

WCF Architecture

Estimated time for completion: 60 minutes

Overview:

In this lab you will experiment with different options to implement services and clients.

Part 1 – Implement a Simple Service

In this part you will implement a service contract for a simple service.

Steps:

1. Open the starter project located at <before/part1>.
2. In the project PetShopContract, open the file Constants.cs. Notice that a constant for a service namespace is defined. Open OrderContract.cs and inspect the definition of the interface IOrderContract – notice the attributes [ServiceContract] and [OperationContract]. Your job is to implement a service that supports this contract.
3. A class library project for the service has already been added to the solution. The project is named "PetShopOrderService". Verify that references to System.ServiceModel.dll and to the PetShopContract project have been added and that the default .NET namespace for the project is set to DM.PetShop.
4. Add a new File OrderService.cs that implements a public class named OrderService in the project's default namespace. Implement IOrderContract in this class. IOrderContract.PlaceOrder should simply write a diagnostic message to the console. Annotate the class OrderService with the ServiceBehavior attribute specifying the service name "DM.PetShop.OrderService" and the service namespace Constants.SERVICE\_NAMESPACE.

Solution

The solution for this part of the lab can be found at [after\part1](after/part1).

Part 2 – Implement a Host Application

In this part you will implement a host application for OrderService.

1. Use the "Console Application" project template to add a new project to the solution. Name the Project ServiceHost1. To avoid confusions with files from projects you will add later, rename the file Program.cs of the new project to ServiceHost1App.cs. For this new project add a reference to System.ServiceModel.dll, the PetShopContracts project, and to the PetShopOrderService project.
2. Implement the service so that the following criteria are met
   1. A ServiceHost variable named host is used.
   2. The base address of the service is "http://localhost:9000/PetShop"
   3. The service exposes the IOrderContract contract via BasicHttpBinding. To call the service, the address "OrderService" (relative to the base address of the service host) is needed.
   4. Call Open on the host to start listening.
   5. After opening the service, add the following code to express the current state of the host and to ensure that the host process is kept alive to handle incoming requests.

Console.Title = String.Format("{0} is running ...",

host.Description.Endpoints[0].Address);

Console.ReadLine();

* 1. Make sure to clean up the host when the user has pressed enter to stop listening.

1. Set ServiceHost1 as the Startup Project of the Solution.
2. Try to start the application. If you are using Windows Vista or Windows Server 2008 as a host, you will likely see an exception, because the current user is not allowed to use port 9000 as a service endpoint. To solve this problem, perform the following steps.
   1. Open a command prompt using an administrator (right click and select “Run as Administrator”
   2. Run the netsh tool to register the endpoint and allow it to pass through to your service. The command line will look something like: netsh http add urlacl url=http://+:9000/PetShop/ user=DOMAIN\user
   3. You should replace the DOMAIN\user with your current machine name and logged on user name.

Solution

The solution for this part of the lab can be found at [after\part2](after/part2).

Part 3 – Implement a Client Application

In this part you will implement a client application that calls the OrderService.

1. Use the "Console Application" project template to add a new project to the solution. Name the Project Client1. For this new project add a reference to System.ServiceModel.dll and to the PetShopContracts project.
2. Rename the file Program.cs from the Client1 project to Client1App.cs. Open the file and add the following using declarations:

using System.ServiceModel;

using System.ServiceModel.Channels;

using IOrderContractChannelFactory =

System.ServiceModel.ChannelFactory<DM.PetShop.IOrderContract>;

1. Using the IOrderContractChannelFactory alias you have just defined create a new channel that can be used to call the OrderService at http://localhost:9000/PetShop/OrderService. To achieve this, you have to create a new instance of BasicHttpBinding and of EndpointAddress.
2. Use this channel to call PlaceOrder passing the string “1 parrot”.
3. To ensure proper cleanup of the channel, use the following code:

IClientChannel clientChannel = (IClientChannel)channel;

try

{

if (clientChannel.State != CommunicationState.Closed)

clientChannel.Close();

}

catch

{

clientChannel.Abort();

}

1. Start ServiceHost1 now. While ServiceHost1 is running, run the Client1 project. You should be able to call the service now.

Solution

The solution for this part of the lab can be found at [after\part3](after/part3).

Part 4 – Implement a Host Application that uses Configuration Files

In this part you will implement a second host application. In contrast to the first host application, this one uses configuration files instead of configuring the service host with code.

1. Use the "Console Application" project template to add a new project to the solution. Name the Project ServiceHost2. Rename Program.cs to ServiceHost2App.cs. For this new project add a reference to System.ServiceModel.dll, to the PetShopContracts project, and to the PetShopOrderService project. Set ServiceHost2 as the startup project for later tests.
2. Add an application configuration file (app.config) to the project.
3. To add the necessary configuration entries, you can either use the text editor (with IntelliSense) or a tool called the Service Configuration Editor. This tool could be started from Visual Studio by choosing the menu item “Edit WCF Configuration …” from the context menu of the configuration file. However, for a better understanding, we will use the text editor now:
   1. Add an element <system.serviceModel> as a child of the root element <configuration>.
   2. Add a child element <services> to the <system.serviceModel> element.
   3. Add a child element <service> to the <services> element. The <service> element needs the attributes name="DM.PetShop.OrderService".
   4. To the <service> element, add a child element <host> containing a child element named <baseAddresses> containing a child element named <add>. The <add> element needs the property baseAddress="http://localhost:9000/PetShop".
   5. Within the <service> element add a sibling to the <host> element. This element must be named <endpoint>. It needs the attributes address="OrderService", binding="basicHttpBinding" and contract="DM.PetShop.IOrderContract".
4. To implement your service host now, simply write the following code in the main function of ServiceHost2:

using (ServiceHost host = new ServiceHost(typeof(DM.PetShop.OrderService)))

{

host.Open();

Console.Title = String.Format("{0} is running ...",

host.Description.Endpoints[0].Address);

Console.ReadLine();

}

Console.Title = "OrderService closed.";

1. Start the ServiceHost2 now. While ServiceHost2 is running start the Client1 again. Client1 should be able to call the service hosted in ServiceHost2 now.

Solution

The solution for this part of the lab can be found at [after\part4](after/part4).

Part 5 – Adding Support for Metadata

In this part you will extend ServiceHost2 to support exporting metadata. Once you have done that, you will implement a second client.

1. To <system.serviceModel>, add a new child element <behaviors> containing an element named <serviceBehaviors>. Add a new child element <behavior> to the <serviceBehaviors> element. The <behavior> element should have the attribute name="OrderServiceBehavior". To specify that the behavior supports metadata via HTTP GET requests, add a child element <serviceMetadata httpGetEnabled="true"/> to <behavior>.
2. To apply this behavior to your service, add the attribute behaviorConfiguration="OrderServiceBehavior" to your <service> element (configuration/system.serviceModel/services/service).
3. Rebuild ServiceHost2 and start it without the debugger.
4. While ServiceHost2 is running, open <http://localhost:9000/PetShop> in Internet Explorer. Notice that an HTML page is returned by the service. Also notice that the HTML page contains a hyperlink to a WebService description document. Keep the ServiceHost2 process running.
5. Use the "Console Application" project template to add a new project to the solution. Name the project Client2. Rename the file Program.cs to Client2App.cs. In the Solution Explorer, open the context menu of the Client2 project and choose “Add Service Reference”. In the dialog, type “http://localhost:9000/PetShop” for the service URI and “PetShop” for the Service Reference Name. Click OK to generate the proxy.
6. Notice that a configuration file was generated by the wizard. Open this file and take a look at the <client> element containing the endpoint specification with the famous ABC of WCF. Also notice that a file named Reference.cs was generated. You can find this file in the Solution Explorer as a sub-node of ServiceReferences/OrderService.svcmap. This file contains a definition of the service contract (IOrderContract) and the service proxy (OrderServiceClient). The OrderServiceClient automatically uses the configurations specified in the configuration file.
7. In Client2’s Main function, implement the following:

PetShop.PetShopOrderContractClient proxy =

new PetShop.PetShopOrderContractClient();

try

{

proxy.PlaceOrder("2 hamsters, please");

}

finally

{

try

{

if (proxy.State != CommunicationState.Closed)

proxy.Close();

}

catch

{

proxy.Abort();

}

}

1. If ServiceHost2 is not running, start it now. While ServiceHost2 is running, start the Client2.

Solution

The solution for this part of the lab can be found at [after\part5](after/part5).