculatorClient** project, and note the client section is also using net.tcp to communicate with the calculator service.
     1. App.config
     2. <client>
     3. <endpoint address="net.tcp://localhost:9090/CalculatorService"
     4. binding="netTcpBinding" bindingConfiguration="CalculatorService"
     5. contract="CalculatorServiceReference.ICalculatorService"
     6. name="CalculatorService" />
     7. </client>
  2. The solution is set to use multiple startup projects. If this is not the case, right-click the solution node in **Solution Explorer**, and select **Set StartUp Projects** to configure them. Press **CTRL+F5** to start the solution. This will launch the **CalculatorService** and **CalculatorClient** applications.
     1. **Note:** Windows Firewall may prompt you to allow the service access to use the network. It is safe to allow this.
  3. The CalculatorClient will be pre-loaded with random values. Press **Invoke Service** to invoke the service over net.tcp. You will see something similar to the following.
     1. 
     2. Figure 29
     3. The Calculator Service and Client Application connected over TCP

Task 2 – Adding the Router Service

* 1. Your calculator works great if you are able to connect with net.tcp, but what if you had a requirement that the calculator application could access the calculator service over the Internet?
  2. In this task, you will change the CalculatorClient to connect to a router service bridging the http and net.tcp protocols.
  3. Right-click the **Ex8-ProtocolBriding** solution, point to **Add** and select **Existing Project**. Browse to **Source\Assets\RouterService (Choosing the folder that matches the language of your preference.)** and add the **RouterService** project.
  4. The routing service has to be configured Open the **Web.config** file from the **RouterService** project.
  5. The routing service is both a service and a client. The first thing you will do is add the client endpoint and binding configuration to the **Web.config** file. Add the following configuration as shown.
     1. *(Code Snippet - What is new in WCF4 Lab – RoutingService calc endpoint XML)*
     2. Web.config
     3. <system.serviceModel>
     4. **<bindings>**
     5. **<netTcpBinding>**
     6. **<binding name="CalculatorService" >**
     7. **<security mode="None" />**
     8. **</binding>**
     9. **</netTcpBinding>**
     10. **</bindings>**
     11. **<client>**
     12. **<endpoint name="regularCalculatorEndpoint"**
     13. **address="net.tcp://localhost:9090/CalculatorService"**
     14. **binding="netTcpBinding"**
     15. **bindingConfiguration="CalculatorService"**
     16. **contract="\*" />**
     17. **</client>**
     18. </system.serviceModel>
  6. The routing service uses a filter table to map incoming messages to client endpoints. The next thing you need to do is to create the filter table and service behavior that will apply it in the configuration. To do this, add the following portion after the **client** section replacing the original **behaviors** node.
     1. *(Code Snippet - What is new in WCF4 Lab – Routing service behavior XML)*
     2. Web.config
     3. </client>
     4. **<behaviors>**
     5. **<serviceBehaviors>**
     6. **<behavior name="routingConfiguration">**
     7. **<routing filterTableName="filterTable1" />**
     8. **<serviceDebug includeExceptionDetailInFaults="true"/>**
     9. **</behavior>**
     10. **<behavior name="">**
     11. **<serviceMetadata httpGetEnabled="true" />**
     12. **<serviceDebug includeExceptionDetailInFaults="false" />**
     13. **</behavior>**
     14. **</serviceBehaviors>**
     15. **</behaviors>**
     16. **<routing>**
     17. **<filters>**
     18. **<filter name="MatchAllFilter" filterType="MatchAll" />**
     19. **</filters>**
     20. **<filterTables>**
     21. **<filterTable name="filterTable1">**
     22. **<add filterName="MatchAllFilter"   
          endpointName="regularCalculatorEndpoint" priority="0"/>**
     23. **</filterTable>**
     24. **</filterTables>**
     25. **</routing>**
     26. </system.serviceModel>
     27.  **What does the filter table do?**
     28. This filter table will match all requests and route them to the regularCalculatorEndpoint listening at **net.tcp://localhost:9090/CalculatorService**.
  7. The last thing you need to do is to add a service definition for the router service. Your router service will be listening on http and routing requests to the calculator service, which is listening on net.tcp – this effectively bridges the protocols. Add the following configuration after the **routing** section.
     1. *(Code Snippet - What is new in WCF4 Lab – Routing service XML)*
     2. Web.config
     3. </routing>
     4. **<services>**
     5. **<service behaviorConfiguration="routingConfiguration"**
     6. **name="System.ServiceModel.Routing.RoutingService">**
     7. **<endpoint address="general"**
     8. **binding="basicHttpBinding"**
     9. **name="routerEndpoint1"**
     10. **contract="System.ServiceModel.Routing.IRequestReplyRouter" />**
     11. **</service>**
     12. **</services>**
     13. </system.serviceModel>

Task 3 – Modifying the Client to Enable Routed HTTP

* 1. Your client application will need a client endpoint definition to use with the router service. Open the **app.config** file from the **CalculatorClient** project, and add the following configuration after the existing endpoint.
     1. *(Code Snippet - What is new in WCF4 Lab – CalcClient endpoint XML)*
     2. XML
     3. **<endpoint address="http://localhost:8000/Router.svc/general"**
     4. **binding="basicHttpBinding"**
     5. **contract="CalculatorServiceReference.ICalculatorService"**
     6. **name="RouterService" />**

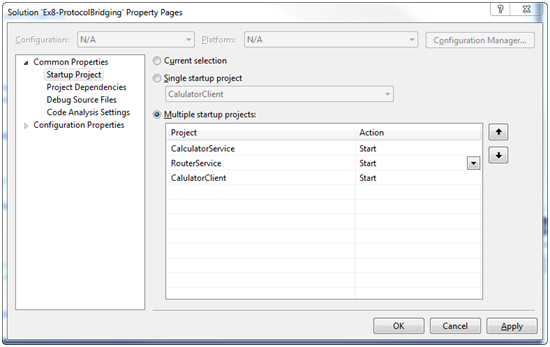
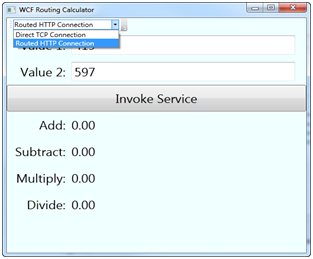
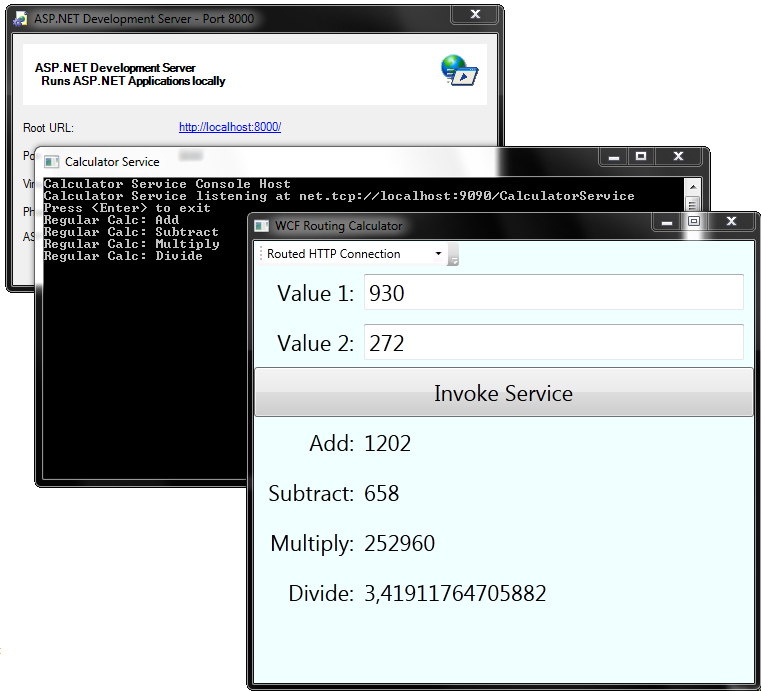
</client>

* + 1.  **Named endpoints in configuration**
    2. If your client section contains more than one endpoint definition, you will have to provide an endpoint name when creating a client proxy. Look at the **CalculateResults** method of MainWindow, which has been written to pass the endpoint name to the proxy constructor.

# Next Step

* 1. Exercise 8: Verification

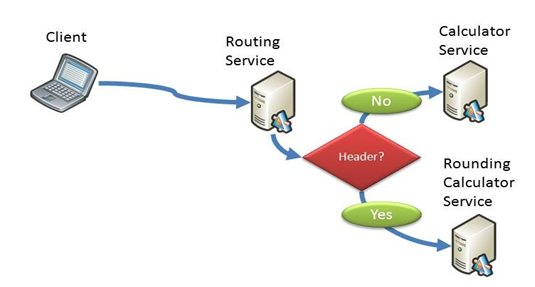
## Exercise 8: Verification

* 1. In this verification, you will test the **CalculatorClient** application to see if it can route requests over HTTP as well as over TCP connections.
  2. Right-click the **Ex-8ProtocolBriding** solution, and select **Set Startup Projects**.
  3. Select the **Multiple startup projects** radio button, and set all three projects to Start.
     1. 
     2. Figure 30
     3. Set Multiple Startup projects
  4. Click **OK** to dismiss the solution properties dialog.
  5. Press **F5** to debug the solution. All three projects should now start.
  6. Select **Routed HTTP Connection** in the **CalculatorClient** toolbar dropdown.
     1. 
     2. Figure 31
     3. Select the Routed HTTP Connection option
  7. Click **Invoke Service** to test the routed connection, it should invoke the service successfully.
     1. 
     2. Figure 32
     3. The CalculatorClient invoking the service using routed HTTP

# Next Step

* 1. Exercise 9: Content Based Routing

Exercise 9: Content Based Routing

* 1. In the previous exercise, you routed all requests bound for your service bridging the protocol from HTTP to TCP. However, there are many cases where you want to route requests based on the content of the message. While you can route based on content from the message body for performance it is recommended that you route based on content in the message headers.
  2. In this exercise, you will add a variation of the calculator service that uses rounding. You will then route to this service messages that contain a special header while other messages that do not have the header will use the regular calculator service.
  3. 
  4. Figure 33
  5. The routing service routing based on message header content

Task 0 – Opening the Solution

* 1. This exercise uses a new begin solution which contains a new service, the RoundingCalculatorService project and some updates to the CalculatorClient application.
  2. Open the starting solution for Exercise 9 located under the **Source\Ex9-ContentRouting\Begin** **(choosing the folder that matches the language of your preference.)** Use it as the starting point for this exercise.
  3. Press **CTRL+SHIFT+B** to build the solution.

Task 1 – Modifying the CalculatorClient to Add a Custom Header

* 1. The CalculatorClient application now has a button on the toolbar to indicate if you want to use the Rounding calculator. Even though this is a different service, you will send the message to the same routing service. In this task, you will write the code to add a custom header that the router will use to route the message.
  2. In the **CalculatorClient** project open **MainWindow.xaml.cs (C#)** or **MainWindow.xaml.vb (Visual Basic)**.
  3. Locate the **AddOptionalRoundingHeader** method and modify it as shown to add a custom header to outgoing messages that will use the RoundingCalculator.
     1. (Code Snippet - What is new in WCF4 Lab –*AddOptionalRoundingHeader method CSharp*)
     2. C#
     3. private void AddOptionalRoundingHeader(
     4. CalculatorServiceClient proxy)
     5. {
     6. **if (this.checkRounding.IsChecked.Value == true)**
     7. **{**
     8. **OperationContext ctx = OperationContext.Current;**
     9. **MessageHeaders messageHeadersElement = ctx.OutgoingMessageHeaders;**
     10. **ctx.OutgoingMessageHeaders.Add(**
     11. **MessageHeader.CreateHeader(**
     12. **"RoundingCalculator",**
     13. **"http://my.custom.namespace/",**
     14. **"1"));**
     15. **}**
     16. }
     17. (Code Snippet - What is new in WCF4 Lab –*AddOptionalRoundingHeader method VB*)
     18. Visual Basic
     19. Private Sub AddOptionalRoundingHeader(ByVal proxy As CalculatorServiceClient)
     20. **If (Me.checkRounding.IsChecked.Value = True) Then**
     21. **Dim ctx = OperationContext.Current**
     22. **Dim messageHeadersElement = ctx.OutgoingMessageHeaders**
     23. **ctx.OutgoingMessageHeaders.Add(**
     24. **MessageHeader.CreateHeader(**
     25. **"RoundingCalculator",**
     26. **"http://my.custom.namespace/",**
     27. **"1"))**
     28. **End If**
     29. End Sub

Task 2 – Adding a New Entry to the Filter Table

* 1. In this task, you will modify the router to detect this new header and route the message to the rounding calculator.
  2. Because your header uses a custom namespace, you will need to create a namespace table. In the **RouterService** project open the **Web.config** file and add the following configuration to the **routing** section.
     1. *(Code Snippet - What is new in WCF4 Lab –Namespace Table XML)*
     2. Web.config
     3. <routing>
     4. **<namespaceTable>**
     5. **<add prefix="custom" namespace="http://my.custom.namespace/"/>**
     6. **</namespaceTable>**
  3. Now you can define a new kind of filter that will match using an XPath expression. Add the following filter definition to the **Web.config** file. This filter will match on messages with the custom RoundingCalculatorHeader with a value of 1.
     1. *(Code Snippet - What is new in WCF4 Lab –XPathFilter XML)*
     2. Web.config
     3. <filters>
     4. **<filter name="XPathFilter"**
     5. **filterType="XPath"**
     6. **filterData="sm:header()/custom:RoundingCalculator = 1"/>**
  4. You need to add a new endpoint address for the *RoundingCalculatorService* so that you can route to it. Modify the **Web.config** file in the **RouterService** project, and add the new endpoint definition to the **client** section as shown.
     1. *(Code Snippet - What is new in WCF4 Lab – roundingCalculatorEndpoint XML)*
     2. Web.config
     3. <client>
     4. **<endpoint name="roundingCalculatorEndpoint"**
     5. **address="net.tcp://localhost:8080/RoundingCalculatorService"**
     6. **binding="netTcpBinding"**
     7. **bindingConfiguration="CalculatorService"**

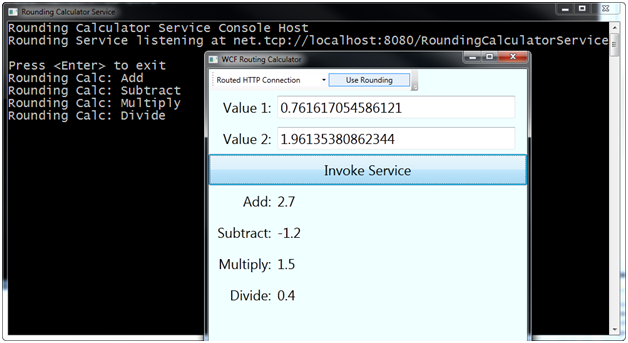
**contract="\*" />**

* 1. Finally, you need to add the new filter to the filter table. The filter table evaluates matches based on the order of priority in the filters with higher numbers having higher priority. Because you want the **XPathFilter** to match messages before the MatchAll filter you need to give it a higher priority. Open the **Web.config** of the RouterService, and modify the **filterTable** configuration as shown.
     1. *(Code Snippet - What is new in WCF4 Lab – filterTable1 XPathFilter XML)*
     2. Web.config
     3. <filterTable name="filterTable1">
     4. **<add filterName="XPathFilter"**
     5. **endpointName="roundingCalculatorEndpoint"**
     6. **priority="1"/>**
     7. <add filterName="MatchAllFilter"
     8. endpointName="regularCalculatorEndpoint"
     9. priority="0"/>
     10. </filterTable>

# Next Step

* 1. Exercise 9: Verification

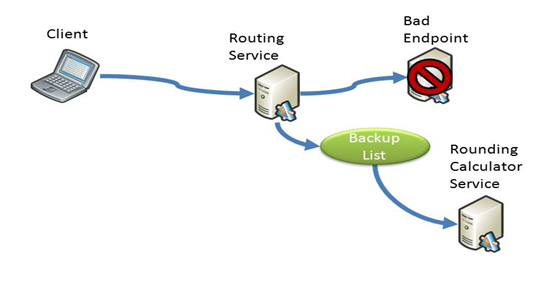
## Exercise 9: Verification

* 1. Configure the solution for multiple startup projects in this order:
     1. CalculatorService
     2. RoundingCalculatorService
     3. RouterService
     4. CalculatorClient
  2. Press **F5** to debug the solution.
  3. In the CalculatorClient select **Routed HTTP connection.**
  4. Click the **Use Rounding** button in the toolbar.
  5. Click the **Invoke Service** button.
  6. You should see values rounded to 1 decimal place.
     1. 
     2. Figure 34
     3. The routing service routed messages to the rounding calculator service
  7. Click the **Use Rounding** button to turn off rounding.
  8. Click the **Invoke Service** button again.
  9. This time the router will select the regular calculator service, and you will see that the values are not rounded to one decimal place.

# Next Step

* 1. Exercise 10: Backup Lists

Exercise 10: Backup Lists

* 1. You have seen how routing can be used to bridge protocols and route based on content. In this exercise, you will see how you can add list of alternate services to be used in the event that the primary service cannot be contacted.
  2. 
  3. Figure 35
  4. The Routing Service using a backup list

Task 0 – Opening the Solution

* 1. To begin this exercise you can use the solution you finished from Exercise 9. Alternatively, you can follow the following steps to begin with Exercise 10.
  2. Open the starting solution for Exercise 10 located under the **Source\Ex10-BackupLists\Begin** **(choosing the folder that matches the language of your preference.)** Use it as the starting point for this exercise.
  3. Press **CTRL+SHIFT+B** to build the solution.

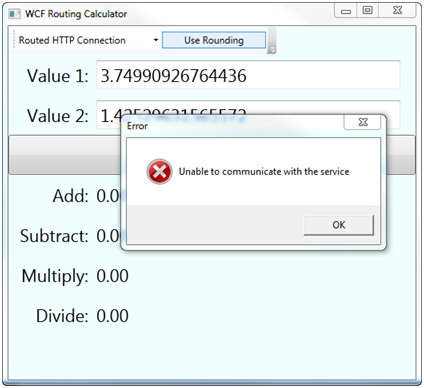
Task 1 – Adding a Bad Endpoint

* 1. In this task, you will modify the configuration to call an endpoint that does not exist. This will allow us to see how the router service behaves when it cannot contact an endpoint.
  2. Open the **Web.config** file from the **RouterService** project.
  3. Add the **badEndpoint** definition to the **client** section as shown.
     1. *(Code Snippet - What is new in WCF4 Lab – badEndpoint XML)*
     2. Web.config
     3. <client>
     4. **<endpoint name="badEndpoint"**
     5. **address="http://localhost:404/bad/endpoint"**
     6. **binding="wsHttpBinding"**

**contract="\*" />**

* 1. Now, in order to get a communication failure, you must actually direct the Routing Service to send to this endpoint. Replace one of the good calculator endpoints in the filter table. In the filter table definition, change the entry that references the *roundingCalculatorEndpoint* to instead reference **badEndpoint**
     1. XML
     2. <filterTables>
     3. <filterTable name="filterTable1">
     4. <add filterName="XPathFilter"
     5. endpointName=**"badEndpoint"**
     6. priority="1"/>

Task 2 – Observing the Routing Fail

* 1. Right-click the **Ex10-BackupLists** solution and select **Set StartUp Projects**.
  2. Select **Multiple startup projects** and set all the projects to **Start**.
  3. Press **F5** to debug the solution.
  4. In the CalculatorClient select **Routed HTTP connection** and click on the **Use Rounding** button to add the rounding message header.
  5. Click on **Invoke Service**.
  6. The client will receive a communication exception because the router tried to route the message to an endpoint that does not exist.
     1. 
     2. Figure 36
     3. The client receives a communication error

Task 3 – Enabling Backup Lists

* 1. The router service can establish a backup list for each endpoint that it routes to. In the event that it is unable to communicate to the primary endpoint, it will begin trying the services in the backup list until it successfully invokes a service or all of them fail.
  2. Open the **Web.config** file from the **RouterService** project.
  3. Add a backup list in the **routing** section as shown.
     1. *(Code Snippet - What is new in WCF4 Lab – BackupLists XML)*
     2. Web.config
     3. <routing>
     4. **<backupLists>**
     5. **<backupList name="backupList1">**
     6. **<add endpointName="roundingCalculatorEndpoint"/>**
     7. **</backupList>**
     8. **</backupLists>**
  4. Filters can have an optional backup list. Modify the XPathFilter that is routing to the **badEndpoint** and add a backup list as shown.
     1. XML
     2. <filterTable name="filterTable1">
     3. <add filterName="XPathFilter"
     4. endpointName="badEndpoint"
     5. **backupList="backupList1"**
     6. priority="1"/>

# Next Step

* 1. Exercise 10: Verification

## Exercise 10: Verification

* 1. Press **F5** to launch the solution in the debugger, all 4 projects should start.
  2. In the CalculatorClient select **Routed HTTP connection** and click on the **Use Rounding** button to add the rounding message header.
  3. The client will work but you will see that it is noticeably slower. This is because each call to the service must first fail; then the router will call the services in the backup list until it succeeds. Each operation from the client is a separate call so there will be four failed calls and four successful ones.

# Next Step

* 1. Exercise 11: Deploying the Web Application

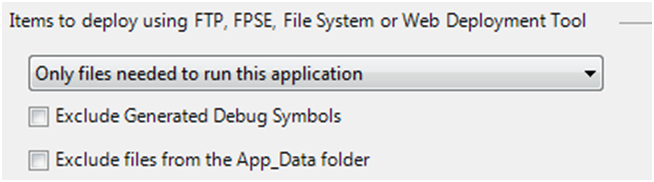
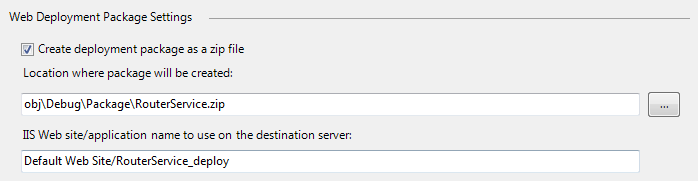
Exercise 11: Deploying the Web Application

* 1. In this exercise, you will deploy the Routing we application project using Windows Server AppFabric; a set of extensions to the Windows Application Server and the Windows Process Activation Service (WAS). The goal of Windows Server AppFabric is to simplify the deployment, configuration, management, and monitoring of Windows Communication Foundation (WCF) and Windows Workflow Foundation (WF) services built on .NET 4.
  2.  Is AppFabric installed?
  3. AppFabric is **not installed** as a part of the .NET Framework 4 installation. It comes as a separate install delivered through a Windows Update package for Windows that you can download from <http://go.microsoft.com/fwlink/?LinkID=169435>
  4. To determine if AppFabric is installed open IIS Manager. If AppFabric is installed when you open the Default Web Site you will see **AppFabric** at the top of the features window.

Task 0 – Opening the Solution

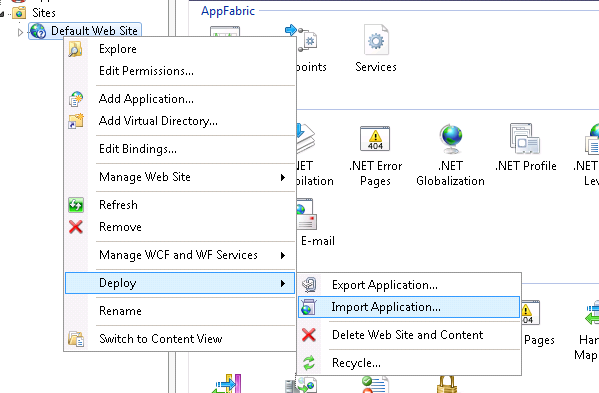
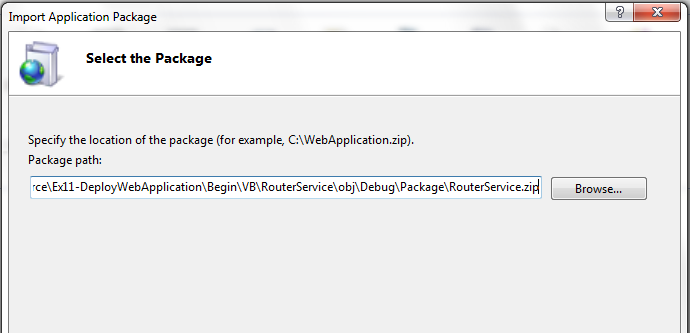
* 1. To begin this exercise you can use the solution you finished from Exercise 10. Alternatively, you can follow the following steps to begin with Exercise 11.
  2. Open the starting solution for Exercise 11 located under the **Source\Ex11-DeployWebApplication\Begin** **(choosing the folder that matches the language of your preference.)** Use it as the starting point for this exercise.
  3. Press **CTRL+SHIFT+B** to build the solution.

Task 1 – Creating the Deployment Package

* 1. In this task, you will use the Web Deployment feature inside Visual Studio to create a deployment package.
  2. In the Solution Explorer right-click the **RouterService** project and select **Package/Publish Settings** set the options as follows.
  3. Items to deploy using FTP, FPSE, File System or Web Deployment Tool Section:
     1. **Only files needed to run this application**
        1. 
        2. Figure 37
        3. Items to deploy
  4. Web Deployment Package Settings section:
     1. Create deployment package as a ZIP file
     2. Location: **obj\Debug\Package\RouterService.zip**
     3. IIS Web Site/Application name to be used on the destination server: **Default Web Site/RouterService**
        1. 
        2. Figure 38
        3. Web Deployment Package Settings
  5. Press **CTRL+SHIFT+S** to save all files – this will save your changes.
  6. In the **Solution Explorer** right-click the **RouterService** project, and select the **Build Deployment** **Package** action. You should see **Publish succeeded** as the status in the bottom left corner when the action has completed.

Task 2 – Deploying the Package

In this task, you will use the Web Deployment feature inside IIS Manager to deploy the package to the local web server.

* 1. Open Internet Information Services (IIS) Manager using the Start menu.
  2. Expand the **Server** node, expand **Sites**, and right-click the **Default Web Site**. Select **Deploy** / **Import Application**.
     1. 
     2. Figure 39
     3. Import Application in the IIS Manager
     4.  **Watch Out**
     5. If IIS was not enabled at the time of Visual Studio 2010 installation you may need to install ASP.NET into IIS by running the following command from an Administrator Visual Studio 2010 command prompt **aspnet\_regiis.exe -iru**
     6. If you don’t see a Deploy menu option it could mean that IIS Web Deployment Tool is not installed or may need to be reconfigured. To fix this you should do the following
     7. Download the IIS Web Deployment tool from <http://www.iis.net/extensions/WebDeploymentTool>
     8. Start the setup
     9. If setup shows you the Change / Repair / Remove screen select *Change* and install the IIS Manager UI module.
  3. For **Package path** browse to the location where the package was created. By default this was in the **obj\Debug\Package** folder for the **RouterService** project. Select the **RouterService.zip** file.
     1. ****
     2. Figure 40
     3. Select the package
  4. Click **Next** 3 times to use the defaults on the rest of the **Import Application Package** wizard pages. When asked if you want to run this application in the default .NET 4 application pool click **Yes**.
  5. Click **Finish** when the wizard has finished.

Task 3 – Changing the Client Application Configuration

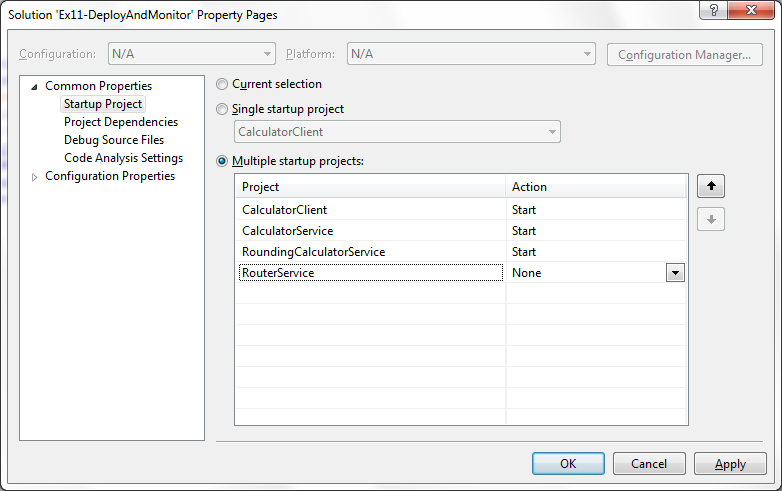
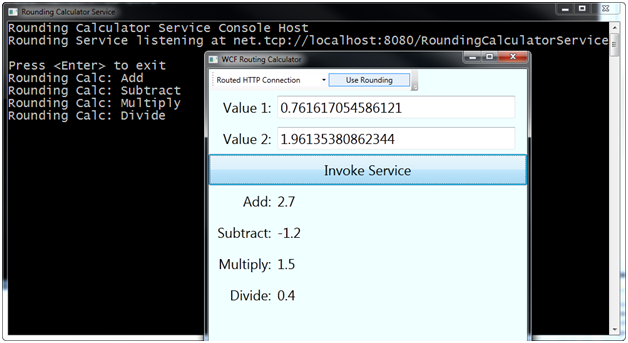
* 1. While you were developing the solution, you were using the Visual Studio Development Web Server. Now that you deploy your solution in the IIS environment, you are going to use a different endpoint for the Router Service.
  2. Open the **app.config** file from the **CalculatorClient** project and modify the **RouterService** endpoint address with the following configuration.
     1. *(Code Snippet - What is new in WCF4 Lab – RouterService endpoint XML)*
     2. App.config
     3. <endpoint **address="http://localhost/RouterService/Router.svc/general"**
     4. binding="basicHttpBinding"
     5. contract="CalculatorServiceReference.ICalculatorService"
     6. name="RouterService" />

</client>

# Next Step

* 1. Exercise 11: Verification

## Exercise 11: Verification

* 1. Right-click the **Ex11-DeployWebApplication** solution, and select **Set Startup Projects**.
  2. Select the Multiple Startup Projects radio button, and set **RouterService** project Action to **None**. Set the CalculatorClient project, CalculatorService project and RoundingCalculatorService project Action to Start.
     1. 
     2. Figure 41
     3. Set Multiple Startup projects
  3. Click OK to dismiss the solution properties dialog.
  4. Press **F5** to debug the solution.
  5. In the CalculatorClient select **Routed HTTP connection**.
  6. Click the **Use Rounding** button in the toolbar.
  7. Click the **Invoke Service** button.
  8. You should see values rounded to 1 decimal place.
     1. 
     2. Figure 42
     3. The routing service routed messages to the rounding calculator service
  9. Click the **Use Rounding** button to turn off rounding.
  10. Click the Invoke Service button again.
  11. This time the router will select the regular calculator service, and you will see that the values are not rounded to one decimal place.

# Next Step

* 1. Summary

Summary

During this lab, you have seen the three new feature areas of WCF including simplified configuration, service discovery and routing.

* 1.  **Feedback**
  2. What do you think of this lab? What do you think of the new Windows Communication Foundation 4? Your feedback is important; it will help us build the best product for you. Please take a moment to provide it. Send your comments to **wfwcfhol@microsoft.com**.