**Controlling XML Serialization Using Attributes**

**.NET Framework 4**

Attributes can be used to control the XML serialization of an object or to create an alternate XML stream from the same set of classes. For more details about creating an alternate XML stream, see [How to: Specify an Alternate Element Name for an XML Stream](http://msdn.microsoft.com/en-us/library/athddy89.aspx).

|  |
| --- |
| **Description: 2baksw0z.note(en-us,VS.100).gifNote:** |
| If the XML generated must conform to section 5 of the World Wide Web Consortium (www.w3.org) document titled "Simple Object Access Protocol (SOAP) 1.1," use the attributes listed in [Attributes That Control Encoded SOAP Serialization](http://msdn.microsoft.com/en-us/library/b29kkt2s.aspx). |

By default, an XML element name is determined by the class or member name. In a simple class named Book, a field named ISBN will produce an XML element tag <ISBN>, as shown in the following example.

**C#**

public class Book

{

public string ISBN;

}

// When an instance of the Book class is serialized, it might

// produce this XML:

// <ISBN>1234567890</ISBN>.

This default behavior can be changed if you want to give the element a new name. The following code shows how an attribute enables this by setting the [ElementName](http://msdn.microsoft.com/en-us/library/system.xml.serialization.xmlelementattribute.elementname.aspx) property of a [XmlElementAttribute](http://msdn.microsoft.com/en-us/library/system.xml.serialization.xmlelementattribute.aspx).

**C#**

public class TaxRates{

[XmlElement(ElementName = "TaxRate")]

public decimal ReturnTaxRate;

}

For more information about attributes, see [Extending Metadata Using Attributes](http://msdn.microsoft.com/en-us/library/5x6cd29c.aspx). For a list of attributes that control XML serialization, see [Attributes That Control XML Serialization](http://msdn.microsoft.com/en-us/library/83y7df3e.aspx).

**Controlling Array Serialization**

The [XmlArrayAttribute](http://msdn.microsoft.com/en-us/library/system.xml.serialization.xmlarrayattribute.aspx) and the [XmlArrayItemAttribute](http://msdn.microsoft.com/en-us/library/system.xml.serialization.xmlarrayitemattribute.aspx) attributes are designed to control the serialization of arrays. Using these attributes, you can control the element name, namespace, and XML Schema (XSD) data type (as defined in the World Wide Web Consortium [www.w3.org] document titled "XML Schema Part 2: Datatypes"). You can also specify the types that can be included in an array.

The **XmlArrayAttribute** will determine the properties of the enclosing XML element that results when an array is serialized. For example, by default, serializing the array below will result in an XML element named Employees. The Employees element will contain a series of elements named after the array type Employee.

**C#**

public class Group{

public Employee[] Employees;

}

public class Employee{

public string Name;

}

A serialized instance might resemble the following.

<Group>

<Employees>

<Employee>

<Name>Haley</Name>

</Employee>

</Employees >

</Group>

By applying a **XmlArrayAttribute**, you can change the name of the XML element, as follows.

**C#**

public class Group{

[XmlArray("TeamMembers")]

public Employee[] Employees;

}

The resulting XML might resemble the following.

[Copy](javascript:CodeSnippet_CopyCode('CodeSnippetContainerCode_b1ed15eb-59b8-4910-b9c8-56e0c7980302');)

<Group>

<TeamMembers>

<Employee>

<Name>Haley</Name>

</Employee>

</TeamMembers>

The **XmlArrayItemAttribute**, on the other hand, controls how the items contained in the array are serialized. Note that the attribute is applied to the field returning the array.

**C#**

public class Group{

[XmlArrayItem("MemberName")]

public Employee[] Employees;

}

The resulting XML might resemble the following.

<Group>

<Employees>

<MemberName>Haley</MemberName>

</Employees>

</Group>

**Serializing Derived Classes**

Another use of the **XmlArrayItemAttribute** is to allow the serialization of derived classes. For example, another class named Manager that derives from Employee can be added to the previous example. If you do not apply the **XmlArrayItemAttribute**, the code will fail at run time because the derived class type will not be recognized. To remedy this, apply the attribute twice, each time setting the [Type](http://msdn.microsoft.com/en-us/library/system.xml.serialization.xmlarrayitemattribute.type.aspx) property for each acceptable type (base and derived).

**C#**

public class Group{

[XmlArrayItem(Type = typeof(Employee)),

XmlArrayItem(Type = typeof(Manager))]

public Employee[] Employees;

}

public class Employee{

public string Name;

}

public class Manager:Employee{

public int Level;

}

A serialized instance might resemble the following.

[Copy](javascript:CodeSnippet_CopyCode('CodeSnippetContainerCode_40e170d4-4aee-4ff4-83dd-5eb307f002eb');)

<Group>

<Employees>

<Employee>

<Name>Haley</Name>

</Employee>

<Employee xsi:type = "Manager">

<Name>Ann</Name>

<Level>3</Level>

<Employee>

</Employees >

</Group>

**Serializing an Array as a Sequence of Elements**

You can also serialize an array as a flat sequence of XML elements by applying a **XmlElementAttribute** to the field returning the array as follows.

**C#**

public class Group{

[XmlElement]

public Employee[] Employees;

}

A serialized instance might resemble the following.

[Copy](javascript:CodeSnippet_CopyCode('CodeSnippetContainerCode_82ae9584-5a25-4dbc-bbbe-c312aec02761');)

<Group>

<Employees>

<Name>Haley</Name>

</Employees>

<Employees>

<Name>Noriko</Name>

</Employees>

<Employees>

<Name>Marco</Name>

</Employees>

</Group>

Another way to differentiate the two XML streams is to use the XML Schema Definition tool to generate the XML Schema (XSD) document files from the compiled code. (For more details on using the tool, see [The XML Schema Definition Tool and XML Serialization](http://msdn.microsoft.com/en-us/library/943242d1.aspx).) When no attribute is applied to the field, the schema describes the element in the following manner.

[Copy](javascript:CodeSnippet_CopyCode('CodeSnippetContainerCode_2668e122-8d54-46d7-a22c-d29758b2207d');)

<xs:element minOccurs="0" maxOccurs ="1" name="Employees" type="ArrayOfEmployee" />

When the **XmlElementAttribute** is applied to the field, the resulting schema describes the element as follows.

[Copy](javascript:CodeSnippet_CopyCode('CodeSnippetContainerCode_7605013a-3993-4875-bed4-3a2d34069090');)

<xs:element minOccurs="0" maxOccurs="unbounded" name="Employees" type="Employee" />

**Serializing an ArrayList**

The [ArrayList](http://msdn.microsoft.com/en-us/library/system.collections.arraylist.aspx) class can contain a collection of diverse objects. You can therefore use a **ArrayList** much as you use an array. Instead of creating a field that returns an array of typed objects, however, you can create a field that returns a single **ArrayList**. However, as with arrays, you must inform the [XmlSerializer](http://msdn.microsoft.com/en-us/library/system.xml.serialization.xmlserializer.aspx) of the types of objects the **ArrayList** contains. To accomplish this, assign multiple instances of the **XmlElementAttribute** to the field, as shown in the following example.

**C#**

public class Group{

[XmlElement(Type = typeof(Employee)),

XmlElement(Type = typeof(Manager))]

public ArrayList Info;

}

**Controlling Serialization of Classes Using XmlRootAttribute and XmlTypeAttribute**

There are two attributes that can be applied to a class (and only a class): [XmlRootAttribute](http://msdn.microsoft.com/en-us/library/system.xml.serialization.xmlrootattribute.aspx) and [XmlTypeAttribute](http://msdn.microsoft.com/en-us/library/system.xml.serialization.xmltypeattribute.aspx). These attributes are very similar. The **XmlRootAttribute** can be applied to only one class: the class that, when serialized, represents the XML document's opening and closing element—in other words, the root element. The **XmlTypeAttribute**, on the other hand, can be applied to any class, including the root class.

For example, in the previous examples, the Group class is the root class, and all its public fields and properties become the XML elements found in the XML document. Therefore, there can be only one root class. By applying the **XmlRootAttribute**, you can control the XML stream generated by the **XmlSerializer**. For example, you can change the element name and namespace.

The **XmlTypeAttribute** allows you to control the schema of the generated XML. This capability is useful when you need to publish the schema through an XML Web service. The following example applies both the **XmlTypeAttribute** and the **XmlRootAttribute** to the same class.

**C#**

[XmlRoot("NewGroupName")]

[XmlType("NewTypeName")]

public class Group{

public Employee[] Employees;

}

If this class is compiled, and the XML Schema Definition tool is used to generate its schema, you would find the following XML describing Group.

<xs:element name="NewGroupName" type="NewTypeName">

In contrast, if you were to serialize an instance of the class, only NewGroupName would be found in the XML document.

<NewGroupName>

. . .

</NewGroupName>

**Preventing Serialization with the XmlIgnoreAttribute**

There might be situations when a public property or field does not need to be serialized. For example, a field or property could be used to contain metadata. In such cases, apply the [XmlIgnoreAttribute](http://msdn.microsoft.com/en-us/library/system.xml.serialization.xmlignoreattribute.aspx) to the field or property and the **XmlSerializer** will skip over it.