DataContract versioning is required when a modification has been made to the existing DataContract exposed to the client from the service or vice versa. If either service or client changes the schema of DataContract exposed, then a new version of DataContract is needed and both parties involved in communication should be able to accommodate the changes done in the DataContract.

The scenario that can cause for a new version of DataContract is as below,

1. Missing a member from existing DataContract.
2. Adding a new member to existing DataContract.
3. Round Tripping

In this article we will explore New Member scenarios.

**Adding a New Member to existing DataContract**

This is most common changes we do to give new version to DataContract.

For example we have a DataContract through which we are exposing Product custom class to the client

**Product.cs**

[DataContract]

public class Product

{

[DataMember(Order = 1)]

public string ProductNumber;

[DataMember(Order = 2)]

public string ProductName;

[DataMember(Order = 3)]

public string ProductPrice;

}

And we have ServiceContract which is returning a Product from the service

**IService1.cs**

namespace WcfService10

{

[ServiceContract]

public interface IService1

{

[OperationContract]

Product GetaProduct();

}

}

Service implementation is as below:

**Service1.svc.cs**

namespace WcfService10

{

public class Service1 : IService1

{

public Product GetaProduct()

{

try

{

Product p = new Product { ProductName = "Pen", ProductPrice = "9", ProductNumber = "1" };

return p;

}

catch (Exception ex)

{

return null;

}

}

}

}

Now at the client side, we can call the service as below. (Call this client as Client1)

**Program.cs**

using ConsoleApplication1.ServiceReference1;

namespace ConsoleApplication1

{

class Program

{

static void Main(string[] args)

{

using (Service1Client proxy = new Service1Client())

{

var res = proxy.GetaProduct();

Console.WriteLine(res.ProductName + res.ProductPrice + res.ProductNumber);

}

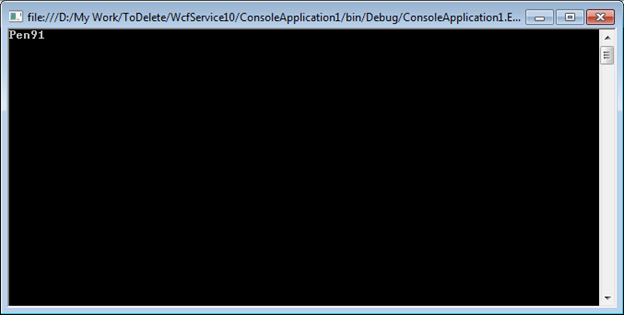
Console.ReadKey(true);

}

}

}

**Output**



Now let us go ahead and modify [the](http://www.watchesn.com) DataContract at the service. Just add one more parameter

[DataMember(Order = 4)]

public string ProductColor;

So now DataContract will look like

**Product.cs**

[DataContract]

public class Product

{

[DataMember(Order = 1)]

public string ProductNumber;

[DataMember(Order = 2)]

public string ProductName;

[DataMember(Order = 3)]

public string ProductPrice;

[DataMember(Order = 4)]

public string ProductColor;

}

And now go ahead and modify the service implementation as below:

**Service1.svc.cs**

namespace WcfService10

{

public class Service1 : IService1

{

public Product GetaProduct()

{

try

{

Product p = new Product { ProductName = "Pen", ProductPrice = "9", ProductNumber = "1",ProductColor="Red" };

return p;

}

catch (Exception ex)

{

return null;

}

}

}

}

Now do not update the service reference at the clien1. What I mean here is that same version of DataContract at the client 1. And create one more client and call it as client 2. Use updated service at the client.

So, now client 1 will look like exactly it was previously

**Program.cs (Client1)**

using ConsoleApplication1.ServiceReference1;

namespace ConsoleApplication1

{

class Program

{

static void Main(string[] args)

{

using (Service1Client proxy = new Service1Client())

{

var res = proxy.GetaProduct();

Console.WriteLine(res.ProductName + res.ProductPrice + res.ProductNumber);

}

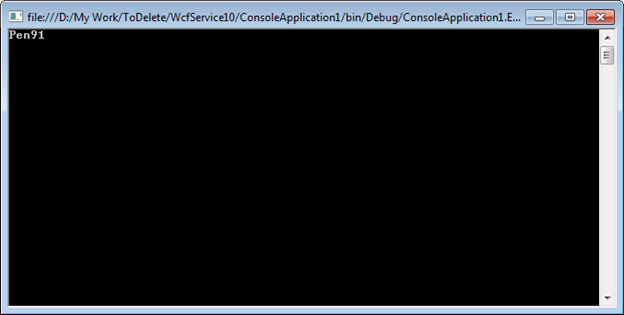
Console.ReadKey(true);

}

}

}

Output would be exactly the same as it was with previous version of DataContract



And new created client (Client2) will look like below,

**Program.cs(Client2)**

using ConsoleApplication2.ServiceReference1;

namespace ConsoleApplication2

{

class Program

{

static void Main(string[] args)

{

using (Service1Client proxy = new Service1Client())

{

var res = proxy.GetaProduct();

Console.WriteLine(res.ProductName + res.ProductPrice + res.ProductNumber+res.ProductColor);

}

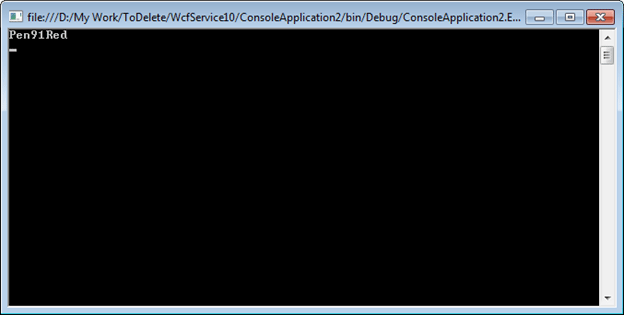
Console.ReadKey(true);

}

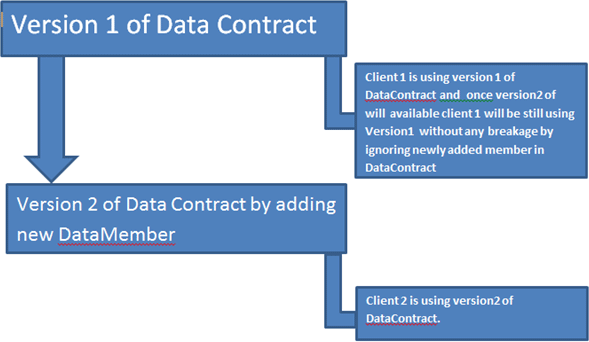
}

}

This client is using version2 of DataContract and output would contain newly added member in DataContract.



So on deseralization new added DataMember will be ingnored by DataContractSeralizer and both party will continue working without breaking.



If we have a DataContract

**Product.cs**

[DataContract]

public class Product

{

[DataMember(Order = 1)]

public string ProductNumber;

[DataMember(Order = 2)]

public string ProductName;

[DataMember(Order = 3)]

public string ProductPrice;

[DataMember(Order = 4)]

public string ProductColor;

}

ServiceContract is as below

**IService1.cs**

namespace WcfService10

{

[ServiceContract]

public interface IService1

{

[OperationContract]

Product GetaProduct(Productp );

}

}

And Service implementation is as below,

**Service1.svc.cs**

using System;

using System.Collections.Generic;

using System.Linq;

using System.Runtime.Serialization;

using System.ServiceModel;

using System.ServiceModel.Web;

using System.Text;

namespace WcfService10

{

public class Service1 : IService1

{

public Product GetaProduct(Productp)

{

Productp1 = new Product

{

ProductNumber = p.ProductNumber,

ProductName = p.ProductName,

ProductPrice = p.ProductPrice,

ProductColor = p.ProductColor

};

returnp1;

}

}

}

Now we have a service.

Now at the client side (Say it as client1) we are calling the service as below

**Program.cs**

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using ConsoleApplication1.ServiceReference1;

namespace ConsoleApplication1

{

class Program

{

static void Main(string[] args)

{

using(Service1Client proxy = new Service1Client())

{

Productp = new Product{ProductNumber="1",

ProductPrice="100",

ProductName ="Bat",

ProductColor ="White"};

varres = proxy.GetaProduct(p);

Console.WriteLine(res.ProductName+"\n" + res.ProductPrice+"\n" + res.ProductNumber+"\n"+res.ProductColor);

}

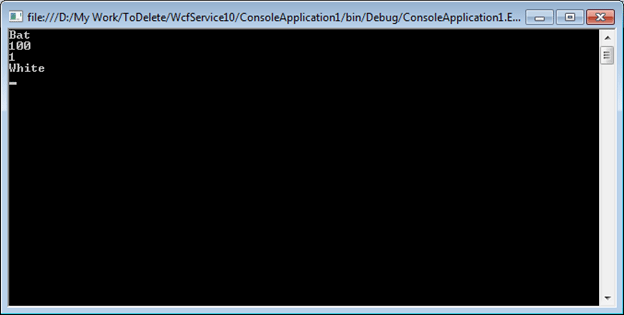
Console.ReadKey(true);

}

}

}

**Output**



Now let us go ahead and modify the service implementation as below,

**Service1.svc.cs**

using System;

using System.Collections.Generic;

using System.Linq;

using System.Runtime.Serialization;

using System.ServiceModel;

using System.ServiceModel.Web;

using System.Text;

namespace WcfService10

{

public class Service1 : IService1

{

public Product GetaProduct(Productp)

{

Productp1 = new Product

{

ProductNumber = p.ProductNumber??"Missing Number",

ProductName = p.ProductName??"Missing Name",

ProductPrice = p.ProductPrice??"Missing Price",

ProductColor = p.ProductColor??"Missing color"

};

returnp1;

}

}

}

If you see the above implementation , we are assigning some default value when value of the property is not provided or in other words Members are missing.

ProductNumber = p.ProductNumber??"Missing Number",

ProductName = p.ProductName??"Missing Name",

ProductPrice = p.ProductPrice??"Missing Price",

ProductColor = p.ProductColor??"Missing color"

So if at the client side DataContract, some DataMember is missing. In this case at the client side member ProductColor is missing

Product p = new Product{ProductNumber="1",

ProductPrice="100",

ProductName ="Bat"};

Now at the service side one DataMember is missing so Service implementation will return default value for that

**Program.cs**

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using ConsoleApplication1.ServiceReference1;

namespace ConsoleApplication1

{

class Program

{

static void Main(string[] args)

{

using(Service1Client proxy = new Service1Client())

{

Productp = new Product{ProductNumber="1",

ProductPrice="100",

ProductName ="Bat"};

varres = proxy.GetaProduct(p);

Console.WriteLine(res.ProductName+"\n" + res.ProductPrice+"\n" + res.ProductNumber+"\n"+res.ProductColor);

}

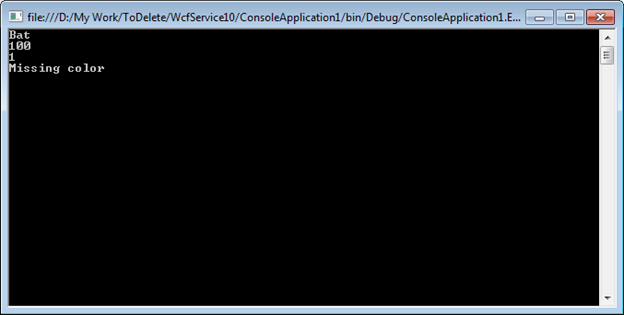
Console.ReadKey(true);

}

}

}

**Output**



**OnDeserializing event**

In case of missing DataMember , we can use OnDeserializing event to share the logic across all the party

**Product.cs**

[DataContract]

public class Product

{

[DataMember(Order = 1)]

public string ProductNumber;

[DataMember(Order = 2)]

public string ProductName;

[DataMember(Order = 3)]

public string ProductPrice;

[DataMember(Order = 4)]

public string ProductColor;

[OnDeserializing]

voidOnDeserializing(StreamingContext context)

{

ProductName = "Name Missing";

ProductNumber = "Number Missing";

ProductColor = "Color Missing ";

ProductPrice = "Price Missing";

}

}

**Serialization order in Data Contract**

If you have a Data Contract in your WCF service as below:

**Student.cs**

[DataContract]

public class Student

{

[DataMember]

public string Name { get; set; }

[DataMember]

public string Address { get; set; }

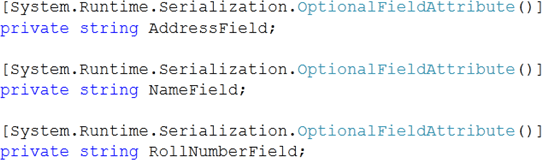
[DataMember]

public string RollNumber { get; set; }

}

Upon serialization, properties of the data contract will get serialized in alphabetical order.

So at the client side serialized data contract will look like



So we can see that in a serialized data contract at the client side, properties are serialized in alphabetical order.

Now if you want to manage the order of serialization, you need to use Order attribute of DataMember.

**Student.cs**

[DataContract]

public class Student

{

[DataMember(Order=3)]

public string Name { get; set; }

[DataMember(Order=2)]

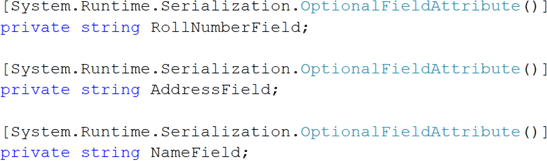
public string Address { get; set; }

[DataMember(Order=1)]

public string RollNumber { get; set; }

}

Upon serialization, properties of the data contract will get serialized in the order specified by the order attribute of the data contract.



**Note**: To see the order of serialization, click on ServiceReference1 in object explorer when you are adding that at the client side. You will get a class Student.cs

**Two important points**

1. If two properties are having same order then they will get serialized in alphabetical order.
2. In inheritance also DataContract will be serialized in alphabetical order, if explicitly order is not specified on DataMember.

**Data Contract in WCF**A data contract is a formal agreement between a service and a client that abstractly describes the data to be exchanged.

1. Data contract can be explicit or implicit. Simple type such as int, string etc has an implicit data contract. User defined object are explicit or Complex type, for which you have to define a Data contract using [DataContract] and [DataMember] attribute.
2. WCF uses a serialization engine called the Data Contract Serializer by default to serialize and deserialize data.We can also explicitly create a data contract by using DataContractAttribute and DataMemberAttribute attributes. This attribute can be applied to classes, structures, and enumerations.
3. The DataMemberAttribute attribute must then be applied to each member of the data contract type to indicate that it is a data member, that is, it should be serialized.
4. Data Contracts can be defined as follows:  
     
   â€¢ It describes the external format of data passed to and from service operations   
   â€¢ It defines the structure and types of data exchanged in service messages  
   â€¢ It maps a CLR type to an XML Schema   
   â€¢ It defines how data types are serialized and deserialized. Through serialization, you convert an object into a sequence of bytes that can be transmitted over a network. Through deserialization, you reassemble an object from a sequence of bytes that you receive from a calling application.  
   â€¢ It is a versioning system that allows you to manage changes to structured data .
5. Namespace for holding DataContract and DataMember attributes.

**System.Runtime.Serialization**  
Example :

1. Create User defined data type called Customer. This data type should be identified for serialization and deserialization by mentioning with [DataContract] and [DataMember] attribute.
2. Add Wcf Service at DataContractinWCF.Web(Silverlight Web Project),We get IService1.cs and Service1.svc files.
3. IService1.cs :  
     
   namespace DataContractinWCF.Web  
   {  
   [ServiceContract]  
   public interface IService1  
   {  
   [OperationContract]  
   List<Customer> GetCustomerData(int CustomerID);  
   }  
   [DataContract]  
   public class Customer  
   {  
   private string m\_Name;  
   private int m\_Age;  
   private int m\_Salary;  
   private string m\_Designation;  
   private string m\_Manager;

[DataMember]  
public string Name  
{  
get { return m\_Name; }  
set { m\_Name = value; }

}

[DataMember]  
public int Age  
{  
get { return m\_Age; }  
set { m\_Age = value; }  
}

[DataMember]  
public int Salary  
{  
get { return m\_Salary; }  
set { m\_Salary = value; }  
}

[DataMember]  
public string Designation  
{  
get { return m\_Designation; }  
set { m\_Designation = value; }  
}

[DataMember]  
public string Manager  
{  
get { return m\_Manager; }  
set { m\_Manager = value; }  
}  
  
}  
  
  
In GetCustomerData method we have created the Customer instance and return to the client.  
As we have created the data contract for the Employee class, client will aware of this instance whenever he creates proxy for the service.

1. Service1.svc.cs :

namespace DataContractinWCF.Web

{

public class Service1 : IService1

{

public Cusomer GetCustomerData (int empId)

{

Cusomer CusDetail = new Cusomer ();

//Do something to get Customer details and assign to CusDetail properties

return CusDetail;

}

}

1. Client Side  
     
   On client side we can create the proxy for the service and make use of it. The client side code is shown below.  
     
   Build Web project and then Add Service reference to client side.Now we can access the service methods at client side.  
     
   On button click event access the service as follows :  
     
   Public void btnGetDetails\_Click(object sender, EventArgs e)

{

ServiceReference1.Service1Client service = new ServiceReference1.Service1Client();

Cusomer CusDetails;

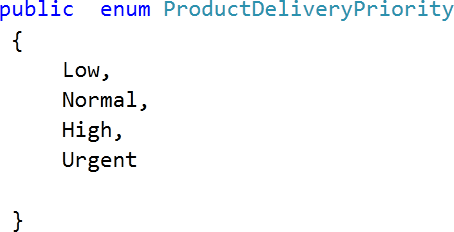
CusDetails = service.GetCustomerData(CusomerId);

//Do something on Customer details

}

**Enumeration in DataContract of WCF**By default **Enums** are **serializable**. If we define Enum at service side and use it in Data Contract, it is exposed at the client side.

For example, if we have an **Enum** as below,

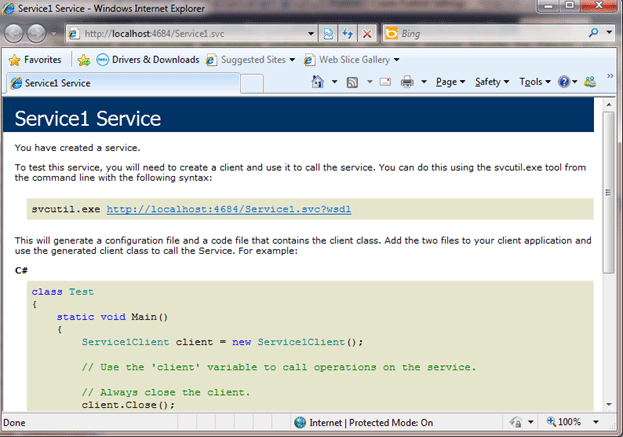


And we are using it in **DataContract** as below

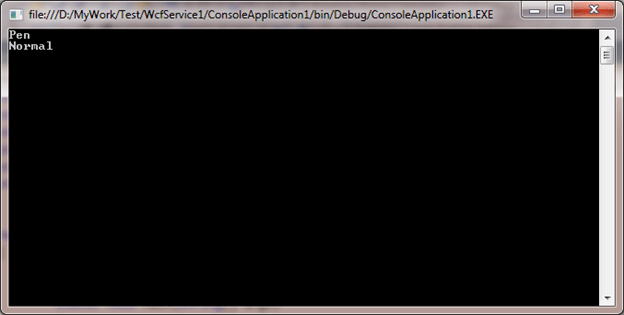
2.gif

By default Enum is serialized.

So, let us see the default behavior of Enum with an example.

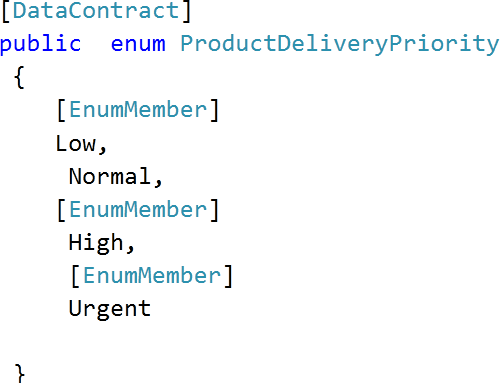
1. Create a DataContract called Product.  
     
   **Product.cs**  
     
   using System;  
   using System.Collections.Generic;  
   using System.Linq;  
   using System.Web;  
   using System.Runtime.Serialization;  
     
   namespace WcfService1  
   {  
   [DataContract]  
   public class Product  
   {  
   [DataMember]  
   public stringProductName;  
   [DataMember]  
   public ProductDeliveryPriorityProductPriority;  
   }  
   public enum ProductDeliveryPriority  
   {  
   Low,  
   Normal,  
   High,  
   Urgent  
   }  
   }
2. Create the Service Contract. We are just returning a Product from the service.  
     
   **IService1.cs**  
     
   using System;  
   using System.Collections.Generic;  
   using System.Linq;  
   using System.Text;  
   using System.ServiceModel;  
     
   namespace WcfService1  
   {  
   [ServiceContract]  
   public interface IService1  
   {  
   [OperationContract]  
   ProductGetProduct();  
   }  
   }
3. Implement the service in service definition .   
     
   **Service1.svc.cs**  
     
   using System;  
   using System.Collections.Generic;  
   using System.Linq;  
   using System.Runtime.Serialization;  
   using System.ServiceModel;  
   using System.ServiceModel.Web;  
   using System.Text;  
     
   namespace WcfService1  
   {  
   public class Service1 : IService1   
   {  
   public Product GetProduct()  
   {  
   Product p = new Product{ ProductName = "Pen", ProductPriority = ProductDeliveryPriority.Normal };  
   return p;  
   }  
   }  
   }
4. Define the EndPoint in config file as below. We are exposing the service with basicHttpBinding.  
     
   **Web.Config**  
     
   <?xml version="1.0"?>  
   <configuration>  
   <system.web>  
   <compilation debug="true" targetFramework="4.0" />  
   </system.web>  
   <system.serviceModel>  
   <behaviors>  
   <serviceBehaviors>  
   <behavior>  
   <!-- 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>  
   <services>  
   <service name ="WcfService1.Service1">  
   <endpoint address ="" binding ="basicHttpBinding" contract ="WcfService1.IService1"/>  
   <endpoint address ="mex" binding ="mexHttpBinding" contract ="IMetadataExchange"/>  
   <host>  
   <baseAddresses>  
   <add baseAddress ="http://localhost:8181/Service1.svc"/>  
   </baseAddresses>  
   </host>  
   </service>   
   </services>  
   <serviceHostingEnvironment multipleSiteBindingsEnabled="true" />  
   </system.serviceModel>  
   <system.webServer>  
   <modules runAllManagedModulesForAllRequests="true"/>  
   </system.webServer>  
   </configuration>
5. Test the service in browser   
     
   
6. Consume the service at the client. Add the service reference at the client side and create the proxy.   
     
   **Program.cs**  
     
   using System;  
   using System.Collections.Generic;  
   using System.Linq;  
   using System.Text;  
   using System.ServiceModel;  
   using ConsoleApplication1.ServiceReference1;  
     
   namespace ConsoleApplication1  
   {  
   class Program  
   {  
   static void Main(string[] args)  
   {  
   Service1Clientproxy = new Service1Client();  
   Productp = proxy.GetProduct();  
   Console.WriteLine(p.ProductName);  
   Console.WriteLine(p.ProductPriority.ToString());  
   Console.ReadKey(true);  
   }  
   }  
   }

We added the service reference and namespace. On running the output we will get is



**Excluding certain ENUM values**

If we want to exclude certain member from ENUM then we need to follow below steps

1. Decorate the Enum with DataContract attribute   
     
   
2. Decorate the entire member we want to expose to client as EnumMember. Any Enum member not decorated with EnumMember will not be exposed to the client.   
     
    

In above Enum definition we are not decorating Normal with EnumMember . and in service we are returning Normal to the client . so when client will try to access normal enum value , it will throw an error.

So let us modify the Enum as below,

Modified Enum definition

[DataContract]

public enum ProductDeliveryPriority

{

[EnumMember]

Low,

Normal,

[EnumMember]

High,

[EnumMember]

Urgent

}

And at the client side while accessing

**Program.cs**

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.ServiceModel;

using ConsoleApplication1.ServiceReference1;

namespace ConsoleApplication1

{

class Program

{

static void Main(string[] args)

{

try

{

Service1Client proxy = new Service1Client();

Product p = proxy.GetProduct();

Console.WriteLine (p.ProductName);

Console.WriteLine (p.ProductPriority.ToString ());

}

catch (Exception ex)

{

Console.WriteLine (ex.Message);

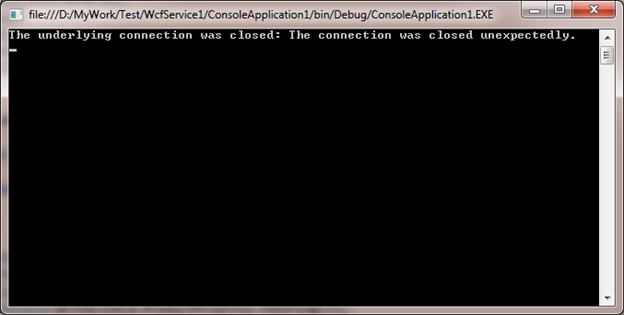
}

Console.ReadKey(true);

}

}

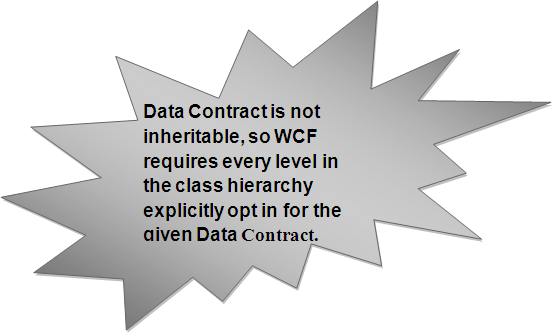
}

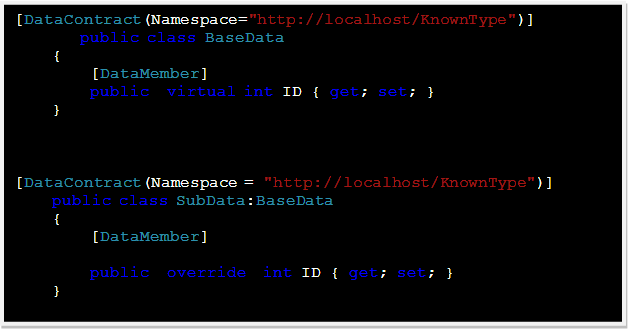
**Output**  
  


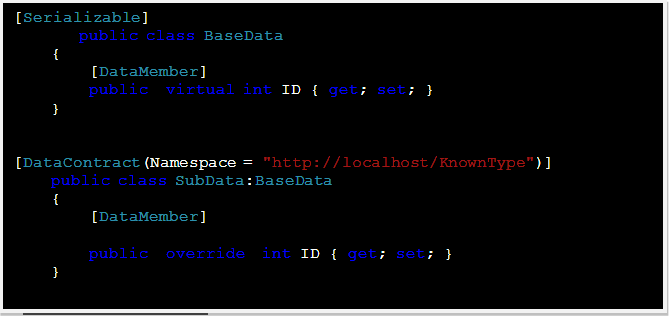
We are getting this error at client side because in service we are retuning a Product with ProductPriority normal but normal is not attributed with EnumMember. So it is not exposed on wire or serialized.

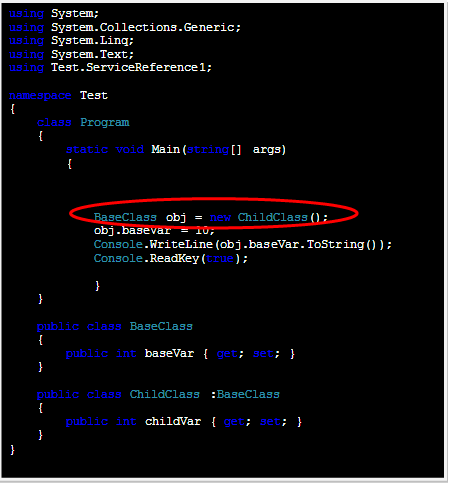
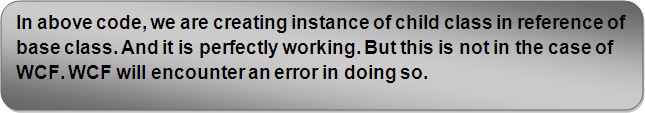
**Data Contract Hierarchy in WCF  
Objective:**  
  
This article will explain

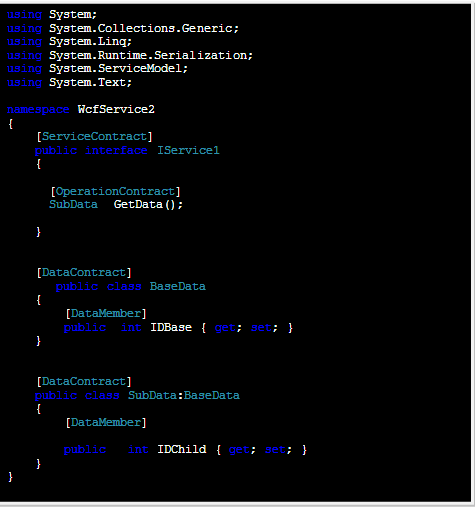
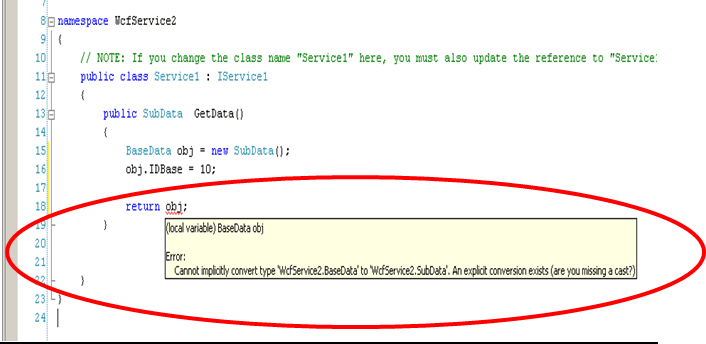
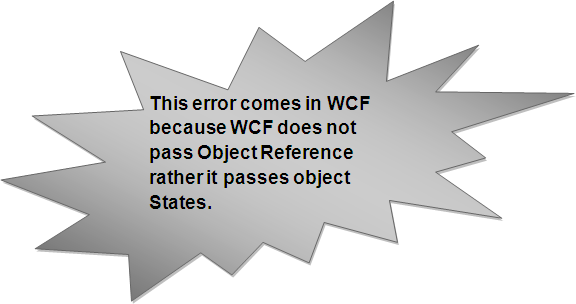
1. Data Contract hierarchy
2. How base class and sub class are getting exposed in WCF?
3. What is Known Type attribute
4. Mixing of sterilization and DataContract

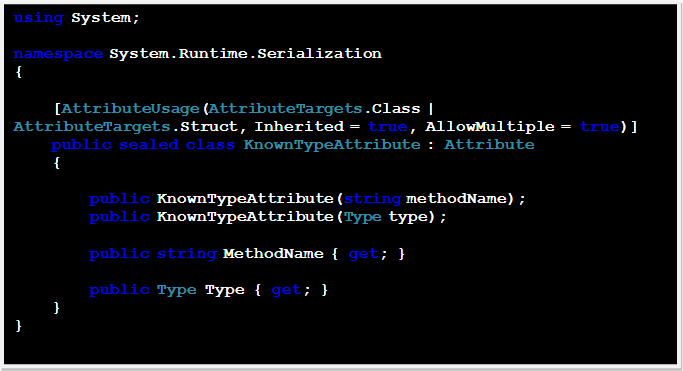
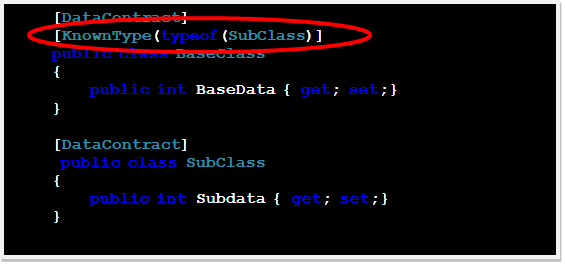
**Data Contract Hierarchy   
**

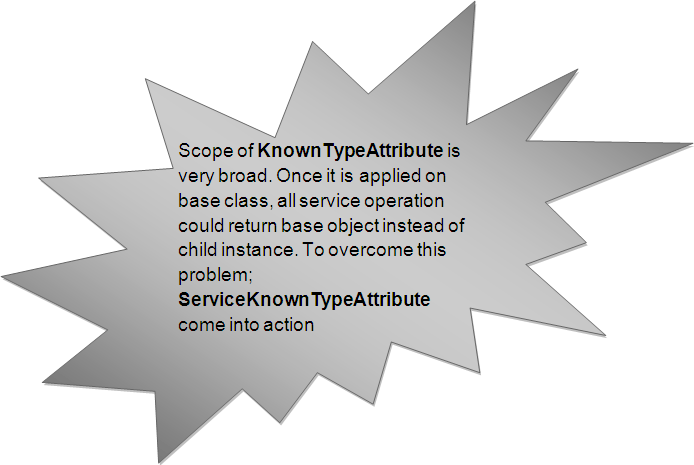
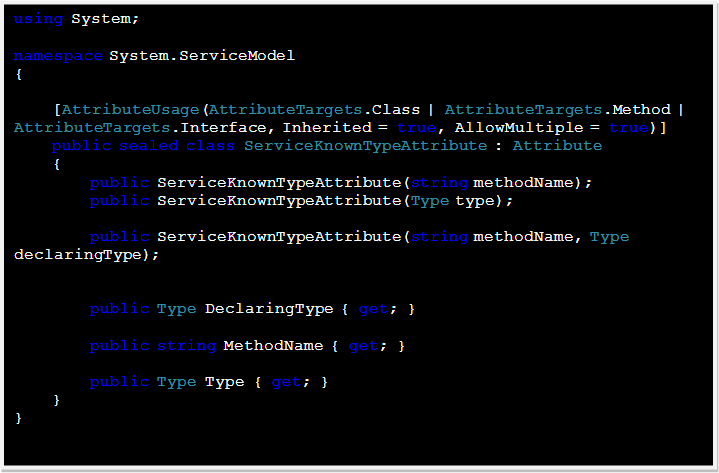
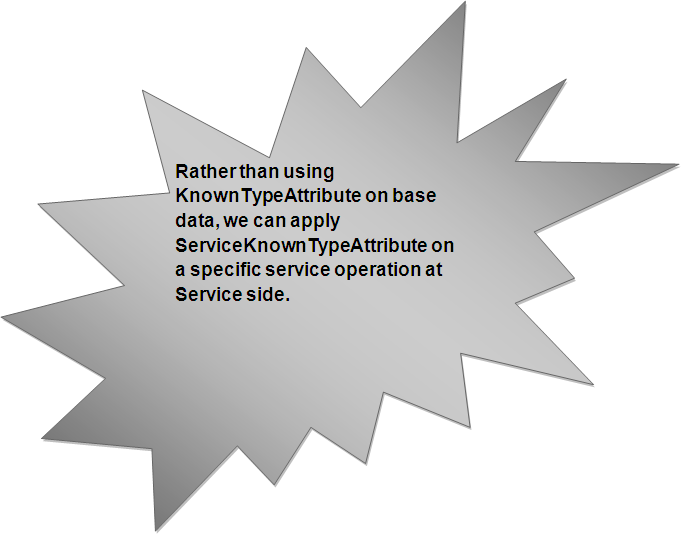
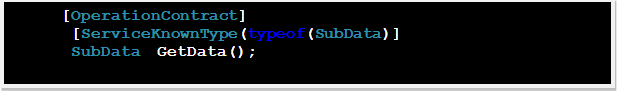
1. If any class in hierarchy is not attributed as **[DataContract]** or **[serilizable]** then **InvalidDataContractException** will occur at service run time.
2. WCF allows mixing **[DataContract]** and **[Serilizable]** attribute. But [Serilizable] should be at the root of the DataContract hierarchy.   
     
   **Examples:  
     
   **

**Mixing of DataContract and Serilizable**  
  
  
Follow the below code,

1. There are two classes in hierarchy
2. Base class is referring instance of derive class. This is perfectly valid.  
     
     
     
     
     
   

Let us try to refer derive child class in base class,   
  
**Contract and classes**  
 **Error Service implantation**  
  
  
  
**How to solve this problem?**   
  
**KnownTypeAttribute**

1. The solution is to explicitly tell WCF about the **sub class** to the **base class.**
2. This is done using **KnownType Attribute class.**  
     
   This class is defined as below in System.Runtime.Serilization namespace.  
     
   
3. The **KnownTypeAttribute** allows designating the sub class.  
     
   **Example**
4. On the host side the **KnownTypeAttribute** affects all contracts and operations using the base class across all services and endpoints. This allows accepting subclass instead of base class.
5. WCF includes sub class in **metadata** such that client can also pass the sub class instead of base class.

**ServiceKnownTypeAttribute**   
  
  
  
**ServiceKnownTypeAttribute class  
  
  
  
**  
  
**ServiceKnownType** could be applied on a particular service operation or to entire service contract.