<https://www.codeproject.com/Tips/601224/What-is-KnownType-Attribute-and>

What is KnownType Attribute and How to Use It in WCF Technology

Prerequisites

You need to be familiar with the structure of the WCF Technology and C# to better understand.

Introduction

The KnownTypeAttribute class allows you to specify, in advance, the types that should be included for consideration during deserialization.

Normally, when passing parameters and return values between a client and a service, both endpoints share all of the data contracts of the data to be transmitted.

When data arrives at a receiving endpoint, the WCF runtime attempts to deserialize the data into an instance of a common language runtime (CLR) type. The type that is instantiated for deserialization is chosen by first inspecting the incoming message to determine the data contract to which the contents of the message conform. The deserialization engine then attempts to find a CLR type that implements a data contract compatible with the message contents. The set of candidate types that the deserialization engine allows for during this process is referred to as the deserializer's set of "known types."

One way to let the deserialization engine know about a type is by using the KnownTypeAttribute. The attribute cannot be applied to individual data members, only to whole data contract types. The attribute is applied to an outer type that can be a class or a structure. In its most basic usage, applying the attribute specifies a type as a "known type." This causes the known type to be a part of the set of known types whenever an object of the outer type or any object referred to through its members is being deserialized.

Let's Do A Practical Example to Follow

Create a new**WCF Service Application**and implement the following model:

C#

Copy Code

[DataContract]

public abstract class Person

{

[DataMember]

public int Code { get; set; }

[DataMember]

public string Name { get; set; }

}

[DataContract]

public class Student : Person

{

[DataMember]

public int StudentId { get; set; }

}

[DataContract]

public class Teacher : Person

{

[DataMember]

public int TeacherId { get; set; }

}

Suppose we want to create a service that can give us lists of all the entities in the system (include Students & Teachers). First, we define the contract in the following way:

C#

Copy Code

[ServiceContract]

public interface IStudentService

{

[OperationContract]

IEnumerable<Person> GetAll();

}

As you can see, the output of GetAll method is Person (**Base type**of Student & Teacher). So the service we'll implement is in the following form:

C#

Copy Code

public class StudentService : IStudentService

{

public IEnumerable<Person> GetAll()

{

List<Person> listOfPerson = new List<Person>();

listOfPerson.Add( new Student()

{ Code = 1, StudentId = 123, Name = "student 1" } );

listOfPerson.Add( new Student()

{ Code = 1, StudentId = 124, Name = "student 2"} );

listOfPerson.Add( new Student()

{ Code = 1, StudentId = 125, Name = "student 3"} );

listOfPerson.Add( new Teacher()

{ Code = 2, TeacherId = 321, Name = "Mehran mousavi"} );

listOfPerson.Add( new Teacher()

{ Code = 2, TeacherId = 322, Name = "Teacher 2" } );

listOfPerson.Add( new Teacher()

{ Code = 2, TeacherId = 323, Name = "Teacher 3"} );

return listOfPerson;

}

To display the output of this WCF Service, create a new ConsoleApplication project and add this service by AddServiceReference to it.

Main function of our ConsoleApplication project is as follows:

C#

Copy Code

class Program

{

static void Main( string[] args )

{

StudentService.StudentServiceClient client =

new StudentService.StudentServiceClient();

client.GetAll().ToList().ForEach( \_record =>

{

Console.Write( "Name : {0}", \_record.Name );

Console.WriteLine( "Code : {0}", \_record.Code );

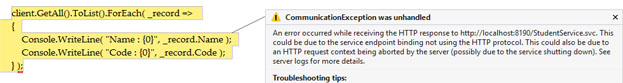
} );

Console.ReadLine();

}

}

After build and run the ConsoleApplication,we get a error !!!



Why !!? The problem is that when you try to invoke the service the actual implementation returns a ChildModel for **WCF Deserialize Engine** has no knowledge. The clients of the service neither have knowledge of this type.

So you need to explicitly indicate this class that you are using in the implementation but is not part of the contract. This could be done by using the KnownType attribute in base Class of Student & Teacher:

C#

Copy Code

[DataContract]

[KnownType( typeof( Student ) )]

[KnownType( typeof(Teacher) )]

public abstract class Person

{

[DataMember]

public int Code { get; set; }

[DataMember]

public string Name { get; set; }

}

After changing your WCF Service, now you can run Client (ConsoleApplication) and see the result successfully ...