**Part 6 - WCF DataContract and DataMember**

Per comprendere gli attributi DataContract e DataMember in WCF, per prima cosa capiamo cosa si intende per Serializzazione. Per quanto riguarda WCF, la Serializzazione è il processo di conversione di un oggetto in una rappresentazione XML. Il processo inverso, ovvero la ricostruzione dello stesso oggetto dall'XML, è chiamato Deserializzazione.

For a complex type like Customer, Employee, Student to be serialized, the complex type can either be decorated with   
**1. SerializableAttribute or**  
**2. DataContractAttribute**

Con .NET 3.5 SP1 e versioni successive, non dobbiamo usare esplicitamente gli attributi DataContract o DataMember. Data Contract Serializer serializzerà tutte le proprietà pubbliche del tuo tipo complesso in ordine alfabetico. Per impostazione predefinita, i campi e le proprietà private non vengono serializzati. Se decoriamo un tipo complesso, con l'attributo [Serializable] DataContractSerializer serializza tutti i campi. Con l'attributo [Serializable] non abbiamo un controllo esplicito su quali campi includere ed escludere nei dati serializzati. Se decoriamo un tipo complesso con l'attributo [Datacontract], DataContractSerializer serializza i campi contrassegnati con l'attributo [DataMember]. I campi che non sono contrassegnati con l'attributo [DataMember] vengono esclusi dalla serializzazione. L'attributo [DataMember] può essere applicato ai campi privati ​​o alle proprietà pubbliche. In WCF, il modo più comune di serializzazione è contrassegnare il tipo con l'attributo DataContract e contrassegnare ogni membro che deve essere serializzato con l'attributo DataMember. Se vuoi avere un controllo esplicito su quali campi e proprietà vengono serializzati, usa gli attributi DataContract e DataMember. 1. Usando DataContractAttribute, puoi definire uno spazio dei nomi XML per i tuoi dati 2. Usando DataMemberAttribute, puoi a) Definire Nome, Ordine e se una proprietà o un campo è obbligatorio b) Inoltre, serializzare campi e proprietà privati

**Code used in the demo:**  
**SQL Script:**  
Create Table tblEmployee  
(  
 Id int,  
 Name nvarchar(50),  
 Gender nvarchar(50),  
 DateOfBirth datetime  
)  
  
Insert into tblEmployee values (1, 'Mark', 'Male', '10/10/1980')  
Insert into tblEmployee values (2, 'Mary', 'Female', '11/10/1981')

Insert into tblEmployee values (3, 'John', 'Male', '8/10/1979')  
  
Create procedure spGetEmployee  
@Id int  
as  
Begin  
 Select Id, Name, Gender, DateOfBirth  
 from tblEmployee   
 where Id = @Id  
End  
  
Create procedure spSaveEmployee  
@Id int,  
@Name nvarchar(50),  
@Gender nvarchar(50),  
@DateOfBirth DateTime  
as  
Begin  
 Insert into tblEmployee  
 values (@Id, @Name, @Gender, @DateOfBirth)  
End

**Employee.cs file in EmployeeService project:**  
using System;  
using System.Runtime.Serialization;  
  
namespace EmployeeService  
{  
    [DataContract(Namespace="http://pragimtech.com/Employee")]  
    public class Employee  
    {  
        private int \_id;  
        private string \_name;  
        private string \_gender;  
        private DateTime \_dateOfBirth;  
  
        [DataMember(Order=1)]  
        public int Id  
        {  
            get { return \_id; }  
            set { \_id = value; }  
        }  
  
        [DataMember(Order = 2)]  
        public string Name  
        {  
            get { return \_name; }  
            set { \_name = value; }  
        }  
  
        [DataMember(Order = 3)]  
        public string Gender  
        {  
            get { return \_gender; }  
            set { \_gender = value; }  
        }  
  
        [DataMember(Order = 4)]  
        public DateTime DateOfBirth  
        {  
            get { return \_dateOfBirth; }  
            set { \_dateOfBirth = value; }  
        }  
    }  
}  
  
**IEmployeeService.cs in EmployeeService project**  
using System.ServiceModel;  
  
namespace EmployeeService  
{  
    [ServiceContract]  
    public interface IEmployeeService  
    {  
        [OperationContract]  
        Employee GetEmployee(int Id);  
  
        [OperationContract]  
        void SaveEmployee(Employee Employee);  
    }  
}

**EmployeeService.cs in EmployeeService project**  
using System;  
using System.Data;  
using System.Data.SqlClient;  
using System.Configuration;  
  
namespace EmployeeService  
{  
    public class EmployeeService : IEmployeeService  
    {  
        public Employee GetEmployee(int Id)  
        {  
            Employee employee = new Employee();  
            string cs = ConfigurationManager.ConnectionStrings["DBCS"].ConnectionString;  
            using (SqlConnection con = new SqlConnection(cs))  
            {  
                SqlCommand cmd = new SqlCommand("spGetEmployee", con);  
                cmd.CommandType = CommandType.StoredProcedure;  
                SqlParameter parameterId = new SqlParameter();  
                parameterId.ParameterName = "@Id";  
                parameterId.Value = Id;  
                cmd.Parameters.Add(parameterId);  
                con.Open();  
                SqlDataReader reader = cmd.ExecuteReader();  
                while (reader.Read())  
                {  
                    employee.Id = Convert.ToInt32(reader["Id"]);  
                    employee.Name = reader["Name"].ToString();  
                    employee.Gender = reader["Gender"].ToString();  
                    employee.DateOfBirth = Convert.ToDateTime(reader["DateOfBirth"]);  
                }  
            }  
            return employee;  
        }  
  
        public void SaveEmployee(Employee employee)  
        {  
            string cs = ConfigurationManager.ConnectionStrings["DBCS"].ConnectionString;  
            using (SqlConnection con = new SqlConnection(cs))  
            {  
                SqlCommand cmd = new SqlCommand("spSaveEmployee", con);  
                cmd.CommandType = CommandType.StoredProcedure;  
                SqlParameter parameterId = new SqlParameter  
                {  
                    ParameterName = "@Id",  
                    Value = employee.Id  
                };  
                cmd.Parameters.Add(parameterId);  
  
                SqlParameter parameterName = new SqlParameter  
                {  
                    ParameterName = "@Name",  
                    Value = employee.Name  
                };  
                cmd.Parameters.Add(parameterName);  
  
                SqlParameter parameterName = new SqlParameter  
                {  
                    ParameterName = "@Gender",  
                    Value = employee.Gender  
                };  
                cmd.Parameters.Add(parameterGender);  
  
                SqlParameter parameterName = new SqlParameter  
                {  
                    ParameterName = "@DateOfBirth",  
                    Value = employee.DateOfBirth  
                };  
                cmd.Parameters.Add(parameterDateOfBirth);  
  
                con.Open();  
                cmd.ExecuteNonQuery();  
            }  
        }  
    }  
}

namespace EmployeeService

{

// NOTA: è possibile utilizzare il comando "Rinomina" del menu "Refactoring" per modificare il nome di classe "EmployeeService" nel codice e nel file di configurazione contemporaneamente.

public class EmployeeService : IEmployeeService

{

public void DoWork()

{

}

public Employee GetEmployee(int Id)

{

Employee employee = new Employee();

string cs = ConfigurationManager.ConnectionStrings["DBCS"].ConnectionString;

using (SqlConnection con = new SqlConnection(cs))

{

SqlCommand cmd = new SqlCommand("spGetEmployee", con);

cmd.CommandType = CommandType.StoredProcedure;

SqlParameter parameterId = new SqlParameter();

parameterId.ParameterName = "@Id";

parameterId.Value = Id;

cmd.Parameters.Add(parameterId);

con.Open();

SqlDataReader reader = cmd.ExecuteReader();

while (reader.Read())

{

employee.Id = Convert.ToInt32(reader["Id"]);

employee.Name = reader["Name"].ToString();

employee.Gender = reader["Gender"].ToString();

employee.DateOfBirth = Convert.ToDateTime(reader["DateOfBirth"]);

}

}

return employee;

}

public void SaveEmployee(Employee employee)

{

string cs = ConfigurationManager.ConnectionStrings["DBCS"].ConnectionString;

using (SqlConnection con = new SqlConnection(cs))

{

SqlCommand cmd = new SqlCommand("spSaveEmployee", con);

cmd.CommandType = CommandType.StoredProcedure;

SqlParameter parameterId = new SqlParameter

{

ParameterName = "@Id",

Value = employee.Id

};

cmd.Parameters.Add(parameterId);

SqlParameter parameterName = new SqlParameter

{

ParameterName = "@Name",

Value = employee.Name

};

cmd.Parameters.Add(parameterName);

SqlParameter parameterGender = new SqlParameter

{

ParameterName = "@Gender",

Value = employee.Gender

};

cmd.Parameters.Add(parameterGender);

SqlParameter parameterDateOfBirth = new SqlParameter

{

ParameterName = "@DateOfBirth",

Value = employee.DateOfBirth

};

cmd.Parameters.Add(parameterDateOfBirth);

con.Open();

cmd.ExecuteNonQuery();

}

}

}

}

Per comprendere gli attributi DataContract e DataMember in WCF, per prima cosa capiamo cosa si intende per Serializzazione. Per quanto riguarda WCF, la Serializzazione è il processo di conversione di un oggetto in una rappresentazione XML. Il processo inverso, ovvero la ricostruzione dello stesso oggetto dall'XML, è chiamato Deserializzazione. Per impostazione predefinita, WCF utilizza DataContractSerializer.

Per comprendere gli attributi DataContract e DataMember in WCF, per prima cosa capiamo cosa si intende per serializzazione. Per serializzare un tipo complesso come Customer, Employee, Student, il tipo complesso può essere decorato con 1. SerializableAttribute o 2. DataContractAttribute Con .NET 3.5 SP1 e versioni successive, non dobbiamo usare esplicitamente gli attributi DataContract o DataMember. Il Data Contract Serializer serializzerà tutte le proprietà pubbliche del tuo tipo complesso in ordine alfabetico. Per impostazione predefinita, i campi e le proprietà private non vengono serializzati.

Se decoriamo un tipo complesso, con l'attributo [Serializable] il DataContractSerializer serializza tutti i campi. Con l'attributo [Serializable] non abbiamo un controllo esplicito su quali campi includere ed escludere nei dati serializzati. Se decoriamo un tipo complesso con l'attributo [Datacontract], il DataContractSerializer serializza i campi contrassegnati con l'attributo [DataMember]. I campi che non sono contrassegnati con l'attributo [DataMember] vengono esclusi dalla serializzazione. L'attributo [DataMember] può essere applicato sia ai campi privati ​​che alle proprietà pubbliche.

In WCF, il modo più comune di serializzazione è contrassegnare il tipo con l'attributo DataContract e contrassegnare ogni membro che deve essere serializzato con l'attributo DataMember. Se vuoi avere un controllo esplicito su quali campi e proprietà vengono serializzati, usa gli attributi DataContract e DataMember. 1. Usando DataContractAttribute, puoi definire uno spazio dei nomi XML per i tuoi dati 2. Usando DataMemberAttribute, puoi a) Definire Nome, Ordine e se una proprietà o un campo è obbligatorio b) Inoltre, serializzare campi e proprietà privati