Host your WCF service with multiple host environment using multiple protocol

By **Faris Y. Elias SHOMOU**, 15 Dec 2008

Download source code - 111.21 KB

Introduction

In the world of distributed application, many technologies has been introduced, implemented and used like DCOM, CORBA,...etc.

And after that, a new concept has been adapted,it was the web services. Web service is simple to develop and implement without paying attention to firewall issues and as a result, communication between systems has been simplified.

The problem while developing a web service is you need to host it inside a web server like IIS or Apache using http or wshttp protocols, and this is the only option you have.
WCF has solved this issue, by providing the possibility to host your service in a different application process using also various protocols.

In this article I'll show how you can achieve this.

The sample code includes a simple WCF service, console windows host, windows service host, IIS6 host, II7 host and client console windows application built in VS 2008 Standard edition and .NET 3.0.

Background

As we know if you adopt web services architecture for your system, you need the flowing:
- create a service like a component library using [WebService] and [WebMethod] attributes
- host it inside a web server like IIS
- generate a proxy (interface or contract) from the WSDL
- distribute this proxy to clients application who needs to call and use this web service.

Multiple Host and Protocol Support with WCF

Microsoft has introduced the WCF concept in order to make distributed application development and deployment simple.

Hosting Environment	Supported protocol
Windows console and form application	HTTP,net.tcp,net.pipe,net.msmq
Windows service application (formerly known as NT services)	HTTP,net.tcp,net.pipe,net.msmq
Web server IIS6	http, wshttp
Web server IIS7 - Windows Process Activation Service (WAS)	HTTP,net.tcp,net.pipe,net.msmq

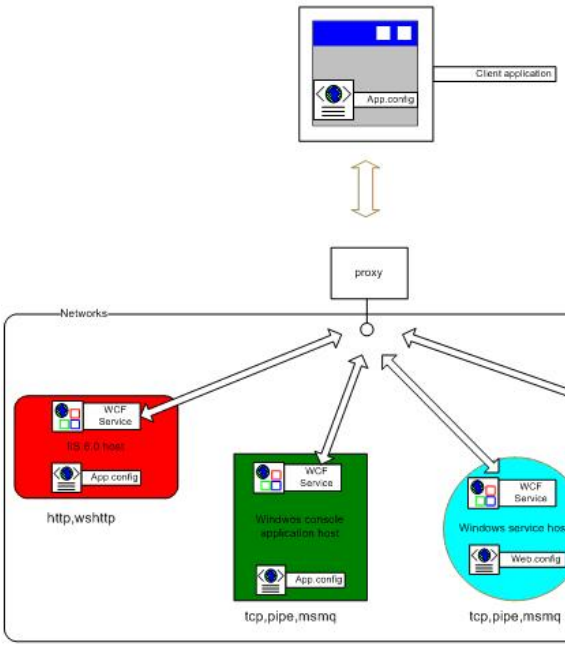
About Article

Exploring multiple hosting environment for WCF service

Type	**Article**
Licence	**CPOL**
First Posted	**10 Nov 2008**
Views	**146,937**
Downloads	**3,120**
Bookmarked	**132 times**

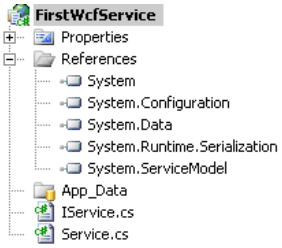
IIS6 .NET3.0 IIS7 VS2008 C# IIS

Print Email



Create a test service

Here I have created a very simple service called "FirstWcfService.Service" as in the following listing -1 :



Collapse | Copy Code

```
// Listing -1 IService.cs file
using System;
using System.Runtime.Serialization;
using System.ServiceModel;
namespace FirstWcfService
{
    [ServiceContract]
    public interface IService
    {

        [OperationContract]
        string Hello();
    }
}
// Service.cs file
using System;
namespace FirstWcfService
{
    public class Service : IService
    {
        public string Hello()
        {
            return ("Hello WCF");
        }
    }
}
```

As we can see, this service contains only on operation contract (Hello), this operation will return simple string "Hello WCF".

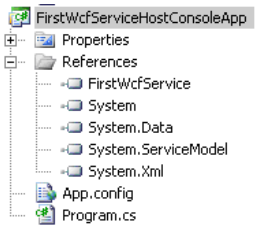
Hosting our service

As I have mentioned in the begging of my article, with the arrival of WCF, now we have multiple options to host this service.

In each hosting environment, we need to provide an endpoint for this service and also we need to reference it, in order the client application can reach this service.
What does that mean practically, it means we have to create a configuration file for our hosted service.

Option 1 – windows console host application

1 - I created here a classic console application (picture 1) and then I hosted my service as in the following listing – 2:



Collapse | Copy Code

```
//Listing - 2 Program.cs
using System;
using System.Collections.Generic;
using System.Text;
using System.ServiceModel;
using System.ServiceModel.Description;
namespace FirstWcfServiceHostConsoleApp
{
    class Program
    {
        static void Main(string[] args)
        {

using (ServiceHost host = new ServiceHost(typeof(FirstWcfService.Service)))
            {
                host.Open();
                Console.WriteLine("Press <Enter> to terminate the Host application.");
                Console.ReadLine();
            }

        }
    }
}
```

In order to host a WCF service inside a console application we need a minimum of work. Let us examine the code in the above listing:
This code ServiceHost host = new ServiceHost(typeof(FirstWcfService.Service)
creates an instane of our service.
This line host.Open() makes the service ready for access inside the hosting enviroemnt.

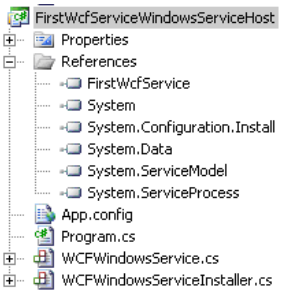
2- Secondly I have created a configuration file with a minimum configuration options as in the following listing - 3:

Collapse | Copy Code

```
//Listing - 3 App.config
<?xml version="1.0"?>
<configuration>
  <system.serviceModel>
    <services>
      <service name="FirstWcfService.Service" behaviorConfiguration="ServiceBehavior">
        <!-- Service Endpoints -->
        <endpoint address="FirstWcfService" binding="netTcpBinding"
            contract="FirstWcfService.IService"/>
        <!-- This Endpoint is used for genertaing the proxy for the client -->
 <!-- To avoid disclosing metadata information, set the value below to false and
       remove the metadata endpoint above before deployment -->
        <endpoint address="mex" contract="IMetadataExchange" binding="mexTcpBinding" />
        <host>
          <baseAddresses>
            <add baseAddress="net.tcp://localhost:9100/"/>
          </baseAddresses>
        </host>
      </service>
    </services>
    <behaviors>
      <serviceBehaviors>
        <behavior name="ServiceBehavior">
          <serviceMetadata httpGetEnabled="false"/>
        </behavior>
      </serviceBehaviors>
    </behaviors>
  </system.serviceModel>
</configuration>
```

Option 2 – windows service host application

1 - I created here a classic windows service application and then I hosted my service as in the following listing – 4:



Collapse | Copy Code

```
//Listing - 4, Program.cs
using System;
using System.ServiceProcess;
namespace Windows_Service
{
    static class Program
    {
        static void Main()
        {
            ServiceBase[] ServicesToRun;
            ServicesToRun = new ServiceBase[] { new WCFWindowsService() };
            ServiceBase.Run(ServicesToRun);
        }
    }
}
//Listing - 4, WCFWindowsService.cs
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Diagnostics;
using System.ServiceProcess;
using System.ServiceModel;
namespace Windows_Service
{
    public partial class WCFWindowsService : ServiceBase
    {
       ServiceHost m_serviceHost;

protected override void OnStart(string[] args)
        {
            m_serviceHost = new ServiceHost(typeof(FirstWcfService.Service));
            m_serviceHost.Open();
        }

        protected override void OnStop()
        {
            if (m_serviceHost != null)
            {
                m_serviceHost.Close();
            }
            m_serviceHost = null;
        }
    }
}
```

In order to host a WCF service inside a windows service application, we need also a minimum of work by implementing the followings functions:
void OnStart(string[] args) and protected override void OnStop()
Let us examine the code in the above listing:
This code:
m_serviceHost = new ServiceHost(typeof(FirstWcfService.Service));
m_serviceHost.Open();
creates an instane of our service and then makes the service ready for access inside the hosting enviroment.

2 - To make this application working like a windows service, we should install it as in Listing – 5. This file is not related to WCF implementation, so I'm not going to explain what does this code does.

Collapse | Copy Code

```
//Listing - 5, WCFWindowsServiceInstaller.cs
using System;
using System.Collections;
using System.Collections.Generic;
using System.ComponentModel;
using System.Configuration.Install;
using System.ServiceProcess;
// To install or uninstall this Windows service via cmd:
//
//C:\Program Files\Microsoft Visual Studio 9.0\VC>installutil.exe /u  yourpath/WCFWindowsService.exe
//C:\Program Files\Microsoft Visual Studio 9.0\VC>installutil.exe     yourpath/WCFWindowsService.exe
namespace Windows_Service
{
    [RunInstaller(true)]
    public partial class WCFWindowsServiceInstaller : Installer
    {
        public WCFWindowsServiceInstaller()
        {
            InitializeComponent();
            ServiceProcessInstaller processInstaller = new ServiceProcessInstaller();
            ServiceInstaller serviceInstaller = new ServiceInstaller();
            processInstaller.Account = ServiceAccount.LocalSystem;
            serviceInstaller.DisplayName = "WCF_WindowsService";
            serviceInstaller.Description = "WCF_WindowsService.";
            serviceInstaller.ServiceName = "WCF_WindowsService";
            serviceInstaller.StartType = ServiceStartMode.Manual;
            Installers.Add(processInstaller);
            Installers.Add(serviceInstaller);
        }
    }
}
```

3 - I have created a configuration file with a minimum configuration options as in the following listing - 6:

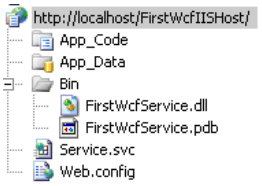
Collapse | Copy Code

```
 //Listing - 6, App.config
<?xml version="1.0" encoding="utf-8" ?>
<configuration>
  <system.serviceModel>
    <services>
      <service name="FirstWcfService.Service" behaviorConfiguration="ServiceBehavior">
        <endpoint address="FirstWcfService" contract="FirstWcfService.IService"
             binding="netTcpBinding" />
        <!-- This Endpoint is used for genertaing the proxy for the client -->
        <endpoint address="mex" contract="IMetadataExchange" binding="mexTcpBinding" />
```

```
        <host>
          <baseAddresses>
            <add baseAddress="net.tcp://localhost:9000"/>
          </baseAddresses>
        </host>
      </service>
    </services>
    <behaviors>
      <serviceBehaviors>
        <behavior name="ServiceBehavior">
          <serviceMetadata httpGetEnabled="false"/>
        </behavior>
      </serviceBehaviors>
    </behaviors>
  </system.serviceModel>
</configuration>
```

Option 3 – IIS6 host application

1 - I created here a simple WCF Service web application and then I have referenced FirstWcfService in order to host it as in the following listing – 7:



Collapse | Copy Code

```
 //Listing - 7, file Service.svc
<%@ ServiceHost Language="C#" Debug="true" Service="FirstWcfService.Service"%>
```

As see can see her, when we host a WCF service inside IIS, we do not need to create a host service and open it as we have done in the console and service windows application hosting, because IIS is responsible for creating the service and make it listening to clients calls, all we need just the above code inside a .svc file.

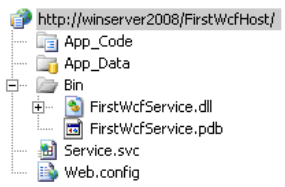
2- I have created a configuration file with a minimum configuration options as in the following listing -8:

Collapse | Copy Code

```
//Listing - 8, Web.config
<?xml version="1.0"?>
<configuration>
  <system.serviceModel>
    <services>
      <service name="FirstWcfService.Service" behaviorConfiguration="ServiceBehavior">
        <!-- Service Endpoints -->
        <endpoint address="" binding="basicHttpBinding"
           contract="FirstWcfService.IService"/>
       <!-- This Endpoint is used for genertaing the proxy for the client -->
<endpoint address="mex" binding="mexHttpBinding" contract="IMetadataExchange"/>
      </service>
    </services>
    <behaviors>
      <serviceBehaviors>
        <behavior name="ServiceBehavior">
          <!-- To avoid disclosing metadata information, set the value below to
              false and remove the metadata endpoint above before deployment -->
          <serviceMetadata httpGetEnabled="true"/>
          <!-- To receive exception details in faults for debugging purposes,
              set the value below to true.  Set to false before deployment to avoid
              disclosing exception information -->
          <serviceDebug includeExceptionDetailInFaults="false"/>
        </behavior>
      </serviceBehaviors>
    </behaviors>
  </system.serviceModel>
</configuration>
```

Option 4 – IIS7 (WAS) host application

1 - I created here a simple WCF Service web application using windows server 2008 (IIS7) and then I have referenced FirstWcfService in order to host it as in the following listing – 9:



Collapse | Copy Code

```
//Listing - 9 file Service.svc
<%@ ServiceHost Language="C#" Debug="true" Service="FirstWcfService.Service"%>
```

2- I have created a configuration file with a minimum configuration options as in the following listing -10:

Collapse | Copy Code

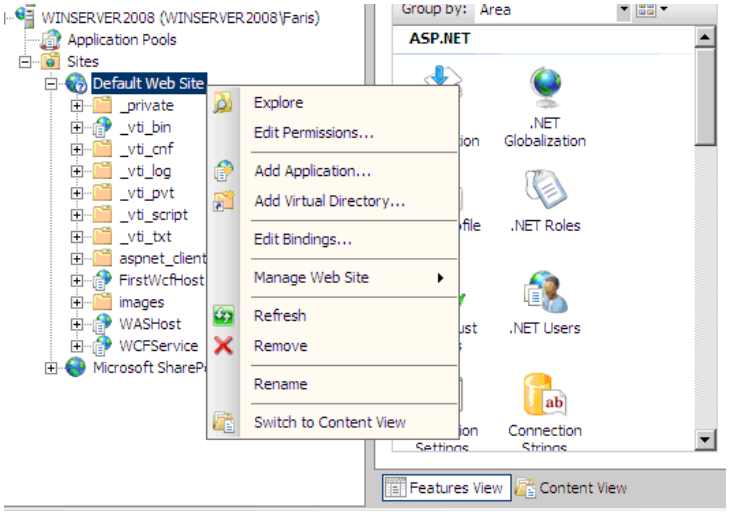
```
//Listing - 10, Web.config
```

```
<?xml version="1.0"?>
<configuration>
  <system.serviceModel>
    <services>
      <service name="FirstWcfService.Service" behaviorConfiguration="ServiceBehavior">
        <!-- Service Endpoints -->
        <endpoint address="" binding="netTcpBinding" contract="FirstWcfService.IService"
            bindingConfiguration="tcpbinding"/>
        <endpoint address="mextcp" binding="mexTcpBinding" contract="IMetadataExchange"/>
      </service>
    </services>
    <behaviors>
      <serviceBehaviors>
        <behavior name="ServiceBehavior">
          <!-- To avoid disclosing metadata information, set the value below to false
              and remove the metadata endpoint above before deployment -->
          <serviceMetadata httpGetEnabled="true"/>
          <!-- To receive exception details in faults for debugging purposes,
               set the value below to true.  Set to false before deployment to avoid
               disclosing exception information -->
          <serviceDebug includeExceptionDetailInFaults="false"/>
        </behavior>
      </serviceBehaviors>
    </behaviors>
    <bindings>
      <netTcpBinding>
        <binding name="tcpbinding">
          <!--<security mode="None"></security>-->
          <security mode="Transport">
            <transport clientCredentialType="Windows" protectionLevel="EncryptAndSign"/>
            <message clientCredentialType="Windows"/>
          </security>
        </binding>
      </netTcpBinding>
    </bindings>
  </system.serviceModel>
</configuration>
```

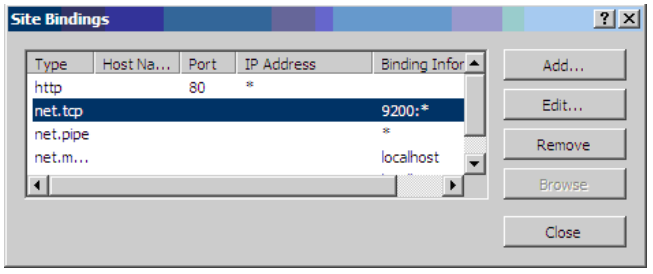
NOTE: As you can see here, I have done the same implementation as I did for option 3 while hosting my SCF service inside II6, the only difference is, I have configured my service to be callable using "net.Tcp" protocol inside IIS7 (WAS) instead of classic "basicHttp" protocol. I'm not going to explain the new architecture of IIS7, because this topic is out of scope in this article.

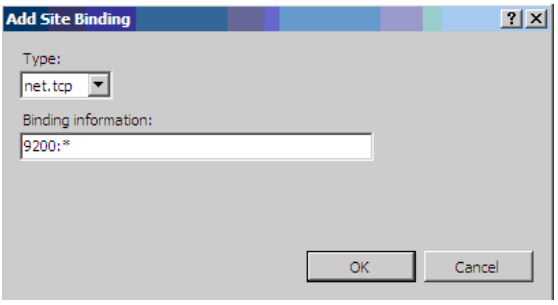
3- When you create a WCF service inside IIS7, by default this service has only http and wshttp enabled. I have configured my service to be callable using net.Tcp protocol as in the following steps:

A - in order to enable a new protocol like net.Tcp, we need to add it to our default web site tree via IIS Manager by clinking on "Edit Bindings"

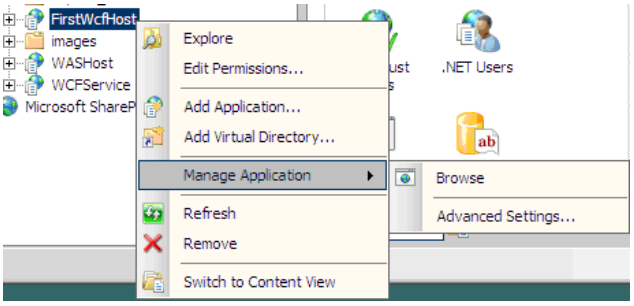


B - Add the new protocol with the desired port also, I added 9200

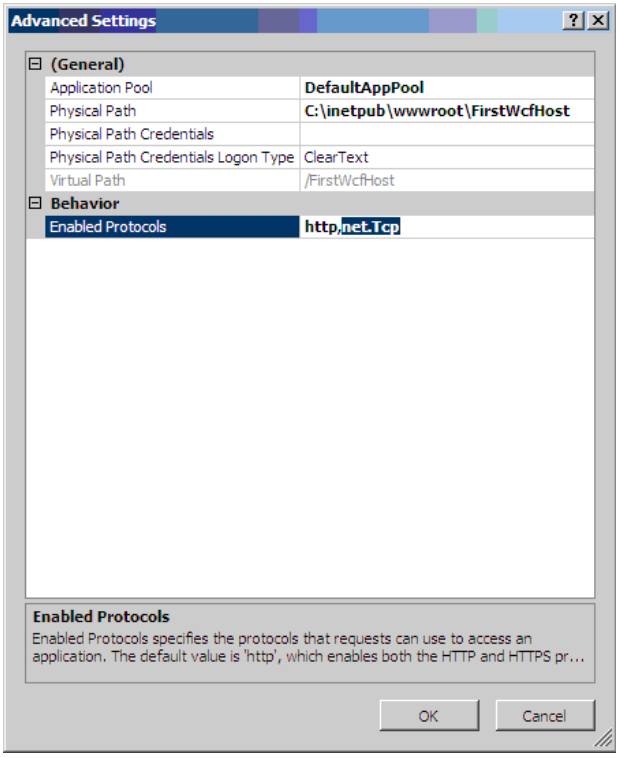




C – Always from IIS Manager, add the net.Tcp to your WCf service by clicking on advanced settings:

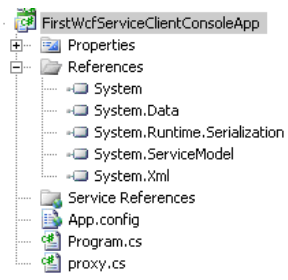


D – write your desired protocol through Enabled protocols settings :



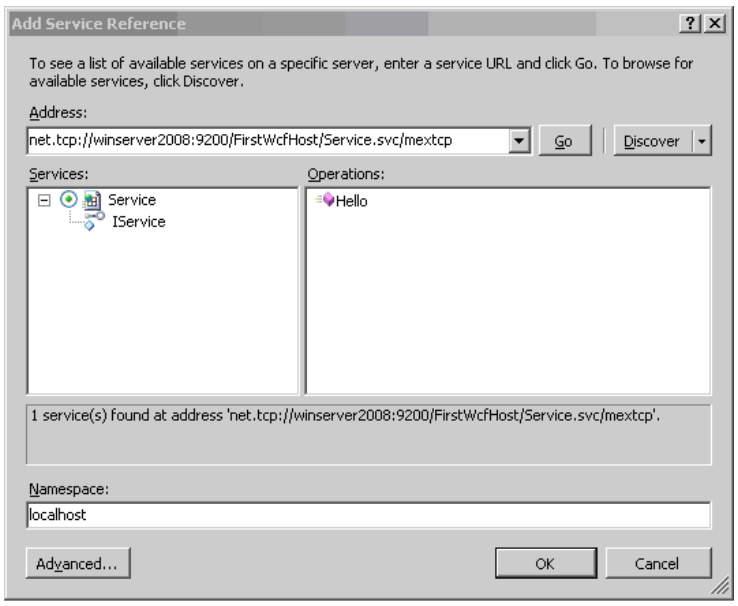
Create the client application

I created a simple client console application in order to show you how we can call our WCF service using multiple hosting options.
My client application looks like this:



Let's examine this application:
1 – We need co create a proxy, to create a proxy we have tow possibilities:

Either from Visual studio by adding a service reference as below :



Or by using the SvcUtil.exe utility, as we have multiple hosting options, we can observe here that we need just to use one of the following created proxy :

• If you create the proxy from a service hosted inside a console application you make: SvcUtil.exe net.tcp://localhost:9100/mex /out:path\proxy.cs /n:*,localhost

• If you create the proxy from a service hosted inside a windows service application you make SvcUtil.exe net.tcp://localhost:9000/mex /out:path\proxy.cs /n:*,localhost

• If you create the proxy from a service hosted inside an IIS 6 you make SvcUtil.exe http://localhost/FirstWcfIISHost/Service.svc /out:path\proxy.cs /n:*,localhost

• If you create the proxy from a service hosted inside an IIS7 using tcp protocol you make SvcUtil.exe net.tcp://winserver2008:9200/FirstWcfHost/Service.svc/ out:path\proxy.cs /n:*,localhost

When you create your proxy for the first time, you can use the same proxy to call the same service inside different hosting environments.
Finally our proxy will be as the following listing – 11

Collapse | Copy Code

```
// Listing - 11 proxy.cs file
namespace localhost
{


    [System.CodeDom.Compiler.GeneratedCodeAttribute("System.ServiceModel", "3.0.0.0")]
    [System.ServiceModel.ServiceContractAttribute(ConfigurationName="localhost.IService")]
    public interface IService
    {

        [System.ServiceModel.OperationContractAttribute(Action =
            "<a href="%22http://tempuri.org/IService/Hello%22">http://tempuri.org/IService/Hello</a>",
            ReplyAction = "<a
href="%22http://tempuri.org/IService/HelloResponse%22">http://tempuri.org/IService/HelloResponse</a>")
]
        string Hello();
    }

    [System.CodeDom.Compiler.GeneratedCodeAttribute("System.ServiceModel", "3.0.0.0")]
    public interface IServiceChannel : localhost.IService,
        System.ServiceModel.IClientChannel
    {
    }

    [System.Diagnostics.DebuggerStepThroughAttribute()]
    [System.CodeDom.Compiler.GeneratedCodeAttribute("System.ServiceModel", "3.0.0.0")]
    public partial class ServiceClient :
        System.ServiceModel.ClientBase<localhost.IService>, localhost.IService
    {

        public ServiceClient()
        {
        }

        public ServiceClient(string endpointConfigurationName) :
                base(endpointConfigurationName)
        {
        }

        public ServiceClient(string endpointConfigurationName, string remoteAddress) :
                base(endpointConfigurationName, remoteAddress)
        {
        }

        public ServiceClient(string endpointConfigurationName,
            System.ServiceModel.EndpointAddress remoteAddress) :
                base(endpointConfigurationName, remoteAddress)
        {
        }

        public ServiceClient(System.ServiceModel.Channels.Binding binding,
            System.ServiceModel.EndpointAddress remoteAddress) :
                base(binding, remoteAddress)
        {
```

```
        }

        public string Hello()
        {
            return base.Channel.Hello();
        }
    }
}
```

2- I have created a configuration file with a minimum configuration options as in the following listing -12:

Collapse | Copy Code

```
//Listing - 12, App.config
<?xml version="1.0" encoding="utf-8" ?>
<configuration>
  <system.serviceModel>
    <client>
      <!-- calling the WCF service which is hosted inside windows console application -->
      <endpoint address="net.tcp://localhost:9100/FirstWcfService" binding="netTcpBinding"
          contract="localhost.IService" name="Windows_Console_Host_TcpBinding">
      </endpoint>
      <!-- calling the WCF service which is hosted inside IIS6 -->
      <endpoint address="<a
href="%22http://localhost/FirstWcfIISHost/Service.svc%22">http://localhost/FirstWcfIISHost/Service.svc
</a>"
          binding="basicHttpBinding"
          contract="localhost.IService" name="IIS_Host_HttpBinding">
```

In this file I have configured my client in order to show you how we can call the same WCF service hosted in different environments. In realty you do not need all these endpoints, you need only one of these endpoints for your client, so your "app.config" would be as in following listing - 13:

Collapse | Copy Code

```
<?x ml version="1.0" encoding="utf-8" ?>
<configuration>
  <system.serviceModel>
    <client>
      <!-- calling the WCF service which is hosted inside windows console application -->
      <endpoint address="net.tcp://localhost:9100/FirstWcfService" binding="netTcpBinding"
          contract="localhost.IService" name="Windows_Console_Host_TcpBinding">
      </endpoint>
    </client>
   </system.serviceModel>
</configuration>
```

3- I have implemented my client as in the following listing -14:

Collapse | Copy Code

```
//Listing - 14, Program.cs
using System;
namespace FirstWcfServiceClientConsoleApp
{
    class Program
    {
        static void Main(string[] args)
        {
            try
            {
                #region Call the hosted service inside IIS6
                localhost.ServiceClient iis6proxy = new localhost.ServiceClient(
                    "IIS_Host_HttpBinding");
                Console.WriteLine("====================");
                Console.WriteLine("Call the hosted service inside IIS6");
                Console.WriteLine(iis6proxy.Hello());
                Console.WriteLine("====================");
                Console.WriteLine();
                #endregion
                #region Call the hosted service inside IIS7 (WAS)
                localhost.ServiceClient iis7proxy = new localhost.ServiceClient(
                    "II7_WAS_Host_TcpBinding");
                Console.WriteLine("====================");
                Console.WriteLine("Call the hosted service inside IIS7 (WAS)");
                Console.WriteLine(iis7proxy.Hello());
                Console.WriteLine("====================");
                Console.WriteLine();
                #endregion
                #region Call the hosted service inside Windows service application
                localhost.ServiceClient tcpWindowsSrviceproxy =
                    new localhost.ServiceClient("Windows_Services_Host_TcpBinding");
                Console.WriteLine("====================");
                Console.WriteLine(
                    "Call the hosted service inside Windows service application");
                Console.WriteLine(tcpWindowsSrviceproxy.Hello());
                Console.WriteLine("====================");
                Console.WriteLine();
                #endregion
                #region Call the hosted service inside Windows Console application
                localhost.ServiceClient tcpproxy = new localhost.ServiceClient(
                     "Windows_Console_Host_TcpBinding");
                Console.WriteLine("====================");
                Console.WriteLine(
                    "Call the hosted service inside Windows Console application");
                Console.WriteLine(tcpproxy.Hello());
                Console.WriteLine("====================");
                Console.WriteLine();
                Console.WriteLine("Press <Enter> to terminate the client application.");
                Console.ReadLine();
                #endregion
            }
            catch (Exception e)
            {
                Console.WriteLine(e.Message);
                Console.ReadLine();
            }
```

```
        }
    }
}
```

In the above code, we notice that in order to call a WCF service, we need to instantiate the proxy class like this :

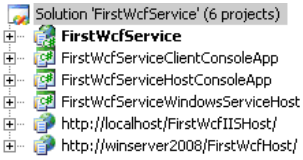
localhost.ServiceClient iis6proxy = new localhost.ServiceClient("IIS_Host_HttpBinding");

and then we can call a function or method we are interested like this:

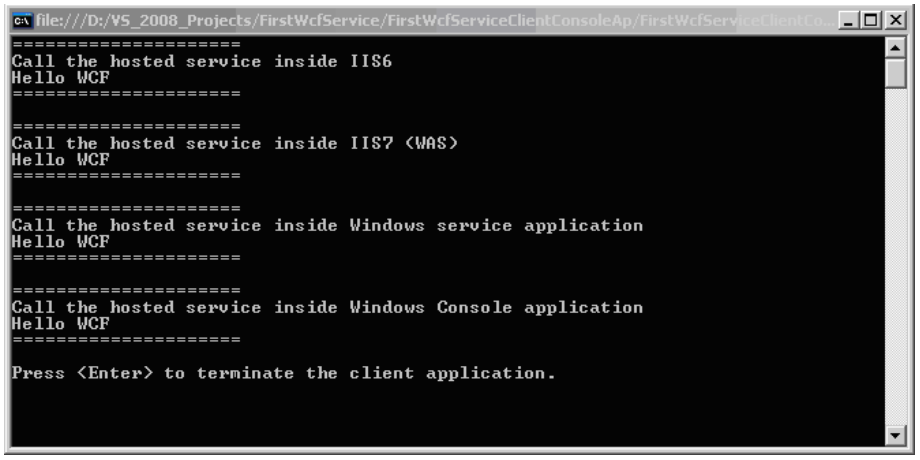
iis6proxy.Hello());

Putting all together and get the result

My final solution will be something like that:



Finally when we run the client application (FirstWcfServiceClientConsoleApp.exe) after verifying that all four hosting environments (IIS6, IIS7, windows service, windows console application) are running and listening, we get the following wonderful result:



As you can notice here, we have created a WCF service, hosted it in four different environments as below using different protocols :

- Windows a console application using tcp protocol
- Windows service process using tcp protocol
- IIS6 using http protocol
- IIS7 using tcp protocol

By changing only the endpoint on the client we have been able to access our WCF service and calling functions. What I do like to precise here, different client applications (Windows, Intranet, Internet, Extranet, ..etc) can call the service in different hosting environments.

Conclusion

We can conclude that WCF provides us a very simple development and deployment framework, but very powerful way of establishing communication between diverse systems using SOA (Service Oriented Architecture) designs and approaches.

License

About the Author

Faris Y. Elias SHOMOU
Architect Apside
France

I'm un IT Consultant (with B. Sc. and M. Sc. Background in Civil Engineering - university of Baghdad). I'm specialized in Microsoft products (ASP.NET, C#, WCF, Ajax.NET, VB.NET, and IIS) and also I'm familiar with various platforms and technologies (XML-XSL, SOA, Java, ORACLE). Good knowledge in n-tiers design and development using Object-Oriented Programming (OOP) methodologies.

I enjoy working on my own initiative as well as being part of a team. Well organized with a determination to succeed in all areas of work to be undertaken. I communicate effectively in four languages (English, French, Arab, Chaldean) in a cross-cultural environment. Currently I am working with Apside as an Senior IT Engineer in France.

Feel free to mail me for anything at fshomou AT hotmail Dot com.

Article Top

Comments and Discussions