
# Introduction to WCF: Part 1

## 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. 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 has 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.

## Solution

The solution for this part of the lab can be found at 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 confusion 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 and to the PetShopOrderService project.
2. Implement the service so that the following criteria are met
    a. A `ServiceHost` variable named `host` is used.
    b. The base address of the service is "http://localhost:9000/PetShop"
    c. 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.
    d. Call `Open` on the host to start listening.
    e. 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.WriteLine("{0} is running ...",
    host.Description.Endpoints[0].Address);
Console.ReadLine();
```

    f. Make sure to clean up the host when the user has pressed enter to stop listening. (Hint: create the `ServiceHost` in a `using` block.)
3. Set ServiceHost1 as the Startup Project of the Solution.
4. 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.
    a. Open a command prompt using an administrator (right click and select "Run as Administrator"
    b. 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`
    c. 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.

## 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.
2. **Copy** OrderContract.cs in the PetShopOrderService project (right-click the file in the Solution Explorer and select Copy), then **paste** it into the Client1 project (right-click the project and select Paste).
3. 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 DM.PetShop;
```

4. Create a generic `ChannelFactory`, specifying `IOrderContract` for the type argument and passing a new `BasicHttpBinding` to the constructor.

```
ChannelFactory<IOrderContract> factory =
    new ChannelFactory<IOrderContract>(new BasicHttpBinding());
```

5. Using the `ChannelFactory<IOrderContract>` 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 `EndpointAddress`.

```
IOrderContract proxy = factory.CreateChannel(
    new EndpointAddress("http://localhost:9000/PetShop/OrderService"));
```

6. Use this channel to call `PlaceOrder` passing the string "1 parrot". To ensure proper cleanup of the channel, use the following code:

```
using ((IDisposable)proxy)
{
    proxy.PlaceOrder("1 parrot");
}
```

7. 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.

## Part 4 – Implement a Host Application that uses Configuration Files

*In this part you will change the host application to use a configuration file instead of configuring the service host with code.*

1. Add an application configuration file (app.config) to the project.
2. 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 selecting "WCF Configuration Editor" from the Tools menu. However, for a better understanding, we will use the text editor now:
    a. Add an element `<system.serviceModel>` as a child of the root element `<configuration>`.
    b. Add a child element `<services>` to the `<system.serviceModel>` element.
    c. Add a child element `<service>` to the `<services>` element. The `<service>` element needs the attributes `name="DM.PetShop.OrderService"`.
    d. 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"`.
    e. 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"`.
3. To implement your service host now, simply write the following code in the main function of ServiceHost1:

```
using (ServiceHost host = new ServiceHost(typeof(DM.PetShop.OrderService)))
{
    host.Open();
    Console.WriteLine("{0} is running ...",
        host.Description.Endpoints[0].Address);
    Console.ReadLine();
}
```

4. Start the ServiceHost1 now. While ServiceHost1 is running start the Client1 again. Client1 should be able to call the service hosted in ServiceHost1 now.

## Solution

The solution for this part of the lab can be found at after\part4.

## Part 5 – Using Data Contracts

*In this part of the lab, we will modify the service contract so that a data contract is used as a method argument.*

1. Add a class file called DataContract.cs to the PetShopOrderService project.
    a. Remove the DataContract class and replace it with a public `Order` class with the following properties: Product (string) and Quantity (int).
    b. Add a reference to System.Runtime.Serialization to the project. Also add a using directive for the namespace to DataContract.cs.
    c. Add a `DataContract` attribute to the `Order` class, specifying "urn:dm:wcf:labs:architecure" for the Namespace parameter.
    d. Add a `DataMember` attribute to both `Product` and `Quantity` properties.
2. Change the data type from `string` to `Order` for the `PlaceOrder` method of the `IOrderContract` interface in OrderContract.cs in the PetShopOrderService project.
    a. Do the same for the PlaceOrder method of the OrderContract class.
    b. Change the implementation to write the order product and quantity to the console.

```
public class OrderService : IOrderContract
{
    public void PlaceOrder(Order orderData)
    {
        Console.WriteLine("Order placed: {0} {1}",
            orderData.Quantity, orderData.Product);
    }
}
```

3. Copy DataContract.cs from the PetShopOrderService project to the Client1 project.
    a. Set a reference to System.Runtime.Serialization.
    b. Run ServiceHost1 to make sure the service is operating properly.
4. Open OrderContract.cs in Client1 project and change the orderData parameter of PlaceOrder from `string` to `Order`.
5. Last but not least, open Client1App.cs and change the call to PlaceOrder to accept a new Order.
    a. Run Client1. It should place an order.

```
proxy.PlaceOrder(new Order{ Quantity=1, Product="parrot" });
```

## Solution

There is a complete solution for this lab at [after/part5](after/part5).