**Examples of XML Serialization**

**.NET Framework 4**

XML serialization can take more than one form, from simple to complex. For example, you can serialize a class that simply consists of public fields and properties, as shown in [Introducing XML Serialization](http://msdn.microsoft.com/en-us/library/182eeyhh.aspx). The following code examples address various advanced scenarios, including how to use XML serialization to generate an XML stream that conforms to a specific XML Schema (XSD) document.

**Serializing a DataSet**

Besides serializing an instance of a public class, an instance of a [DataSet](http://msdn.microsoft.com/en-us/library/system.data.dataset.aspx) can also be serialized, as shown in the following code example.

**C#**

private void SerializeDataSet(string filename){

XmlSerializer ser = new XmlSerializer(typeof(DataSet));

// Creates a DataSet; adds a table, column, and ten rows.

DataSet ds = new DataSet("myDataSet");

DataTable t = new DataTable("table1");

DataColumn c = new DataColumn("thing");

t.Columns.Add(c);

ds.Tables.Add(t);

DataRow r;

for(int i = 0; i<10;i++){

r = t.NewRow();

r[0] = "Thing " + i;

t.Rows.Add(r);

}

TextWriter writer = new StreamWriter(filename);

ser.Serialize(writer, ds);

writer.Close();

}

**Serializing an XmlElement and XmlNode**

You can also serialize instances of a [XmlElement](http://msdn.microsoft.com/en-us/library/system.xml.xmlelement.aspx) or [XmlNode](http://msdn.microsoft.com/en-us/library/system.xml.xmlnode.aspx) class, as shown in the following code example.

**C#**

private void SerializeElement(string filename){

XmlSerializer ser = new XmlSerializer(typeof(XmlElement));

XmlElement myElement=

new XmlDocument().CreateElement("MyElement", "ns");

myElement.InnerText = "Hello World";

TextWriter writer = new StreamWriter(filename);

ser.Serialize(writer, myElement);

writer.Close();

}

private void SerializeNode(string filename){

XmlSerializer ser = new XmlSerializer(typeof(XmlNode));

XmlNode myNode= new XmlDocument().

CreateNode(XmlNodeType.Element, "MyNode", "ns");

myNode.InnerText = "Hello Node";

TextWriter writer = new StreamWriter(filename);

ser.Serialize(writer, myNode);

writer.Close();

}

**Serializing a Class that Contains a Field Returning a Complex Object**

If a property or field returns a complex object (such as an array or a class instance), the [XmlSerializer](http://msdn.microsoft.com/en-us/library/system.xml.serialization.xmlserializer.aspx) converts it to an element nested within the main XML document. For example, the first class in the following code example returns an instance of the second class.

**C#**

public class PurchaseOrder

{

public Address MyAddress;

}

public class Address

{

public string FirstName;

}

The serialized XML output might resemble the following.

<PurchaseOrder>

<Address>

<FirstName>George</FirstName>

</Address>

</PurchaseOrder>

**Serializing an Array of Objects**

You can also serialize a field that returns an array of objects, as shown in the following code example.

**C#**

public class PurchaseOrder

{

public Item [] ItemsOrders

}

public class Item

{

public string ItemID

public decimal ItemPrice

}

The serialized class instance might resemble the following, if two items are ordered.

<PurchaseOrder xmlns:xsi=http://www.w3.org/2001/XMLSchema-instance xmlns:xsd="http://www.w3.org/2001/XMLSchema">

<Items>

<Item>

<ItemID>aaa111</ItemID>

<ItemPrice>34.22</ItemPrice>

<Item>

<Item>

<ItemID>bbb222</ItemID>

<ItemPrice>2.89</ItemPrice>

<Item>

</Items>

</PurchaseOrder>

**Serializing a Class that Implements the ICollection Interface**

You can create your own collection classes by implementing the [ICollection](http://msdn.microsoft.com/en-us/library/system.collections.icollection.aspx) interface, and use the [XmlSerializer](http://msdn.microsoft.com/en-us/library/system.xml.serialization.xmlserializer.aspx) to serialize instances of these classes. Note that when a class implements the **ICollection** interface, only the collection contained by the class is serialized. Any public properties or fields added to the class will not be serialized. The class must include an **Add** method and an **Item** property (C# indexer) to be serialized.

**C#**

using System;

using System.IO;

using System.Collections;

using System.Xml.Serialization;

public class Test{

static void Main(){

Test t = new Test();

t.SerializeCollection("coll.xml");

}

private void SerializeCollection(string filename){

Employees Emps = new Employees();

// Note that only the collection is serialized -- not the

// CollectionName or any other public property of the class.

Emps.CollectionName = "Employees";

Employee John100 = new Employee("John", "100xxx");

Emps.Add(John100);

XmlSerializer x = new XmlSerializer(typeof(Employees));

TextWriter writer = new StreamWriter(filename);

x.Serialize(writer, Emps);

}

}

public class Employees:ICollection{

public string CollectionName;

private ArrayList empArray = new ArrayList();

public Employee this[int index]{

get{return (Employee) empArray[index];}

}

public void CopyTo(Array a, int index){

empArray.CopyTo(a, index);

}

public int Count{

get{return empArray.Count;}

}

public object SyncRoot{

get{return this;}

}

public bool IsSynchronized{

get{return false;}

}

public IEnumerator GetEnumerator(){

return empArray.GetEnumerator();

}

public void Add(Employee newEmployee){

empArray.Add(newEmployee);

}

}

public class Employee{

public string EmpName;

public string EmpID;

public Employee(){}

public Employee(string empName, string empID){

EmpName = empName;

EmpID = empID;

}

}

**Purchase Order Example**

You can cut and paste the following example code into a text file renamed with a .cs or .vb file name extension. Use the C# or Visual Basic compiler to compile the file. Then run it using the name of the executable.

This example uses a simple scenario to demonstrate how an instance of an object is created and serialized into a file stream using the [Serialize](http://msdn.microsoft.com/en-us/library/system.xml.serialization.xmlserializer.serialize.aspx) method. The XML stream is saved to a file, and the same file is then read back and reconstructed into a copy of the original object using the [Deserialize](http://msdn.microsoft.com/en-us/library/system.xml.serialization.xmlserializer.deserialize.aspx) method.

In this example, a class named PurchaseOrder is serialized and then deserialized. A second class named Address is also included because the public field named ShipTo must be set to an Address. Similarly, an OrderedItem class is included because an array of OrderedItem objects must be set to the OrderedItems field. Finally, a class named Test contains the code that serializes and deserializes the classes.

The CreatePO method creates the PurchaseOrder, Address, and OrderedItem class objects, and sets the public field values. The method also constructs an instance of the **XmlSerializer** class that is used to serialize and deserialize the PurchaseOrder. Note that the code passes the type of the class that will be serialized to the constructor. The code also creates a [FileStream](http://msdn.microsoft.com/en-us/library/system.io.filestream.aspx) that is used to write the XML stream to an XML document.

The ReadPo method is a little simpler. It just creates objects to deserialize and reads out their values. As with the CreatePo method, you must first construct a **XmlSerializer**, passing the type of the class to be deserialized to the constructor. Also, a [FileStream](http://msdn.microsoft.com/en-us/library/system.io.filestream.aspx) is required to read the XML document. To deserialize the objects, call the **Deserialize** method with the **FileStream** as an argument. The deserialized object must be cast to an object variable of type PurchaseOrder. The code then reads the values of the deserialized PurchaseOrder. Note that you can also read the PO.xml file that is created to see the actual XML output.

**C#**

using System;

using System.Xml;

using System.Xml.Serialization;

using System.IO;

// The XmlRootAttribute allows you to set an alternate name

// (PurchaseOrder) for the XML element and its namespace. By

// default, the XmlSerializer uses the class name. The attribute

// also allows you to set the XML namespace for the element. Lastly,

// the attribute sets the IsNullable property, which specifies whether

// the xsi:null attribute appears if the class instance is set to

// a null reference.

[XmlRootAttribute("PurchaseOrder", Namespace="http://www.cpandl.com",

IsNullable = false)]

public class PurchaseOrder

{

public Address ShipTo;

public string OrderDate;

// The XmlArrayAttribute changes the XML element name

// from the default of "OrderedItems" to "Items".

[XmlArrayAttribute("Items")]

public OrderedItem[] OrderedItems;

public decimal SubTotal;

public decimal ShipCost;

public decimal TotalCost;

}

public class Address

{

// The XmlAttribute instructs the XmlSerializer to serialize the

// Name field as an XML attribute instead of an XML element (the

// default behavior).

[XmlAttribute]

public string Name;

public string Line1;

// Setting the IsNullable property to false instructs the

// XmlSerializer that the XML attribute will not appear if

// the City field is set to a null reference.

[XmlElementAttribute(IsNullable = false)]

public string City;

public string State;

public string Zip;

}

public class OrderedItem

{

public string ItemName;

public string Description;

public decimal UnitPrice;

public int Quantity;

public decimal LineTotal;

// Calculate is a custom method that calculates the price per item

// and stores the value in a field.

public void Calculate()

{

LineTotal = UnitPrice \* Quantity;

}

}

public class Test

{

public static void Main()

{

// Read and write purchase orders.

Test t = new Test();

t.CreatePO("po.xml");

t.ReadPO("po.xml");

}

private void CreatePO(string filename)

{

// Creates an instance of the XmlSerializer class;

// specifies the type of object to serialize.

XmlSerializer serializer =

new XmlSerializer(typeof(PurchaseOrder));

TextWriter writer = new StreamWriter(filename);

PurchaseOrder po=new PurchaseOrder();

// Creates an address to ship and bill to.

Address billAddress = new Address();

billAddress.Name = "Teresa Atkinson";

billAddress.Line1 = "1 Main St.";

billAddress.City = "AnyTown";

billAddress.State = "WA";

billAddress.Zip = "00000";

// Sets ShipTo and BillTo to the same addressee.

po.ShipTo = billAddress;

po.OrderDate = System.DateTime.Now.ToLongDateString();

// Creates an OrderedItem.

OrderedItem i1 = new OrderedItem();

i1.ItemName = "Widget S";

i1.Description = "Small widget";

i1.UnitPrice = (decimal) 5.23;

i1.Quantity = 3;

i1.Calculate();

// Inserts the item into the array.

OrderedItem [] items = {i1};

po.OrderedItems = items;

// Calculate the total cost.

decimal subTotal = new decimal();

foreach(OrderedItem oi in items)

{

subTotal += oi.LineTotal;

}

po.SubTotal = subTotal;

po.ShipCost = (decimal) 12.51;

po.TotalCost = po.SubTotal + po.ShipCost;

// Serializes the purchase order, and closes the TextWriter.

serializer.Serialize(writer, po);

writer.Close();

}

protected void ReadPO(string filename)

{

// Creates an instance of the XmlSerializer class;

// specifies the type of object to be deserialized.

XmlSerializer serializer = new XmlSerializer(typeof(PurchaseOrder));

// If the XML document has been altered with unknown

// nodes or attributes, handles them with the

// UnknownNode and UnknownAttribute events.

serializer.UnknownNode+= new

XmlNodeEventHandler(serializer\_UnknownNode);

serializer.UnknownAttribute+= new

XmlAttributeEventHandler(serializer\_UnknownAttribute);

// A FileStream is needed to read the XML document.

FileStream fs = new FileStream(filename, FileMode.Open);

// Declares an object variable of the type to be deserialized.

PurchaseOrder po;

// Uses the Deserialize method to restore the object's state

// with data from the XML document. \*/

po = (PurchaseOrder) serializer.Deserialize(fs);

// Reads the order date.

Console.WriteLine ("OrderDate: " + po.OrderDate);

// Reads the shipping address.

Address shipTo = po.ShipTo;

ReadAddress(shipTo, "Ship To:");

// Reads the list of ordered items.

OrderedItem [] items = po.OrderedItems;

Console.WriteLine("Items to be shipped:");

foreach(OrderedItem oi in items)

{

Console.WriteLine("\t"+

oi.ItemName + "\t" +

oi.Description + "\t" +

oi.UnitPrice + "\t" +

oi.Quantity + "\t" +

oi.LineTotal);

}

// Reads the subtotal, shipping cost, and total cost.

Console.WriteLine(

"\n\t\t\t\t\t Subtotal\t" + po.SubTotal +

"\n\t\t\t\t\t Shipping\t" + po.ShipCost +

"\n\t\t\t\t\t Total\t\t" + po.TotalCost

);

}

protected void ReadAddress(Address a, string label)

{

// Reads the fields of the Address.

Console.WriteLine(label);

Console.Write("\t"+

a.Name +"\n\t" +

a.Line1 +"\n\t" +

a.City +"\t" +

a.State +"\n\t" +

a.Zip +"\n");

}

protected void serializer\_UnknownNode

(object sender, XmlNodeEventArgs e)

{

Console.WriteLine("Unknown Node:" + e.Name + "\t" + e.Text);

}

protected void serializer\_UnknownAttribute

(object sender, XmlAttributeEventArgs e)

{

System.Xml.XmlAttribute attr = e.Attr;

Console.WriteLine("Unknown attribute " +

attr.Name + "='" + attr.Value + "'");

}

}

The XML output might resemble the following.

<?xml version="1.0" encoding="utf-8"?>

<PurchaseOrder xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns:xsd="http://www.w3.org/2001/XMLSchema" xmlns="http://www.cpandl.com">

<ShipTo Name="Teresa Atkinson">

<Line1>1 Main St.</Line1>

<City>AnyTown</City>

<State>WA</State>

<Zip>00000</Zip>

</ShipTo>

<OrderDate>Wednesday, June 27, 2001</OrderDate>

<Items>

<OrderedItem>

<ItemName>Widget S</ItemName>

<Description>Small widget</Description>

<UnitPrice>5.23</UnitPrice>

<Quantity>3</Quantity>

<LineTotal>15.69</LineTotal>

</OrderedItem>

</Items>

<SubTotal>15.69</SubTotal>

<ShipCost>12.51</ShipCost>

<TotalCost>28.2</TotalCost>

</PurchaseOrder>