Creating a WCF service without Visual Studio

Download sample code - 47 KB

WCF is the first step in building a truly service oriented application for you. It is the most important when working with distributed architecture. The sophisticated design of WCF made me think of using it always when I need any Web service to be installed in the server. Visual Studio is capable of creating its own configuration settings that helps in developing our application with ease. But what if you don't have Visual Studio? In this post, I am going to implement one of the most basic WCF service without using Visual Studio and show you how to interact with the service. Let's do it step by step.

Server Side

Steps to Create the Service definition (Contract)

1. Open Notepad and add namespace `System` and `System.ServiceModel`. We need `ServiceModel` to specify a WCF service.
2. For WCF service, we need to create an interface which will act as a proxy to the client. From the client, we need to replicate the proxy object and build the same interface again. After we declare the Interface, we mark the Interface with `ServiceContract`, and the method to be exposed to the outside using `OperationContract`.
3. Next, we create a concrete class for the same to define the class implementing the interface.

So our server is Ready.

Hide Copy Code

```
[ServiceContract]
public interface IOperationSimpleWCF
{
    [OperationContract]
    string MySimpleMethod(string inputText);
}

public class OperationSampleWCF : IOperationSimpleWCF
{
    public string MySimpleMethod(string inputText)
    {
        Console.WriteLine("Message Received : {0}", inputText);
        Console.WriteLine
    (OperationContext.Current.RequestContext.RequestMessage.ToString());
        return string.Format("Message from Server {0}", inputText);
    }
}
```

Steps to Host the Service (Address, Binding)

To host the service in the server, you need to know three inputs:

1. **Binding**: This indicates how the service will be hosted. For basic soap operation with no security, we need `HttpBinding`. We can also use other bindings as well.

2. **Address**: Represents the location to host the service. When the service is hosted, you can specify the qualified service path where the service will be hosted.
3. Host the service using `ServiceHost` and Open the connection.

Hide Copy Code

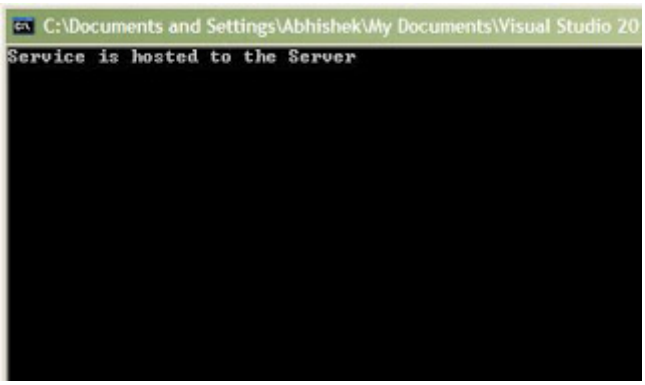
```
static void Main(string[] args)
{
    BasicHttpBinding binding = new BasicHttpBinding();

    Uri serviceUri = new Uri("http://localhost:8000");
    ServiceHost host = new ServiceHost(typeof(OperationSampleWCF), serviceUri);
    host.AddServiceEndpoint(typeof(IOperationSimpleWCF), binding, "OperationService");

    host.Open();

    Console.WriteLine("Service is hosted to the Server");
    Console.ReadLine();

    host.Close();
}
```

So the URI where the service is hosted is ***http://localhost:8000/OperationService***. You should notice that the call `AddServiceEndpoint` is used to give the qualified address for the service. If you don't specify `AddServiceEndpoint`, the service will be posted to root path of URI specified.

To Compile the Server Application

1. Open Command prompt and navigate to the location where you save the File with code.
2. To compile the program, use csc with `/r:` to reference the *System.ServiceModel.dll* using (Say the file we save for code is *ServerProgram.cs*) *csc /r:" C:\windows\Microsoft.NET\Framework\v3.0\Windows Communication Foundation\System.ServiceModel.dll" ServerProgram.cs*
3. You will get the program compiled, and produce the executable. Run the program.

After you run the program, it will show the following screen:



Hence the service is running and waiting for the client to receive requests.

Now it's time to create our client application.

Client Side

Steps to Create Client Application

1. First, you need to mimic the `ServiceContract` interface in the client. If you want to check the Request Body, you can use `OperationContext.Current.RequestContext.RequestMessage`, or you can directly parse the soap contract using and create the `Contract`class yourself.
2. Once we have created the `Contract`, we create an object of `ChannelFactory` which actually creates the interface between the client and the server. The `ChannelFactory` requires the `Binding`, and `EndPointAddress`. The endpoint address for our client will be ***http://localhost:8000/OperationService***.
3. Finally, we call `CreateChannel` to get the actual proxy object which is capable of calling the Remote server.
4. Compile the Client.

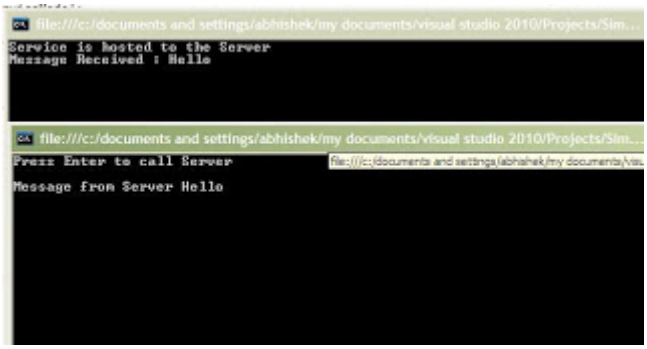
Hence when we call the server, the output will be shown on the client.

Hide Copy Code

```
static void Main(string[] args)
{
    Console.WriteLine("Press Enter to call Server");
    Console.ReadLine();


    BasicHttpBinding binding = new BasicHttpBinding();
    ChannelFactory<IOperationSimpleWCF> factory =
        new ChannelFactory<IOperationSimpleWCF>(binding,
                  new EndpointAddress("http://localhost:8000/OperationService"));

    IOperationSimpleWCF proxy = factory.CreateChannel();
    string methodFromServer = proxy.MySimpleMethod("Hello");

    Console.WriteLine(methodFromServer);

    Console.ReadLine();
}

[ServiceContract]
public interface IOperationSimpleWCF
{
    [OperationContract]
    string MySimpleMethod(string inputText);
}
```

Once we are done, we need to compile the client again using csc. *csc /r:" C:\windows\Microsoft.NET\Framework\v3.0\Windows Communication Foundation\System.ServiceModel.dll" ClientProgram.cs*.

Now open the two consoles side by side, and when you call the server from Client console, it updates the server console. Hence the server has been called properly.



If you open the call to `OperationContext.Current.RequestContext.RequestMessage.ToString()`, you can see the Soap envelope that is passed as request to the server from the client. It will look like:

Hide Copy Code

```
<s:Envelope xmlns:s=\"http://schemas.xmlsoap.org/soap/envelope/\">
 <s:Header>
    <To s:mustUnderstand=\"1\"
    xmlns=\"http://schemas.microsoft.com/ws/2005/05/addressing/none\">
    http://localhost:8000/OperationService</To>
    <Action s:mustUnderstand=\"1\"
    xmlns=\"http://schemas.microsoft.com/ws/2005/05/addressing/none\">
    http://tempuri.org/IOperationSimpleWCF/MySimpleMethod</Action>
  </s:Header>
  <s:Body>
    <MySimpleMethod xmlns=\"http://tempuri.org/\">
      <inputText>Hello</inputText>
    </MySimpleMethod>
  </s:Body>
</s:Envelope>
```

Now if you change the `Binding` from `BasicHttpBinding` to `WSHttpBinding`, the program outputs the same while the `Request` body changes with all the support of security features and the message passed within the Soap envelope will automatically be encrypted.

Hide Shrink Copy Code

```
<s:Envelope xmlns:s=\"http://www.w3.org/2003/05/soap-envelope\"
   xmlns:a=\"http://www.w3.org/2005/08/addressing\"
   xmlns:u=\"http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-
    wssecurity-utility-1.0.xsd\">
  <s:Header>
    <a:Action s:mustUnderstand=\"1\" u:Id=\"_2\">
    http://tempuri.org/IOperationSimpleWCF/MySimpleMethod</a:Action>
    <a:MessageID u:Id=\"_3\">urn:uuid:b3edb9bc-6043-4c05-b540-794e5d61d505</a:MessageID>
    <a:ReplyTo u:Id=\"_4\">
      <a:Address>http://www.w3.org/2005/08/addressing/anonymous</a:Address>
    </a:ReplyTo>
    <a:To s:mustUnderstand=\"1\" u:Id=\"_5\">http://localhost:8000/OperationService</a:To>
    <o:Security s:mustUnderstand=\"1\" xmlns:o=\"http://docs.oasis-open.org/
    wss/2004/01/oasis-200401-wss-wssecurity-secext-1.0.xsd\">
      <u:Timestamp u:Id=\"uuid-ce693b53-159f-492e-b79a-f904abb878d8-11\">
        <u:Created>2010-09-14T23:26:22.015Z</u:Created>
        <u:Expires>2010-09-14T23:31:22.015Z</u:Expires>
      </u:Timestamp>
      <
c:SecurityContextToken u:Id=\"uuid-ff9818d2-87c0-4daa-ba54-ce07e283ca38-4\"
    xmlns:c=\"http://schemas.xmlsoap.org/ws/2005/02/sc\">
        <c:Identifier>urn:uuid:2a3567f3-73e7-4f8d-9a5d-f2be8defae2f</c:Identifier>
      </c:SecurityContextToken>
      <c:DerivedKeyToken u:Id=\"uuid-ce693b53-159f-492e-b79a-f904abb878d8-9\"
    xmlns:c=\"http://schemas.xmlsoap.org/ws/2005/02/sc\">
        <o:SecurityTokenReference>
          <o:Reference ValueType=\http://schemas.xmlsoap.org/ws/2005/02/sc/sct\
    URI=\"#uuid-ff9818d2-87c0-4daa-ba54-ce07e283ca38-4\" />
        </o:SecurityTokenReference>
        <c:Offset>0</c:Offset>
        <c:Length>24</c:Length>
        <c:Nonce>zIN3JJx6Ce3YudbqTPApWQ==</c:Nonce>
      </c:DerivedKeyToken>
      <c:DerivedKeyToken u:Id=\"uuid-ce693b53-159f-492e-b79a-f904abb878d8-10\"
    xmlns:c=\"http://schemas.xmlsoap.org/ws/2005/02/sc\">
        <o:SecurityTokenReference>
          <o:Reference ValueType=\http://schemas.xmlsoap.org/ws/2005/02/sc/sct\ URI=\"#u
uid-ff9818d2-87c0-4daa-ba54-ce07e283ca38-4\" />
        </o:SecurityTokenReference>
        <c:Nonce>+ie7ja4mY4bPB5iwgiZHCg==</c:Nonce>
      </c:DerivedKeyToken>
      <e:ReferenceList xmlns:e=\"http://www.w3.org/2001/04/xmlenc#\">
        <e:DataReference URI=\"#_1\" />
        <e:DataReference URI=\"#_6\" />
      </e:ReferenceList>
      <e:EncryptedData Id=\"_6\" Type=\http://www.w3.org/2001/04/xmlenc#Element\
```

```
    xmlns:e=\"http://www.w3.org/2001/04/xmlenc#\">
        <e:EncryptionMethod Algorithm=\"http://www.w3.org/2001/04/xmlenc#aes256-cbc\" />
        <KeyInfo xmlns=\"http://www.w3.org/2000/09/xmldsig#\">
          <o:SecurityTokenReference>
            <o:Reference ValueType=\http://schemas.xmlsoap.org/ws/2005/02/sc/dk\
    URI=\"#uuid-ce693b53-159f-492e-b79a-f904abb878d8-10\" />
          </o:SecurityTokenReference>
        </KeyInfo>
        <e:CipherData>
          <e:CipherValue>Mv6dlbn+FvqRk3Xq6uwsomzyqW3VJ9RrfxhhA0mzrUgA854c4JG8Tslyw06QhGMi
              BJ/QZBZ0ReWbKUWgqciewW/m89H82zOpD5RlJVSXI+XrGyEwT9uvfye28V1lNgsKfJ04GbWNs1
              vvVFsBeytCt6XZORUiCrK7i+uUDPM0nSaNQ9ojBfhC9RJd9ri1vg+u5K5KG/NVyE4Sse72Phg7J
              d4oZDqbyQneSVpKiVrjvWrw3FOgSyGdQuITCblPYJATWXXyo5ELeNjm+gqVFPuLujoHa3Cuzdod
              JW2C5RrEipXdsfsT2tw2/Vv52SujKAzm01SktUG/V7Z6S4lLXc7fH7oGxK4aKy3gwJrApbqQtwa
              LaNXYwU+sSV+umSv2fmtQXrPH8fH/cq+KcSoCiZptJC3npBPjAD94oVNG4UvuZ5P9py5XV6nJ1
              mFKcZWRXqJSJzhWuGUkGYQja78UsXKDmFcYrQ7or8REdMuywKoIpqTA1PGD76EZJjgYQbwcfyAu
              3quhEJBhm1uUBB+Tg7ealiqkFRLzi0XPEbouMsFLwiYBhHF163ultLVWXUt6i6DdbPccX5DwRuv
              JAp8VU3h8Z6pOfNiwmCMA/OnLEvG7GCBT7BLwfwU9z4TZiIo9ybpdRZ2gBoS41niELH0Mx6jOZ/3
              kLObSTXHC+PQJZPcBM/+HXqniq7q4+3lS5eTAhlBZuAe07IA5GjTD/aL2x79QQxKoqCZy1OG00HF
              uE0fT6fb1lLQqX6Dd6XfxF5YCGjZKKyK043G95sjGmRXggZABIkgIbFVfcZB43cWvDTvRMvJ7h87
              p5KLCBoyWfVuhlAWXtiSoeI8FR9EZ7LVKPktlCS0N3pclN3m7YsN+xGOR9HFxWkq16KQpUqLxiye
              Sf1YazIXc0jnFTgxr8h60f2mR5S5YTXt3douotytWqLv27NjuCBUdFGaCWEbI+azwAl2cm1zdxt
              fqg+ppin/jqVxDgENC/ko0hPpQg7S4yTSxEBOlSavAcRUGsou8eBdSTi4uSea5JzNUHKzJgpRp8
              LUui7Eiw+fwXWj7mciYZfB0T3H9Pz/gbrrduNFIqZSGJ1YcSjX36lxIrxM1
              tGxqHFBeUh53ioOOXkSv49oZIkvXWrA3dnI63UXgm6hjhJnvdZruRqgIbtsv+Ui14WTlG4ojM
              McHN0ArlFGdgHG0iSiFYgNWeHYugFSXUhY950ZCb/xXZl8FywNe6ZjysA7pbXP+/rImDgFwOns
              UXfothJnFrNwLJIOXx3M3qvttxBT9Ha0qGb4yLJevx3kNS6cz0LX7Ol5O2GeKqj9eTccBVUa1
              zsEHeuFxV007WPfJuCCsPeQy+EbNLYTcv95FjdU/8T1lUfsflab6TBiY1DNui8MNVdJbP8Rqqm
              v9SWn5Db9dvMZPR/Xc+RK9V2RGP1i8v5VWBkMA4q6pssc5HGb/z3LbXNXTBQJpOnMtJuJrM3TGi
              4W+Y3bE9gf94yoXaSsR5i7cc63R2tW7GSuKXZjnUXTod++a/TFXUJGmvIt1XtqcpEijaEtxWNrC
              sSIF59YHMgmD7UY6hq2789n5q8sqXVLZbYMtH/YbbK1jjNsMiMIT3HqdBPfiTCzqstwm58GRbVh
              ZdpTSWfm3LfdPiy2HAhK8COITWa2MrwcW65NbvqUvWoWw4Mm1K/gzEW15KAVyQtxHqEoCMmqbTS
              FTzqeaRYhtKKQ54JR1vcHJfl/WNN+bjAqfj3f3IZCZjU21amvwIMuapKcnn1d5EpaoyLvnXGntM
              fQhTC7y5NhBbEtZcUaoT56WbUa0cusfcYWi5RQmCgWVMgbSBhEHyOFsW7h1ETwmRMblNZh2RGbn
              zgSw3wl4EjESrZ6aMLXCZjrVvrPMT9z23/UwWKXno+ofCJW3lqiyssaC5QMUEwZBk887+Ebecxq
              oRxeejfNt9yUb3LY8Riv5iSnpFU8WhtAJac8rVCu3btO49fUUrFs56VHjXni2U4WIrvyfbF+HOn
              vJd8TBDc9KZBiq9Quyrl/VMJGO53wUXEEkF0HUtt6e1tohsGUhwhYOVEOv5seKvl2Tn7yZqoV
              9n/ICyOWckmSbb2328QJvj3kGU7dU89omPD9/4VCuTarPRRJKzG0zMdkPO1GADvoZve5kGyu739
              ARCzt6QBKM2x68Gtq/DipgPcb15nEoLzfpFBW5VQbHzxznm9igmkKkM/y6qxkElPlK2DidxRLU
              pkpngavLnGbmTiFAg5LaT/1o57q7JIx+A9VTzbqvjHWND6UI30IcY4/24xnVg2LKnURrSm6S1kY
              lWNyRHYU/j4Es5KLQNY88Dmypc5D0BDR8UFWALQlnH8elYwLHZH2mo6sm40/60YrRBUlL0ifi
              Hb6jNrx2QlqIehuvtCfwl2PKc1uPTWlJC69Ymvfgo79d1eDmsguJ4k600bBepdjQ789ykdWBMCJ
              g8qYba+hvpFvS20MlgfzNp7QRfkx6LmZPKgePrpen94n6jaWGn7+5XpkByaRESPZxuOnQ2eZZgQ
              pfZtgnhzcNka6xGrUSjJpSMYdYL42Yqoe4W4/DACgzsbHks1WfOuKLF0K9yayzTriWAheJNXeth
              1tGFvc2lI8xna00cwCHhm1OLFU0yFz/2hNdWVKJk15Tcrn3YS66OBnmHUbRcNBV57j69/fg5XQv
              21ku/rBTLCZE7qw2ckgvt1b9wRnOlExUGQekMnFDndyASfwhBUZtS14Uit3w/Tqijp/oBuGD6nE
              jufNXYCqxiMPStm+oz+us/I6+uGGyQbfVTG7IUuBXpERvWrroJU6hQf+odpDo/jFrWU9QCSbNa
              QRG</e:CipherValue>
        </e:CipherData>
      </e:EncryptedData>
    </o:Security>
  </s:Header>
  <s:Body u:Id=\"_0\">
    <MySimpleMethod xmlns=\"http://tempuri.org/\">
      <inputText>Hello</inputText>
    </MySimpleMethod>
  </s:Body>
</s:Envelope>
```

Similarly, if you have just changed the `Binding` and the address to `NetTcpBinding` and *net:tcp://localhost:8000*, it will work very similarly.

In this way, you can create your own WCF service and call it from the client without even using Visual Studio.

I hope this will help you. Thanks for reading.

License