**Serialization**

Serialization is the process of converting the state of an object into a form that can be persisted or transported. The complement of serialization is deserialization, which converts a stream into an object. Together, these processes allow data to be easily stored and transferred.

The .NET Framework features two serializing technologies:

* Binary serialization preserves type fidelity, which is useful for preserving the state of an object between different invocations of an application. For example, you can share an object between different applications by serializing it to the Clipboard. You can serialize an object to a stream, to a disk, to memory, over the network, and so forth. Remoting uses serialization to pass objects "by value" from one computer or application domain to another.
* XML serialization serializes only public properties and fields and does not preserve type fidelity. This is useful when you want to provide or consume data without restricting the application that uses the data. Because XML is an open standard, it is an attractive choice for sharing data across the Web. SOAP is likewise an open standard, which makes it an attractive choice.

**Binary Serialization**

Serialization can be defined as the process of storing the state of an object to a storage medium. During this process, the public and private fields of the object and the name of the class, including the assembly containing the class, are converted to a stream of bytes, which is then written to a data stream. When the object is subsequently deserialized, an exact clone of the original object is created.

When implementing a serialization mechanism in an object-oriented environment, you have to make a number of tradeoffs between ease of use and flexibility. The process can be automated to a large extent, provided you are given sufficient control over the process. For example, situations may arise where simple binary serialization is not sufficient, or there might be a specific reason to decide which fields in a class need to be serialized. The following sections examine the robust serialization mechanism provided with the .NET Framework and highlight a number of important features that allow you to customize the process to meet your needs.

**XML and SOAP Serialization**

XML serialization converts (serializes) the public fields and properties of an object, or the parameters and return values of methods, into an XML stream that conforms to a specific XML Schema definition language (XSD) document. XML serialization results in strongly typed classes with public properties and fields that are converted to a serial format (in this case, XML) for storage or transport.

Because XML is an open standard, the XML stream can be processed by any application, as needed, regardless of platform. For example, XML Web services created using ASP.NET use the [XmlSerializer](http://msdn.microsoft.com/en-us/library/system.xml.serialization.xmlserializer.aspx) class to create XML streams that pass data between XML Web service applications throughout the Internet or on intranets. Conversely, deserialization takes such an XML stream and reconstructs the object.

XML serialization can also be used to serialize objects into XML streams that conform to the SOAP specification. SOAP is a protocol based on XML, designed specifically to transport procedure calls using XML.

To serialize or deserialize objects, use the **XmlSerializer** class. To create the classes to be serialized, use the XML Schema Definition tool.

**Custom Serialization**

Custom serialization is the process of controlling the serialization and deserialization of a type. By controlling serialization, it is possible to ensure serialization compatibility, which is the ability to serialize and deserialize between versions of a type without breaking the core functionality of the type. For example, in the first version of a type, there may be only two fields. In the next version of a type, several more fields are added. Yet the second version of an application must be able to serialize and deserialize both types.

**Examples of XML Serialization**

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.

Visual Basic

[Copy Code](javascript:CopyCode('ctl00_MTCS_main_ctl03_code');" \o "Copy Code)

Private Sub SerializeDataSet(filename As String)

Dim ser As XmlSerializer = new XmlSerializer(GetType(DataSet))

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

Dim ds As DataSet = new DataSet("myDataSet")

Dim t As DataTable = new DataTable("table1")

Dim c As DataColumn = new DataColumn("thing")

t.Columns.Add(c)

ds.Tables.Add(t)

Dim r As DataRow

Dim i As Integer

for i = 0 to 10

r = t.NewRow()

r(0) = "Thing " & i

t.Rows.Add(r)

Next

Dim writer As TextWriter = new StreamWriter(filename)

ser.Serialize(writer, ds)

writer.Close()

End Sub

C#

[Copy Code](javascript:CopyCode('ctl00_MTCS_main_ctl04_code');" \o "Copy Code)

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.

Visual Basic

[Copy Code](javascript:CopyCode('ctl00_MTCS_main_ctl07_code');" \o "Copy Code)

private Sub SerializeElement(filename As String)

Dim ser As XmlSerializer = new XmlSerializer(GetType(XmlElement))

Dim myElement As XmlElement = \_

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

myElement.InnerText = "Hello World"

Dim writer As TextWriter = new StreamWriter(filename)

ser.Serialize(writer, myElement)

writer.Close()

End Sub

Private Sub SerializeNode(filename As String)

Dim ser As XmlSerializer = \_

new XmlSerializer(GetType(XmlNode))

Dim myNode As XmlNode = new XmlDocument(). \_

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

myNode.InnerText = "Hello Node"

Dim writer As TextWriter = new StreamWriter(filename)

ser.Serialize(writer, myNode)

writer.Close()

End Sub

C#

[Copy Code](javascript:CopyCode('ctl00_MTCS_main_ctl08_code');" \o "Copy Code)

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.

Visual Basic

[Copy Code](javascript:CopyCode('ctl00_MTCS_main_ctl10_code');" \o "Copy Code)

Public Class PurchaseOrder

Public MyAdress As Address

End Class

Public Class Address

Public FirstName As String

End Class

C#

[Copy Code](javascript:CopyCode('ctl00_MTCS_main_ctl11_code');" \o "Copy Code)

public class PurchaseOrder

{

public Address MyAddress;

}

public class Address

{

public string FirstName;

}

The serialized XML output might resemble the following.

[Copy Code](javascript:CopyCode('ctl00_MTCS_main_ctl12_code');" \o "Copy Code)

<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.

Visual Basic

[Copy Code](javascript:CopyCode('ctl00_MTCS_main_ctl13_code');" \o "Copy Code)

Public Class PurchaseOrder

public ItemsOrders () As Item

End Class

Public Class Item

Public ItemID As String

Public ItemPrice As decimal

End Class

C#

[Copy Code](javascript:CopyCode('ctl00_MTCS_main_ctl14_code');" \o "Copy Code)

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.

[Copy Code](javascript:CopyCode('ctl00_MTCS_main_ctl15_code');" \o "Copy Code)

<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.

Visual Basic

[Copy Code](javascript:CopyCode('ctl00_MTCS_main_ctl18_code');" \o "Copy Code)

Imports System

Imports System.IO

Imports System.Collections

Imports System.Xml.Serialization

Public Class Test

Shared Sub Main()

Dim t As Test= new Test()

t.SerializeCollection("coll.xml")

End Sub

Private Sub SerializeCollection(filename As String)

Dim Emps As Employees = new Employees()

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

' CollectionName or any other public property of the class.

Emps.CollectionName = "Employees"

Dim John100 As Employee = new Employee("John", "100xxx")

Emps.Add(John100)

Dim x As XmlSerializer = new XmlSerializer(GetType(Employees))

Dim writer As TextWriter = new StreamWriter(filename)

x.Serialize(writer, Emps)

writer.Close()

End Sub

End Class

Public Class Employees

Implements ICollection

Public CollectionName As String

Private empArray As ArrayList = new ArrayList()

Public ReadOnly Default Overloads \_

Property Item(index As Integer) As Employee

get

return CType (empArray(index), Employee)

End Get

End Property

Public Sub CopyTo(a As Array, index As Integer) \_

Implements ICollection.CopyTo

empArray.CopyTo(a, index)

End Sub

Public ReadOnly Property Count () As integer Implements \_

ICollection.Count

get

Count = empArray.Count

End Get

End Property

Public ReadOnly Property SyncRoot ()As Object \_

Implements ICollection.SyncRoot

get

return me

End Get

End Property

Public ReadOnly Property IsSynchronized () As Boolean \_

Implements ICollection.IsSynchronized

get

return false

End Get

End Property

Public Function GetEnumerator() As IEnumerator \_

Implements IEnumerable.GetEnumerator

return empArray.GetEnumerator()

End Function

Public Function Add(newEmployee As Employee) As Integer

empArray.Add(newEmployee)

return empArray.Count

End Function

End Class

Public Class Employee

Public EmpName As String

Public EmpID As String

Public Sub New ()

End Sub

Public Sub New (newName As String , newID As String )

EmpName = newName

EmpID = newID

End Sub

End Class

C#

[Copy Code](javascript:CopyCode('ctl00_MTCS_main_ctl19_code');" \o "Copy Code)

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.

Visual Basic

[Copy Code](javascript:CopyCode('ctl00_MTCS_main_ctl24_code');" \o "Copy Code)

Imports System

Imports System.Xml

Imports System.Xml.Serialization

Imports System.IO

Imports Microsoft.VisualBasic

' 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 ShipTo As Address

Public OrderDate As String

' The XmlArrayAttribute changes the XML element name

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

<XmlArrayAttribute("Items")> \_

Public OrderedItems() As OrderedItem

Public SubTotal As Decimal

Public ShipCost As Decimal

Public TotalCost As Decimal

End Class

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 Name As String

Public Line1 As String

' 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 City As String

Public State As String

Public Zip As String

End Class

Public Class OrderedItem

Public ItemName As String

Public Description As String

Public UnitPrice As Decimal

Public Quantity As Integer

Public LineTotal As Decimal

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

' and stores the value in a field.

Public Sub Calculate()

LineTotal = UnitPrice \* Quantity

End Sub

End Class

Public Class Test

Public Shared Sub Main()

' Read and write purchase orders.

Dim t As New Test()

t.CreatePO("po.xml")

t.ReadPO("po.xml")

End Sub

Private Sub CreatePO(filename As String)

' Creates an instance of the XmlSerializer class;

' specifies the type of object to serialize.

Dim serializer As New XmlSerializer(GetType(PurchaseOrder))

Dim writer As New StreamWriter(filename)

Dim po As New PurchaseOrder()

' Creates an address to ship and bill to.

Dim billAddress As New Address()

billAddress.Name = "Teresa Atkinson"

billAddress.Line1 = "1 Main St."

billAddress.City = "AnyTown"

billAddress.State = "WA"

billAddress.Zip = "00000"

' Set ShipTo and BillTo to the same addressee.

po.ShipTo = billAddress

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

' Creates an OrderedItem.

Dim i1 As New OrderedItem()

i1.ItemName = "Widget S"

i1.Description = "Small widget"

i1.UnitPrice = CDec(5.23)

i1.Quantity = 3

i1.Calculate()

' Inserts the item into the array.

Dim items(0) As OrderedItem

items(0) = i1

po.OrderedItems = items

' Calculates the total cost.

Dim subTotal As New Decimal()

Dim oi As OrderedItem

For Each oi In items

subTotal += oi.LineTotal

Next oi

po.SubTotal = subTotal

po.ShipCost = CDec(12.51)

po.TotalCost = po.SubTotal + po.ShipCost

' Serializes the purchase order, and close the TextWriter.

serializer.Serialize(writer, po)

writer.Close()

End Sub

Protected Sub ReadPO(filename As String)

' Creates an instance of the XmlSerializer class;

' specifies the type of object to be deserialized.

Dim serializer As New XmlSerializer(GetType(PurchaseOrder))

' If the XML document has been altered with unknown

' nodes or attributes, handles them with the

' UnknownNode and UnknownAttribute events.

AddHandler serializer.UnknownNode, AddressOf serializer\_UnknownNode

AddHandler serializer.UnknownAttribute, AddressOf \_

serializer\_UnknownAttribute

' A FileStream is needed to read the XML document.

Dim fs As New FileStream(filename, FileMode.Open)

' Declare an object variable of the type to be deserialized.

Dim po As PurchaseOrder

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

' with data from the XML document.

po = CType(serializer.Deserialize(fs), PurchaseOrder)

' Reads the order date.

Console.WriteLine(("OrderDate: " & po.OrderDate))

' Reads the shipping address.

Dim shipTo As Address = po.ShipTo

ReadAddress(shipTo, "Ship To:")

' Reads the list of ordered items.

Dim items As OrderedItem() = po.OrderedItems

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

Dim oi As OrderedItem

For Each oi In items

Console.WriteLine((ControlChars.Tab & oi.ItemName & \_

ControlChars.Tab & \_

oi.Description & ControlChars.Tab & oi.UnitPrice & \_

ControlChars.Tab & \_

oi.Quantity & ControlChars.Tab & oi.LineTotal))

Next oi

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

Console.WriteLine((ControlChars.Cr & New String \_

(ControlChars.Tab, 5) & \_

" Subtotal" & ControlChars.Tab & po.SubTotal & ControlChars.Cr & \_

New String(ControlChars.Tab, 5) & " Shipping" & ControlChars.Tab & \_

po.ShipCost & ControlChars.Cr & New String(ControlChars.Tab, 5) & \_

" Total" & New String(ControlChars.Tab, 2) & po.TotalCost))

End Sub

Protected Sub ReadAddress(a As Address, label As String)

' Reads the fields of the Address.

Console.WriteLine(label)

Console.Write((ControlChars.Tab & a.Name & ControlChars.Cr & \_

ControlChars.Tab & a.Line1 & ControlChars.Cr & ControlChars.Tab & \_

a.City & ControlChars.Tab & a.State & ControlChars.Cr & \_

ControlChars.Tab & a.Zip & ControlChars.Cr))

End Sub

Protected Sub serializer\_UnknownNode(sender As Object, e As \_

XmlNodeEventArgs)

Console.WriteLine(("Unknown Node:" & e.Name & \_

ControlChars.Tab & e.Text))

End Sub

Protected Sub serializer\_UnknownAttribute(sender As Object, \_

e As XmlAttributeEventArgs)

Dim attr As System.Xml.XmlAttribute = e.Attr

Console.WriteLine(("Unknown attribute " & attr.Name & "='" & \_

attr.Value & "'"))

End Sub 'serializer\_UnknownAttribute

End Class 'Test

C#

[Copy Code](javascript:CopyCode('ctl00_MTCS_main_ctl25_code');" \o "Copy Code)

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.

[Copy Code](javascript:CopyCode('ctl00_MTCS_main_ctl26_code');" \o "Copy Code)

<?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>

**How to: Serialize an Object**

To serialize an object, first create the object that is to be serialized and set its public properties and fields. To do this, you must determine the transport format in which the XML stream is to be stored, either as a stream or as a file. For example, if the XML stream must be saved in a permanent form, create a [FileStream](http://msdn.microsoft.com/en-us/library/system.io.filestream.aspx) object.

|  |
| --- |
| **Note:** |
| For more examples of XML serialization, see [Examples of XML Serialization](http://msdn.microsoft.com/en-us/library/58a18dwa.aspx). |

**To serialize an object**

1. Create the object and set its public fields and properties.
2. Construct a [XmlSerializer](http://msdn.microsoft.com/en-us/library/system.xml.serialization.xmlserializer.aspx) using the type of the object. For more information, see the **XmlSerializer** class constructors.
3. Call the [Serialize](http://msdn.microsoft.com/en-us/library/system.xml.serialization.xmlserializer.serialize.aspx) method to generate either an XML stream or a file representation of the object's public properties and fields. The following example creates a file.

Visual Basic

[Copy Code](javascript:CopyCode('ctl00_MTCS_main_ctl06_code');" \o "Copy Code)

Dim myObject As MySerializableClass = New MySerializableClass()

' Insert code to set properties and fields of the object.

Dim mySerializer As XmlSerializer = New XmlSerializer(GetType(MySerializableClass))

' To write to a file, create a StreamWriter object.

Dim myWriter As StreamWriter = New StreamWriter("myFileName.xml")

mySerializer.Serialize(myWriter, myObject)

myWriter.Close()

C#

[Copy Code](javascript:CopyCode('ctl00_MTCS_main_ctl07_code');" \o "Copy Code)

MySerializableClass myObject = new MySerializableClass();

// Insert code to set properties and fields of the object.

XmlSerializer mySerializer = new

XmlSerializer(typeof(MySerializableClass));

// To write to a file, create a StreamWriter object.

StreamWriter myWriter = new StreamWriter("myFileName.xml");

mySerializer.Serialize(myWriter, myObject);

myWriter.Close();

**How to: Deserialize an Object**

When you deserialize an object, the transport format determines whether you will create a stream or file object. After the transport format is determined, you can call the [Serialize](http://msdn.microsoft.com/en-us/library/system.xml.serialization.xmlserializer.serialize.aspx) or [Deserialize](http://msdn.microsoft.com/en-us/library/system.xml.serialization.xmlserializer.deserialize.aspx) methods, as required.

## To deserialize an object

1. Construct a [XmlSerializer](http://msdn.microsoft.com/en-us/library/system.xml.serialization.xmlserializer.aspx) using the type of the object to deserialize.
2. Call the **Deserialize** method to produce a replica of the object. When deserializing, you must cast the returned object to the type of the original, as shown in the following example, which deserializes the object into a file (although it could also be deserialized into a stream).

Visual Basic

[Copy Code](javascript:CopyCode('ctl00_MTCS_main_ctl04_code');" \o "Copy Code)

Dim myObject As MySerializableClass

' Construct an instance of the XmlSerializer with the type

' of object that is being deserialized.

Dim mySerializer As XmlSerializer = New XmlSerializer(GetType(MySerializableClass))

' To read the file, create a FileStream.

Dim myFileStream As FileStream = \_

New FileStream("myFileName.xml", FileMode.Open)

' Call the Deserialize method and cast to the object type.

myObject = CType( \_

mySerializer.Deserialize(myFileStream), MySerializableClass)

C#

[Copy Code](javascript:CopyCode('ctl00_MTCS_main_ctl05_code');" \o "Copy Code)

MySerializableClass myObject;

// Construct an instance of the XmlSerializer with the type

// of object that is being deserialized.

XmlSerializer mySerializer =

new XmlSerializer(typeof(MySerializableClass));

// To read the file, create a FileStream.

FileStream myFileStream =

new FileStream("myFileName.xml", FileMode.Open);

// Call the Deserialize method and cast to the object type.

myObject = (MySerializableClass)

mySerializer.Deserialize(myFileStream)

**Basic Serialization**

The easiest way to make a class serializable is to mark it with the [Serializable](http://msdn.microsoft.com/en-us/library/system.serializableattribute.aspx) attribute as follows.

C#

[Copy Code](javascript:CopyCode('ctl00_MTCS_main_ctl02_code');" \o "Copy Code)

[Serializable]

public class MyObject {

public int n1 = 0;

public int n2 = 0;

public String str = null;

}

The code example below shows how an instance of this class can be serialized to a file.

C#

[Copy Code](javascript:CopyCode('ctl00_MTCS_main_ctl03_code');" \o "Copy Code)

MyObject obj = new MyObject();

obj.n1 = 1;

obj.n2 = 24;

obj.str = "Some String";

IFormatter formatter = new BinaryFormatter();

Stream stream = new FileStream("MyFile.bin", FileMode.Create, FileAccess.Write, FileShare.None);

formatter.Serialize(stream, obj);

stream.Close();

This example uses a binary formatter to do the serialization. All you need to do is create an instance of the stream and the formatter you intend to use, and then call the **Serialize** method on the formatter. The stream and the object to serialize are provided as parameters to this call. Although it is not explicitly demonstrated in this example, all member variables of a class will be serialized—even variables marked as private. In this aspect, binary serialization differs from the [XMLSerializer Class](http://msdn.microsoft.com/en-us/library/system.xml.serialization.xmlserializer.aspx), which only serializes public fields. For information on excluding member variables from binary serialization, see [Selective Serialization](http://msdn.microsoft.com/en-us/library/axwwbcs6.aspx).

Restoring the object back to its former state is just as easy. First, create a stream for reading and a [formatter](http://msdn.microsoft.com/en-us/library/system.runtime.serialization.formatter.aspx), and then instruct the formatter to deserialize the object. The code example below shows how this is done.

C#

[Copy Code](javascript:CopyCode('ctl00_MTCS_main_ctl07_code');" \o "Copy Code)

IFormatter formatter = new BinaryFormatter();

Stream stream = new FileStream("MyFile.bin", FileMode.Open, FileAccess.Read, FileShare.Read);

MyObject obj = (MyObject) formatter.Deserialize(stream);

stream.Close();

// Here's the proof.

Console.WriteLine("n1: {0}", obj.n1);

Console.WriteLine("n2: {0}", obj.n2);

Console.WriteLine("str: {0}", obj.str);

The [BinaryFormatter](http://msdn.microsoft.com/en-us/library/system.runtime.serialization.formatters.binary.binaryformatter.aspx) used above is very efficient and produces a compact byte stream. All objects serialized with this formatter can also be deserialized with it, which makes it an ideal tool for serializing objects that will be deserialized on the .NET Framework. It is important to note that constructors are not called when an object is deserialized. This constraint is placed on deserialization for performance reasons. However, this violates some of the usual contracts the runtime makes with the object writer, and developers should ensure that they understand the ramifications when marking an object as serializable.

If portability is a requirement, use the [SoapFormatter](http://msdn.microsoft.com/en-us/library/system.runtime.serialization.formatters.soap.soapformatter.aspx) instead. Simply replace the **BinaryFormatter** in the code above with **SoapFormatter,** and call **Serialize** and **Deserialize** as before. This formatter produces the following output for the example used above.

[Copy Code](javascript:CopyCode('ctl00_MTCS_main_ctl10_code');" \o "Copy Code)

<SOAP-ENV:Envelope

xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"

xmlns:xsd="http://www.w3.org/2001/XMLSchema"

xmlns:SOAP- ENC="http://schemas.xmlsoap.org/soap/encoding/"

xmlns:SOAP- ENV="http://schemas.xmlsoap.org/soap/envelope/"

SOAP-ENV:encodingStyle=

"http://schemas.microsoft.com/soap/encoding/clr/1.0"

"http://schemas.xmlsoap.org/soap/encoding/"

xmlns:a1="http://schemas.microsoft.com/clr/assem/ToFile">

<SOAP-ENV:Body>

<a1:MyObject id="ref-1">

<n1>1</n1>

<n2>24</n2>

<str id="ref-3">Some String</str>

</a1:MyObject>

</SOAP-ENV:Body>

</SOAP-ENV:Envelope>

It is important to note that the **Serializable** attribute cannot be inherited. If you derive a new class from MyObject, the new class must be marked with the attribute as well, or it cannot be serialized. For example, when you attempt to serialize an instance of the class below, you will get a [SerializationException](http://msdn.microsoft.com/en-us/library/system.runtime.serialization.serializationexception.aspx) informing you that the MyStuff type is not marked as serializable.

C#

[Copy Code](javascript:CopyCode('ctl00_MTCS_main_ctl12_code');" \o "Copy Code)

public class MyStuff : MyObject

{

public int n3;

}

Using the **Serializable** attribute is convenient, but it has limitations as demonstrated above. Refer to the [Serialization Guidelines](http://msdn.microsoft.com/en-us/library/6exf3h2k.aspx) for information about when you should mark a class for serialization; serialization cannot be added to a class after it has been compiled.

**Custom Serialization**

Custom serialization is the process of controlling the serialization and deserialization of a type. By controlling serialization, it is possible to ensure serialization compatibility, which is the ability to serialize and deserialize between versions of a type without breaking the core functionality of the type. For example, in the first version of a type, there may be only two fields. In the next version of a type, several more fields are added. Yet the second version of an application must be able to serialize and deserialize both types. The following sections describe how to control serialization.

|  |
| --- |
| **Note:** |
| In versions previous to.NET Framework 4.0, serialization of custom user data in a partially trusted assembly was accomplished using the GetObjectData method. Starting with version 4.0, that method is marked with the [SecurityCriticalAttribute](http://msdn.microsoft.com/en-us/library/system.security.securitycriticalattribute.aspx) attribute which prevents execution in partially trusted assemblies. To work around this condition, implement the [ISafeSerializationData](http://msdn.microsoft.com/en-us/library/system.runtime.serialization.isafeserializationdata.aspx) interface. |

## Running Custom Methods During and After Serialization

The best practice and easiest way (introduced in version 2.0 of the .NET Framework) is to apply the following attributes to methods that are used to correct data during and after serialization:

* [OnDeserializedAttribute](http://msdn.microsoft.com/en-us/library/system.runtime.serialization.ondeserializedattribute.aspx)
* [OnDeserializingAttribute](http://msdn.microsoft.com/en-us/library/system.runtime.serialization.ondeserializingattribute.aspx)
* [OnSerializedAttribute](http://msdn.microsoft.com/en-us/library/system.runtime.serialization.onserializedattribute.aspx)
* [OnSerializingAttribute](http://msdn.microsoft.com/en-us/library/system.runtime.serialization.onserializingattribute.aspx)

These attributes allow the type to participate in any one of, or all four of the phases, of the serialization and deserialization processes. The attributes specify the methods of the type that should be invoked during each phase. The methods do not access the serialization stream but instead allow you to alter the object before and after serialization, or before and after deserialization. The attributes can be applied at all levels of the type inheritance hierarchy, and each method is called in the hierarchy from the base to the most derived. This mechanism avoids the complexity and any resulting issues of implementing the [ISerializable](http://msdn.microsoft.com/en-us/library/system.runtime.serialization.iserializable.aspx) interface by giving the responsibility for serialization and deserialization to the most derived implementation. Additionally, this mechanism allows the formatters to ignore the population of fields and retrieval from the serialization stream. For details and examples of controlling serialization and deserialization, click any of the previous links.

In addition, when adding a new field to an existing serializable type, apply the [OptionalFieldAttribute](http://msdn.microsoft.com/en-us/library/system.runtime.serialization.optionalfieldattribute.aspx) attribute to the field. The [BinaryFormatter](http://msdn.microsoft.com/en-us/library/system.runtime.serialization.formatters.binary.binaryformatter.aspx) and the [SoapFormatter](http://msdn.microsoft.com/en-us/library/system.runtime.serialization.formatters.soap.soapformatter.aspx) ignores the absence of the field when a stream that is missing the new field is processed.

## Implementing the ISerializable Interface

The other way to control serialization is achieved by implementing the [ISerializable](http://msdn.microsoft.com/en-us/library/system.runtime.serialization.iserializable.aspx) interface on an object. Note, however, that the method in the previous section supersedes this method to control serialization.

In addition, you should not use default serialization on a class that is marked with the [Serializable](http://msdn.microsoft.com/en-us/library/system.serializableattribute.aspx) attribute and has declarative or imperative security at the class level or on its constructors. Instead, these classes should always implement the **ISerializable** interface.

Implementing **ISerializable** involves implementing the [GetObjectData](http://msdn.microsoft.com/en-us/library/system.runtime.serialization.iserializable.getobjectdata.aspx) method and a special constructor that is used when the object is deserialized. The following sample code shows how to implement **ISerializable** on the MyObject class from a previous section.

C#

[Copy Code](javascript:CopyCode('ctl00_MTCS_main_ctl16_code');" \o "Copy Code)

[Serializable]

public class MyObject : ISerializable

{

public int n1;

public int n2;

public String str;

public MyObject()

{

}

protected MyObject(SerializationInfo info, StreamingContext context)

{

n1 = info.GetInt32("i");

n2 = info.GetInt32("j");

str = info.GetString("k");

}

[SecurityPermissionAttribute(SecurityAction.Demand,

SerializationFormatter =true)]

public virtual void GetObjectData(SerializationInfo info, StreamingContext context)

{

info.AddValue("i", n1);

info.AddValue("j", n2);

info.AddValue("k", str);

}

}

Visual Basic

[Copy Code](javascript:CopyCode('ctl00_MTCS_main_ctl17_code');" \o "Copy Code)

<Serializable()> \_

Public Class MyObject

Implements ISerializable

Public n1 As Integer

Public n2 As Integer

Public str As String

Public Sub New()

End Sub

Protected Sub New(ByVal info As SerializationInfo, \_

ByVal context As StreamingContext)

n1 = info.GetInt32("i")

n2 = info.GetInt32("j")

str = info.GetString("k")

End Sub 'New

<SecurityPermissionAttribute(SecurityAction.Demand, SerializationFormatter := True)> \_

Public Overridable Sub GetObjectData(ByVal info As \_

SerializationInfo, ByVal context As StreamingContext)

info.AddValue("i", n1)

info.AddValue("j", n2)

info.AddValue("k", str)

End Sub

End Class

When **GetObjectData** is called during serialization, you are responsible for populating the [SerializationInfo](http://msdn.microsoft.com/en-us/library/system.runtime.serialization.serializationinfo.aspx) provided with the method call. Add the variables to be serialized as name and value pairs. Any text can be used as the name. You have the freedom to decide which member variables are added to the **SerializationInfo**, provided that sufficient data is serialized to restore the object during deserialization. Derived classes should call the **GetObjectData** method on the base object if the latter implements **ISerializable.**

Note that serialization can allow other code to see or modify object instance data that is otherwise inaccessible. Therefore, code that performs serialization requires the [SecurityPermission](http://msdn.microsoft.com/en-us/library/system.security.permissions.securitypermission.aspx) with the [SerializationFormatter](http://msdn.microsoft.com/en-us/library/system.security.permissions.securitypermissionflag.aspx) flag specified. Under default policy, this permission is not given to Internet-downloaded or intranet code; only code on the local computer is granted this permission. The **GetObjectData** method must be explicitly protected either by demanding the **SecurityPermission** with the **SerializationFormatter** flag specified or by demanding other permissions that specifically help protect private data.

If a private field stores sensitive information, you should demand the appropriate permissions on **GetObjectData** to protect the data. Remember that code that has been granted **SecurityPermission** with the **SerializationFormatter** flag specified can view and modify the data stored in private fields. A malicious caller granted this **SecurityPermission** can view data such as hidden directory locations or granted permissions and use the data to exploit a security vulnerability on the computer. For a complete list of the security permission flags you can specify, see the [SecurityPermissionFlag Enumeration](http://msdn.microsoft.com/en-us/library/system.security.permissions.securitypermissionflag.aspx).

It is important to stress that when **ISerializable** is added to a class you must implement both **GetObjectData** and the special constructor. The compiler warns you if **GetObjectData** is missing. However, because it is impossible to enforce the implementation of a constructor, no warning is provided if the constructor is absent, and an exception is thrown when an attempt is made to deserialize a class without the constructor.

The current design was favored above a [SetObjectData](http://msdn.microsoft.com/en-us/library/system.runtime.serialization.iserializationsurrogate.setobjectdata.aspx) method to get around potential security and versioning problems. For example, a [SetObjectData](http://msdn.microsoft.com/en-us/library/system.runtime.serialization.iserializationsurrogate.setobjectdata.aspx) method must be public if it is defined as part of an interface; thus users must write code to defend against having the **SetObjectData** method called multiple times. Otherwise, a malicious application that calls the **SetObjectData** method on an object in the process of executing an operation can cause potential problems.

During deserialization, **SerializationInfo** is passed to the class using the constructor provided for this purpose. Any visibility constraints placed on the constructor are ignored when the object is deserialized; so you can mark the class as public, protected, internal, or private. However, it is a best practice to make the constructor protected unless the class is sealed, in which case the constructor should be marked private. The constructor should also perform thorough input validation. To avoid misuse by malicious code, the constructor should enforce the same security checks and permissions required to obtain an instance of the class using any other constructor. If you do not follow this recommendation, malicious code can preserialize an object, obtain control with the **SecurityPermission** with the **SerializationFormatter** flag specified and deserialize the object on a client computer bypassing any security that would have been applied during standard instance construction using a public constructor.

To restore the state of the object, simply retrieve the values of the variables from **SerializationInfo** using the names used during serialization. If the base class implements **ISerializable**, the base constructor should be called to allow the base object to restore its variables.

When you derive a new class from one that implements **ISerializable,** the derived class must implement both the constructor as well as the **GetObjectData** method if it has variables that need to be serialized. The following code example shows how this is done using the MyObject class shown previously.

C#

[Copy Code](javascript:CopyCode('ctl00_MTCS_main_ctl24_code');" \o "Copy Code)

[Serializable]

public class ObjectTwo : MyObject

{

public int num;

public ObjectTwo() : base()

{

}

protected ObjectTwo(SerializationInfo si,

StreamingContext context) : base(si,context)

{

num = si.GetInt32("num");

}

[SecurityPermissionAttribute(SecurityAction.Demand,

SerializationFormatter = true)]

public override void GetObjectData(SerializationInfo si,

StreamingContext context)

{

base.GetObjectData(si,context);

si.AddValue("num", num);

}

}

Visual Basic

[Copy Code](javascript:CopyCode('ctl00_MTCS_main_ctl25_code');" \o "Copy Code)

<Serializable()> \_

Public Class ObjectTwo

Inherits MyObject

Public num As Integer

Public Sub New()

End Sub

Protected Sub New(ByVal si As SerializationInfo, \_

ByVal context As StreamingContext)

MyBase.New(si, context)

num = si.GetInt32("num")

End Sub

<SecurityPermissionAttribute(SecurityAction.Demand, \_

SerializationFormatter := True)> \_

Public Overrides Sub GetObjectData(ByVal si As \_

SerializationInfo, ByVal context As StreamingContext)

MyBase.GetObjectData(si, context)

si.AddValue("num", num)

End Sub

End Class

Do not forget to call the base class in the deserialization constructor. If this is not done, the constructor on the base class is never called, and the object is not fully constructed after deserialization.

Objects are reconstructed from the inside out; and calling methods during deserialization can have undesirable side effects, because the methods called might refer to object references that have not been deserialized by the time the call is made. If the class being deserialized implements the [IDeserializationCallback](http://msdn.microsoft.com/en-us/library/system.runtime.serialization.ideserializationcallback.aspx), the [OnDeserialization](http://msdn.microsoft.com/en-us/library/system.runtime.serialization.ideserializationcallback.ondeserialization.aspx) method is automatically called when the entire object graph has been deserialized. At this point, all the child objects referenced have been fully restored. A hash table is a typical example of a class that is difficult to deserialize without using the event listener. It is easy to retrieve the key and value pairs during deserialization, but adding these objects back to the hash table can cause problems, because there is no guarantee that classes that derived from the hash table have been deserialized. Calling methods on a hash table at this stage is therefore not advisable.