This page is specific to

Microsoft Visual Studio 2010/.NET Framework 4

**Using Data Contracts**

A *data contract* is a formal agreement between a service and a client that abstractly describes the data to be exchanged. That is, to communicate, the client and the service do not have to share the same types, only the same data contracts. A data contract precisely defines, for each parameter or return type, what data is serialized (turned into XML) to be exchanged.

**Data Contract Basics**

Windows Communication Foundation (WCF) uses a serialization engine called the Data Contract Serializer by default to serialize and deserialize data (convert it to and from XML). All .NET Framework primitive types, such as integers and strings, as well as certain types treated as primitives, such as [DateTime](http://msdn.microsoft.com/en-us/library/system.datetime.aspx) and [XmlElement](http://msdn.microsoft.com/en-us/library/system.xml.xmlelement.aspx), can be serialized with no other preparation and are considered as having default data contracts. Many .NET Framework types also have existing data contracts. For a full list of serializable types, see [Types Supported by the Data Contract Serializer](http://msdn.microsoft.com/en-us/library/ms731923.aspx).

New complex types that you create must have a data contract defined for them to be serializable. By default, the [DataContractSerializer](http://msdn.microsoft.com/en-us/library/system.runtime.serialization.datacontractserializer.aspx) infers the data contract and serializes all publicly visible types. All public read/write properties and fields of the type are serialized. You can opt out members from serialization by using the [IgnoreDataMemberAttribute](http://msdn.microsoft.com/en-us/library/system.runtime.serialization.ignoredatamemberattribute.aspx). You can also explicitly create a data contract by using [DataContractAttribute](http://msdn.microsoft.com/en-us/library/system.runtime.serialization.datacontractattribute.aspx) and [DataMemberAttribute](http://msdn.microsoft.com/en-us/library/system.runtime.serialization.datamemberattribute.aspx) attributes. This is normally done by applying the **DataContractAttribute** attribute to the type. This attribute can be applied to classes, structures, and enumerations. 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. For more information, see [Serializable Types](http://msdn.microsoft.com/en-us/library/cc656732.aspx).

**Example**

The following example shows a service contract (an interface) to which the [ServiceContractAttribute](http://msdn.microsoft.com/en-us/library/system.servicemodel.servicecontractattribute.aspx) and [OperationContractAttribute](http://msdn.microsoft.com/en-us/library/system.servicemodel.operationcontractattribute.aspx) attributes have been explicitly applied. The example shows that primitive types do not require a data contract, while a complex type does.

Visual Basic

<ServiceContract()> \_

Public Interface ISampleInterface

' No data contract is requred since both the parameter and return

' types are both primitive types.

<OperationContract()> \_

Function SquareRoot(ByVal root As Integer) As Double

' No Data Contract required because both parameter and return

' types are marked with the SerializableAttribute attribute.

<OperationContract()> \_

Function GetPicture(ByVal pictureUri As System.Uri) As System.Drawing.Bitmap

' The MyTypes.PurchaseOrder is a complex type, and thus

' requires a data contract.

<OperationContract()> \_

Function ApprovePurchaseOrder(ByVal po As MyTypes.PurchaseOrder) As Boolean

End Interface

C#

[ServiceContract]

public interface ISampleInterface

{

// No data contract is requred since both the parameter

// and return types are primitive types.

[OperationContract]

double SquareRoot(int root);

// No Data Contract required because both parameter and return

// types are marked with the SerializableAttribute attribute.

[OperationContract]

System.Drawing.Bitmap GetPicture(System.Uri pictureUri);

// The MyTypes.PurchaseOrder is a complex type, and thus

// requires a data contract.

[OperationContract]

bool ApprovePurchaseOrder(MyTypes.PurchaseOrder po);

}

The following example shows how a data contract for the MyTypes.PurchaseOrder type is created by applying the **DataContractAttribute** and **DataMemberAttribute** attributes to the class and its members.

Visual Basic

Namespace MyTypes

<System.Runtime.Serialization.DataContractAttribute()> \_

Public Class PurchaseOrder

Private poId\_value As Integer

' Apply the DataMemberAttribute to the property.

<DataMember()> \_

Public Property PurchaseOrderId() As Integer

Get

Return poId\_value

End Get

Set

poId\_value = value

End Set

End Property

End Class

End Namespace

C#

namespace MyTypes

{

[DataContract]

public class PurchaseOrder

{

private int poId\_value;

// Apply the DataMemberAttribute to the property.

[DataMember]

public int PurchaseOrderId

{

get { return poId\_value; }

set { poId\_value = value; }

}

}

}

**Notes**

The following notes provide items to consider when creating data contracts:

* The **IgnoreDataMemberAttribute** attribute is only honored when used with unmarked types. This includes types that are not marked with one of the **DataContractAttribute**, [SerializableAttribute](http://msdn.microsoft.com/en-us/library/system.serializableattribute.aspx), [CollectionDataContractAttribute](http://msdn.microsoft.com/en-us/library/system.runtime.serialization.collectiondatacontractattribute.aspx), or [EnumMemberAttribute](http://msdn.microsoft.com/en-us/library/system.runtime.serialization.enummemberattribute.aspx) attributes, or marked as serializable by any other means (such as [IXmlSerializable](http://msdn.microsoft.com/en-us/library/system.xml.serialization.ixmlserializable.aspx)).
* You can apply the **DataMemberAttribute** attribute to fields, and properties.
* Member accessibility levels (internal, private, protected, or public) do not affect the data contract in any way.
* The **DataMemberAttribute** attribute is ignored if it is applied to static members.
* During serialization, property-get code is called for property data members to get the value of the properties to be serialized.
* During deserialization, an uninitialized object is first created, without calling any constructors on the type. Then all data members are deserialized.
* During deserialization, property-set code is called for property data members to set the properties to the value being deserialized.
* For a data contract to be valid, it must be possible to serialize all of its data members. For a full list of serializable types, see [Types Supported by the Data Contract Serializer](http://msdn.microsoft.com/en-us/library/ms731923.aspx).  
    
  Generic types are handled in exactly the same way as non-generic types. There are no special requirements for generic parameters. For example, consider the following type.

Visual Basic

<DataContract()> \_

Public Class MyGenericType1 (Of T)

' Code not shown.

End Class

C#

[DataContract]

public class MyGenericType1<T>

{

// Code not shown.

}

This type is serializable whether the type used for the generic type parameter (*T*) is serializable or not. Because it must be possible to serialize all data members, the following type is serializable only if the generic type parameter is also serializable, as shown in the following code.

Visual Basic

<DataContract()> Public Class MyGenericType2(Of T)

<DataMember()> Dim theData As T

End Class

C#

[DataContract]

public class MyGenericType2<T>

{

[DataMember]

T theData;

}