

## WCF 의 Contract2

이번 포스팅은 저번 주제에 이어서 Contract 중 Data Contract 에 대해 이야기 해볼까 합니다.

간단히 얘기하면, Data Contract 는 WCF 서비스에서 사용하는 개체에 대한 정보를 클라이언트에서 인지할 수 있게끔 XSD 형태로 매핑시켜주는 역할을 합니다.

저번 포스팅에서 충분히 설명했듯이 클라이언트에서는 서비스에 대한 정보를 WSDL 을 통해 얻게 됩니다. 이때, 서비스에서 사용하는 개체에 대한 정보 역시 이 WSDL 을 통해 전달되는데, 이렇게 XSD 형태로 매핑시켜주는 역할을 하는 것이 Data Contract 입니다.

저번 포스팅에서 만들었던 서비스에 다음과 같이 Product 라는 이름의 새로운 클래스를 추가하였습니다.

```
using System.ServiceModel;
using System.Runtime.Serialization;

namespace MyService
{
    [DataContract]
    public class Product
    {
        [DataMember]
        public int ProductId;

        [DataMember]
        public string ProductName;

        [DataMember]
        public string Company;

        [DataMember]
        public double Price;

        [DataMember]
        public DateTime CreateDate;
    }
}
```

ServiceContract 속성을 줬듯이 DataContract 속성(attribute)은 서비스에서 사용할 새로운 클래스에 지정해 주면 됩니다. 그리고 이 클래스의 필드들에겐 DataMember 속성을 줬습니다.

이제 이 클래스를 서비스에서 사용하도록 기존 서비스의 코드를 다음과 같이 살짝 바꾸었습니다.

```
[ServiceContract(Namespace = "http://RuAAService.co.kr/")]
interface IProductService
{
    [OperationContract]
    Product GetProduct();
}

class ProductService : IProductService
{
    public Product GetProduct()
    {
        Product p = new Product();
        p.ProductId = 1234;
        p.ProductName = "ABC Chocolate";
        p.Price = 1500.0;
    }
}
```

```

        p.Company = "Lotteee";
        p.CreateDate = DateTime.Parse("2010-01-22");

        return p;
    }
}

```

여기서 조금 주목해야 할 점은 **GetProduct 메소드의 반환 타입이 우리가 새로 생성한 Product 클래스** 라는 것입니다.

Product 클래스는 서비스에서 생성한 클래스이기 때문에 클라이언트에선 알 수 없는 타입이겠죠. 서비스를 이용하는 프로그램이 아닐 때는 그냥 참조 추가를 이용해 dll 파일을 참조하여 다른 클래스를 사용할 수 있지만, 서비스를 사용할 땐 이 방법을 쓸 수는 없습니다.

이러한 문제(?)를 해결하는 방법이 바로 wsdl에 이 클래스에 대한 정보를 추가하여 클라이언트에서 인지할 수 있게 하는 것인데, 이것 Data Contract가 해주게 되는 것입니다.

그럼, 이 클래스에 대한 정보를 닷넷 컴파일러가 어떤 형태의 xsd로 매핑시켜주는지 눈으로 확인해 보겠습니다.

서비스를 실행시킨 상태에서 브라우저를 이용하여 <http://localhost:8000/ProductService?wsdl=wsdl0> 위치로 이동해 봅니다. 그럼, <wsdl:types> 라는 엘리먼트를 볼 수 있고, 그 밑에 서비스에서 사용하는 여러 가지 스키마에 대한 정보를 명시한 엘리먼트를 확인할 수 있습니다.

```

- <wsdl:types>
- <xsd:schema targetNamespace="http://RuAAService.co.kr/Imports">
  <xsd:import schemaLocation="http://localhost:8000/ProductService?xsd=xsd0"
    namespace="http://RuAAService.co.kr/" />
  <xsd:import schemaLocation="http://localhost:8000/ProductService?xsd=xsd1"
    namespace="http://schemas.microsoft.com/2003/10/Serialization/" />
  <xsd:import schemaLocation="http://localhost:8000/ProductService?xsd=xsd2"
    namespace="http://schemas.datacontract.org/2004/07/MyService" />
</xsd:schema>
</wsdl:types>

```

이 스키마들 중에 <http://localhost:8000/ProductService?xsd=xsd2> 위치로 이동해보겠습니다. 그럼 다음과 같은 내용을 확인할 수 있을 것입니다.

```

- <xs:complexType name="Product">
- <xs:sequence>
  <xs:element minOccurs="0" name="Company" nillable="true" type="xs:string" />
  <xs:element minOccurs="0" name="CreateDate" type="xs:dateTime" />
  <xs:element minOccurs="0" name="Price" type="xs:double" />
  <xs:element minOccurs="0" name="ProductId" type="xs:int" />
  <xs:element minOccurs="0" name="ProductName" nillable="true" type="xs:string" />
</xs:sequence>
</xs:complexType>

```

complexType 엘리먼트는 우리가 생성한 클래스의 이름을 나타내고 있고, 그 밑으로는 클래스의 멤버들에 대한 정의가 엘리먼트로 포함되어 있는 것을 확인할 수 있습니다.

이제 어떻게 클라이언트에서 이러한 클래스에 대해 인지할 수 있는지,, 아시겠죠? ^^

아~ 참고로 클래스의 필드들을 DataMember 속성을 이용해 노출을 했었는데, **만약 필드에 DataMember 속성을 적용시키지 않으면 아예 노출이 안된다** 는 것을 유념해주시기 바랍니다.

그리고, 필드들의 한정자가 public 이든, private 이든 이 역시 상관없다는 것도 알아두시면 좋을 것 같습니다. 오로지 DataMember 속성이 적용되었는지만 신경을 써주시면 됩니다.

DataContract 와 DataMember 는 몇 개의 프로퍼티(property)를 가지고 있습니다.

DataContract 는 **Name** 과 **Namespace** 프로퍼티를 가지고 있는데, 이는 ServiceContract 가 가지고 있는 프로퍼티와 같은 역할을 하는 녀석들이기 때문에 따로 긴 설명을 필요 없을 듯 합니다.

그리고, **DataMember** 는 **Name**, **Order**, **IsRequired** 라는 프로퍼티 들을 가지고 있습니다.

- **Name** : 클라이언트에 노출되는 멤버의 이름을 명시적으로 설정 하는 역할
- **Order** : 클라이언트에 멤버들이 노출될 때 그 순서를 정하는 역할

(이 순서를 명시적으로 설정하지 않으면 기본적으로 멤버들은 알파벳 순으로 xsd 에 나타나게 됨)

사실 .NET 어플리케이션 끼리는 이 순서가 그리 중요하지 않습니다. 하지만 다른 플랫폼의 어플리케이션 간의 상호 운용성을 고려할 땐 중요할 수가 있습니다. 클라이언트로 부터 메시지를 받았을 때, 이 순서가 다른 서버에선 인식할 수가 없게 되어버리거든요.

- **IsRequired** 는 멤버가 적어도 한번은 포함이 되어야 하는지에 대한 정의를 내려주는 역할

이 프로퍼티의 값이 true 인 경우엔 설정된 멤버는 메시지에 적어도 한번은 꼭 포함되어야 함을 의미하는 것이죠~

그럼, 이러한 속성들을 적용했을 때, 어떻게 xsd 로 표현되는지 한번 보도록 하겠습니다.  
Product 클래스를 다음과 같이 살짝 수정해보았습니다.

```
[DataContract (Namespace="http://RuAAService.co.kr/Product", Name="ProductInfo")]
public class Product
{
    [DataMember(Name = "ID", Order = 1, IsRequired = true)]
    public int ProductId;

    [DataMember(Name = "Name", Order = 2, IsRequired = true)]
    public string ProductName;

    [DataMember(Order = 3)]
    public string Company;

    [DataMember(Name = "Value", Order = 4, IsRequired = true)]
    public double Price;

    [DataMember(Order = 5, IsRequired = true)]
    public DateTime CreateDate;
}
```

이제 xsd 를 어떻게 확인하시는지 아시죠? xsd 를 확인해 보면 다음과 같이 나타남을 확인할 수 있으실 겁니다.

```
<xs:schema elementFormDefault="qualified" targetNamespace="http://RuAAService.co.kr/Product"
  xmlns:xs="http://www.w3.org/2001/XMLSchema" xmlns:tns="http://RuAAService.co.kr/Product">
  - <xs:complexType name="ProductInfo">
    - <xs:sequence>
      <xs:element name="ID" type="xs:int" />
      <xs:element name="Name" nillable="true" type="xs:string" />
      <xs:element minOccurs="0" name="Company" nillable="true" type="xs:string" />
      <xs:element name="Value" type="xs:double" />
      <xs:element name="CreateDate" type="xs:dateTime" />
    </xs:sequence>
  </xs:complexType>
  <xs:element name="ProductInfo" nillable="true" type="tns:ProductInfo" />
</xs:schema>
```

complexType 엘리먼트에 name 속성의 값이 바뀌고, 네임스페이스의 값도 바뀌고,, 여러가지가 바뀐 것을 확인할 수가 있습니다. 프로퍼티 설정과 xsd 를 비교해가면서 어떻게 적용되는지 유심히 살펴보시기 바랍니다. ^^

Company 멤버의 경우 IsRequired 속성을 따로 명시해주지 않았는데, 이 경우엔 minOccurs=0 으로 표현되는 것을 볼 수가 있습니다. 이게 기본값이며, 포함되지 않아도 됨을 나타내주는 것이죠~

이렇게 해서 Data Contract 에 관한 기본적인 내용은 끝이 난 것 같습니다.

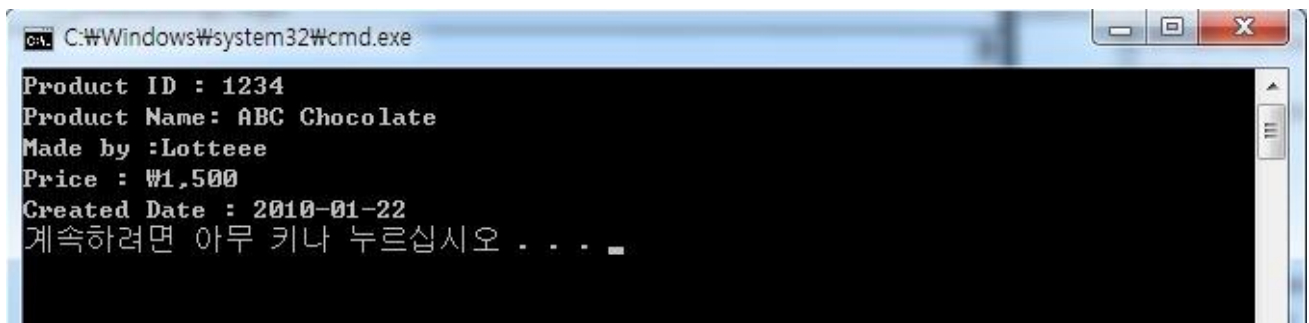
마지막으로, 수정한 이 서비스를 이용하도록 클라이언트 코드 역시 수정을 조금 해보구요~ 결과 화면을 보는 것으로 이번 포스팅을 마치도록 하겠습니다.

수정한 클라이언트의 코드는 다음과 같습니다.

```
static void Main(string[] args)
{
    // 프록시 클래스 인스턴스 생성
    ProductServiceClient myService = new ProductServiceClient();
    // 서비스 Operation 호출
    ProductInfo product = myService.GetProduct();

    Console.WriteLine("Product ID : {0}", product.ID);
    Console.WriteLine("Product Name: {0}", product.Name);
    Console.WriteLine("Made by :{0}", product.Company);
    Console.WriteLine("Price : {0:c}", product.Value);
    Console.WriteLine("Created Date : {0}", product.CreateDate.ToShortDateString());
}
```

수정을 모두 완료한 후에 서비스와 클라이언트를 동시에 실행시키면 다음과 같은 결과 화면을 확인하실 수 있으실 겁니다.



```
C:\Windows\system32\cmd.exe
Product ID : 1234
Product Name: ABC Chocolate
Made by :Lotteee
Price : ₩1,500
Created Date : 2010-01-22
계속하려면 아무 키나 누르십시오 . . .
```