

Essential WCF

Practical Implementation

By Akshay Patel

Essential WCF

Practical Implementation

*This free book is provided by courtesy of [C# Corner](#) and Mindcracker Network and its authors.
Feel free to share this book with your friends and co-workers.*

Please do not reproduce, republish, edit or copy this book.



Akshay Patel

Sam Hobbs
Editor, C# Corner

Table of Contents

This Book is about the basic introduction of the “**WCF (Windows Communication Foundation)**” for the beginners.

1. WCF Introduction and Contracts.
2. WCF Fault Contracts.
3. WCF Message Exchange pattern.
 - 3.1. Request/Response.
 - 3.2. One way
 - 3.3. Duplex.
4. WCF Data Contracts
 - 4.1 DataContract Attribute
 - 4.2 DataMemberAttribute
5. WCF Difference between Service allocation and Service library
6. WCF Serialization Part1
 - 6.1 XML serialize
 - 6.2 NetDataContractSerializer
 - 6.3 DataContractSerializer
7. WCF serialization Part2
8. WCF Opt-In Vs. Opt-Out
 - 8.1 Opt-Out Approach
 - 8.2 Opt-In Approach
9. WCF Message Contract
 - 9.1 MessageContractAttribute
 - 9.2 MessageHeaderAttribute
 - 9.3 MessageBodyAttribute
10. WCF Address Binding and Contract
 - 10.1 BasicHttpBinding
 - 10.2 wsDualHttpBinding
 - 10.3 webHttpBinding
 - 10.4 NetTcpBinding
 - 10.5 NetPeerTcpBinding
 - 10.6 ntNamedPipeBinding
 - 10.7 netNamedPipeBinding
 - 10.8 netMSMQBinding
11. WCF Service Configuration Using Web.Config
12. WCF Service Configuration Using Configuration Editor

1 WCF Introduction and Contracts

Introduction

Defining the basic information of all the contracts and a code demonstration for creating the WCF service application.

What is WCF?

WCF is a combined feature of Web Service, Remoting, MSMQ and COM+. WCF provides a common platform for all .NET communication. It is a part of .Net 3.0.



Difference between WCF and Web service

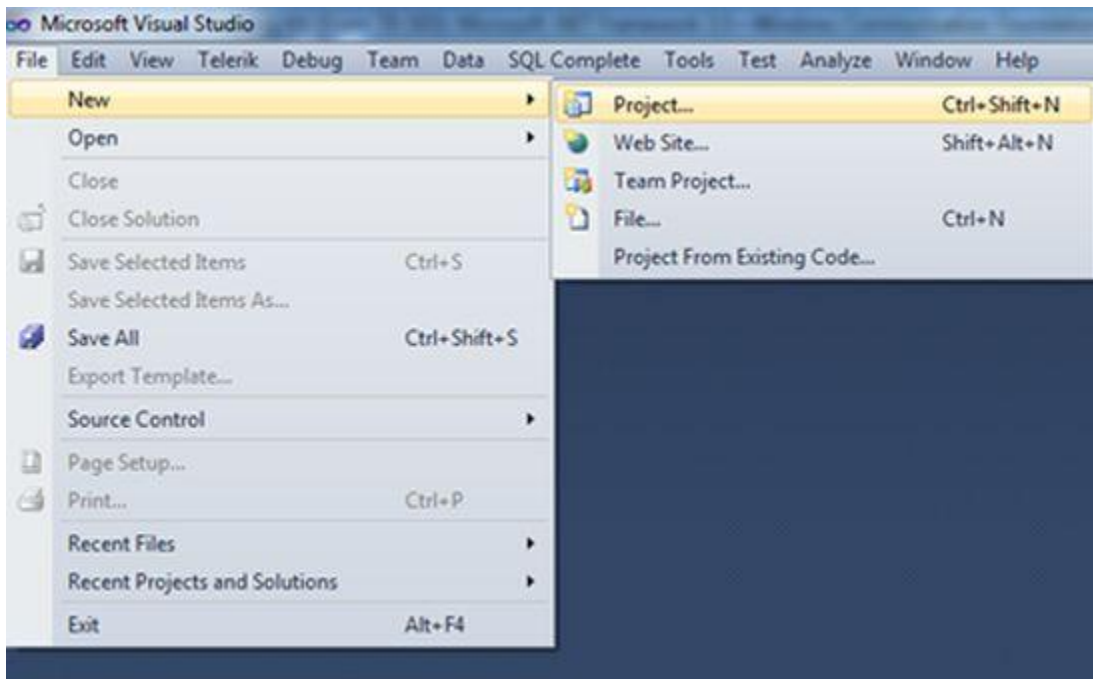
- In a web service we need to add the [WebService] attribute to the class.
In WCF we need to add the [ServiceContract] attribute to the class.
- Add the [WebMethod] attribute to the method in a web service.
Add the [OperationContract] attribute to the method in WCF.
- For serialization in a web service use the System.Xml.serialization namespace.
WCF uses the System.Runtime.Serialization namespace for serialization.
- We can host a web service in IIS.
We can host WCF in IIS, WAS (Windows Activation Service), self-hosting and a Windows Service.

Let's see how to create a WCF service application step-by-step.

Step 1

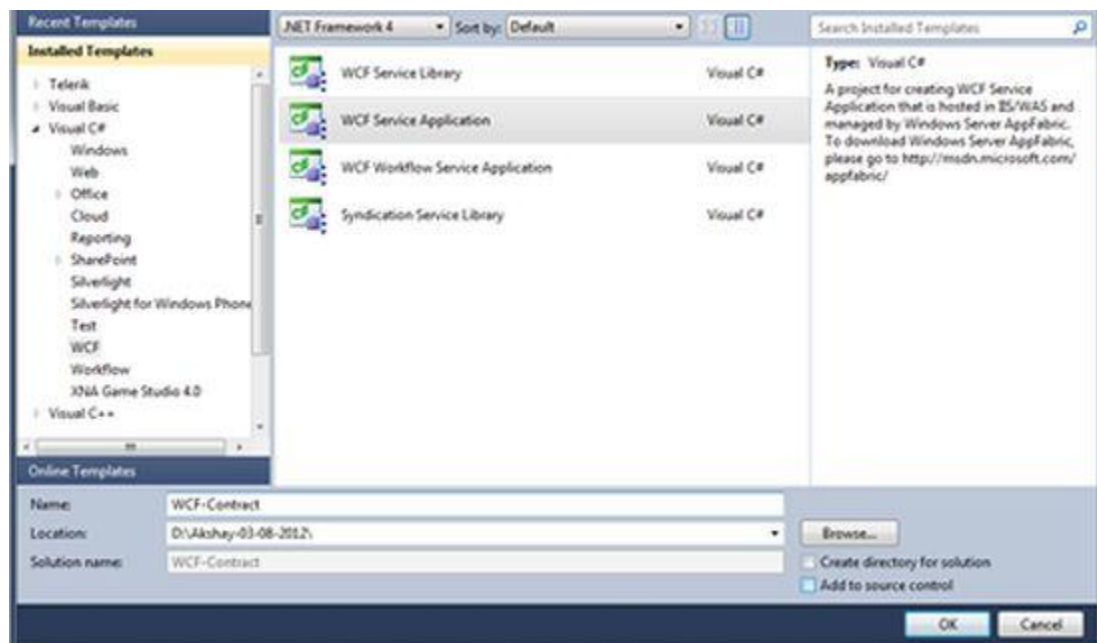
Start Menu >> All Programs >> Microsoft Visual Studio 2010 >> Microsoft Visual Studio 2010

In that "File" >> "New" >> "Project..."



Step 2

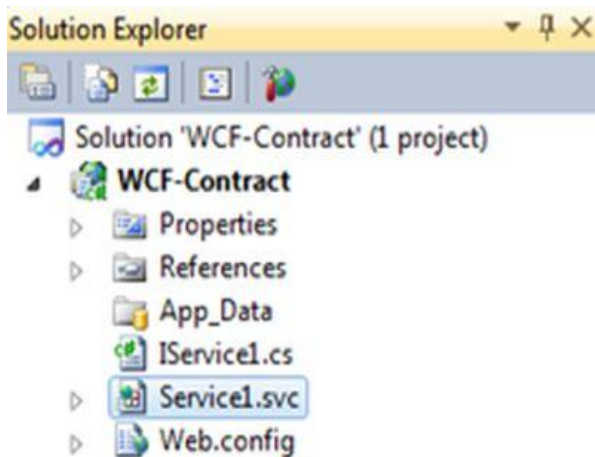
- Select WCF from Installed Templates
- Select .NET Framework 4 from dropdown list
- Select WCF Service Application
- Give desired name and select the location
- Click on OK button



Step 3

Now your Windows Communication Foundation service application is ready as a default service. You will see in Solution Explorer Service1.svc and IService1.cs

Open the IService1.cs file, as in:



In this file you will find a ServiceContract, OperationContract and DataContract.

Service Contract

Service contract is an attribute applied to an interface i.e. IService1. It describes which operations the client can perform on the services.

```
namespace WCF_Contract
{
    // NOTE: You can use the "Rename" command on the "Refactor" menu to ch
    [ServiceContract]
    public interface IService1
    {
        [OperationContract]
        string GetData(int value);

        [OperationContract]
        CompositeType GetDataUsingDataContract(CompositeType composite);

        // TODO: Add your service operations here
    }
}
```

Operation Contract

Operation Contract is an attribute applied to methods in interfaces i.e. IService1. It is used to define a method in an interface.

```
namespace WCF_Contract
{
    // NOTE: You can use the "Rename" command on the "Refactor" menu to
    [ServiceContract]
    public interface IService1
    {
        [OperationContract]
        string GetData(int value);

        [OperationContract]
        CompositeType GetDataUsingDataContract(CompositeType composite);

        // TODO: Add your service operations here
    }
}
```

Data Contract

DataContract defines which data types are passed to and from the service. It is used to define a class and the DataMember attribute is used to define the properties.

```
[DataContract]
public class CompositeType
{
    bool boolValue = true;
    string stringValue = "Hello ";

    [DataMember]
    public bool BoolValue
    {
        get { return boolValue; }
        set { boolValue = value; }
    }

    [DataMember]
    public string StringValue
    {
        get { return stringValue; }
        set { stringValue = value; }
    }
}
```

WCF Contracts map directly to a corresponding web services standard:

- ServiceContracts map to WSDL
- DataContracts map to XSD
- MessageContracts map to SOAP

Step 4

There is one method "GetData" which accepts parameters of type in

```
[ServiceContract]
public interface IService1
{
    [OperationContract]
    string GetData(int value);

    [OperationContract]
    CompositeType GetDataUsingDataContract(CompositeType composite);

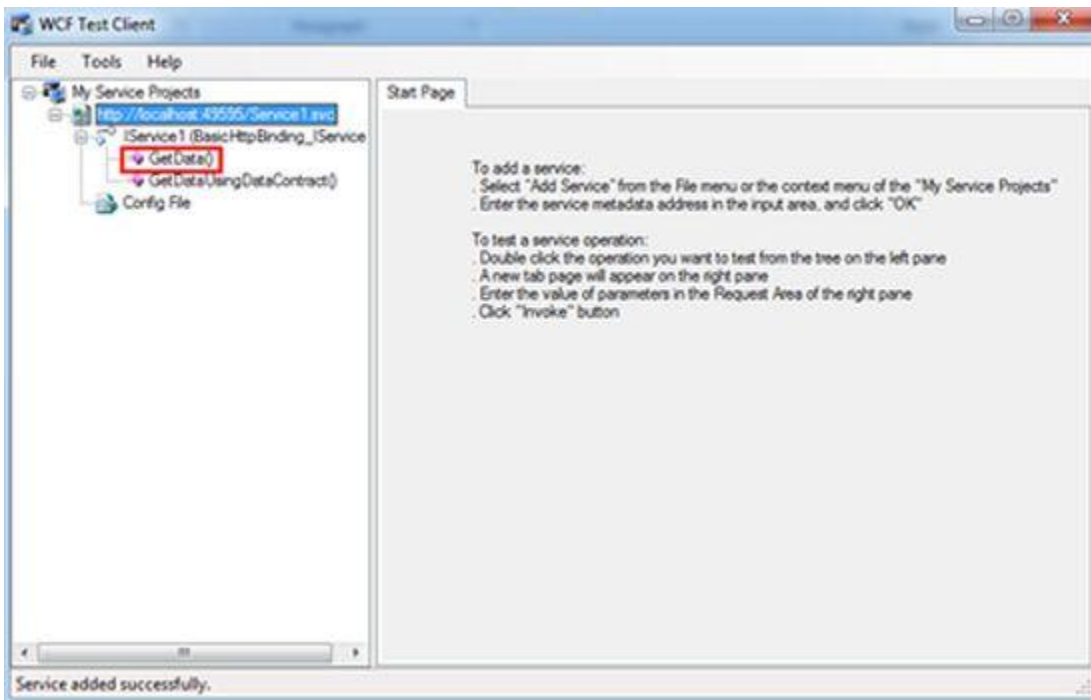
    // TODO: Add your service operations here
}
```

An implementation of this method is in the Service1.svc.cs file. This method returns a string with the value you passed as a parameter.

```
public class Service1 : IService1
{
    public string GetData(int value)
    {
        return string.Format("You entered: {0}", value);
    }
}
```

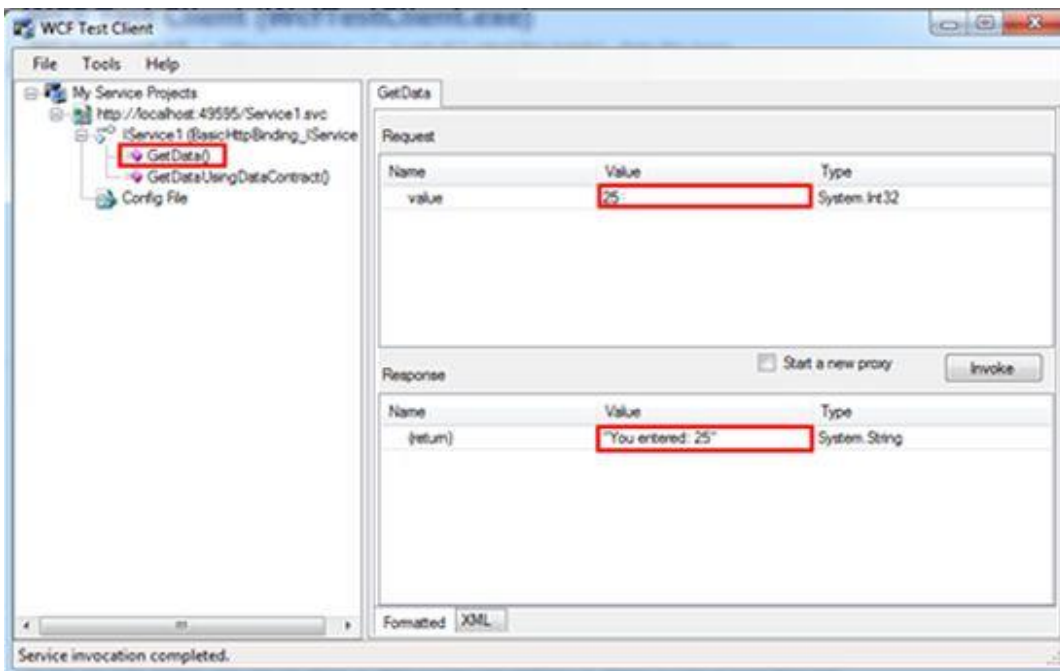
Step 5

Now let us run this service



You can test your service in WCF Test Client. WCF Test Client is a GUI tool which enables us to enter parameters, invoke services with these parameters and view the responses returned by services.

Now double-click on the "GetData" method and enter a parameter value of type System.int32. Here I enter "25" as a parameter value and press the "Invoke" button. You will get "You entered: 25" as a response.



Conclusion

We have seen a very basic example of a WCF Service application. We entered a parameter value of type int and got the entered value as a response.

2 WCF Fault Contracts

Introduction

Discuss about the fault contracts property in Windows Communication Foundation and also explains the difference between faults & exceptions and provides sample code for implementing fault contracts.

Fault Contracts

Windows Communication Foundation enables us to specify the fault behavior of our service.

For example

We create a service and this service is consumed by a client. Suppose this service throws an exception. But how can the client be notified of this exception? Because whenever an exception is thrown from a service, it cannot be sent to the client. But if you want to know the reason then a fault contract can help us. We can send information regarding this with the fault.

Faults Vs. Exceptions

The main thing in faults and exceptions is that "faults and exceptions are not the same thing".

An exception is a .Net mechanism. It is used when the program encounters an error. The .Net language allows us to throw, catch and ignore the exception. At some point they should be handled or the .Net runtime will terminate the thread in which the exception was thrown.

Faults refer to the SOAP fault mechanism to transfer error information from a service to a client. WCF provides the `FaultException` class. Whenever a service throws a `FaultException`, WCF serializes the information sent to the client.

FaultException

A `FaultException` is used to send untyped fault data to the client.

FaultException<TDetail>

This generic version is used to send typed fault data to the client. `TDetail` represents the type parameter for the fault information. Now let's see the example of typed fault data.

Create one WCF Service Application. Add the following segment of code in the Interface. In this code we are using the `FaultContract` attribute. It is applied to the interface only. Suppose we want to return various types of faults then we need to set the `FaultContract` attribute multiple times.

```
[ServiceContract]
public interface IService1
{
    [OperationContract]
    [FaultContract(typeof(FaultInfo))]
    string Topup(string Operator, string MobileNumber, double Amount);
}
```

```
}  
[DataContract()  
public class FaultInfo  
{  
    [DataMember()  
    public string Reason = null;  
}
```

If we choose to use a typed fault then we need to define a DataContract for the type that holds the error information. In our example we create a datacontract for the FaultInfo class and we create one data-member i.e. reason to hold the error information.

Add the following lines of code in the .svc.cs file. In this method we check whether the amount is double or not. If the amount is double then we throw a fault exception.

```
public string Topup(string Operator, string MobileNumber, double Amount)  
{  
    if (Amount == 0.0d)  
    {  
        FaultInfo fi = new FaultInfo();  
        fi.Reason = "Amount should not be in double.";  
        throw new FaultException<FaultInfo>(fi, new FaultReason(fi.Reason));  
    }  
    return "Recharge Successful.";  
}
```

If the amount is double then we create an object of the "FaultInfo" class and set the information we want to send to the client in the datamember i.e. the Reason. Finally we throw a fault exception of the fault info type.

Conclusion

Defining the FaultException and the difference between faults and exceptions and also defines the sample code to implement fault exceptions. You can download the code and check the application.

3 WCF Message Exchange pattern

Introduction

Simple introduction about the WCF Message exchange patterns. Various types of message exchange patterns and code demonstration for each message exchange pattern.

MEPs

MEP stands for Message Exchange Pattern. In WCF we are creating services. What does a service do for us? A service does some task on behalf of us or sends a response back from our request. All such communications are done using messages. We send some message as a request and get a message as a response from the service. WCF supports three types of MEPs:

- Request / Response
- One-Way
- Duplex

Now let us see each with an example.

Request / Response

Request / Response is a very common MEP. To set this pattern we need to set the `IsOneWay` property of `OperationContractAttribute` as false. And the default value of the `IsOneWay` property is false. So ultimately we do not need to set this property. It is set by default. That is why it is very easy to implement.

One more thing we should keep in mind is that we should not use the Duplex Channel setting. We will see about Duplex later in this article.

```
[ServiceContract]
public interface IService1
{
    [OperationContract]
    string GetTransaction(int value);
}
```

Here you may assume that since we are sending a request and against this request we are getting something in response, we cannot use a void return type. This is absolutely wrong. You can use a void return type. In this case the response message is still being sent back to the client. The difference is it sends an empty SOAP body.

```
[ServiceContract]
public interface IService1
{
    [OperationContract]
    void DoTransaction(string value);
}
```

Oneway

Set the `IsOneWay` property of `OperationContractAttribute` to true for a OneWay message exchange pattern. Suppose you send a message to a service. This pattern is used when the service does some operation and you do not want a response back. For example you want to change the status of a transaction from pending to completed and you do not want to get a confirmation from the service that the status is changed. You can use a OneWay pattern.

```
[ServiceContract]
public interface IService1
{
    [OperationContract(IsOneWay=true)]
    void ChangeTransactionStatus(int value);
}
```

Before using this pattern, keep following points in your mind

You cannot use FaultContract with this pattern. In my previous article we have seen FaultContract. For FaultContract there should be a two-way channel. So it is not possible in a Oneway pattern.

It is dangerous to send a oneway message since there is no assurance that the operation is processed. In our above example we sent a message to change the transaction status but there is no assurance that the transaction is changed or not.

Duplex

The duplex MEP is a two-way message channel. When the client sends a message to the service to instantiate some long-running processing and requires notification back from the service, a duplex MEP is applicable.

There are two types of contracts, ServiceContract and CallbackContract required to implement a duplex MEP. This MEP is declared by associating a CallbackContract with the ServiceContract. It is done through a CallbackContract property of ServiceContract.

```
namespace WcfService1
{
    [ServiceContract]
    public interface ITransactionHandler
    {
        [OperationContract (IsOneWay=true)]
        void TransactionDone(int value);
    }
    [ServiceContract(CallbackContract = typeof(ITransactionHandler))]
    interface ITransactionService
    {
        [OperationContract(IsOneWay = true)]
        void DoTransaction(string value);
    }
}

public class TransactionService : ITransactionService
{
    public void DoTransaction(string value)
    {
        int TransactionId = 1; //Get TransactionId from database

        ITransactionHandler callbackHandler =
            OperationContext.Current.GetCallbackChannel<ITransactionHandler>();

        callbackHandler.TransactionDone(TransactionId);
    }
}
```

In the preceding example the client sends a one-way message to the TransactionService. After some period of time the service calls back to the client and says TransactionDone.

Before using this pattern, keep the following points in mind.

In the real world this pattern is problematic because it requires a connection back to the client. And there may be a chance to not connect back due to firewall and Network Address Translation problems.

This pattern doesn't scale very well due to long-running sessions maintained between client and service.

Threading problems can occur if either of the Callback channels are not empty.

Conclusion

The Request / Response MEP are used in most cases, because some kind of acknowledgement is required. In our case if the client gets TransactionId back at the time of changing the status, he can query further with this TransactionId. So acknowledgement is mandatory.

4 WCF Data Contracts

DataContract

Data contracts are the conceptual agreement of the format and structure of the data in the message exchanged between a service and client. It is a format of data passed to and from the service operations. It defines the structure and types of data exchanged in service messages.

DataContractAttribute

The DataContractAttribute is defined in the System.Runtime.Serialization namespace. It is used to declare a type as a DataContract.

Properties of DataContractAttribute

```
namespace System.Runtime.Serialization
{
    // Summary:
    //     Specifies that the type defines or implements a data contract and is serializable
    //     by a serializer, such as the System.Runtime.Serialization.DataContractSerializer.
    //     To make their type serializable, type authors must define a data contract
    //     for their type.
    [AttributeUsage(AttributeTargets.Class | AttributeTargets.Struct | AttributeTargets.Enum, Inherited = false, AllowMultiple = false)]
    public sealed class DataContractAttribute : Attribute
    {
        // Summary:
        //     Initializes a new instance of the System.Runtime.Serialization.DataContractAttribute
        //     class.
        public DataContractAttribute();

        // Summary:
        //     Gets or sets a value that indicates whether to preserve object reference
        //     data.
        //
        // Returns:
        //     true to keep object reference data using standard XML; otherwise, false.
        //     The default is false.
        public bool IsReference { get; set; }

        // Summary:
        //     Gets or sets the name of the data contract for the type.
        //
        // Returns:
        //     The local name of a data contract. The default is the name of the class that
        //     the attribute is applied to.
        public string Name { get; set; }

        // Summary:
        //     Gets or sets the namespace for the data contract for the type.
        //
        // Returns:
        //     The namespace of the contract.
        public string Namespace { get; set; }
    }
}
```

- **Name** : It determines the type name as it is generated in the resulting schema.
- **Namespace** : It sets the target namespace of the schema.

DataMemberAttribute

The DataMemberAttribute is also the part of the System.Runtime.Serialization namespace. It is applied to the members and it is used to declare that the members should be included in the serialization.

Properties of DataMemberAttribute

```

namespace System.Runtime.Serialization
{
    // Summary:
    //     When applied to the member of a type, specifies that the member is part of
    //     a data contract and is serializable by the System.Runtime.Serialization.DataContractSerializer.
    [AttributeUsage(AttributeTargets.Property | AttributeTargets.Field, Inherited = false, AllowMultiple = false)]
    public sealed class DataMemberAttribute : Attribute
    {
        // Summary:
        //     Initializes a new instance of the System.Runtime.Serialization.DataMemberAttribute
        //     class.
        public DataMemberAttribute();

        // Summary:
        //     Gets or sets a value that specifies whether to serialize the default value
        //     for a field or property being serialized.
        //
        // Returns:
        //     true if the default value for a member should be generated in the serialization
        //     stream; otherwise, false. The default is true.
        public bool EmitDefaultValue { get; set; }

        // Summary:
        //     Gets or sets a value that instructs the serialization engine that the member
        //     must be present when reading or deserializing.
        //
        // Returns:
        //     true, if the member is required; otherwise, false.
        public bool IsRequired { get; set; }

        // Summary:
        //     Gets or sets a data member name.
        //
        // Returns:
        //     The name of the data member. The default is the name of the target that the
        //     attribute is applied to.
        public string Name { get; set; }

        // Summary:
        //     Gets or sets the order of serialization and deserialization of a member.
        //
        // Returns:
        //     The numeric order of serialization or deserialization.
        public int Order { get; set; }
    }
}

```

- **EmitDefaultValue** : If you want to add default values in the serialization then set the EmitDefaultValue to true else EmitDefaultValue to false. By default it is true.
- **IsRequired** : It controls the minOccurs attribute for the schema element. The default value is false.
- **Name** : It creates the schema element name generated for the member.
- **Order** : It maintains the order of each element in the schema. By default it appears alphabetically.

Note

- For example you set the EmitDefaultValue to false and IsRequired to true and you are not assigning any value, at that time it creates a problem. Because the serializer emits the default value and we are not passing any value, on the other side it is required. That is why it throws an error.

- If you apply a data member attribute to both property and field, then it generates duplicate members in the schema.

Sample Code

- Create one DataContract for Place. Add the data members PlaceCode and PlaceName.
- Set the order for the data members and set IsRequired to true for PlaceCode.
- Create one OperationContract with return type Place.

```
namespace DataContractServiceApp
{
    [ServiceContract]
    public interface IService1
    {
        [OperationContract]
        Place GetPlace(string PlaceCode);
    }

    [DataContract(Name = "Place", Namespace = "")]
    public class Place
    {
        [DataMember(Name = "PlaceCode", Order = 1, IsRequired = true)]
        public string PlaceCode;
        [DataMember(Name = "PlaceName", Order = 2)]
        public string PlaceName;
    }
}
```

- In the service class, the GetPlace method is implemented and in that we create an object of place, assign some values to it and return a Place object.

```
namespace DataContractServiceApp
{
    public class Service1 : IService1
    {
        public Place GetPlace(string PlaceCode)
        {
            Place place = new Place();

            // Get placename by placecode from database.

            return place;
        }
    }
}
```

5 WCF Difference between Service and Service Library

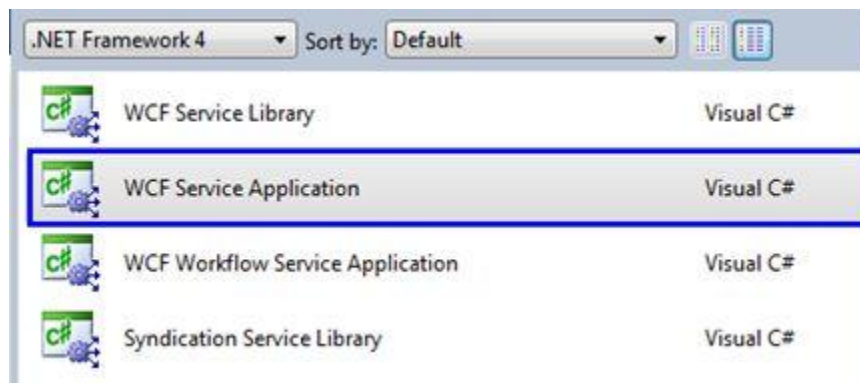
Introduction

Difference between a Service Application and a Service Library. And explains where we should use Service Applications and where to use a Service Library.

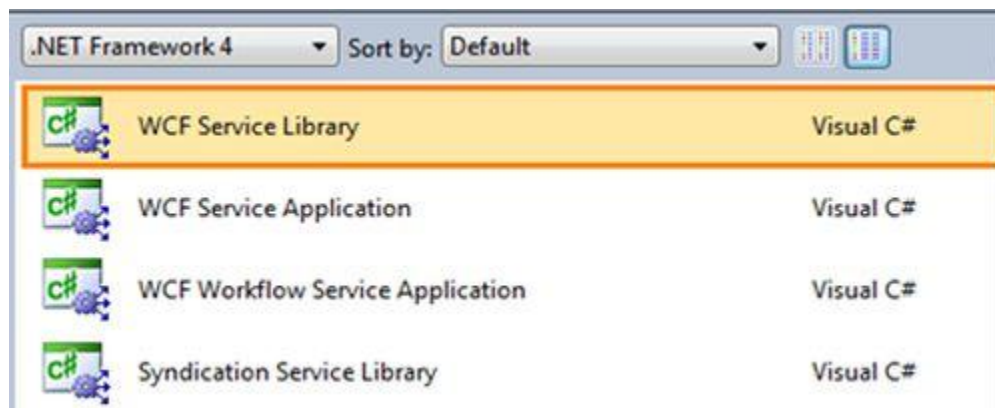
When you try to create a WCF service using Visual Studio IDE, you will have a choice of two types of WCF services, WCF Service Application and WCF Service Library. Now let us see the differences between them. We will do this by generating each of them and comparing the results.

Differences

1. The very basic point is that you need to select a "WCF Service Application" template under the WCF project type at the time of project creation. In other words when you create the project from File >> New >> Project, select the WCF as project type from the left side of the screen and select "WCF Service Application" from the template. Select your desired location and provide an appropriate name for your project and press the "OK" button.

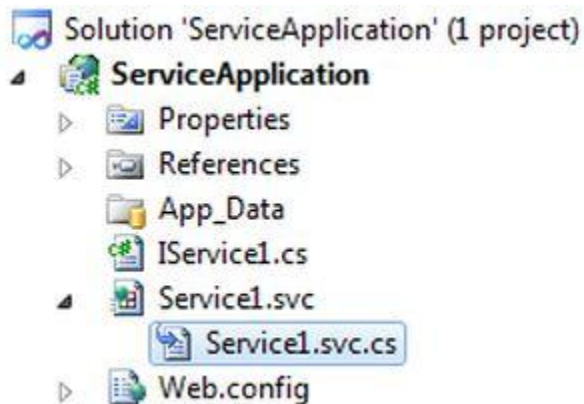


For a WCF Service Library you need to select "WCF Service Library" template at the time of project creation.

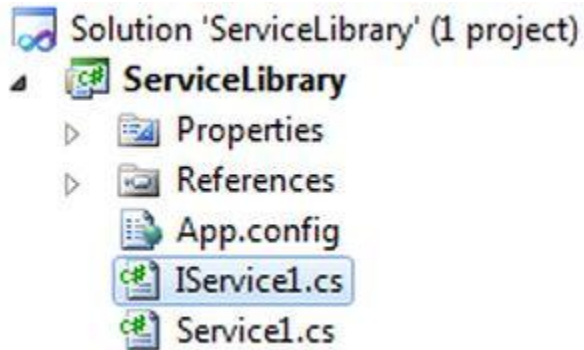


2. After creating the projects, compare the files created by Visual Studio IDE.

In the Service Application we have a service contract i.e. IService1 for the service implementation and Service1 as a Web.config file.



In the Service Library we also have a service contract i.e. IService1 for the service implementation and Service1 as an App.config file for the configuration (instead of web.config as in the Service Application).



The major difference is that the WCF Service Application has a .svc file, whereas the Service Library does not have a .svc file. Suppose we want to host this service application in IIS, then you need to specify for IIS the execution runtime environment requirements.

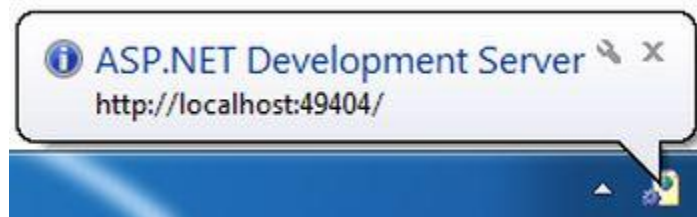
In a web application we set an .aspx, similarly for a Service Application we need to set a .svc.

3. Run the WCF Service Application and it will show a directory listing, as in:

Directory Listing -- /

Wednesday, October 03, 2012 11:10 AM	<dir> App_Data
Wednesday, October 03, 2012 11:16 AM	<dir> bin
Wednesday, October 03, 2012 11:10 AM	1,243 IService1.cs
Wednesday, October 03, 2012 11:10 AM	<dir> obj
Wednesday, October 03, 2012 11:10 AM	<dir> Properties
Wednesday, October 03, 2012 11:10 AM	115 Service1.svc
Wednesday, October 03, 2012 11:10 AM	936 Service1.svc.cs
Wednesday, October 03, 2012 11:10 AM	4,452 ServiceApplication.csproj
Wednesday, October 03, 2012 11:10 AM	1,086 ServiceApplication.csproj.user
Wednesday, October 03, 2012 11:10 AM	925 ServiceApplication.sln
Wednesday, October 03, 2012 11:16 AM	10,752 ServiceApplication.suo
Wednesday, October 03, 2012 11:10 AM	960 Web.config
Wednesday, October 03, 2012 11:10 AM	247 Web.Debug.config
Wednesday, October 03, 2012 11:10 AM	343 Web.Release.config

Version Information: ASP.NET Development Server 10.0.0.0



Now double-click on Service1.svc and it is hosted by default by the ASP.Net Development server. You can see in the above popup window that the ASP.Net development server has been started.

Service1 Service

You have created a service.

To test this service, you will need to create a client and use it to call the service. You can do this using the svcutil.exe tool from the command line with the following syntax:

```
svcutil.exe http://localhost:49404/Service1.svc?wsdl
```

You can also access the service description as a single file:

```
http://localhost:49404/Service1.svc?singleWsdl
```

This will generate a configuration file and a code file that contains the client class. Add the two files to your client application and use the generated client class to call the Service. For example:

C#

```
class Test
{
    static void Main()
    {
        ServiceClient client = new ServiceClient();

        // Use the 'client' variable to call operations on the service.

        // Always close the client.
        client.Close();
    }
}
```

Visual Basic

```
Class Test
Shared Sub Main()
    Dim client As ServiceClient = New ServiceClient()
    ' Use the 'client' variable to call operations on the service.

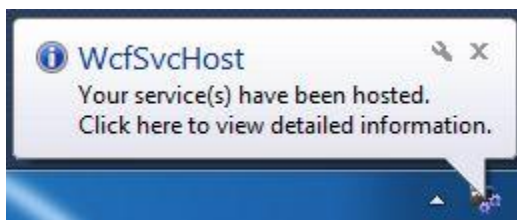
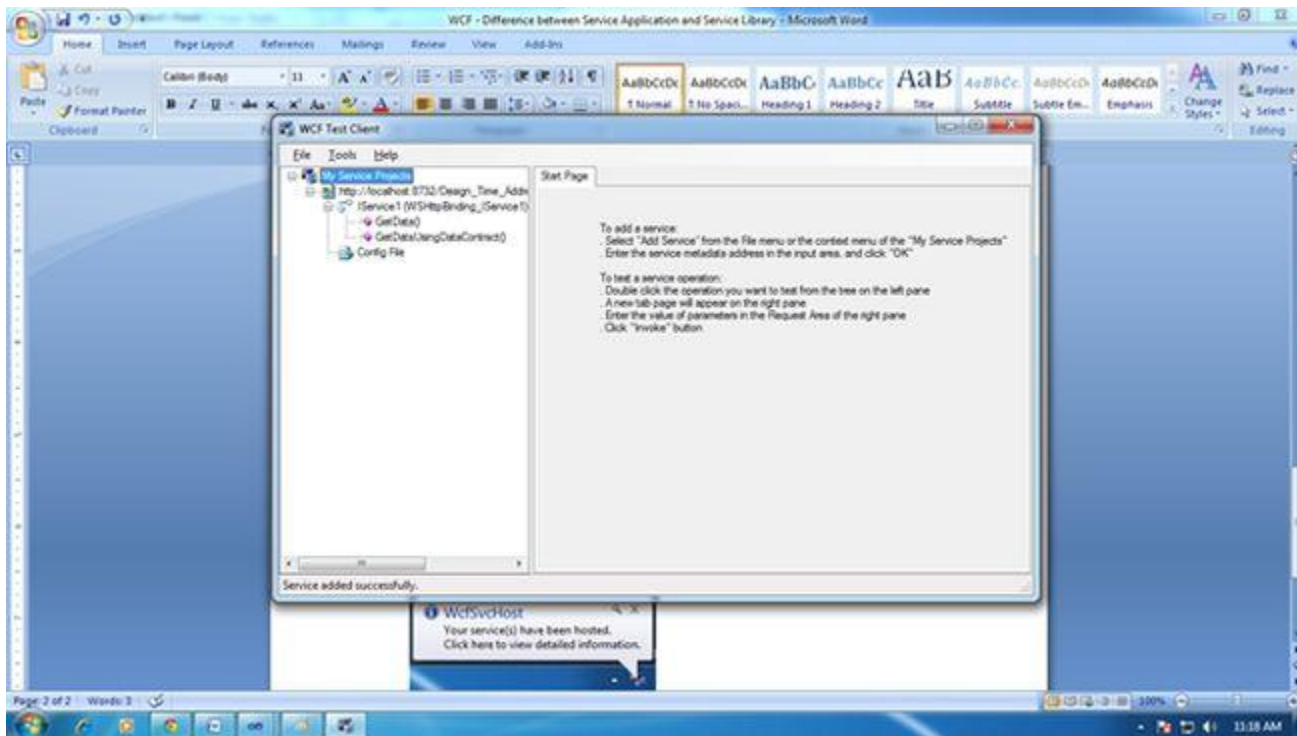
    ' Always close the client.
    client.Close()
End Sub
End Class
```

A WCF Service Library is not hosted by the ASP.Net development server. It is hosted by the Test Client, since a Service

©2013 C# CORNER.

SHARE THIS DOCUMENT AS IT IS. PLEASE DO NOT REPRODUCE, REPUBLISH, CHANGE OR COPY.

Library does not have .svc file.



If you want to host your service in IIS then you should select the WCF Service Application template and if you want to host it as a Windows Service or set a reference of your library then you should select the WCF Service Library template.

Conclusion

The WCF Service Application template can be used to create WCF services with a hosting website created within the project

The WCF Service Library template can be used to create WCF services that are hosted by the WCF Service Host, and these can be tested using the WCF service Test Client.

6 WCF Serialization Part1

Introduction

Here we will discuss about serialization in WCF, including the default serializer in WCF and the various kinds of serializers

that WCF supports.

WCF provides a message-oriented programming framework. We use WCF to transmit messages between applications. Internally WCF represents all the messages by a Message class. When WCF transmits a message it takes a logical message object and encodes it into a sequence of bytes. After that WCF reads those bytes and decodes them in a logical object. The process forms a sequence of bytes into a logical object; this is called an encoding process. At runtime when WCF receives the logical message, it transforms them back into corresponding .Net objects. This process is called serialization.



WCF supports three basic serializers:

- XmlSerializer
- NetDataContractSerializer
- DataContractSerializer

WCF deserializes WCF messages into .Net objects and serializes .Net objects into WCF messages. WCF provides DataContractSerializer by default with a servicecontract. We can change this default serializer to a custom serializer like XmlSerializer.

```

[XmlSerializerFormat]
[ServiceContract]
public interface IService1
{
    [OperationContract]
    void AddAuthor(Author author);
}
  
```

The XmlSerializerFormat attribute above the ServiceContract means this serializer is for all operation contracts. You can also set a separate contract for each operation.

Each serializer calls different algorithms for mapping between .Net object and WCF messages. And each serializer produces a slightly different message.

We can use SvcUtil.exe to move between .Net types and XSD. We can export an XSD that describes what the serializer expects and can import an XSD to product types for a specific serializer.

XMLSerializer

We can find XmlSerializer in the System.Xml.Serialization namespace. WCF supports this serialization from .Net 1.0. It is used by default with the ASP.Net webservice (ASMX).

Usage

- We can use this serializer whenever we want to get backward compatibility with ASMX.

- It can also be used when integrating with non WCF Services.

NetDataContractSerializer

NetDataContractSerializer is analogous to .Net Remoting Formatters. It implements IFormatter and it is compatible with [Serializable] types. It is not recommended for service oriented design.

Usage

- It is used when we have WCF services at both sides.
- It is easier when we want to migrate .Net remoting applications.

DataContractSerializer

DataContractSerializer is a default serializer for WCF. We need not to mention DataContractSerializer attribute above the service contract.

Usage

- Used by default unless we specify otherwise.
- It is the best choice for code-first designs.

In my next article we will see step-by-step implementation of DataContractSerializer and code demonstration.

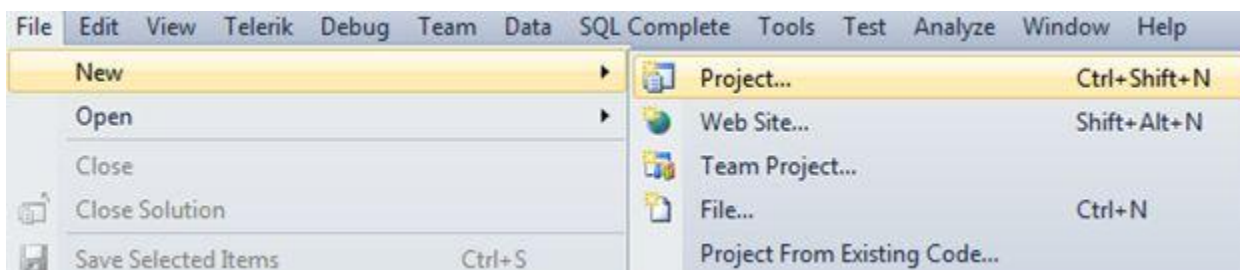
7 WCF Serialization

Introduction

Create an application, in this application we use a DataContractSerializer to serialize and deserialize data. We also see how we can use SvcUtil.exe.

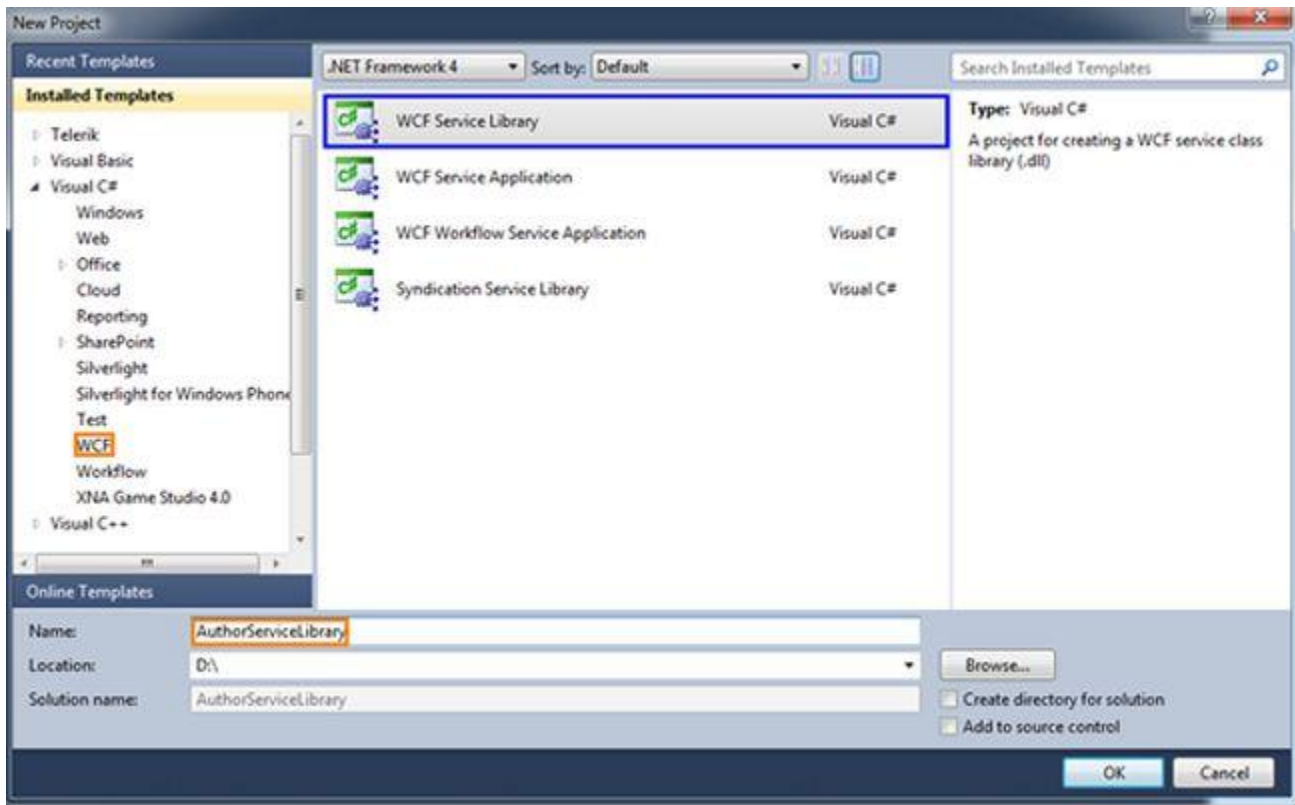
Step-by-step implementation of DataContractSerializer

File -> New -> Project



Select "WCF" from Installed Templates.

Select "WCF Service Library" and give a proper name, for example "AuthorSerializationLibrary".



Create one DataContract in the Interface i.e. IService1.cs using the following code:

```
namespace AuthorServiceLibrary
{
    [ServiceContract]
    public interface IService1
    {
    }

    [DataContract]
    public class Author
    {
        [DataMember]
        public string FirstName;

        [DataMember]
        public string LastName;

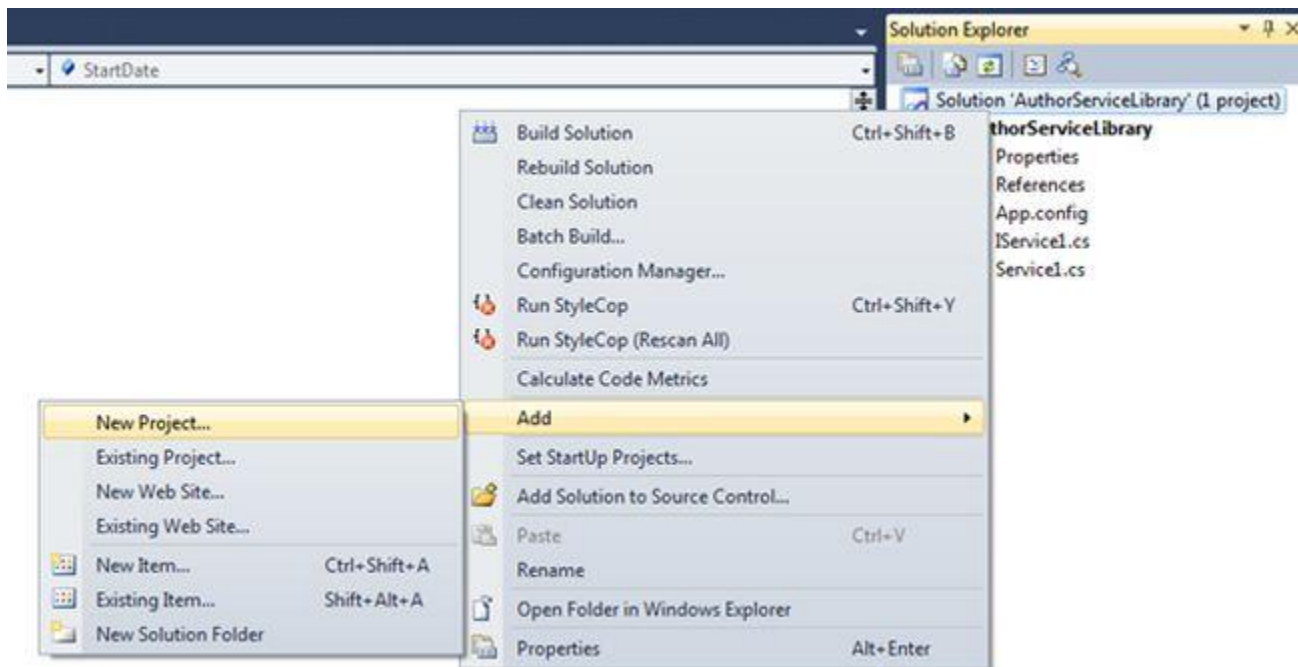
        [DataMember]
        public DateTime StartDate;

        [DataMember]
        public string ArticleName;
    }
}
```


I have not created an operation contract. So there is no code in the service class.

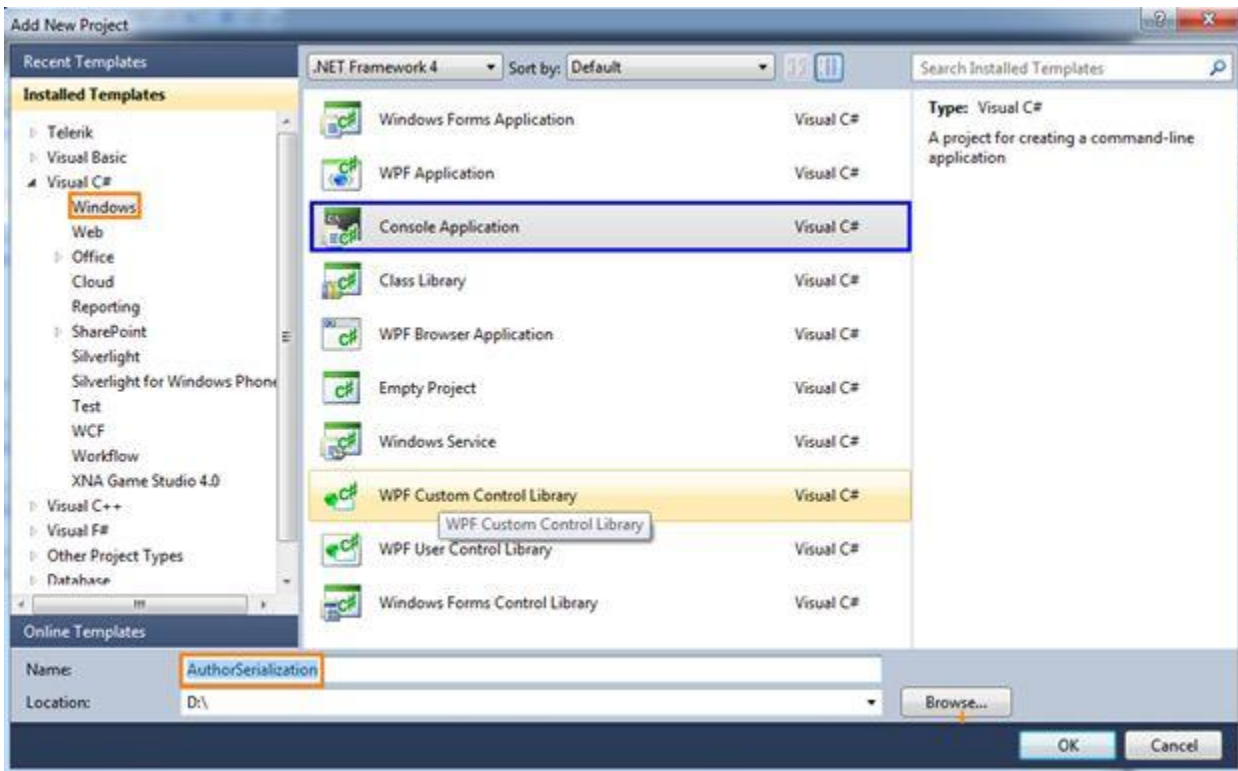
```
namespace AuthorServiceLibrary
{
    public class Service1 : IService1
    {
    }
}
```

Now right-click on the Solution file and select Add > New Project, as in:



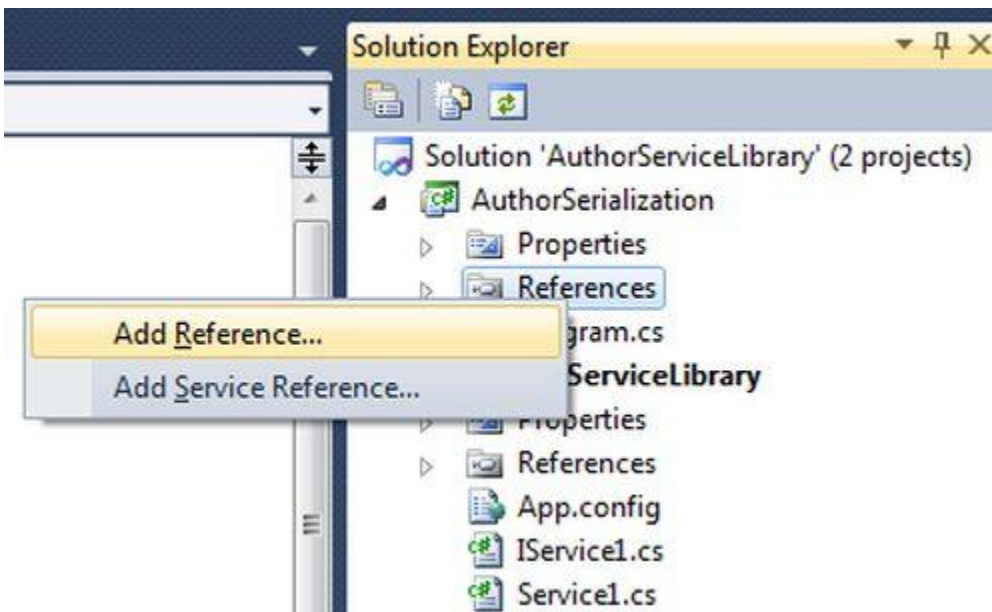
Select "Windows" from the Installed Templates.

Select "Console Application" and give it the name "AuthorSerialization", as in:



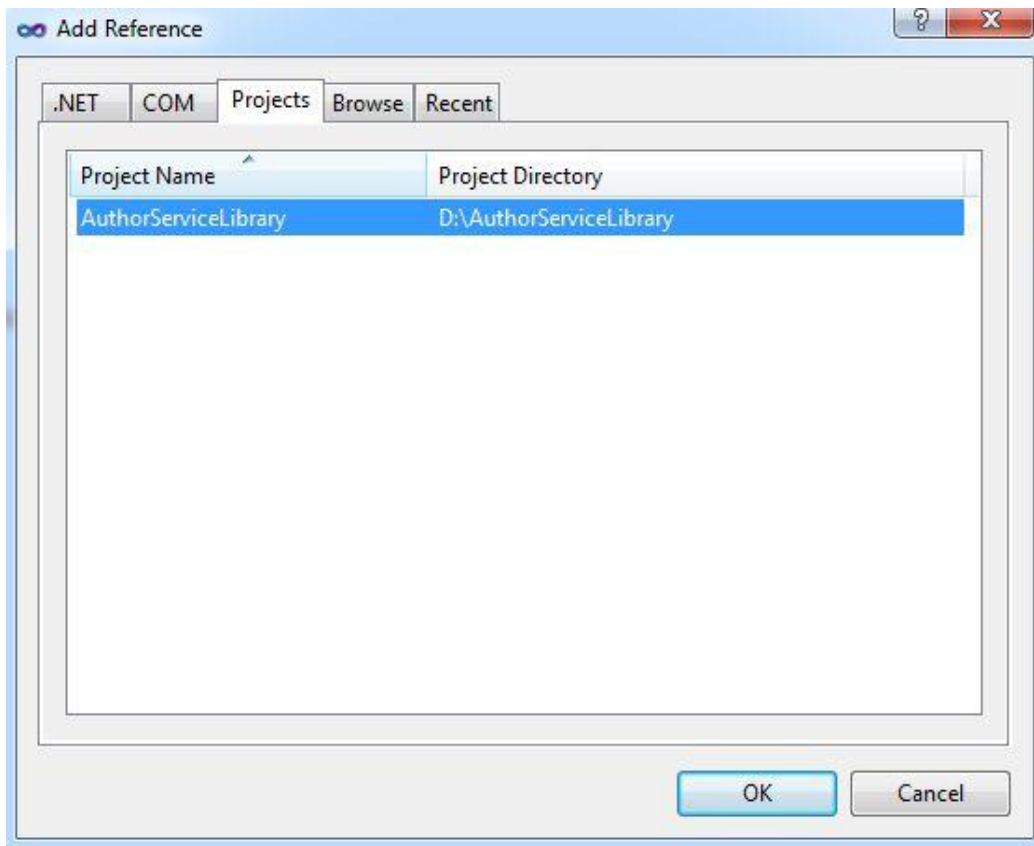
Add a reference for "AuthServiceLibrary" into "AuthorSerialization".

Right-click on "References" in AuthorSerialization and select "Add Reference...", as in:

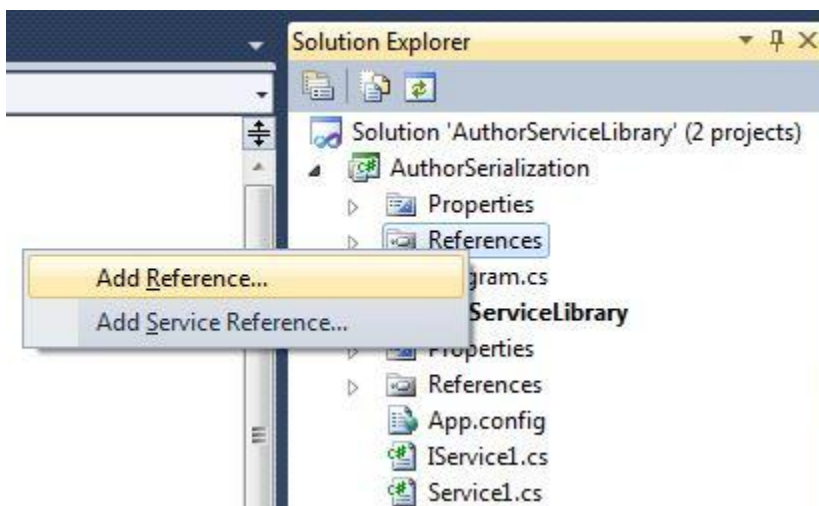


Select "AuthServiceLibrary" from the Projects tab.

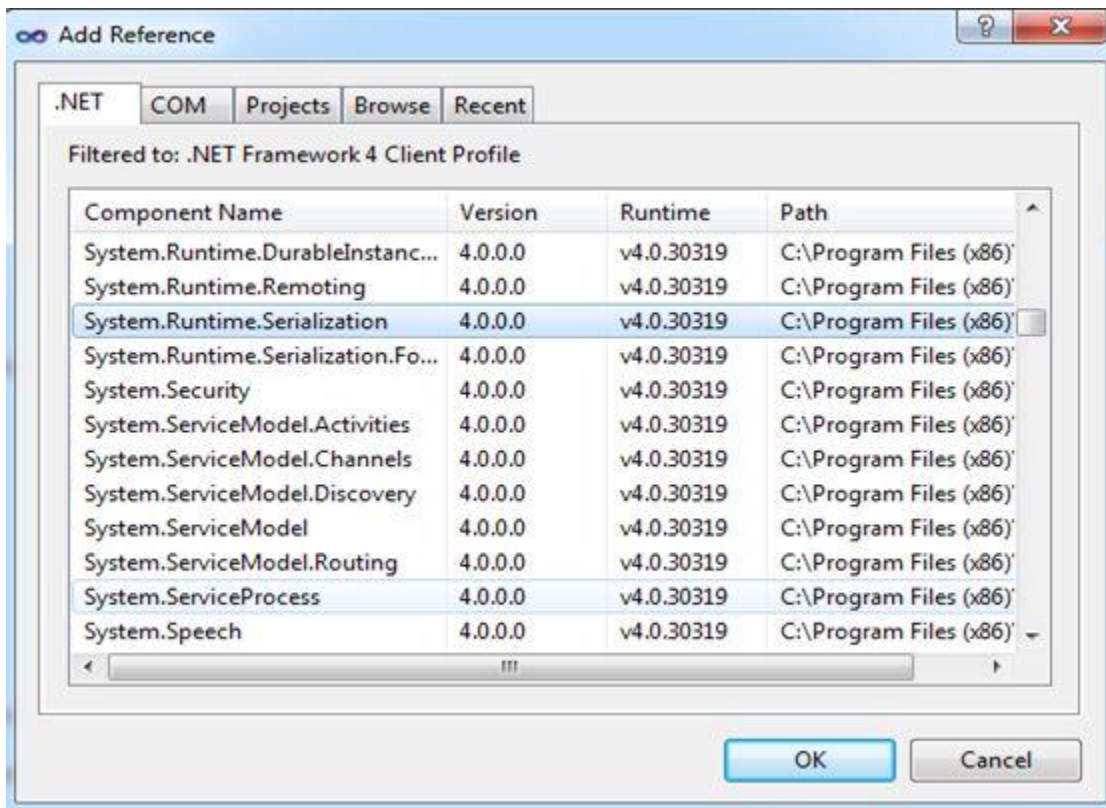
Press the "OK" button.



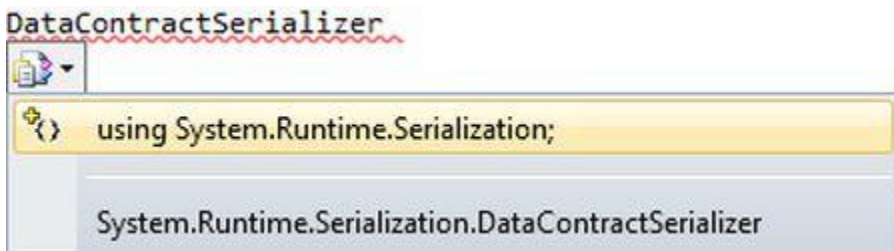
One more time right-click on references and select "Add Reference", as in:



Select "System.Runtime.Serialization" from the .NET tab, as in:



Import the System.Runtime.Serialization namespace into the console application "AuthorSerialization"; see:



Create a static function to serialize data in the program.cs file of "AuthorSerialization" using:

```

static void Serialize()
{
    Author author = new Author();
    author.FirstName = "Akshay";
    author.LastName = "Patel";
    author.StartDate = DateTime.Now;
    author.ArticleName = "WCF - Serialization - Day 6";

    using (FileStream fs = new FileStream("author.xml", FileMode.Create))
    {
  
```

```
DataContractSerializer dcSerializer = new DataContractSerializer(typeof(Author));
    dcSerializer.WriteObject(fs, author);
}
}
```

In the preceding Serialize function we create an object of the Author class and assign some values to each field. We create a .xml file using a FileStream object. After that we serialize the class using DataContractSerializer and write into the .xml file.

Create one more function to deserialize the data under the preceding function; the code is:

```
static void Deserialize()
{
    using (FileStream fs = new FileStream("author.xml", FileMode.Open))
    {
        DataContractSerializer dcSerializer = new DataContractSerializer(typeof(Author));
        Author author = dcSerializer.ReadObject(fs) as Author;
        Console.WriteLine("Name: {0}, Article: {1}", author.FirstName, author.ArticleName);
    }
}
```

In the preceding code we open author.xml with the help of a FileStream object. We create an object of DataContractSerializer and read the file.

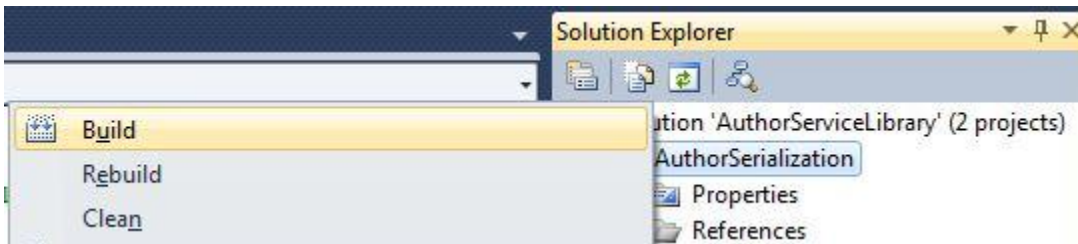
Add the following lines of code in the main function:

```
static void Main(string[] args)
{
    if (args.Length > 0 && args[0].Equals("ds"))
        Deserialize();
    else
        Serialize();
}
```

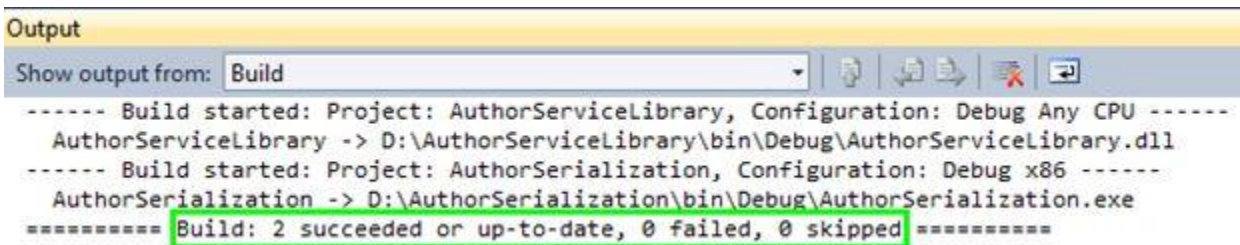
In the preceding code we check the length of args and args equal to ds, if it fulfills the condition then call the Deserialize function and if not then call the Serialize function. In other words if you want to call the Deserialize function then pass "ds" as an argument .

Now build the console application "AuthorSerialization".

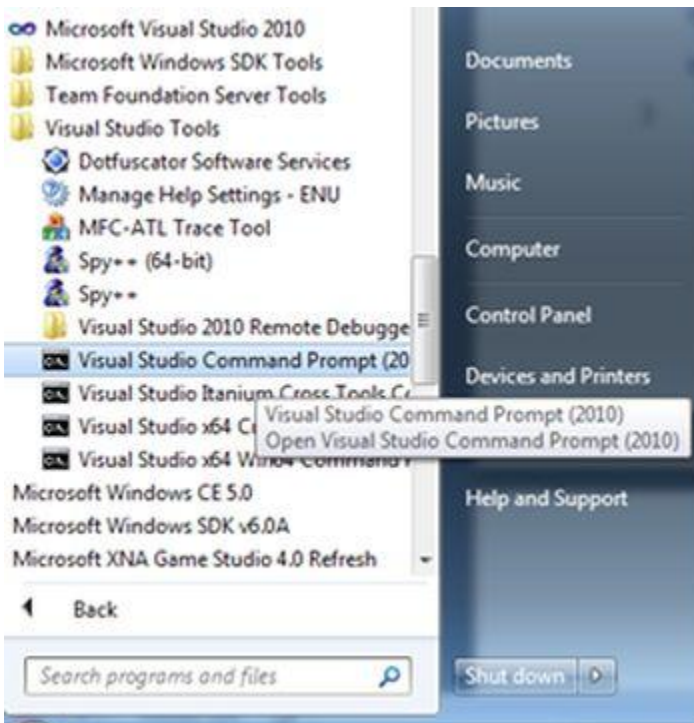
For that right-click on the AuthorSerialization project and select "Build".



Now you will see in the output window that "Build: 2 succeeded or up-to-date".



Open a Visual Studio Command Prompt; see:



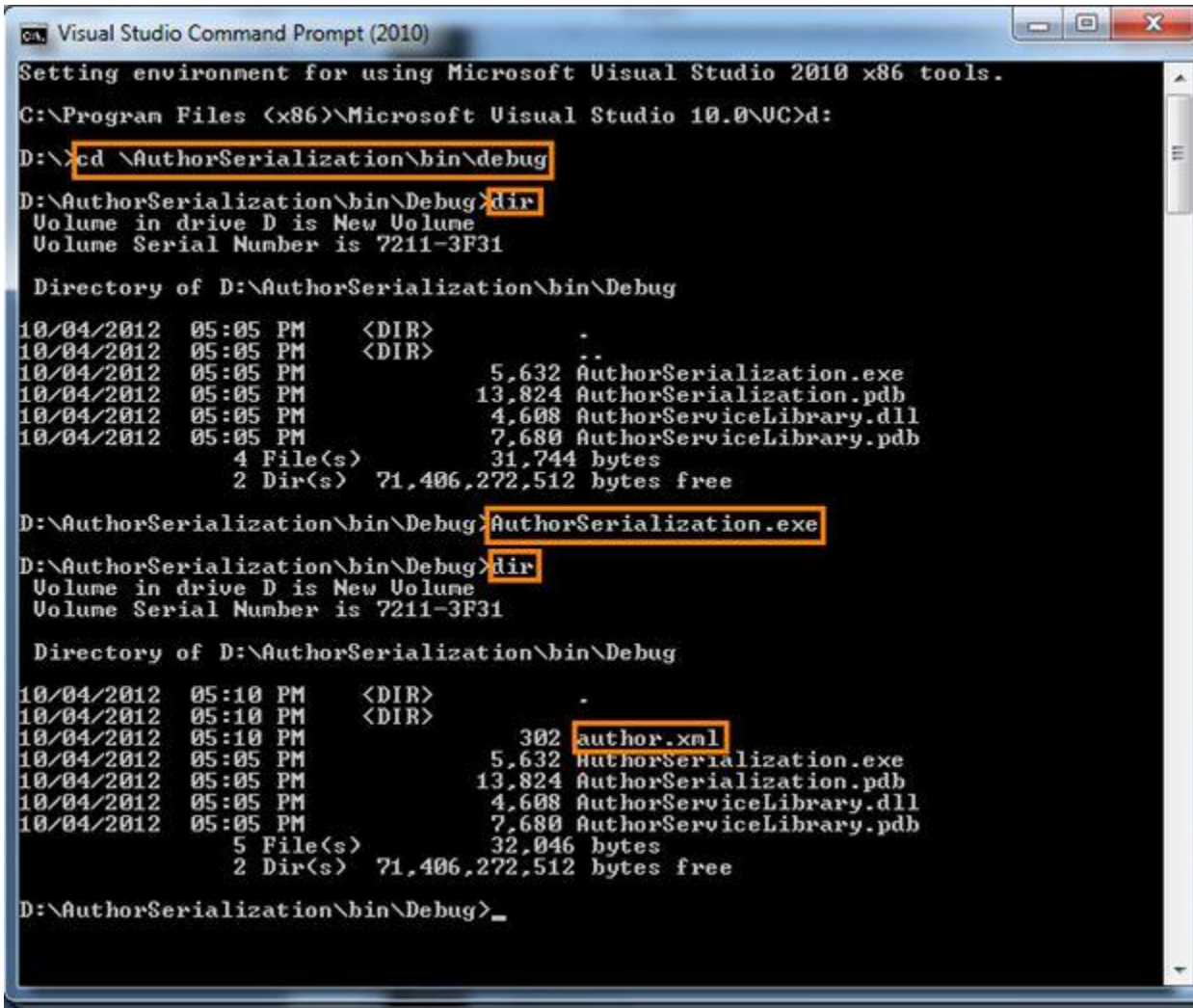
Now navigate to the path where you created your AuthorSerialization application. Navigate to bin and in that go to debug.

Run the command dir, it will show all files and directories contained in the debug folder.

Run AuthorSerialization.exe.

Once you have executed it successfully, run the dir command.

You will see that an author.xml file has been created.



```

Visual Studio Command Prompt (2010)
Setting environment for using Microsoft Visual Studio 2010 x86 tools.
C:\Program Files (x86)\Microsoft Visual Studio 10.0\VC>d:
D:\>cd \AuthorSerialization\bin\debug
D:\AuthorSerialization\bin\Debug>dir
Volume in drive D is New Volume
Volume Serial Number is 7211-3F31

Directory of D:\AuthorSerialization\bin\Debug

10/04/2012  05:05 PM    <DIR>          .
10/04/2012  05:05 PM    <DIR>          ..
10/04/2012  05:05 PM                5,632 AuthorSerialization.exe
10/04/2012  05:05 PM            13,824 AuthorSerialization.pdb
10/04/2012  05:05 PM             4,608 AuthorServiceLibrary.dll
10/04/2012  05:05 PM             7,680 AuthorServiceLibrary.pdb
               4 File(s)              31,744 bytes
               2 Dir(s)  71,406,272,512 bytes free

D:\AuthorSerialization\bin\Debug>AuthorSerialization.exe
D:\AuthorSerialization\bin\Debug>dir
Volume in drive D is New Volume
Volume Serial Number is 7211-3F31

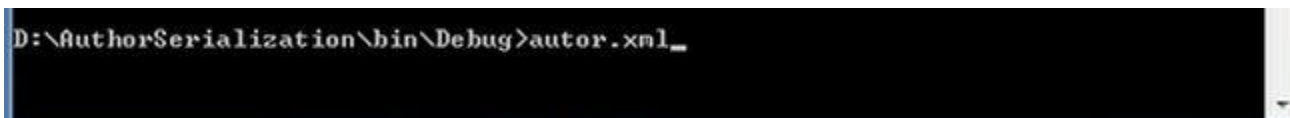
Directory of D:\AuthorSerialization\bin\Debug

10/04/2012  05:10 PM    <DIR>          .
10/04/2012  05:10 PM    <DIR>          ..
10/04/2012  05:10 PM                302 author.xml
10/04/2012  05:05 PM                5,632 AuthorSerialization.exe
10/04/2012  05:05 PM            13,824 AuthorSerialization.pdb
10/04/2012  05:05 PM             4,608 AuthorServiceLibrary.dll
10/04/2012  05:05 PM             7,680 AuthorServiceLibrary.pdb
               5 File(s)              32,046 bytes
               2 Dir(s)  71,406,272,512 bytes free

D:\AuthorSerialization\bin\Debug>_

```

Now open the author.xml file in Internet Explorer.



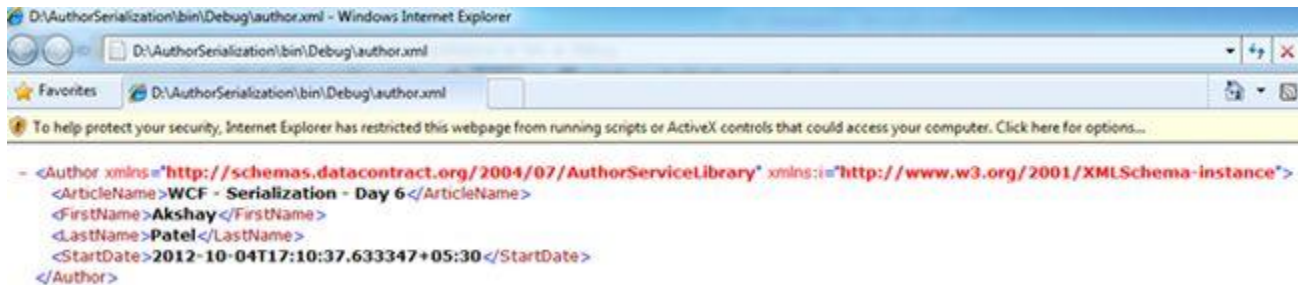
```

D:\AuthorSerialization\bin\Debug>autor.xml_

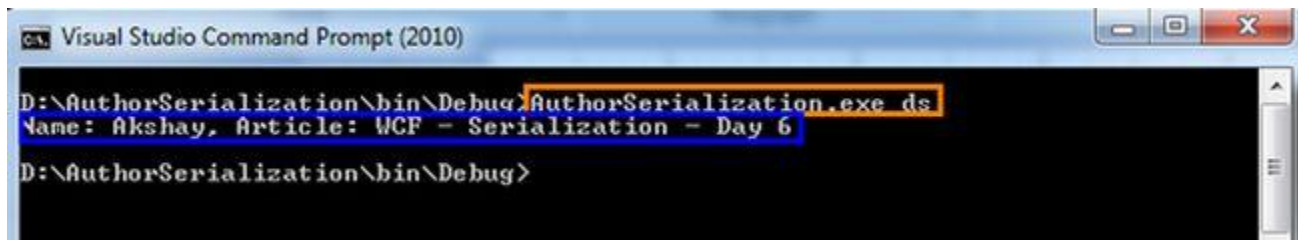
```

Here in author.xml "Author" is a root element because we declared a DataContract with this name. In this element you can see ArticleName, FirstName, LastName, StartDate and their values which we have

supplied. The very important thing you can observe is that all elements in the author element is in alphabetical order. Here we are not setting the order attribute so by default the DataContract is generated in alphabetical order.



Previously we ran AuthorSerialization.exe without a parameter i.e. we do a serialization process; now the deserialize method is called. For that run AuthorSerialization.exe with the "ds" parameter.

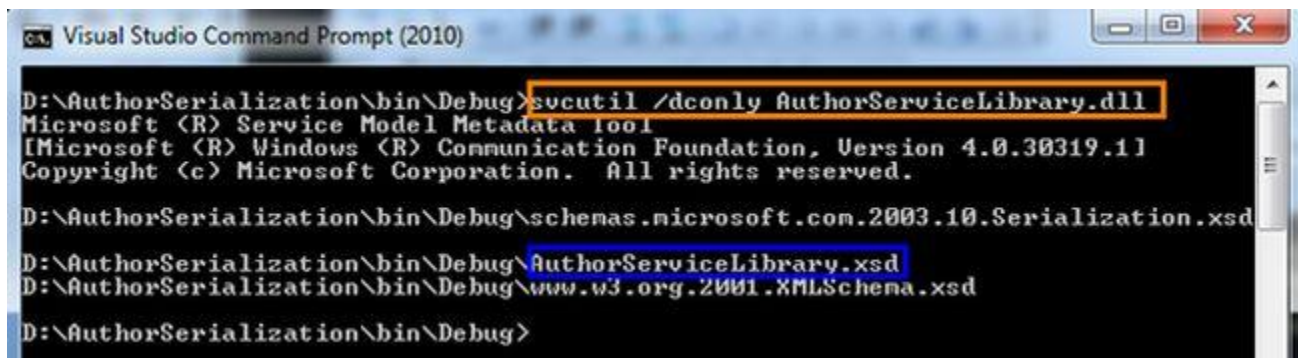


Now generate a XSD of our DataContract with the help of svcutil.exe.

Run the following command: svcutil /dconly AuthorServiceLibrary.dll

In the above command "dconly" means DataContract only.

Once you run this command, it will generate a .xsd file.



Now open AuthorServiceLibrary.xsd in notepad.




```

AuthorServiceLibrary - Notepad
File Edit Format View Help
<?xml version="1.0" encoding="utf-8"?>
<xs:schema xmlns:tns="http://schemas.datacontract.org/2004/07/AuthorServiceLibrary"
elementFormDefault="qualified"
targetNamespace="http://schemas.datacontract.org/2004/07/AuthorServiceLibrary"
xmlns:xs="http://www.w3.org/2001/XMLSchema">
  <xs:complexType name="Author">
    <xs:sequence>
      <xs:element minOccurs="0" name="ArticleName" nillable="true" type="xs:string" />
      <xs:element minOccurs="0" name="FirstName" nillable="true" type="xs:string" />
      <xs:element minOccurs="0" name="LastName" nillable="true" type="xs:string" />
      <xs:element minOccurs="0" name="StartDate" type="xs:dateTime" />
    </xs:sequence>
  </xs:complexType>
  <xs:element name="Author" nillable="true" type="tns:Author" />
  <xs:complexType name="Service1">
    <xs:sequence />
  </xs:complexType>
  <xs:element name="Service1" nillable="true" type="tns:Service1" />
</xs:schema>

```

Now with the same svcutil generate the .cs file. You can see in the command prompt.

```

Visual Studio Command Prompt (2010)
D:\AuthorSerialization\bin\Debug>svcutil /donly AuthorServiceLibrary.xsd
Microsoft (R) Service Model Metadata Tool
[Microsoft (R) Windows (R) Communication Foundation, Version 4.0.30319.11
Copyright (c) Microsoft Corporation. All rights reserved.

Generating files...
D:\AuthorSerialization\bin\Debug\AuthorServiceLibrary.cs
D:\AuthorSerialization\bin\Debug>

```

Now open the AuthorServiceLibrary.cs file in Notepad.

```

Visual Studio Command Prompt (2010)
D:\AuthorSerialization\bin\Debug>notepad AuthorServiceLibrary.cs
D:\AuthorSerialization\bin\Debug>

```

This is our original code generated back by the tool.

```

//-----
// <auto-generated>
//   This code was generated by a tool.
//   Runtime Version:4.0.30319.17020
//
//   Changes to this file may cause incorrect behavior and will be lost if
//   the code is regenerated.
// </auto-generated>
//-----

```

```

namespace AuthorServiceLibrary
{

```

```
using System.Runtime.Serialization;

[System.Diagnostics.DebuggerStepThroughAttribute()]
[System.CodeDom.Compiler.GeneratedCodeAttribute("System.Runtime.Serialization", "4.0.0.0")]
[System.Runtime.Serialization.DataContractAttribute(Name = "Author", Namespace =
"http://schemas.datacontract.org/2004/07/AuthorServiceLibrary")]
public partial class Author : object, System.Runtime.Serialization.IExtensibleDataObject
{

    private System.Runtime.Serialization.ExtensionDataObject extensionDataField;

    private string ArticleNameField;

    private string FirstNameField;

    private string LastNameField;

    private System.DateTime StartDateField;

    public System.Runtime.Serialization.ExtensionDataObject ExtensionData
    {
        get
        {
            return this.extensionDataField;
        }
        set
        {
            this.extensionDataField = value;
        }
    }

    [System.Runtime.Serialization.DataMemberAttribute()]
    public string ArticleName
    {
        get
        {
            return this.ArticleNameField;
        }
        set
        {
            this.ArticleNameField = value;
        }
    }

    [System.Runtime.Serialization.DataMemberAttribute()]
    public string FirstName
```

```
{  
    get  
    {  
        return this.FirstNameField;  
    }  
    set  
    {  
        this.FirstNameField = value;  
    }  
}
```

```
[System.Runtime.Serialization.DataMemberAttribute()]  
public string LastName  
{  
    get  
    {  
        return this.LastNameField;  
    }  
    set  
    {  
        this.LastNameField = value;  
    }  
}
```

```
[System.Runtime.Serialization.DataMemberAttribute()]  
public System.DateTime StartDate  
{  
    get  
    {  
        return this.StartDateField;  
    }  
    set  
    {  
        this.StartDateField = value;  
    }  
}
```

```
[System.Diagnostics.DebuggerStepThroughAttribute()]  
[System.CodeDom.Compiler.GeneratedCodeAttribute("System.Runtime.Serialization", "4.0.0.0")]  
[System.Runtime.Serialization.DataContractAttribute(Name = "Service1", Namespace =  
"http://schemas.datacontract.org/2004/07/AuthorServiceLibrary")]  
public partial class Service1 : object, System.Runtime.Serialization.IExtensibleDataObject  
{  
  
    private System.Runtime.Serialization.ExtensionDataObject extensionDataField;
```

```
public System.Runtime.Serialization.ExtensionDataObject ExtensionData
{
    get
    {
        return this.extensionDataField;
    }
    set
    {
        this.extensionDataField = value;
    }
}
```

8 WCF Opt-In Vs. Opt-Out

Introduction

Discuss about two approaches of WCF i.e. Opt-In and Opt-Out used by DataContractSerializer and XmlSerializer. We will also see the difference between DataContractSerializer and XmlSerializer.

Opt-Out Approach

In this approach, members are assumed to be serialized except we mention it as NonSerialized. XmlSerializer uses this approach.

```
namespace AuthorServiceLibrary
{
    [ServiceContract]
    public interface IService1
    {
    }

    [Serializable()]
    public class Author
    {
        public string FirstName;
        public string LastName;

        [NonSerialized()]
        public string Article;
    }
}
```

In the above example, we set the Serializable attribute to the Author class. In other words all class members are going to be serialized but suppose we do not want to serialize the article member then set the NonSerialized attribute on article member. In this case only the firstname and lastname are serialized.

Opt-In Approach

In this approach we need to specify each member, which we want to serialize. For example we want to serialize the firstname and lastname, not an article member then we need to set the DataMember attribute to the firstname and lastname.

```
namespace AuthorServiceLibrary
{
    [ServiceContract]
    public interface IService1
    {
    }

    [DataContract]
    public class Author
    {
        [DataMember()]
        public string FirstName;

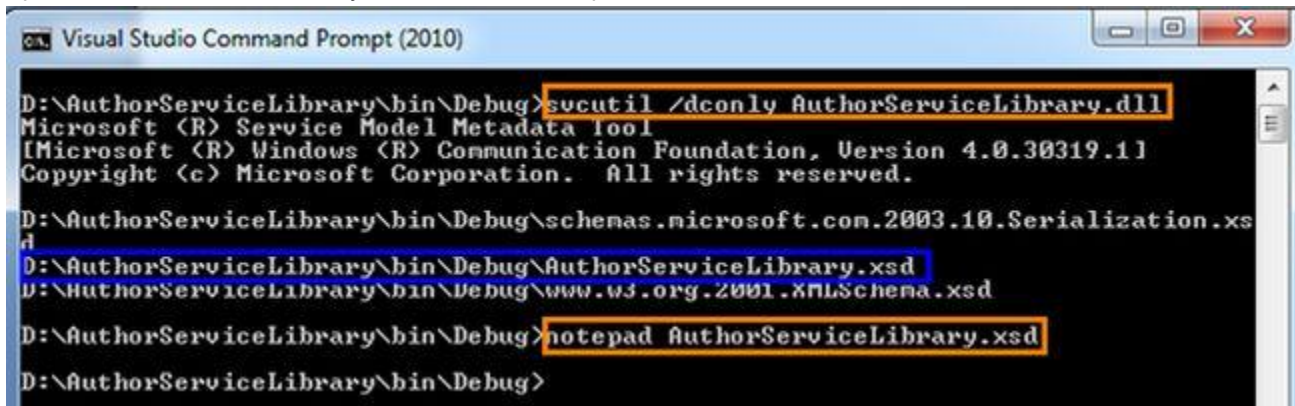
        [DataMember()]
        public string LastName;

        public string Article;
    }
}
```

In the above code snippet you can see that we are not applying the DataMember attribute to the Article member. That's why this member is not serialized.

Now check this with svcutil.exe from the command prompt. It will generate an AuthorServiceLibrary.xsd file.

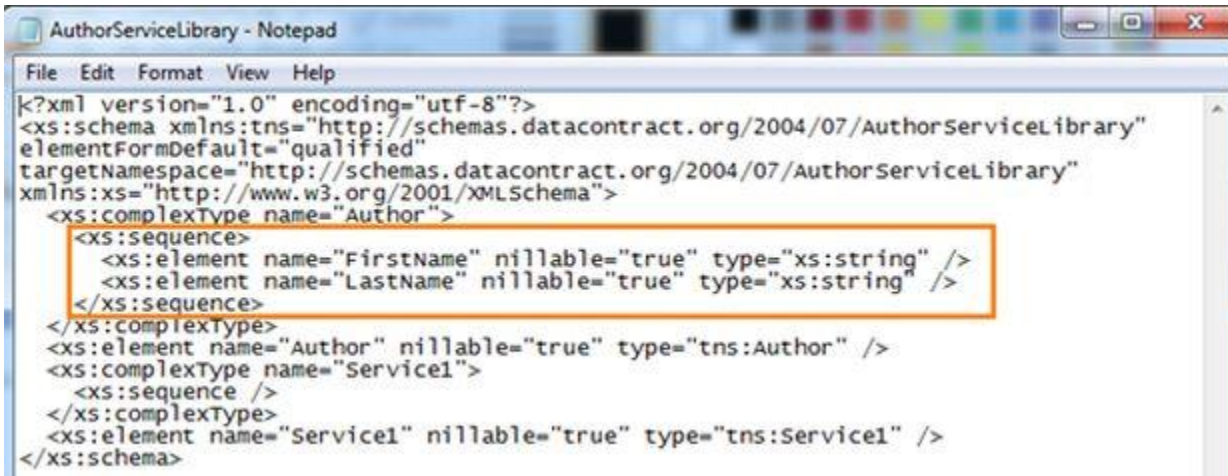
Open the AuthorServiceLibrary.xsd file in the Notepad.



```
Visual Studio Command Prompt (2010)
D:\AuthorServiceLibrary\bin\Debug>svcutil /donly AuthorServiceLibrary.dll
Microsoft (R) Service Model Metadata Tool
[Microsoft (R) Windows (R) Communication Foundation, Version 4.0.30319.1]
Copyright (c) Microsoft Corporation. All rights reserved.

D:\AuthorServiceLibrary\bin\Debug>notepad AuthorServiceLibrary.xsd
D:\AuthorServiceLibrary\bin\Debug>
```

Check the result in the Notepad file. In this file you will see only FirstName and LastName. The Article member is not serialized.



```
<?xml version="1.0" encoding="utf-8"?>
<xs:schema xmlns:tns="http://schemas.datacontract.org/2004/07/AuthorServiceLibrary"
elementFormDefault="qualified"
targetNamespace="http://schemas.datacontract.org/2004/07/AuthorServiceLibrary"
xmlns:xs="http://www.w3.org/2001/XMLSchema">
  <xs:complexType name="Author">
    <xs:sequence>
      <xs:element name="FirstName" nillable="true" type="xs:string" />
      <xs:element name="LastName" nillable="true" type="xs:string" />
    </xs:sequence>
  </xs:complexType>
  <xs:element name="Author" nillable="true" type="tns:Author" />
  <xs:complexType name="Service1">
    <xs:sequence />
  </xs:complexType>
  <xs:element name="Service1" nillable="true" type="tns:Service1" />
</xs:schema>
```

DataContractSerializer

DataContractSerializer uses the Opt-In approach. This approach serializes properties as well as fields. We can serialize protected and private members also. The DataContractSerializer is faster than XmlSerializer because we don't have full control over serialization.

XmlSerializer

XmlSerializer uses The Opt-Out approach. This approach serializes properties only and it must be a public. It cannot understand the DataContractSerializer attribute. It will not serialize unless we apply the serializable attribute.

Conclusion

The DataContractSerializer is always able to serialize to XML, but it is for very small and simple XML. It focuses on speed instead of on being comprehensive. And XmlSerializer is used for comprehensive XML schemas.

9 WCF Message Contract

Introduction

Defining WCF Message Contracts and MessageContractAttributes, MessageHeaderAttributes, MessageBodyMemberAttributes and a sample code demonstration.

In WCF services, a DataContract enables us to define the structure of the data. This data is sent in the body of our SOAP (Simple Object Access Protocol) messages. These messages may be of either inbound (request) or outbound (response) type. A message is nothing but a packet and WCF uses this packet to transfer information from source to destination. This message contains an envelope, header and body.



A Message Contract is used to control the structure of a message body and serialization process. It is also used to send / access information in SOAP headers.

Message Contract Attributes

- MessageContractAttribute
- MessageHeaderAttribute
- MessageBodyMemberAttribute

MessageContractAttribute

The MessageContract Attribute is applied to a class or structure to define our own message structure, as in:

```
[MessageContract()]
public class AuthorRequest
{
}
}
```

Properties of MessageContractAttribute

1. public bool HasProtectionLevel { get; }

HasProtectionLevel gets a value that indicates whether the message has a protection level.

It returns true if the message must be encrypted, signed, or both; otherwise false. The default is false.

2. public bool IsWrapped { get; set; }

IsWrapped gets or sets a value that specifies whether the message body has a wrapper element.

It returns true if the message body has a wrapper element; otherwise, false. The default is true.

3. `public ProtectionLevel ProtectionLevel { get; set; }`

ProtectionLevel gets or sets a value that specified whether the message must be encrypted, signed, or both.

It returns one of the System.Net.Security.ProtectionLevel values. The default is System.Net.Security.ProtectionLevel.None.

4. `public string WrapperName { get; set; }`

WrapperName gets or sets the name of the wrapper element of the message body.

It returns the name of the wrapper element in the message body.

5. `public string WrapperNamespace { get; set; }`

WrapperNamespace gets or sets the namespace of the message body wrapper element.

It returns the wrapper element namespace.

For example:

```
[MessageContract(IsWrapped = false,ProtectionLevel=ProtectionLevel.None)]
```

```
public class AuthorRequest  
{  
  
}
```

MessageHeaderAttribute

MessageHeaderAttribute is applied to the members of the MessageContract to declare the members within the message header; see:

```
[MessageContract(IsWrapped = false,ProtectionLevel=ProtectionLevel.None)]
```

```
public class AuthorRequest  
{  
    [MessageHeader()]  
    public string AuthorId;  
}
```

Properties of MessageHeaderAttribute

1. `public string Name { get; set; }`

Name specifies the name of the element that corresponds to this member.

It returns the name of the element that corresponds to this member. This string must be a valid XML element name.

2. `public string Namespace { get; set; }`

Namespace specifies the namespace of the element that corresponds to this member.

It returns a namespace URI of the element that corresponds to this member.

3. `public ProtectionLevel ProtectionLevel { get; set; }`

ProtectionLevel specifies whether the member is to be transmitted as-is, signed, or signed and encrypted.

It returns one of the `System.Net.Security.ProtectionLevel` values. The default is `System.Net.Security.ProtectionLevel.None`.

4. `public string Actor { get; set; }`

Actor gets or sets a URI that indicates the node at which this header is targeted. It maps to the role header attribute when SOAP 1.2 is used and the actor header attribute when SOAP 1.1 is used.

It returns a URI that indicates the node at which this header is targeted. This URI maps to the role header attribute when SOAP 1.2 is used and the actor header attribute when SOAP 1.1 is used.

5. `public bool MustUnderstand { get; set; }`

MustUnderstand specifies whether the node acting in the `System.ServiceModel.MessageHeaderAttribute.Actor` role must understand this header. This is mapped to the `mustUnderstand` SOAP header attribute.

It returns true if the node acting in the `System.ServiceModel.MessageHeaderAttribute.Actor` role must understand this header; otherwise, false.

6. `public bool Relay { get; set; }`

Relay specifies whether this header is to be relayed to downstream nodes. This is mapped to the `relay` SOAP header attribute.

It returns true if this header is to be relayed to downstream nodes; otherwise, false.

MessageBodyMemberAttribute

`MessageBodyMemberAttribute` is applied to the members of message contracts to declare the members within the message body.

Properties of MessageBodyMemberAttribute

Name, Namespace, ProtectionLevel and Order are the properties of `MessageBodyMemberAttribute`. As we are now

©2013 C# CORNER.

SHARE THIS DOCUMENT AS IT IS. PLEASE DO NOT REPRODUCE, REPUBLISH, CHANGE OR COPY.

familiar with all the properties, I am not explaining here once again.

Why should we use MessageContract

Suppose we are not using MessageContract in our service, by default SOAP body element contains a child wrapper element of operation name and wraps parameters. If there is no parameter then this element will be empty in the inbound (request) message. In the same way, in an outbound (response) message, if it is of type void or say nothing to return then this element will be empty.

For example we create a service and set MessageContract with the property IsWrapped to false. Then there is no child element in the SOAP body. You will see a collection of MessageBodyMembers directly under the SOAP body within the MessageContract.

Controlling wrapping is important when we deal with the other platforms except WCF because they might serialize their SOAP messages differently.

Sample Code

Consider a scenario where we create a service and in this service we have one operation contract which returns the author information. Now we want to set some authentication like AuthorId should be

matched. In this case we store AuthorId in MessageHeader because it is safe. Here we create two MessageContracts, one is for the request i.e. AuthorRequest and another is for the response i.e. AuthorResponse.

While creating a MessageContract consider the following two points:

1. When we pass a parameter of type MessageContract, only one parameter is used in the service operation.

```
[ServiceContract]
public interface IService1
{
    [OperationContract]
    [FaultContract(typeof(string))]
    AuthorResponse GetAuthorInfo(AuthorRequest Request);
}
```

2. The return type of the OperationContract is of MessageContractType or Void type.

```
[ServiceContract]
public interface IService1
{
    [OperationContract]
    [FaultContract(typeof(string))]
    AuthorResponse GetAuthorInfo(AuthorRequest Request);
}
```

Now add the following lines of code to your Interface.

```
[ServiceContract]
public interface IService1
{
    [OperationContract]
    [FaultContract(typeof(string))]
    AuthorResponse GetAuthorInfo(AuthorRequest Request);
}

[DataContract]
public class Author
{
    [DataMember(Name="FirstName",Order=1)]
    public string FirstName;

    [DataMember(Name = "LastName", Order = 2)]
    public string LastName;

    [DataMember(Name = "ArticleName", Order = 3)]
    public string Article;
}

[MessageContract(IsWrapped = false)]
public class AuthorRequest
{
    [MessageHeader(Name="AuthorIdentity")]
    public string AuthorId;
}

[MessageContract(IsWrapped = false)]
public class AuthorResponse
{
    [MessageBodyMember]
    public Author AuthorInfo;
}
```

Now add an implementation of the above operation contract in the service class.

```
private const string AuthorId = "db2972";
public AuthorResponse GetAuthorInfo(AuthorRequest Request)
{
    if (Request.AuthorId != AuthorId)
    {
        string Error = "Invalid Author Id.";
        throw new FaultException<string>(Error);
    }

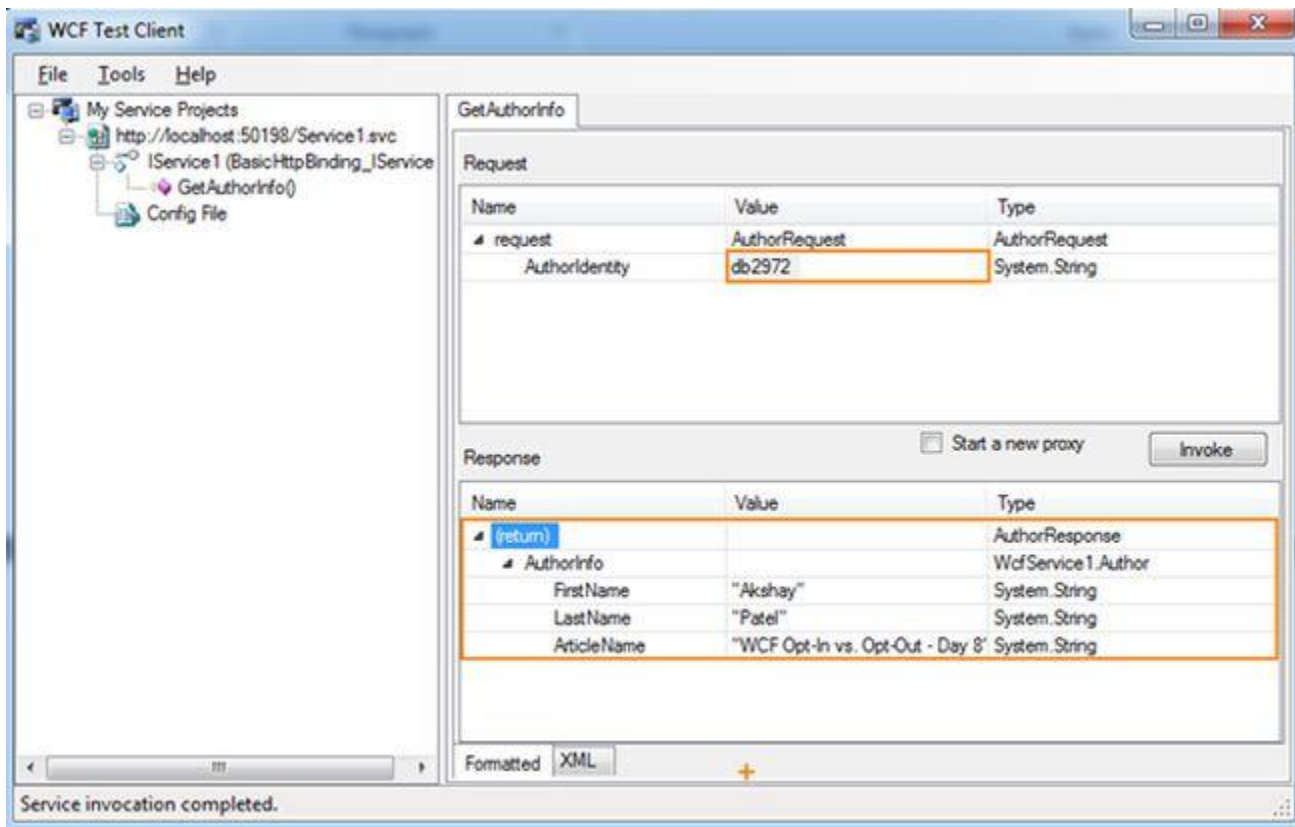
    AuthorResponse Response = new AuthorResponse();
    Response.AuthorInfo = new Author();

    Response.AuthorInfo.FirstName = "Akshay";
    Response.AuthorInfo.LastName = "Patel";
    Response.AuthorInfo.Article = "WCF Opt-In vs. Opt-Out - Day 8";

    return Response;
}
```

Test Client Output (IsWrapped = false)

Insert 'db2972' as the input for the AuthorIdentity parameter; see:



SOAP Request

An AuthorIdentity element is in the SOAP request as we pass this under the MessageHeader. And the Body element is

©2013 C# CORNER.

SHARE THIS DOCUMENT AS IT IS. PLEASE DO NOT REPRODUCE, REPUBLISH, CHANGE OR COPY.

empty because we are not passing any value.

```
Request
<s:Envelope xmlns:s="http://schemas.xmlsoap.org/soap/envelope/">
  <s:Header>
    <Action s:mustUnderstand="1" xmlns="http://schemas.microsoft.com/ws/2005/05/addressing/none" http://tempuri.org/IService1/GetAuthorInfo/>
    <h:AuthIdentity xmlns:h="http://tempuri.org/">db2972</h:AuthIdentity>
  </s:Header>
  <s:Body />
</s:Envelope>
```

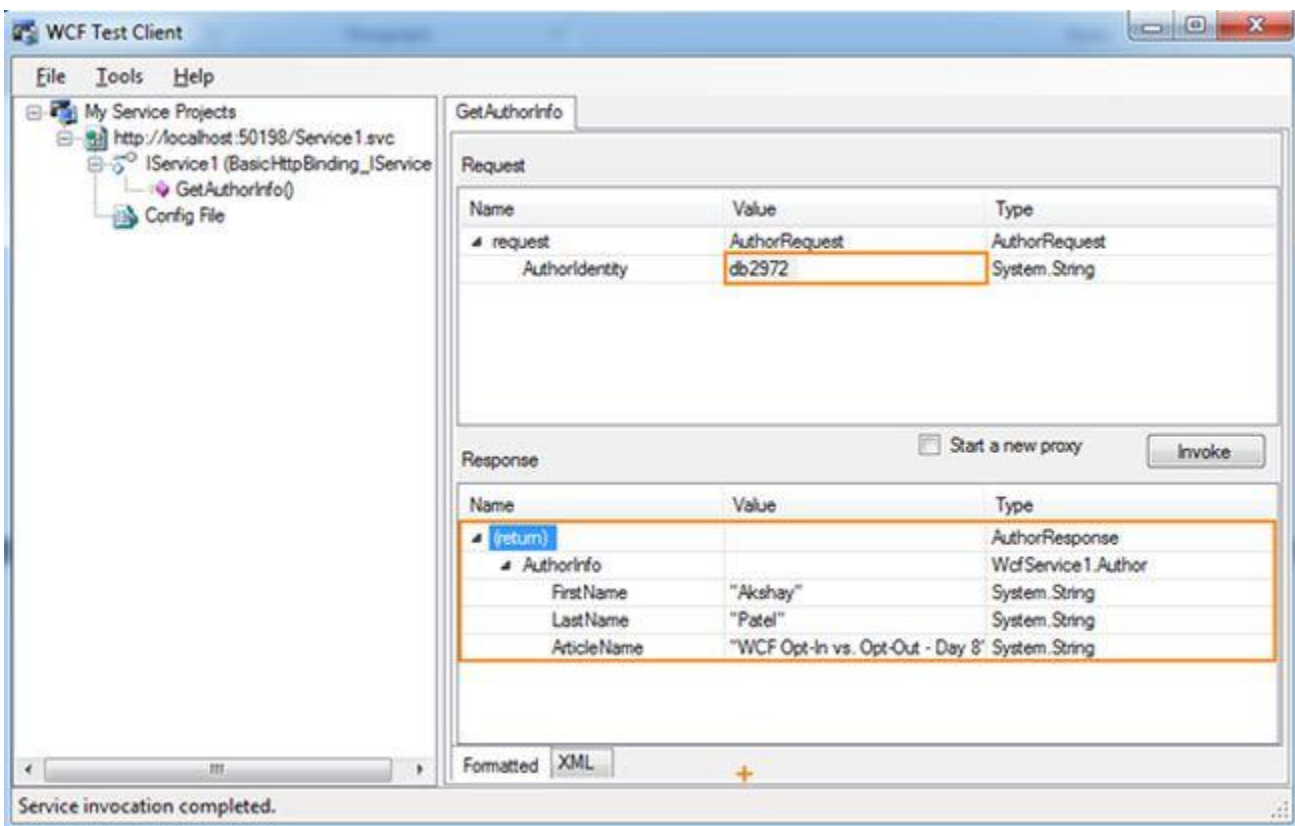
SOAP Response

In response we get author information directly under the body element as we set IsWrapped to false.

```
Response
<s:Envelope xmlns:s="http://schemas.xmlsoap.org/soap/envelope/">
  <s:Header />
  <s:Body>
    <AuthorInfo xmlns="http://tempuri.org/" xmlns:a="http://schemas.datacontract.org/2004/07/WcfService1" xmlns:i="http://www.w3.org/2001/XMLSchema-instance">
      <a:FirstName>Akshay</a:FirstName>
      <a:LastName>Patel</a:LastName>
      <a:ArticleName>WCF Opt-In vs. Opt-Out - Day 8</a:ArticleName>
    </AuthorInfo>
  </s:Body>
</s:Envelope>
```

Test Client Output (IsWrapped = true)

Now change the code, set IsWrapped to true in the MessageContract and once again check the result in Test Client.



The screenshot shows the WCF Test Client interface. On the left, the service 'http://localhost:50198/Service1.svc' is selected, and the 'GetAuthorInfo()' operation is chosen. The 'Request' tab is active, showing a table with the following data:

Name	Value	Type
request	AuthorRequest	AuthorRequest
AuthIdentity	db2972	System.String

Below the request, the 'Response' tab is active, showing a table with the following data:

Name	Value	Type
(return)		AuthorResponse
AuthorInfo		WcfService1.Author
FirstName	"Akshay"	System.String
LastName	"Patel"	System.String
ArticleName	"WCF Opt-In vs. Opt-Out - Day 8"	System.String

The status bar at the bottom indicates 'Service invocation completed.'

SOAP Request

47

AuthorRequest element is empty and is added under the body element because we set IsWrapped to true.

Request

```
<s:Envelope xmlns:s="http://schemas.xmlsoap.org/soap/envelope/">
  <s:Header>
    <Action s:mustUnderstand="1" xmlns="http://schemas.microsoft.com/ws/2005/05/addressing/none">http://tempuri.org/Service1/GetAuthorInfo</Action>
    <h:AuthIdentity xmlns:h="http://tempuri.org/">db2972</h:AuthIdentity>
  </s:Header>
  <s:Body>
    <AuthorRequest xmlns="http://tempuri.org/" />
  </s:Body>
</s:Envelope>
```

SOAP Response

In the same way in the response also the AuthorResponse element is added under the body element.

Response

```
<s:Envelope xmlns:s="http://schemas.xmlsoap.org/soap/envelope/">
  <s:Header />
  <s:Body>
    <AuthorResponse xmlns="http://tempuri.org/">
      <AuthorInfo xmlns:a="http://schemas.datacontract.org/2004/07/WcfService1" xmlns:i="http://www.w3.org/2001/XMLSchema-instance">
        <a:FirstName>Akshay</a:FirstName>
        <a:LastName>Patel</a:LastName>
        <a:ArticleName>WCF Opt-In vs. Opt-Out - Day 8</a:ArticleName>
      </AuthorInfo>
    </AuthorResponse>
  </s:Body>
</s:Envelope>
```

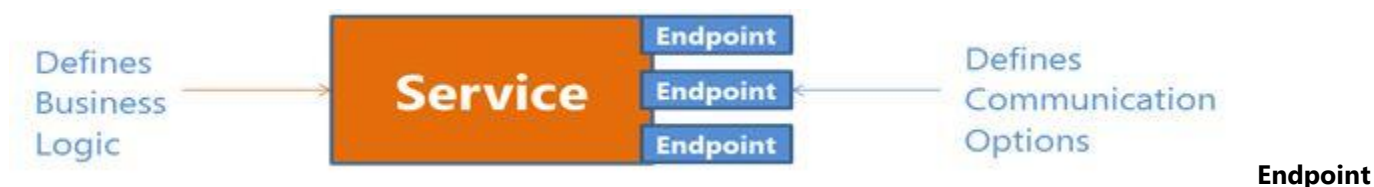
Conclusion

I explained the IsWrapped property with an example. You can download the source code and check the output by changing other properties values. The ProtectionLevel property is very important and we will discuss it later on in future articles.

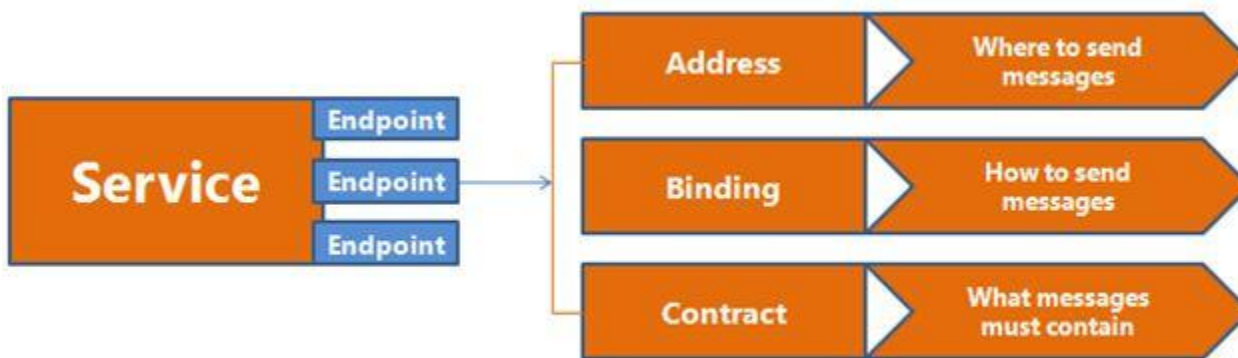
10 WCF Address Binding and Contract

Introduction

We will discuss endpoints which consists of Address, Binding and Contract.



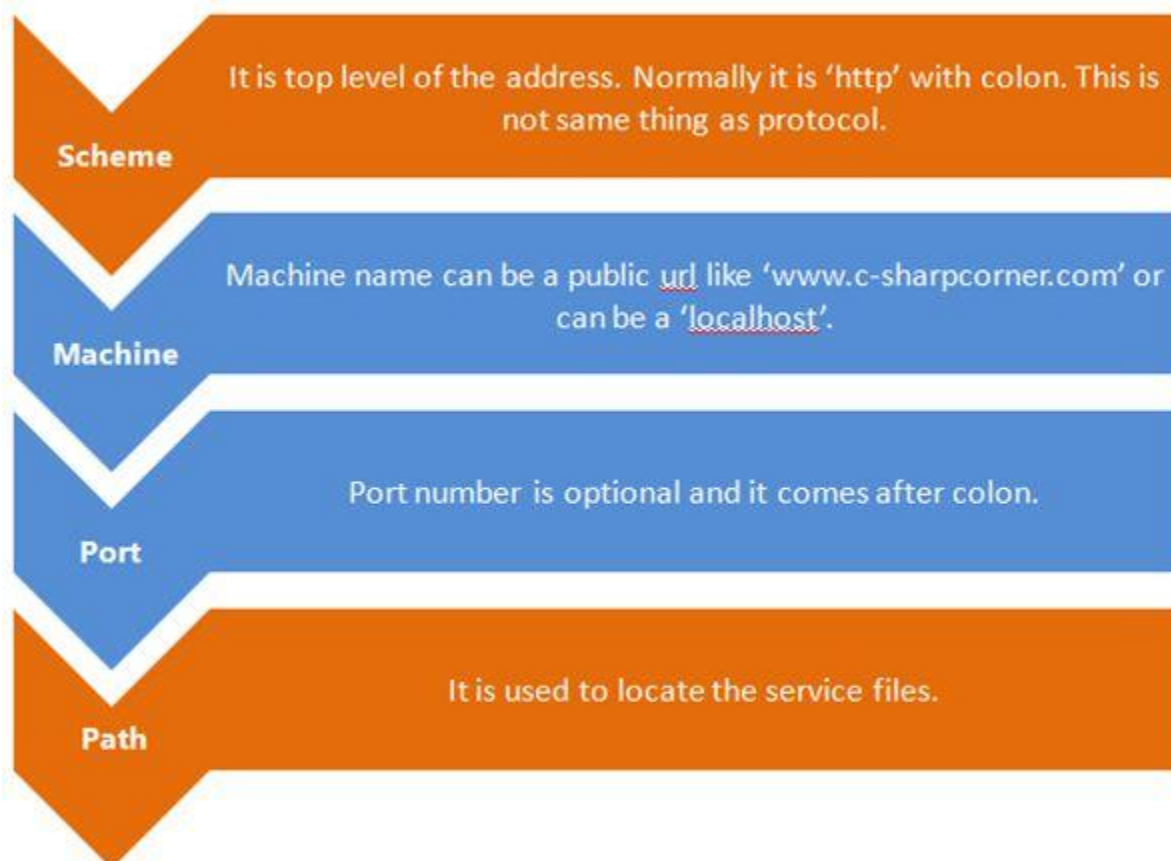
An Endpoint is a piece of information that tells WCF how to build the runtime communication channels to send and receive messages. An endpoint consists of the three things address, binding and contract.



Address

Address - Where - Where to send messages

An Address is a unique Uniform Resource Locator (URI) that identifies the location of the service. It defines the network address for sending and receiving the messages. It is divided into four parts:



Binding

Binding - How - How to send messages

A Binding specifies which transport protocol to use, what message format and which any of the ws* protocols we want to use to a particular endpoint.

BasicHttpBinding

- Replacement for earlier Web Service based on ASMX (Active Server Methods)
- Supports:
 - HTTP - Hypertext Transfer Protocol
 - HTTPS - Hypertext Transfer Protocol over SSL
 - MTOM - Message Transmission Optimization Mechanism encoding methods.

wsHttpBinding

- Uses SOAP over HTTP and supports reliability, transactions and security over the internet.
- Supports:
 - HTTP - Hypertext Transfer Protocol
 - HTTPS - Hypertext Transfer Protocol over SSL
 - MTOM - Message Transmission Optimization Mechanism encoding methods.

wsDualHttpBinding

- Used for duplex service contract because it supports bidirectional communication.

webHttpBinding

- Sends information directly over HTTP or HTTPS without creating a SOAP envelope
- It is a good choice when SOAP is not required by the client

NetTcpBinding

- Used to send binary-encoded SOAP messages from one computer to another
- Uses TCP (Transmission Control Protocol) and includes support for reliability, transactions and security

NetPeerTcpBinding

- Used for peer-to-peer communication over TCP
- Communication should occur between two or more computers

netNamedPipeBinding

- Binary encoded SOAP messages are sent over named pipes
- Used on a single WCF computer

netMSMQBinding

- Queued binding is used to send binary-encoded SOAP messages over MSMQ
- Communication should occur between two computers

Contract

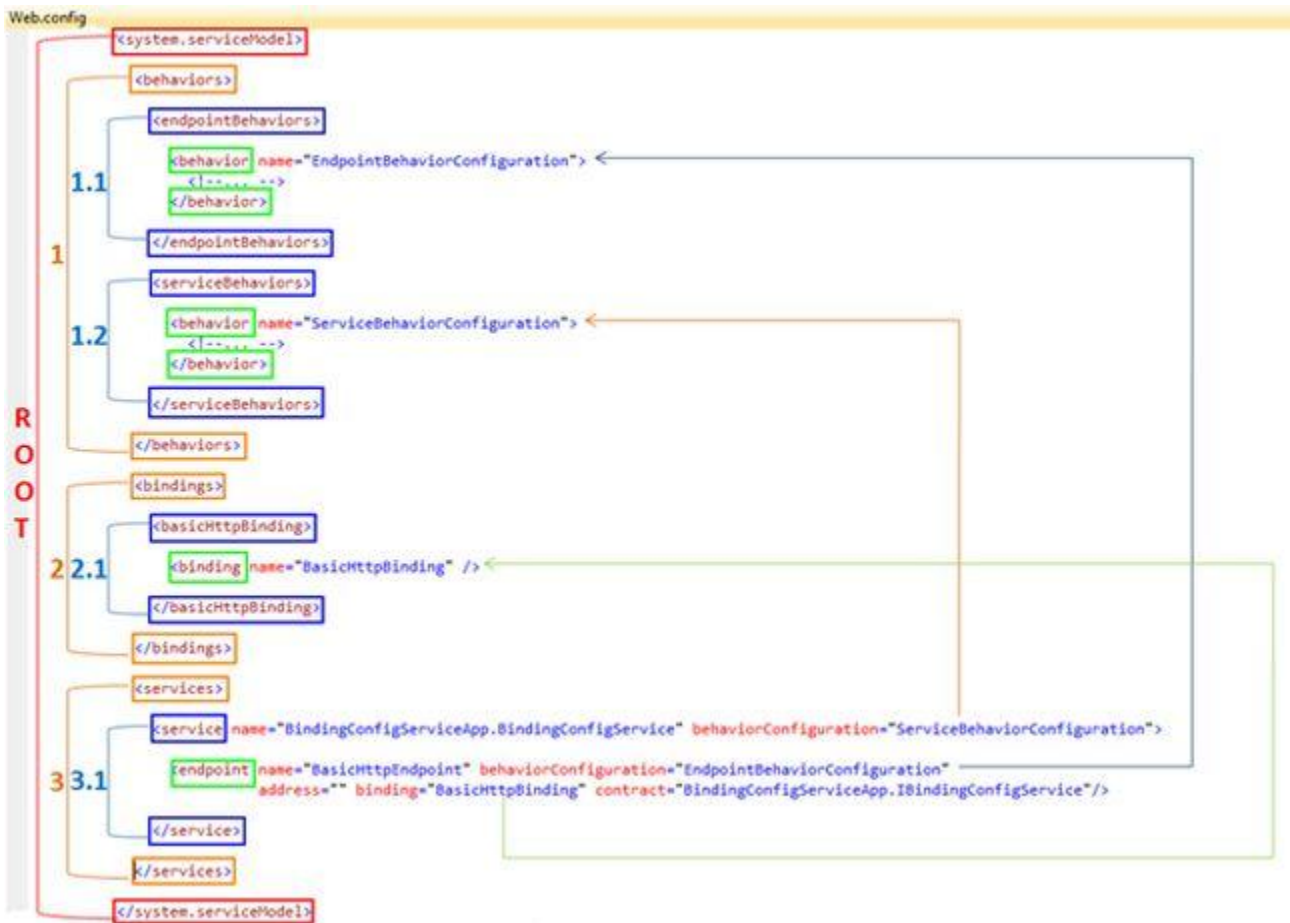
A Contract provides the additional detail about the structuring contents of the various messages that will be used by the various operations exposed to the particular endpoints. For example we create a service "TopupService", the interface used to define the service is "ITopupService" and the project name is "Utility". The contract for this service would be Utility.ITopupService.

11 WCF Service Configuration Using Web.Config

Introduction

Here we will see how to configure a service using a XML-based configuration file i.e. web.config. We will also see definition of endpoints, multiple endpoints and publishing metadata.

To understand service configuration, first create a new project using the WCF Service Application template. Once you have completed the service application creation part, open the web.config file. The configuration information for a service is contained within a system.servicemodel element in the web.config file. Under the first level of the system.servicemodel element, there are behaviours, bindings and services. We can define various behaviours like endpoint behaviours and servicebehaviors under the behaviours element. In the binding section different bindings like basicHttpBinding, wsHttpBinding etc. In our example we define basicHttpBinding. Under the services element, there is a service element and under this element we can define endpoints. We must specify at least one endpoint, otherwise we will get an error during runtime.



The <system.serviceModel> Element

- system.serviceModel is a root element of all WCF configuration elements. The configuration information for a service is contained within a system.servicemodel element.

<behaviors> Element

- <endpointBehaviors>

This configuration section represents all the behaviors defined for a specific endpoint.

- <serviceBehaviors>

This configuration section represents all the behaviors defined for a specific service.

<bindings> Element

The element contains the specifications for all bindings that can be used by any endpoint defined in any service.

- **<basicHttpBinding>**

This element represents a binding that a WCF service can use to configure and expose endpoints.

- **<binding>**

The binding element can be a system provided binding or can be a custom binding.

<services> Element

The services element contains the specifications for all services the application hosts.

- **<service>**

This element contains two attributes, name and behaviorConfiguration. The Name specifies the type that provides an implementation of the ServiceContract and behaviorConfiguration specifies the name of the behaviour elements found in the behaviors element.

- **<endpoint>**

Endpoint requires address, binding and contract. We have already seen these in my previous article.

Multiple Bindings

In the preceding example we are defining a single binding i.e. basicHttpBinding. Now you may ask, can we define multiple bindings? The answer is yes, we can define multiple bindings for different clients. To define multiple bindings we need to define multiple endpoints.

When creating multiple endpoints, we should remember that the address should be unique. If the two endpoints use the same address, an error will be thrown at runtime.

You can define the same address when you have a service that uses two interfaces. Here you can have two endpoints with the same address, but the contract name must be different.

Now let's see how to create multiple bindings.

```
<services>
<service name="BindingConfigServiceApp.BindingConfigService" behaviorConfiguration="ServiceBehaviorConfiguration">
  <endpoint address="http://localhost:8080/BindingConfigService"
    binding="basicHttpBinding"
    contract="BindingConfigServiceApp.IBindingConfigService"/>
  <endpoint address="http://localhost:8080/BindingConfigService/Secure"
    binding="wsHttpBinding"
    contract="BindingConfigServiceApp.IBindingConfigService" />
  <endpoint address="net.tcp://localhost:8001/BindingConfigService"
    binding="netTcpBinding"
    contract="BindingConfigServiceApp.IBindingConfigService" />
</service>
</services>
```


Base Address

In the preceding example we specify an address as an absolute address. It is a very easy method to understand. We can also specify a relative address. When we are using multiple endpoints then it is an efficient approach to specify the relative address method.

Specify the base address under the host element for each service. We can also add multiple base addresses by using the add method.

```
<services>

<service name="BindingConfigServiceApp.BindingConfigService" behaviorConfiguration="ServiceBehaviorConfiguration">

  <host>
    <baseAddresses>
      <add baseAddress="http://localhost:8000/BindingConfigService"/>
      <add baseAddress="net.tcp://localhost:8001/BindingConfigService"/>
    </baseAddresses>
  </host>

  <endpoint address=""
    binding="basicHttpBinding"
    contract="BindingConfigServiceApp.IBindingConfigService"/>

  <endpoint address="/Secure"
    binding="wsHttpBinding"
    contract="BindingConfigServiceApp.IBindingConfigService" />

  <endpoint address=""
    binding="netTcpBinding"
    contract="BindingConfigServiceApp.IBindingConfigService" />

</service>

</services>
```

In the previous example we define multiple endpoints and there is an address like "http://localhost:8000/BindingConfigService". This portion of address is common in the first two endpoints. So we can set this common portion as the base address under the host element.

Publishing Metadata

We can publish service metadata using the HTTP-GET protocol. To expose this metadata we need to create a metadata exchange endpoint. This endpoint can append "mex" to the HTTP address. The endpoint uses mex as the address, mexHttpBinding as the binding and IMetadataExchange interface as a contract. We also need to set the attribute of the servicemetadata i.e. HttpGetEnabled to true in the servicebehaviors.

The client can access this metadata using a HTTP-GET request with a ?wsdl query string appended.

```

<system.serviceModel>
  <behaviors>
    <serviceBehaviors>
      <behavior name="ServiceBehaviorConfiguration">
        <serviceMetadata httpGetEnabled="True"/>
      </behavior>
    </serviceBehaviors>
  </behaviors>
  <services>
    <service name="BindingConfigServiceApp.BindingConfigService" behaviorConfiguration="ServiceBehaviorConfiguration">
      <endpoint address=""
        binding="basicHttpBinding"
        contract="BindingConfigServiceApp.IBindingConfigService"/>
      <endpoint address="mex"
        binding="mexHttpBinding"
        contract="IMetadataExchange" />
    </service>
  </services>
</system.serviceModel>

```

Conclusion

It is normally a good practice to use a configuration file when specifying endpoints because we can make changes in endpoints without a code recompile.

12 WCF Service Configuration Using Configure Editor

Introduction

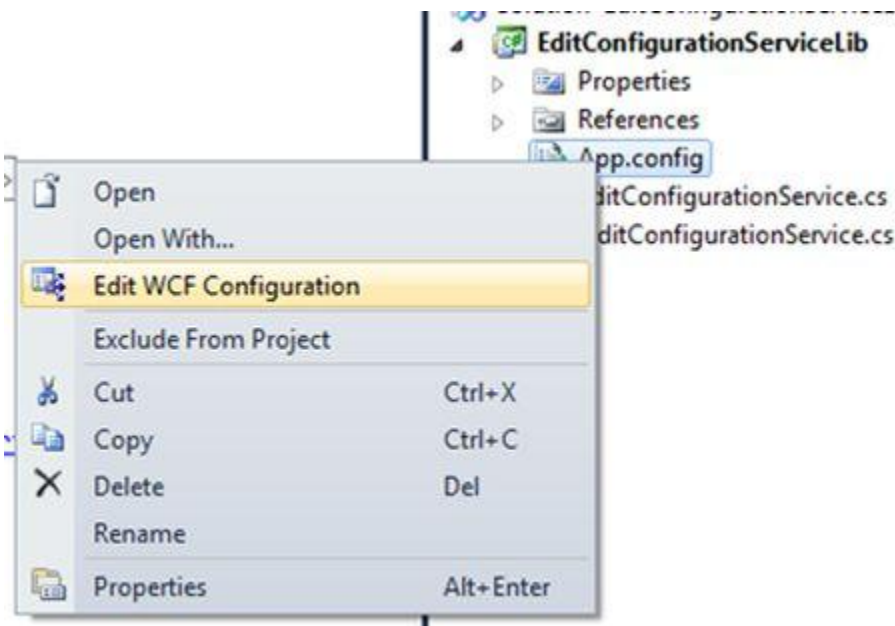
Let us configure a WCF Service using the WCF Service Configuration Editor.

Open the Visual Studio editor and select "File" -> "New" -> "Project...". Select the WCF Service Library from the template and provide an appropriate name for it.

In the web.config file you can see under the system.serviceModel element, there are two endpoints specified. One is for wsHttpBinding and another is to expose metadata using the IMetadataExchange interface. The base address is also specified under the host element. In service behavior with the serviceMetadata element set httpGetEnabled attribute to true. This is the default configuration in the WCF Service Library.

```
<system.serviceModel>
  <services>
    <service name="EditConfigurationServiceLib.EditConfigurationService">
      <endpoint address="" binding="wsHttpBinding" contract="EditConfigurati">...</endpoint>
      <endpoint address="mex" binding="mexHttpBinding" contract="IMetadataExchange" />
    </service>
  </services>
  <behaviors>
    <serviceBehaviors>
      <behavior>
        <serviceMetadata httpGetEnabled="True"/>
        <serviceDebug includeExceptionDetailInFaults="False" />
      </behavior>
    </serviceBehaviors>
  </behaviors>
</system.serviceModel>
```

Right-click on App.config and select Edit WCF Configuration to open the WCF Configuration Editor.



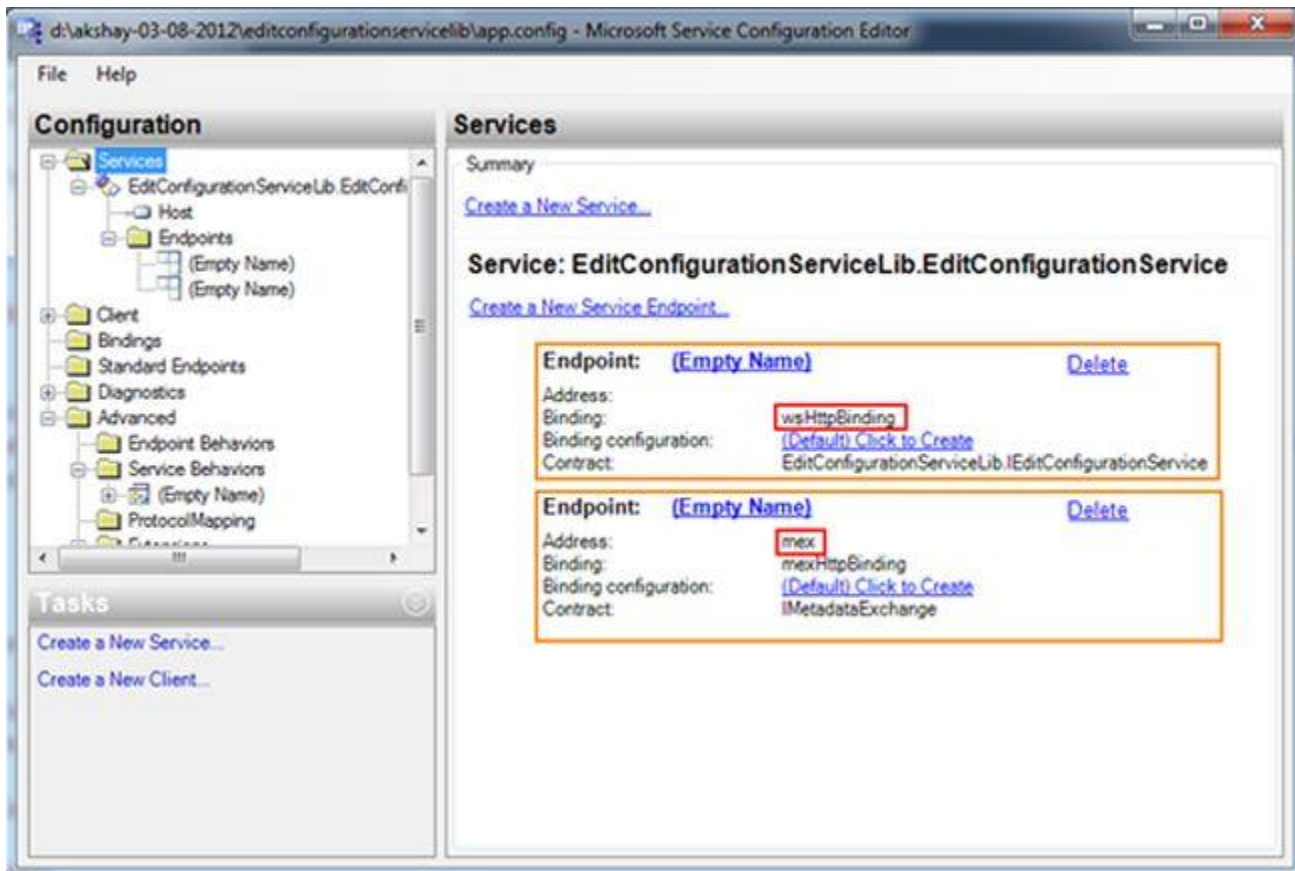
The WCF Service configuration information is contained under the system.servicemodel element. So now check how this configuration is shown in the configuration editor.

Endpoints

App.config

```
<endpoint address="" binding="wsHttpBinding" contract="EditConfigurati">...</endpoint>
<endpoint address="mex" binding="mexHttpBinding" contract="IMetadataExchange" />
```

Configuration Editor

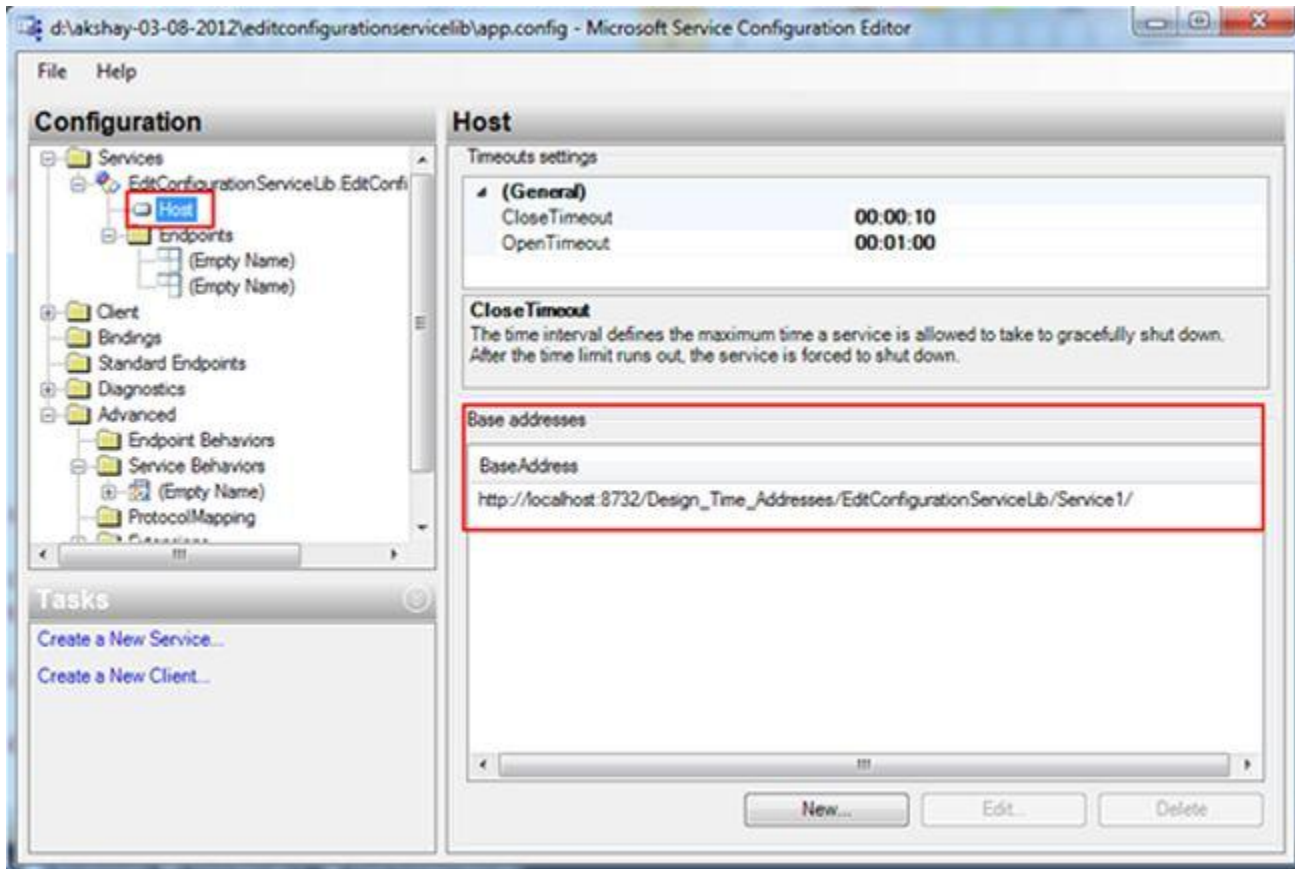


Base Address

App.config

```
<host>
  <baseAddresses>
    <add baseAddress="http://localhost:8732/Design_Time_Addresses/EditConfigurationServiceLib/Service1/" />
  </baseAddresses>
</host>
```

Configuration Editor

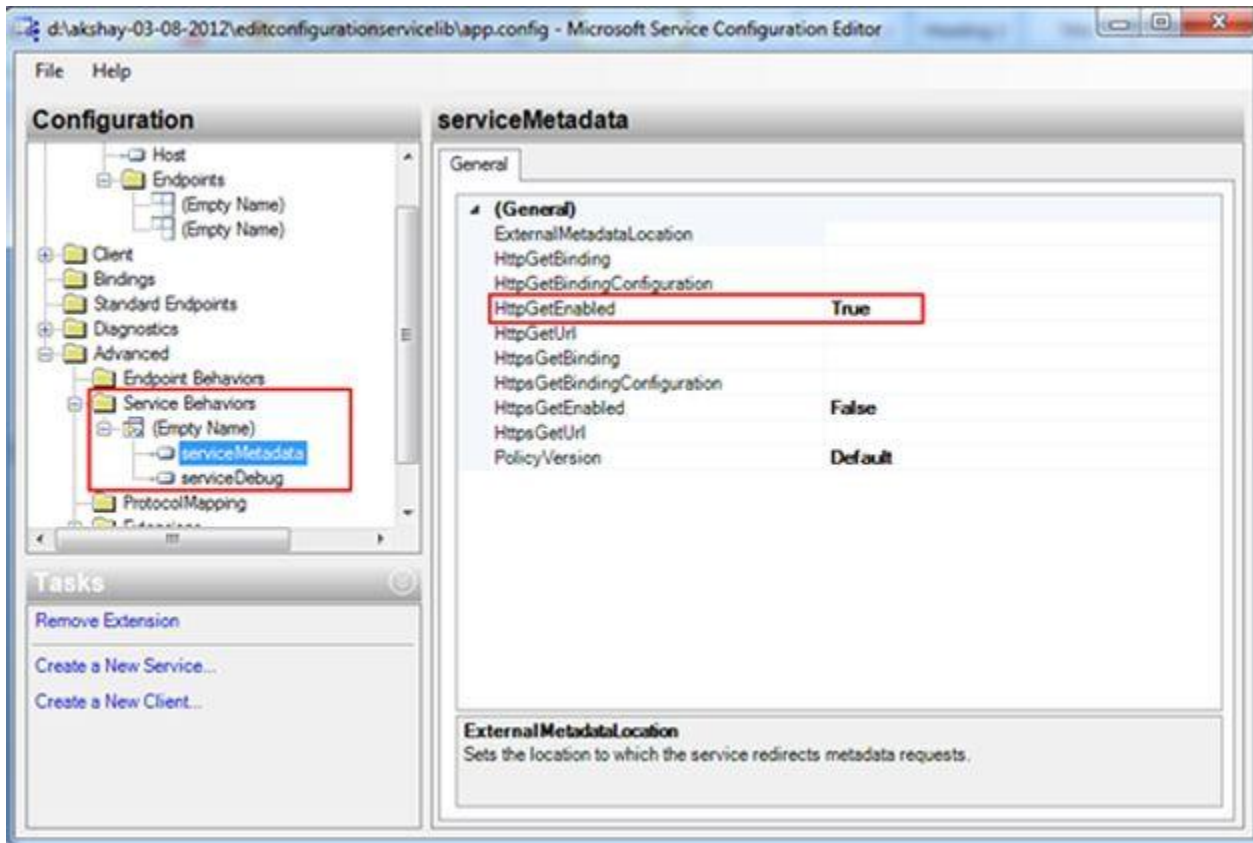


Service Behavior

App.config

```
<behaviors>
  <serviceBehaviors>
    <behavior>
      <serviceMetadata httpGetEnabled="True"/>
      <serviceDebug includeExceptionDetailInFaults="False" />
    </behavior>
  </serviceBehaviors>
</behaviors>
```

Configuration Editor

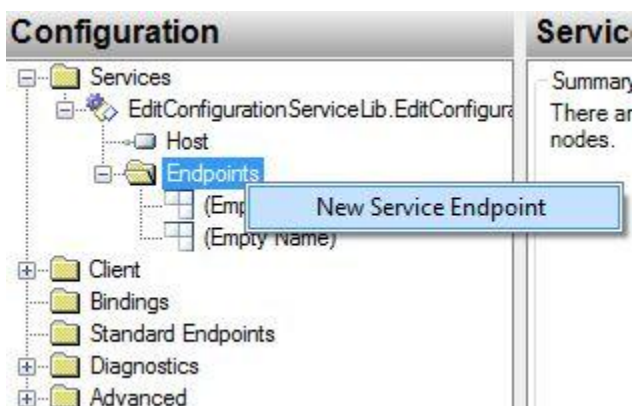


How to add new endpoint?

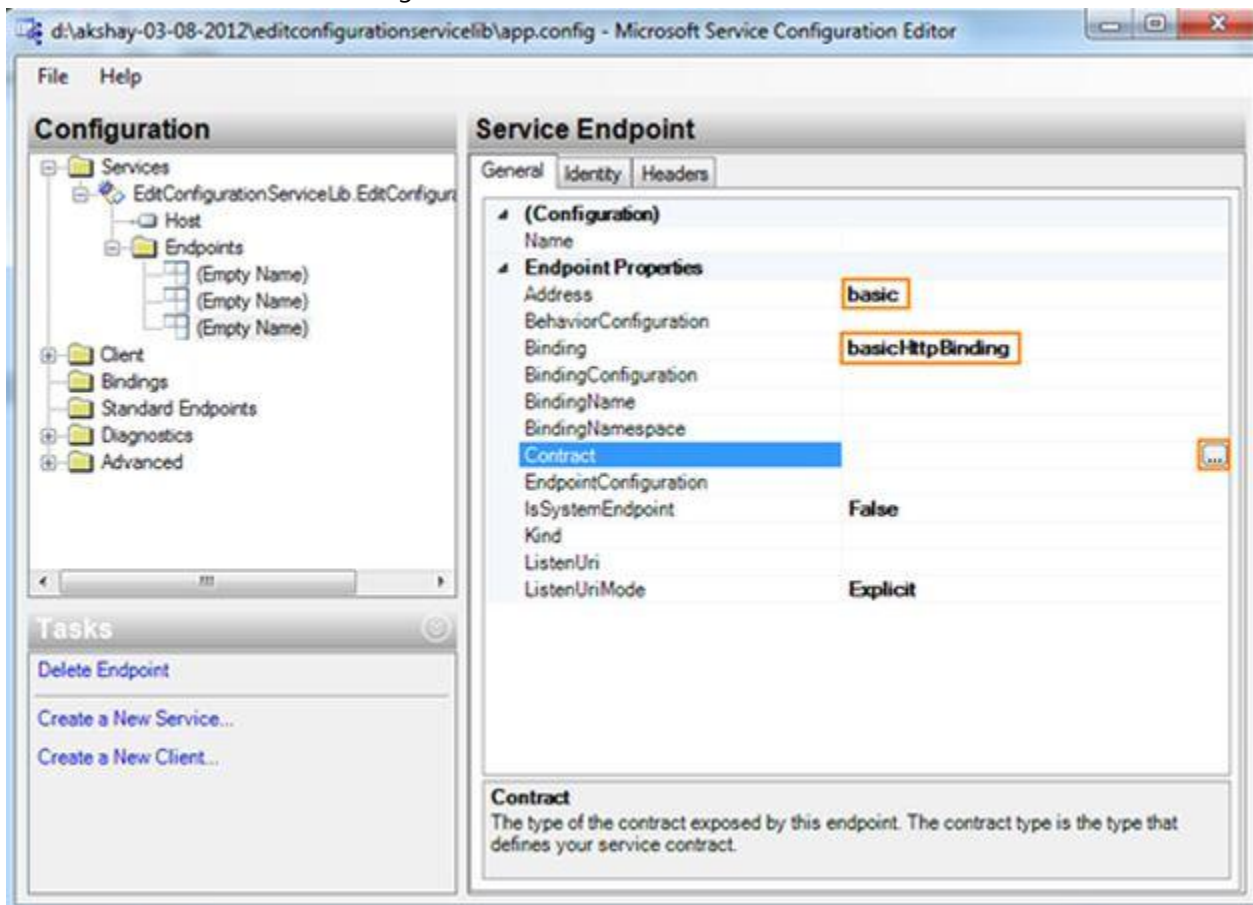
There are two approaches to add a new endpoint using this configuration editor. Use the following to create a new endpoint.

Approach 1

Right-click on the Endpoints folder, select "New Service Endpoint".



In the general tab, insert the address as "basic" and select "basicHttpBinding" for the binding. For contract click on the button which is available on the right side.

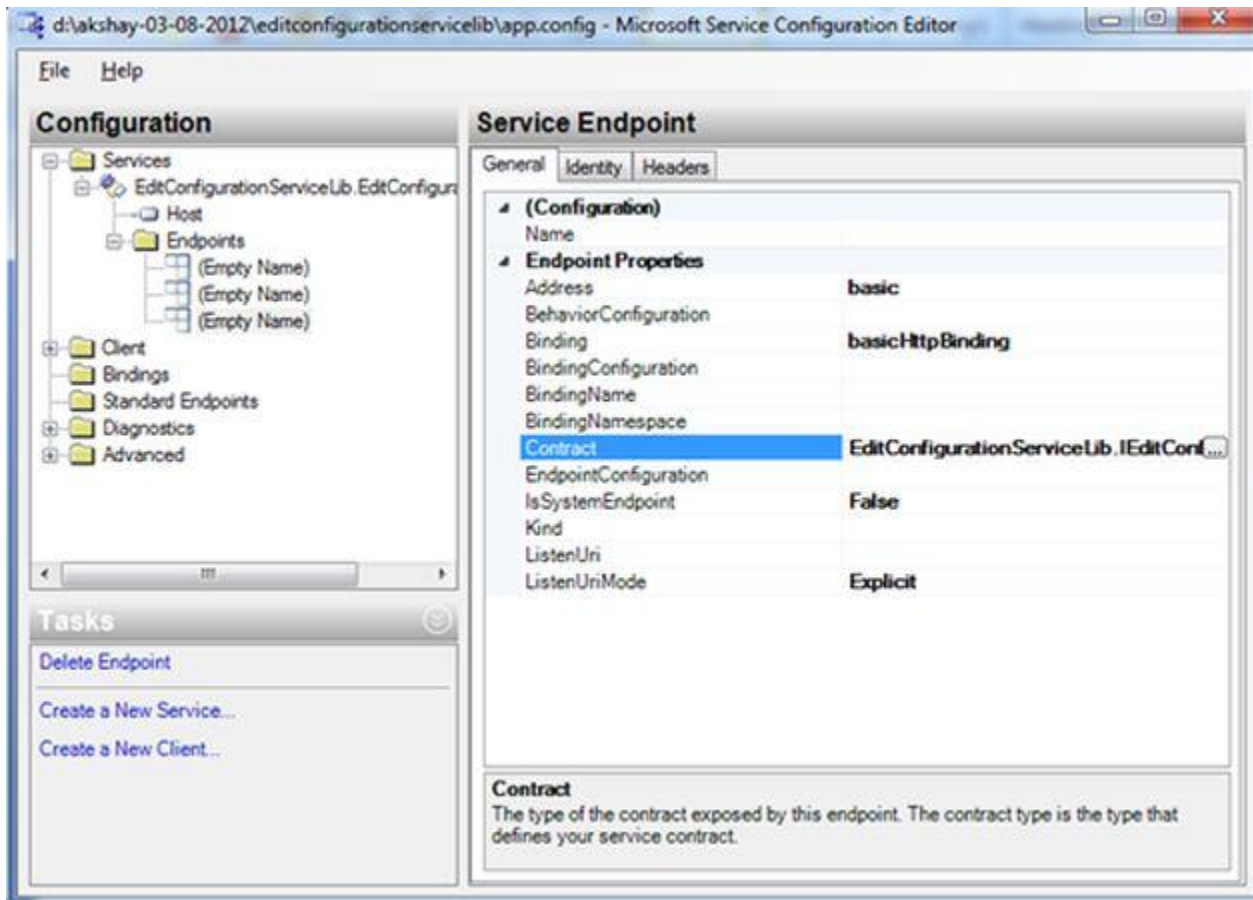


Now navigate to the path "\\bin\\Debug\\EditConfigurationServiceLib.dll" and select the contract.

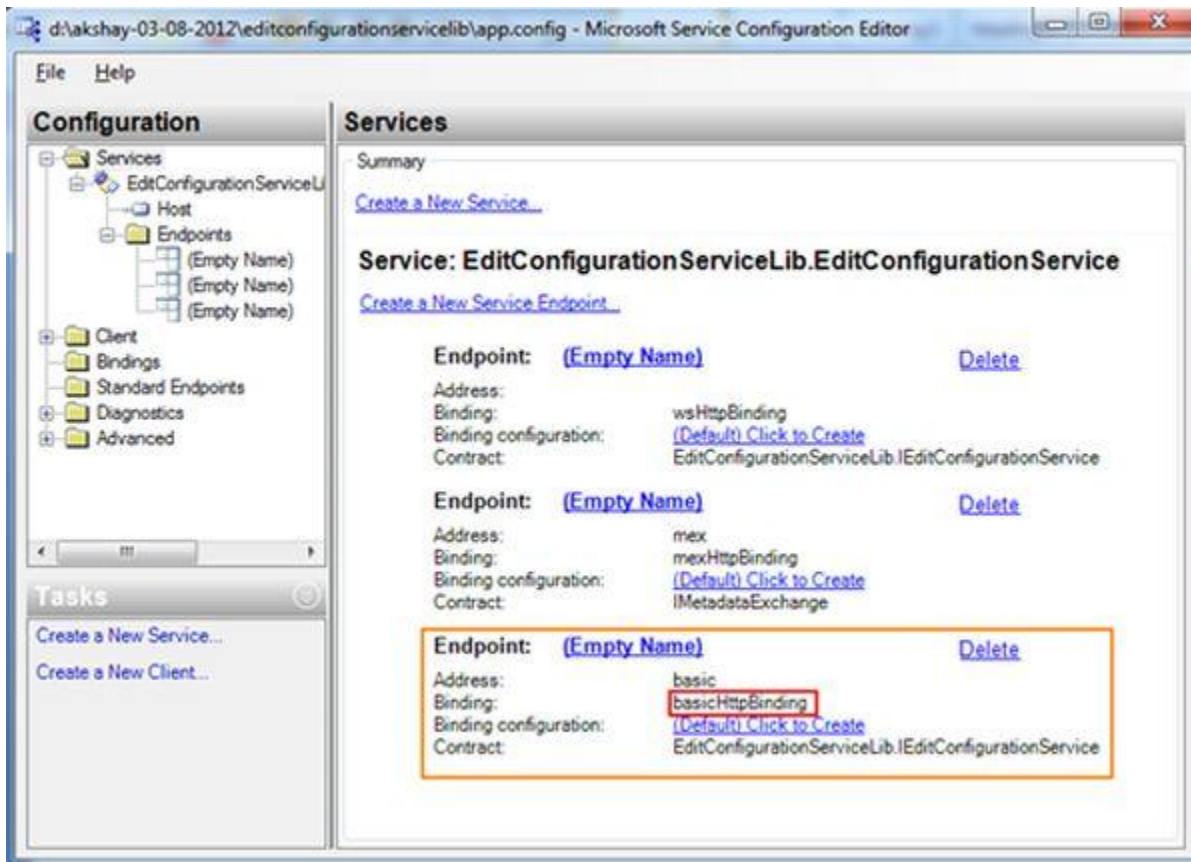


And click on the "Open" button.

This will add a contract to your binding. Now you will see an address, binding and contract for basicHttpBinding.



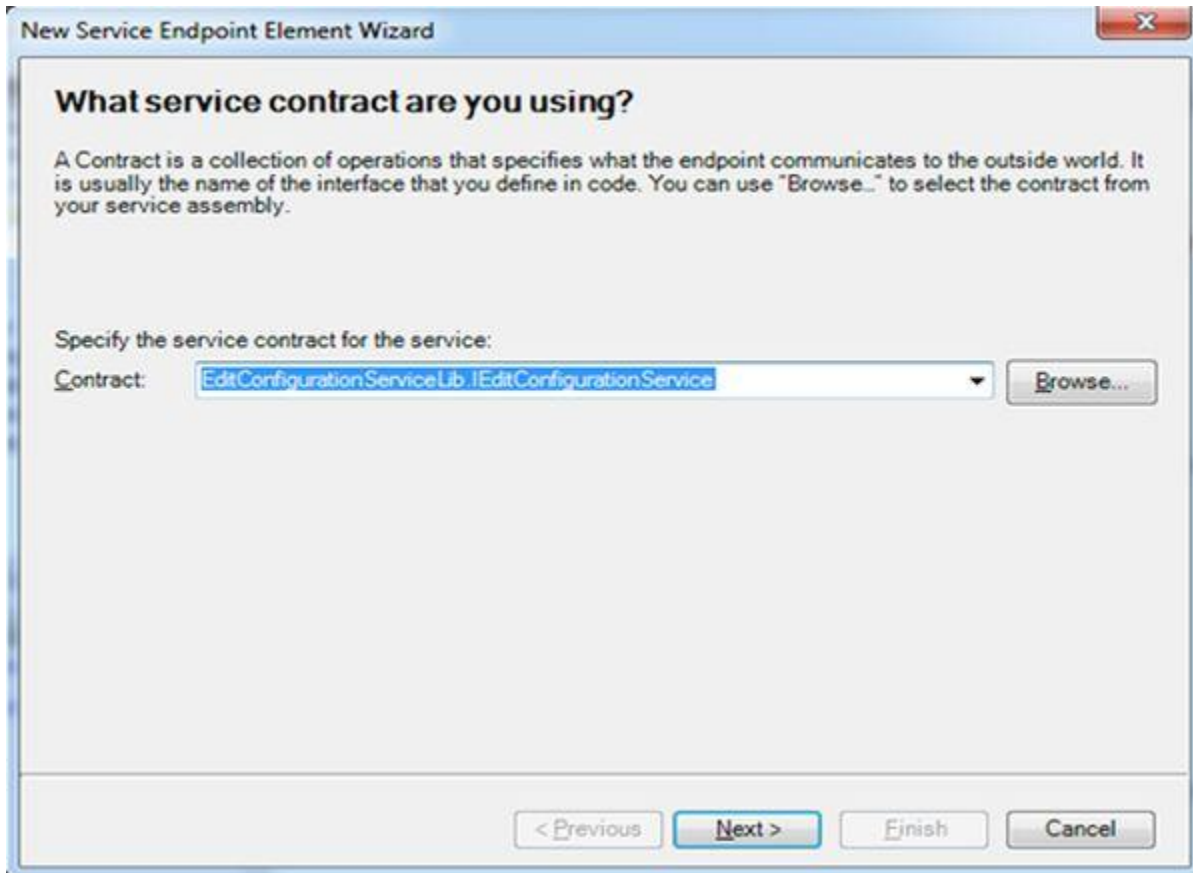
Now click on services and it will show endpoints for the service, as in:



Here you will see that one more endpoint is added, as the one covered in an orange rectangle in the image.

Approach 2

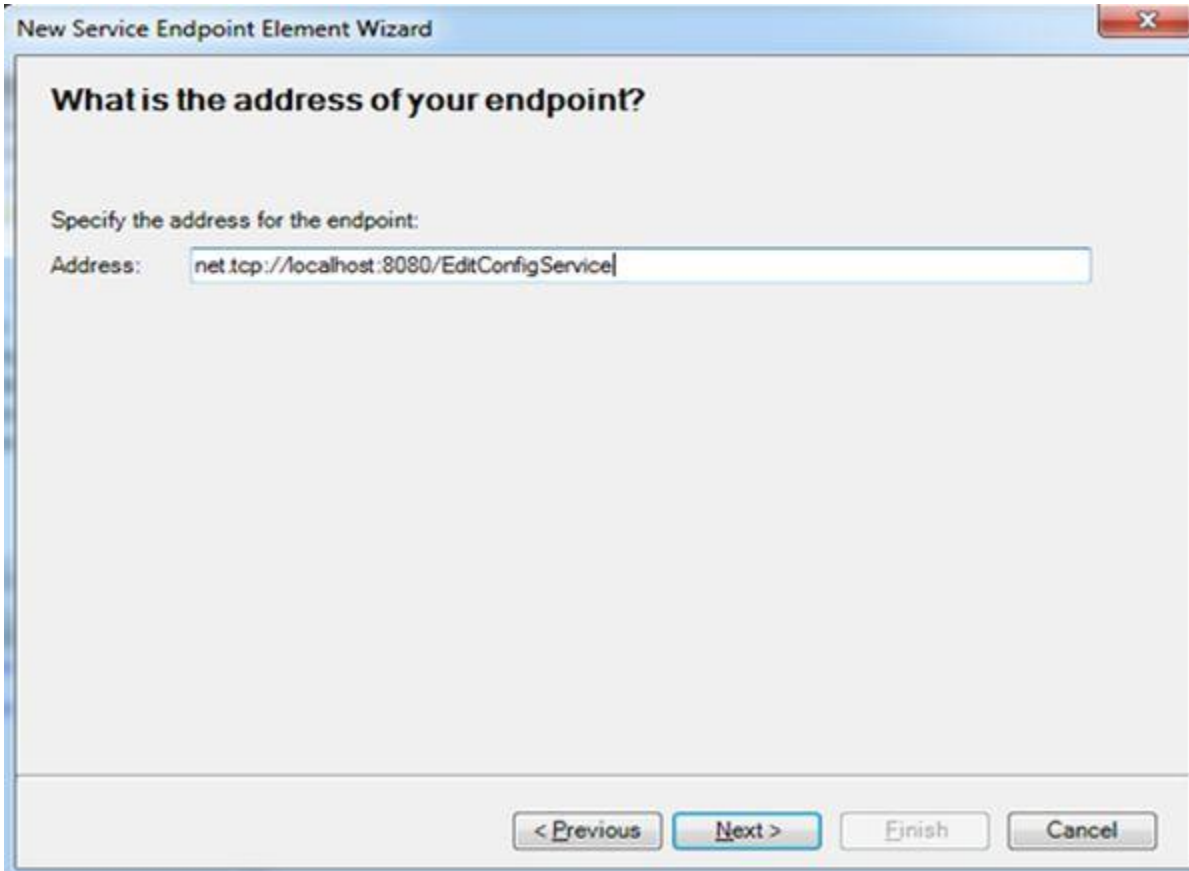
On the preceding image, there is a link "Create New Service Endpoint" to create a new endpoint. So click on this link. The Contract is selected by default. Now click on the "Next" button.



Select TCP and click on "Next" button, as in:

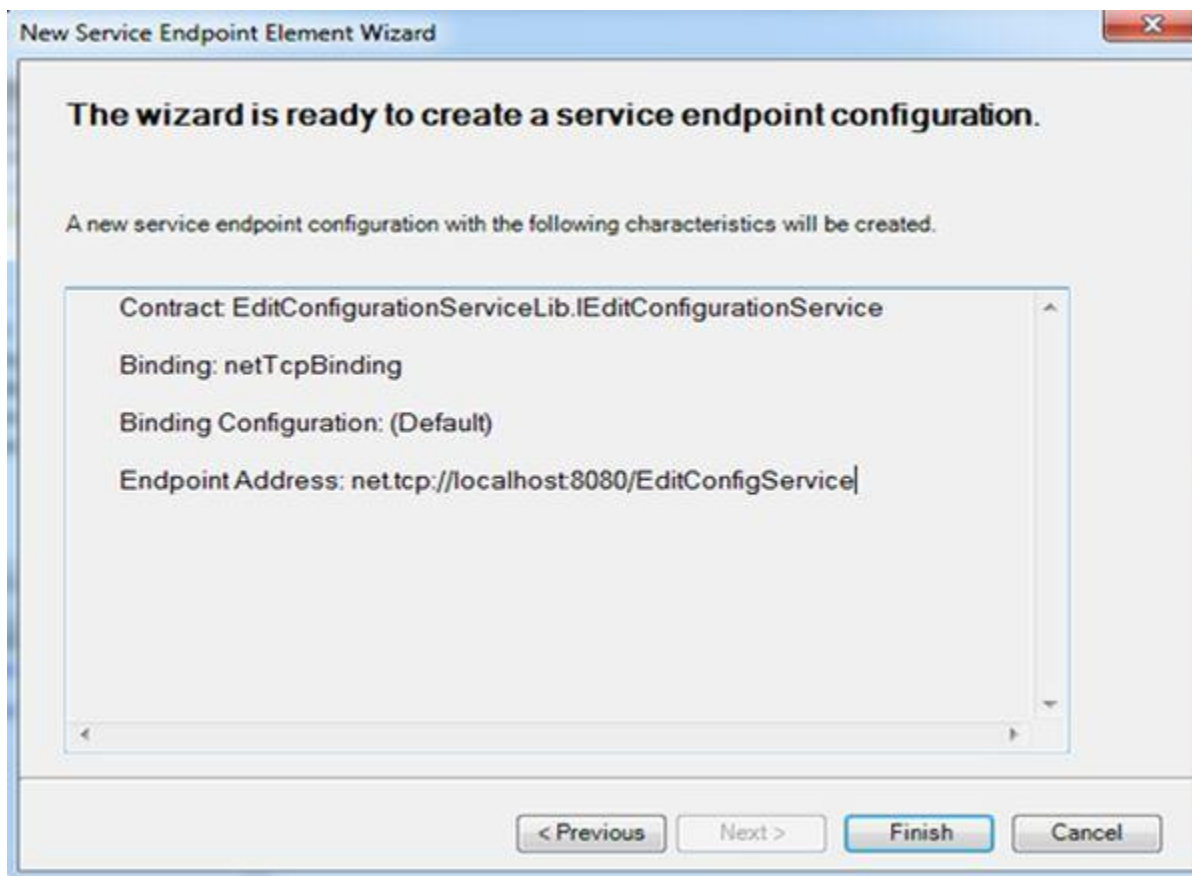


Specify the address in the address text box.

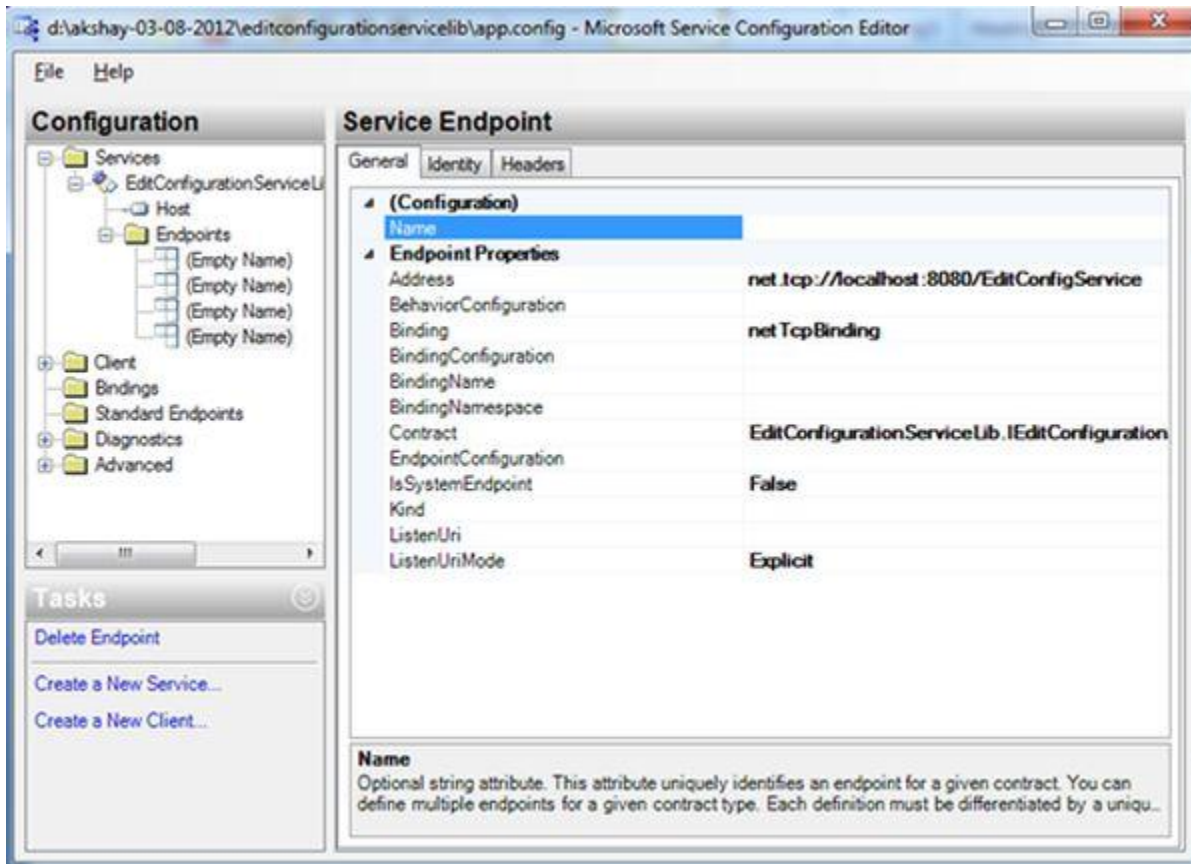


The image shows a Windows-style dialog box titled "New Service Endpoint Element Wizard". The main heading inside is "What is the address of your endpoint?". Below this, it says "Specify the address for the endpoint:". There is a text input field labeled "Address:" containing the text "net.tcp://localhost:8080/EditConfigService". At the bottom of the dialog, there are four buttons: "< Previous", "Next >", "Finish", and "Cancel". The "Next >" button is highlighted with a blue border.

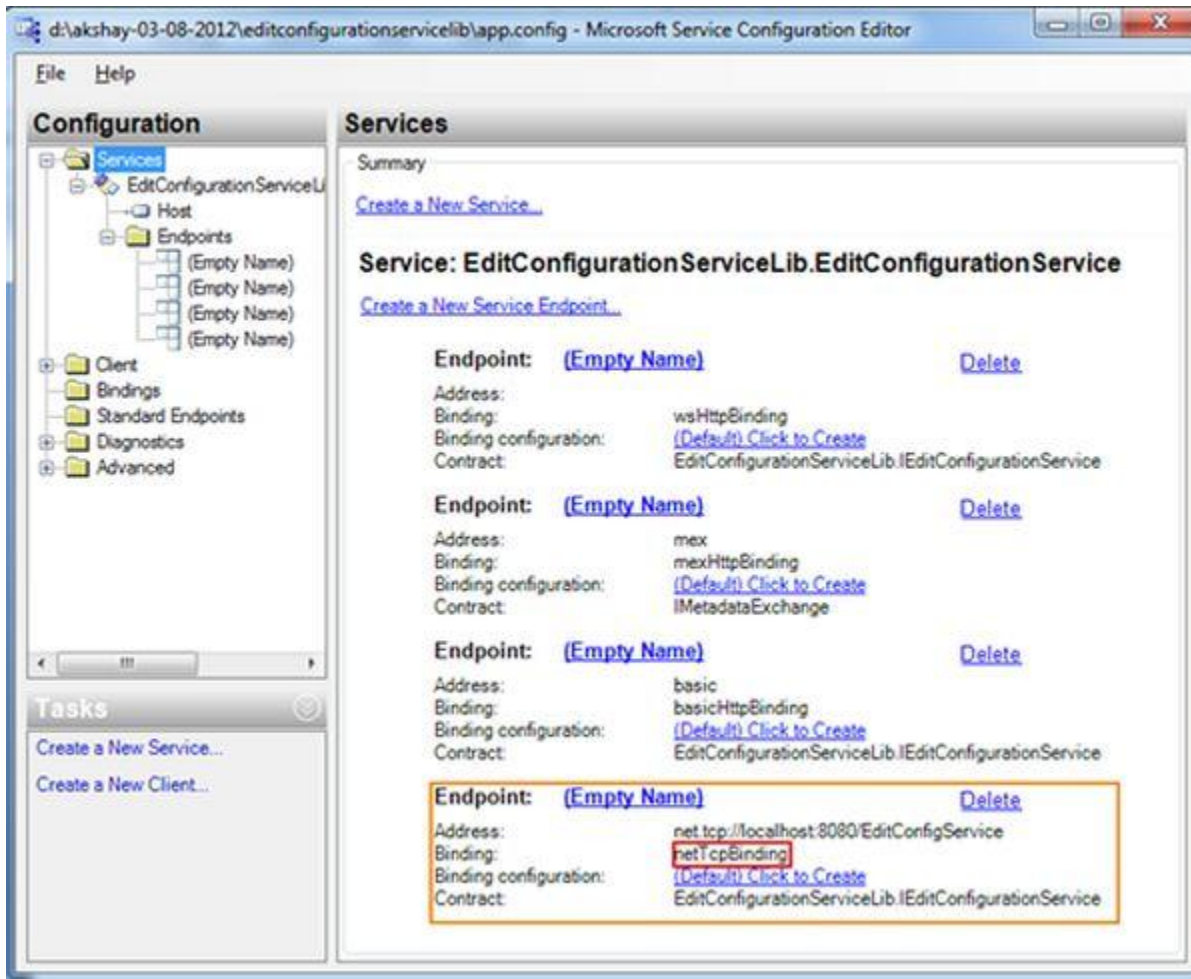
Click on the "Finish" button.



It will show an address, binding and contract for netTcpBinding, which we just created.



Click on services, which now shows one more binding i.e. `netTcpBinding`; see:

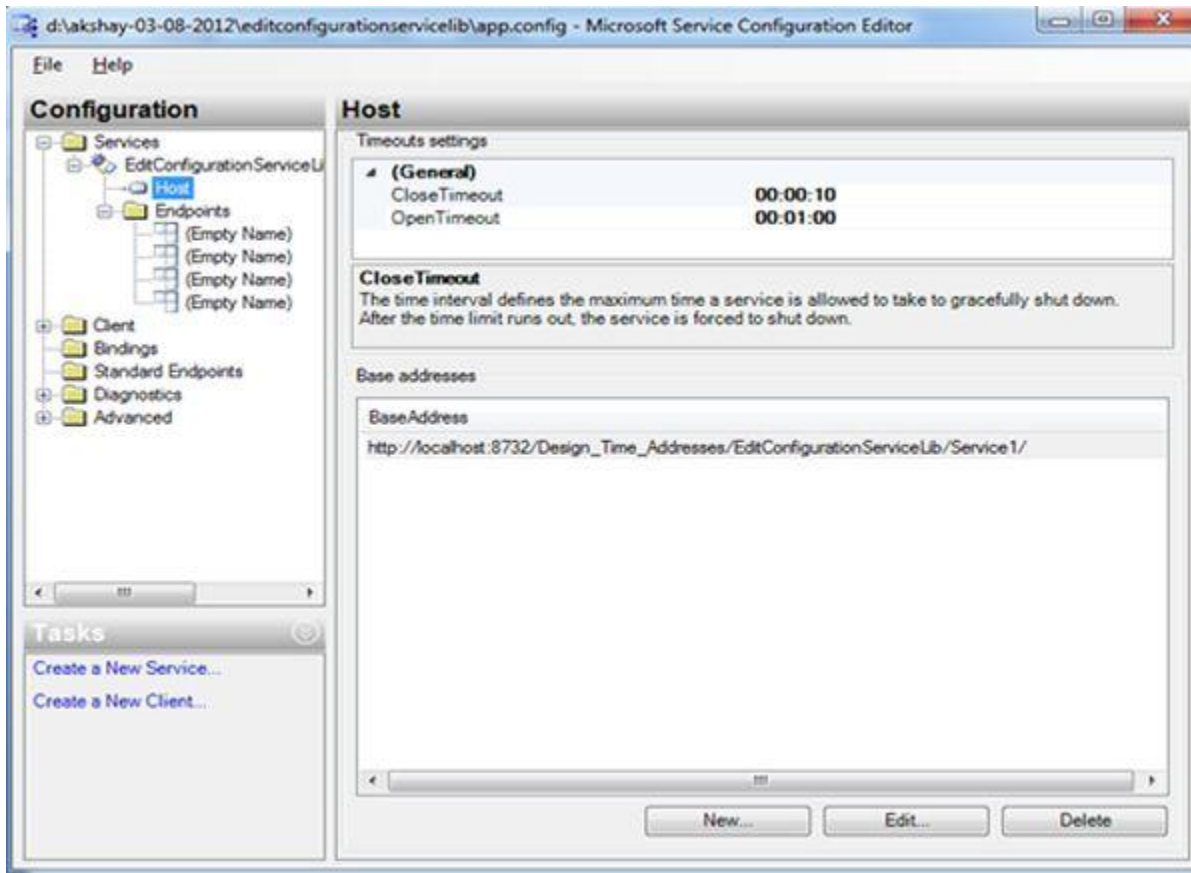


We have exposed two more endpoints i.e. for basicHttpBinding and netTcpBinding. Now go to the file menu and save these changes. Now open the app.config file. It will reflect all the changes done through the configuration editor.

```
<system.serviceModel>
  <services>
    <service name="EditConfigurationServiceLib.EditConfigurationService">
      <clear />
      <endpoint binding="wsHttpBinding" contract="EditConfigurationServiceLib.IEditConfigurationService"
        listenUriMode="Explicit">
        <identity>...</identity>
      </endpoint>
      <endpoint address="mex" binding="mexHttpBinding" contract="IMetadataExchange"
        listenUriMode="Explicit">
        <identity>
          <certificateReference storeName="My" storeLocation="LocalMachine"
            x509FindType="FindBySubjectDistinguishedName" />
        </identity>
      </endpoint>
      <endpoint address="basic" binding="basicHttpBinding" contract="EditConfigurationServiceLib.IEditConfigurationService"
        listenUriMode="Explicit">
        <identity>
          <certificateReference storeName="My" storeLocation="LocalMachine"
            x509FindType="FindBySubjectDistinguishedName" />
        </identity>
      </endpoint>
      <endpoint address="net.tcp://localhost:8080/EditConfigService"
        binding="netTcpBinding" bindingConfiguration="" contract="EditConfigurationServiceLib.IEditConfigurationService" />
      <host>
        <baseAddresses>
          <add baseAddress="http://localhost:8732/Design_Time_Addresses/EditConfigurationServiceLib/Service1/" />
        </baseAddresses>
      </host>
    </service>
  </services>
  <behaviors>
    <serviceBehaviors>
      <behavior>
        <serviceMetadata httpGetEnabled="True"/>
        <serviceDebug includeExceptionDetailInFaults="false" />
      </behavior>
    </serviceBehaviors>
  </behaviors>
</system.serviceModel>
```

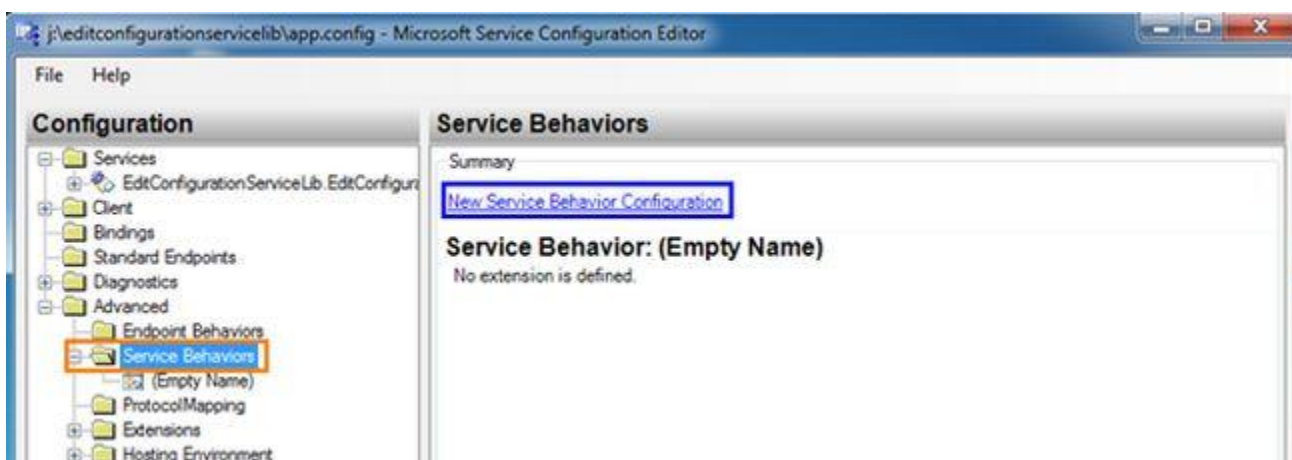
How to specify base address?

Select host under the services and in the bottom-right side there are buttons to add new base addresses and to update base addresses. You can select any according to your requirement. After creating or changing the base address, save these changes and check in the app.config file for reflection of these changes.

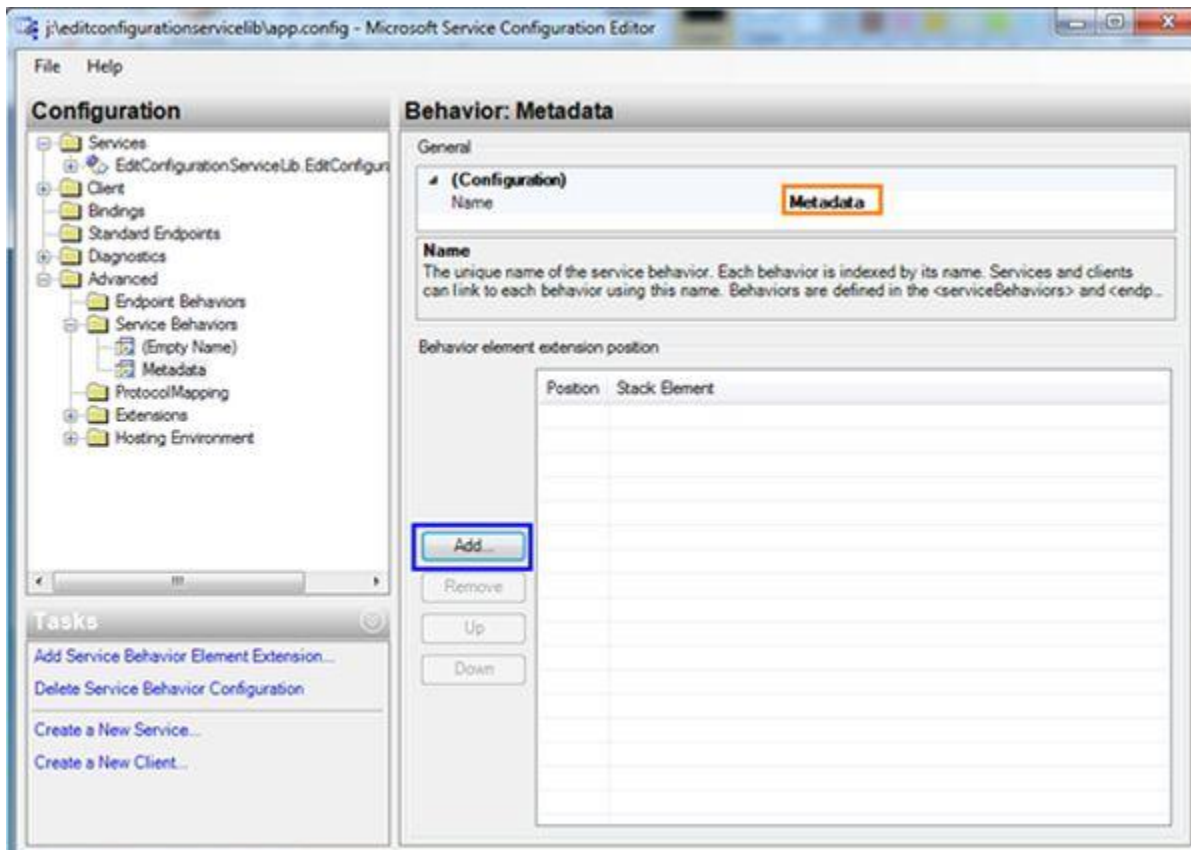


How to set service behavior?

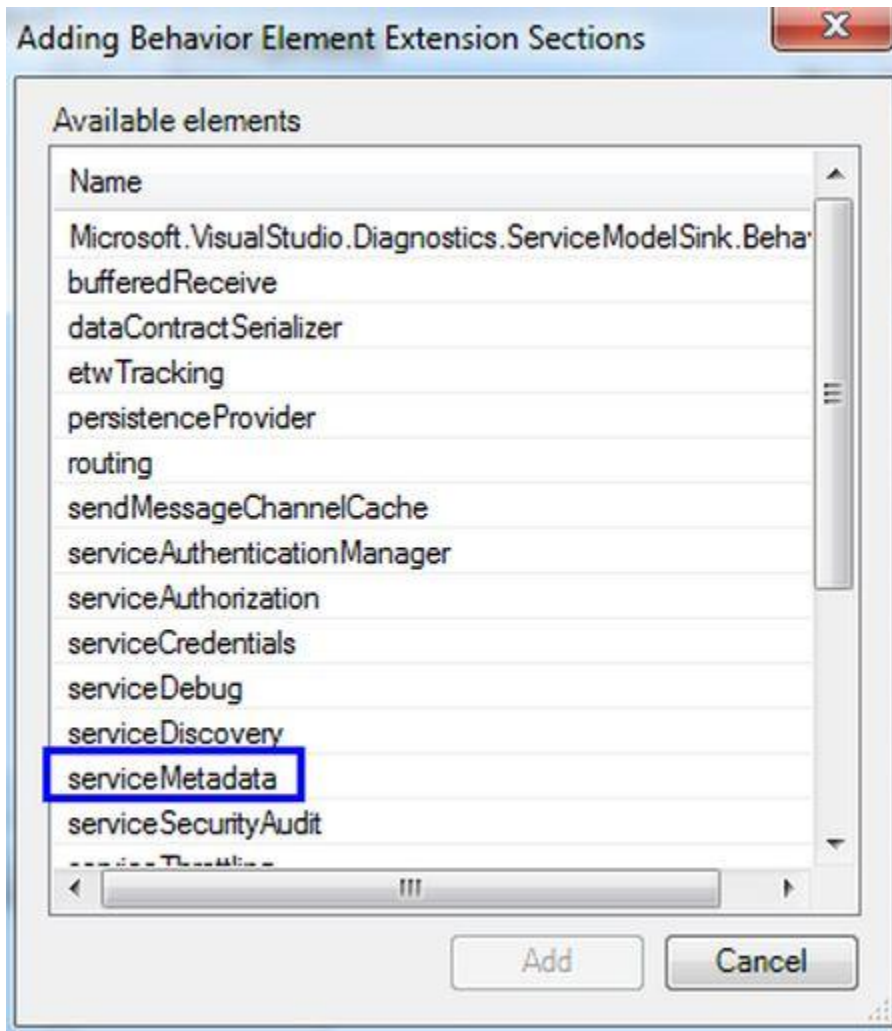
Go to the Advanced section and select Service Behavior. On the right panel click on the link "New Service Behavior Configuration".



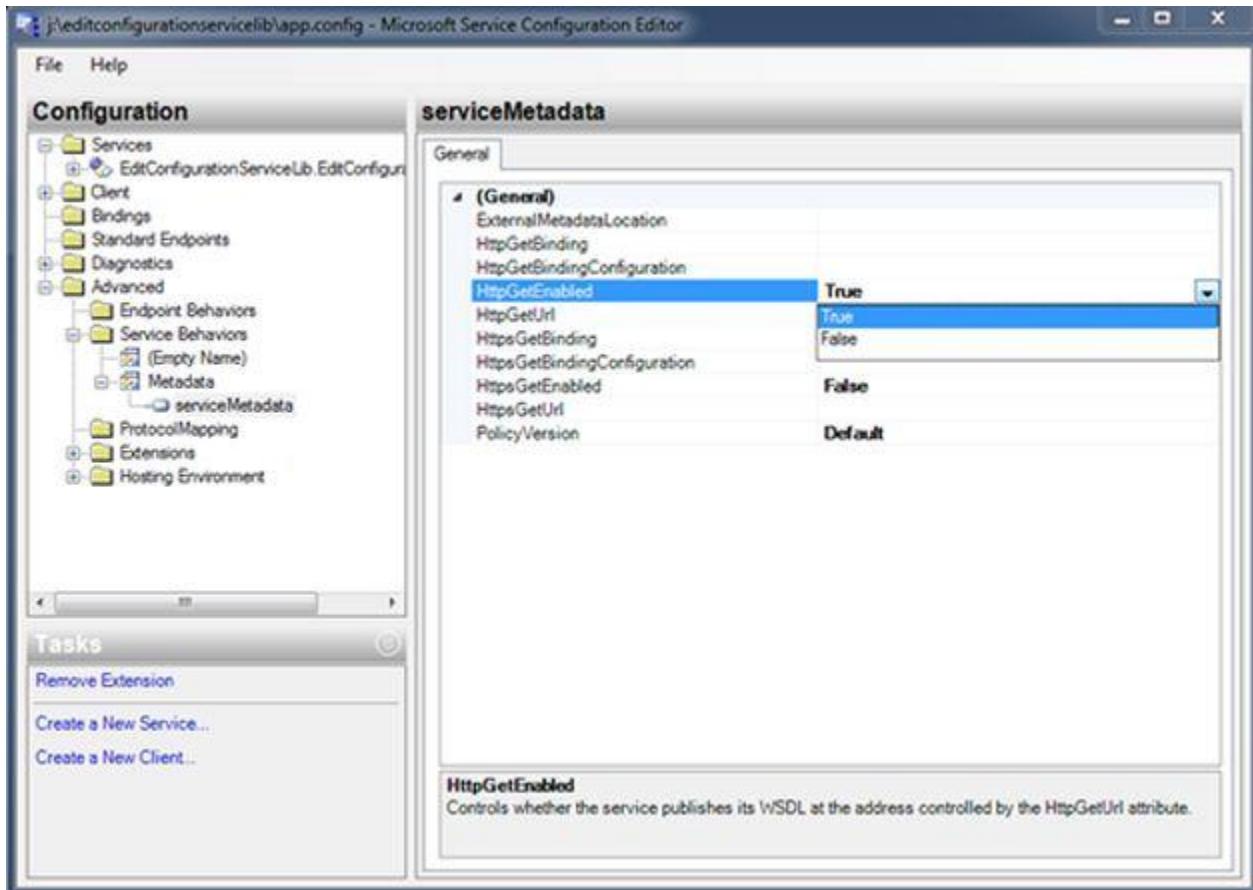
Give the configuration name as "Metadata" and click on the "Add" button



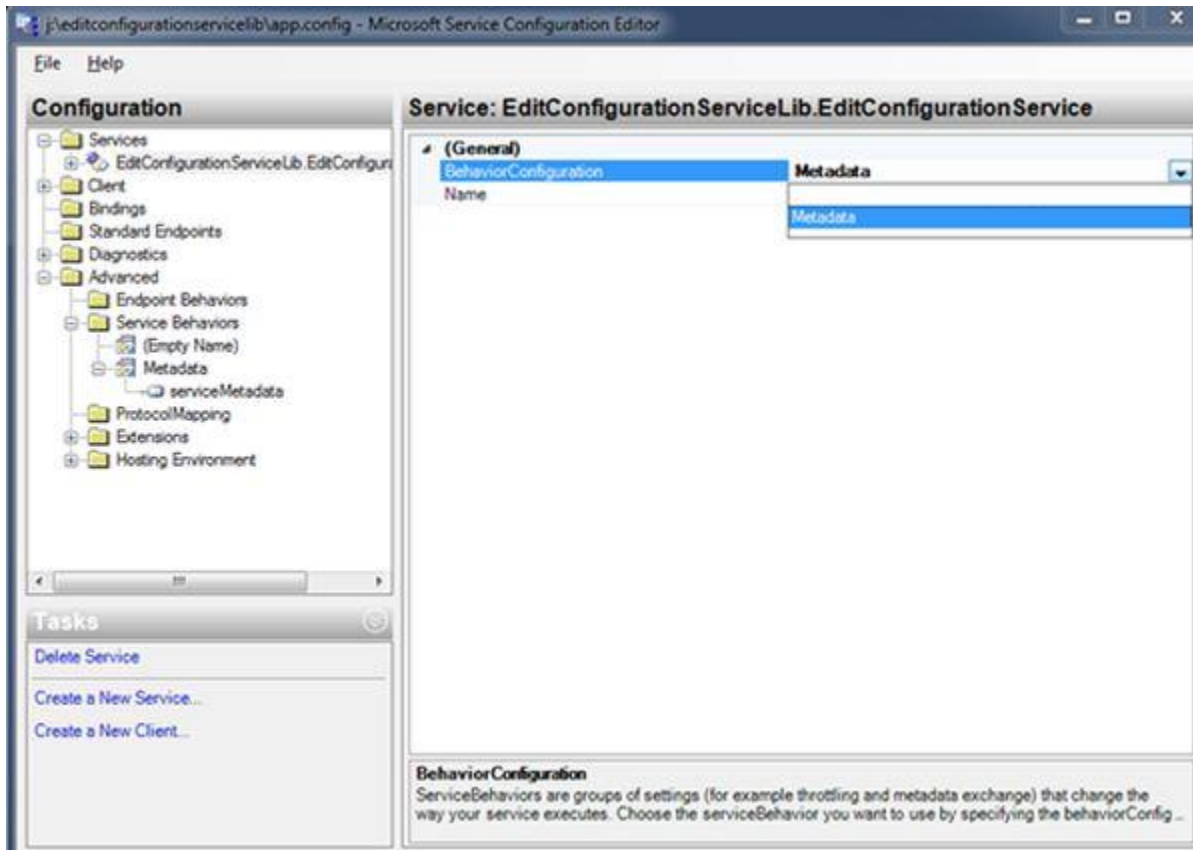
Select "serviceMetadata" from different behaviors.



Now select "serviceMetadata" from the left panel and set the "HttpGetEnabled" attribute to true.



The "Metadata" behavior is created now. The next step is to assign this behavior to a service. For that select the service "EditConfigurationServiceLib.EditConfigurationService" and select a newly created behavior for the BehaviorConfiguration attribute.



Finally save the changes and check the app.config file for these changes.