**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.

# **In This Section**

[Serialization How-to Topics](http://msdn.microsoft.com/en-us/library/ms172360(VS.85).aspx)

Lists links to How-to topics contained in this section.

[Binary Serialization](http://msdn.microsoft.com/en-us/library/72hyey7b(VS.85).aspx)

Describes the binary serialization mechanism that is included with the common language runtime.

[XML and SOAP Serialization](http://msdn.microsoft.com/en-us/library/90c86ass(VS.85).aspx)

Describes the XML and SOAP serialization mechanism that is included with the common language runtime.

# **Reference**

[System.Runtime.Serialization](http://msdn.microsoft.com/en-us/library/system.runtime.serialization(VS.85).aspx)

Contains classes that can be used for serializing and deserializing objects.

[System.Xml.Serialization](http://msdn.microsoft.com/en-us/library/system.xml.serialization(VS.85).aspx)

Contains classes that can be used to serialize objects into XML format documents or streams.

# **Related Sections**

[.NET Remoting](http://msdn.microsoft.com/en-us/library/72x4h507(VS.85).aspx)

Describes the various communications methods available in the .NET Framework for remote communications.

[Advanced Development Technologies](http://msdn.microsoft.com/en-us/library/a493kdy6(VS.85).aspx)

Provides links to more information on sophisticated development tasks and techniques in the .NET Framework.

[XML Web Services Created Using ASP.NET and XML Web Service Clients](http://msdn.microsoft.com/en-us/library/7bkzywba(VS.85).aspx)

Provides topics that describe and explain how to program XML Web services created using ASP.NET.

# Serialization

In [computer science](http://en.wikipedia.org/wiki/Computer_science), in the context of data storage and transmission, **serialization** is the process of converting an [object](http://en.wikipedia.org/wiki/Object_(computer_science)) into a sequence of bits so that it can be stored on a storage medium (such as a [file](http://en.wikipedia.org/wiki/Computer_file), or a memory buffer) or transmitted across a [network](http://en.wikipedia.org/wiki/Computer_network) connection link. When the resulting series of bits is reread according to the serialization format, it can be used to create a semantically identical clone of the original object. For many complex objects, such as those that make extensive use of [references](http://en.wikipedia.org/wiki/Reference_(computer_science)), this process is not straightforward.

This process of serializing an object is also called **deflating** or [**marshalling**](http://en.wikipedia.org/wiki/Marshalling_(computer_science)) an object. The opposite operation, extracting a data structure from a series of bytes, is **deserialization** (which is also called **inflating** or **unmarshalling**).

### [[edit](http://en.wikipedia.org/w/index.php?title=Serialization&action=edit&section=6)] .NET Framework

In the [.NET](http://en.wikipedia.org/wiki/.NET_Framework) languages, classes can be serialized and deserialized by adding the Serializable attribute to the class.

*'VB Example*

<Serializable()> Class Employee

*// C# Example*

[Serializable]

class Employee

If new members are added to a serializable class, they can be tagged with the OptionalField attribute to allow previous versions of the object to be deserialized without error. This attribute affects only deserialization, and prevents the runtime from throwing an exception if a member is missing from the serialized stream. A member can also be marked with the NonSerialized attribute to indicate that it should not be serialized. This will allow the details of those members to be kept secret.

To modify the default deserialization (for example, to automatically initialize a member marked NonSerialized), the class must implement the IDeserializationCallback interface and define the IDeserializationCallback.OnDeserialization method.

Objects may be serialized in binary format for deserialization by other [.NET](http://en.wikipedia.org/wiki/.NET_Framework) applications. The framework also provides the SoapFormatter and XmlSerializer objects to support serialization in human-readable, cross-platform XML.

**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(VS.85).aspx) object.

|  |
| --- |
| **NoteNote** |
| For more examples of XML serialization, see [Examples of XML Serialization](http://msdn.microsoft.com/en-us/library/58a18dwa(VS.85).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(VS.85).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(VS.85).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

[[](javascript:CopyCode('ctl00_rs1_mainContentContainer_ctl05VisualBasic');)Copy Code](javascript:CopyCode('ctl00_rs1_mainContentContainer_ctl05VisualBasic');)

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#

[[](javascript:CopyCode('ctl00_rs1_mainContentContainer_ctl06CSharp');)Copy Code](javascript:CopyCode('ctl00_rs1_mainContentContainer_ctl06CSharp');)

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(VS.85).aspx) or [Deserialize](http://msdn.microsoft.com/en-us/library/system.xml.serialization.xmlserializer.deserialize(VS.85).aspx) methods, as required.

### To deserialize an object

1. Construct a [XmlSerializer](http://msdn.microsoft.com/en-us/library/system.xml.serialization.xmlserializer(VS.85).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

[[](javascript:CopyCode('ctl00_rs1_mainContentContainer_ctl04VisualBasic');)Copy Code](javascript:CopyCode('ctl00_rs1_mainContentContainer_ctl04VisualBasic');)

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#

[[](javascript:CopyCode('ctl00_rs1_mainContentContainer_ctl05CSharp');)Copy Code](javascript:CopyCode('ctl00_rs1_mainContentContainer_ctl05CSharp');)

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)

**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.

|  |
| --- |
| **NoteNote** |
| The state of a UTF-8 or UTF-7 encoded object is not preserved if the object is serialized and deserialized using different .NET Framework versions. |

**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(VS.85).aspx) attribute as follows.

C#

[[](javascript:CopyCode('ctl00_rs1_mainContentContainer_ctl02CSharp');)Copy Code](javascript:CopyCode('ctl00_rs1_mainContentContainer_ctl02CSharp');)

[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#

[[](javascript:CopyCode('ctl00_rs1_mainContentContainer_ctl03CSharp');)Copy Code](javascript:CopyCode('ctl00_rs1_mainContentContainer_ctl03CSharp');)

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(VS.85).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(VS.85).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(VS.85).aspx), and then instruct the formatter to deserialize the object. The code example below shows how this is done.

C#

[[](javascript:CopyCode('ctl00_rs1_mainContentContainer_ctl07CSharp');)Copy Code](javascript:CopyCode('ctl00_rs1_mainContentContainer_ctl07CSharp');)

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(VS.85).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(VS.85).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.

[[](javascript:CopyCode('ctl00_rs1_mainContentContainer_ctl10other');)Copy Code](javascript:CopyCode('ctl00_rs1_mainContentContainer_ctl10other');)

<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(VS.85).aspx) informing you that the MyStuff type is not marked as serializable.

C#

[[](javascript:CopyCode('ctl00_rs1_mainContentContainer_ctl12CSharp');)Copy Code](javascript:CopyCode('ctl00_rs1_mainContentContainer_ctl12CSharp');)

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(VS.85).aspx) for information about when you should mark a class for serialization; serialization cannot be added to a class after it has been compiled.

**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(VS.85).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.

**Introducing XML Serialization**

Serialization is the process of converting an object into a form that can be readily transported. For example, you can serialize an object and transport it over the Internet using HTTP between a client and a server. On the other end, deserialization reconstructs the object from the stream.

XML serialization serializes only the public fields and property values of an object into an XML stream. XML serialization does not include type information. For example, if you have a Book object that exists in the Library namespace, there is no guarantee that it will be deserialized into an object of the same type.

|  |
| --- |
| **NoteNote** |
| XML serialization does not convert methods, indexers, private fields, or read-only properties (except read-only collections). To serialize all an object's fields and properties, both public and private, use the [BinaryFormatter](http://msdn.microsoft.com/en-us/library/system.runtime.serialization.formatters.binary.binaryformatter(VS.85).aspx) instead of XML serialization. |

The central class in XML serialization is the [XmlSerializer](http://msdn.microsoft.com/en-us/library/system.xml.serialization.xmlserializer(VS.85).aspx) class, and its most important methods are the **Serialize** and **Deserialize** methods. The [XmlSerializer](http://msdn.microsoft.com/en-us/library/system.xml.serialization.xmlserializer(VS.85).aspx) creates C# files and compiles them into .dll files to perform this serialization. In .NET Framework 2.0, the [XML Serializer Generator Tool (Sgen.exe)](http://msdn.microsoft.com/en-us/library/bk3w6240(VS.85).aspx) is designed to generate these serialization assemblies in advance to be deployed with your application, and improve startup performance. The XML stream generated by the **XmlSerializer** is compliant with the World Wide Web Consortium (www.w3.org) XML Schema definition language (XSD) 1.0 recommendation. Furthermore, the data types generated are compliant with the document titled "XML Schema Part 2: Datatypes."

The data in your objects is described using programming language constructs like classes, fields, properties, primitive types, arrays, and even embedded XML in the form of **XmlElement** or **XmlAttribute** objects. You have the option of creating your own classes, annotated with attributes, or using the XML Schema Definition tool to generate the classes based on an existing XML Schema.

If you have an XML Schema, you can run the XML Schema Definition tool to produce a set of classes that are strongly typed to the schema and annotated with attributes. When an instance of such a class is serialized, the generated XML adheres to the XML Schema. Provided with such a class, you can program against an easily manipulated object model while being assured that the generated XML conforms to the XML schema. This is an alternative to using other classes in the .NET Framework, such as the **XmlReader** and **XmlWriter** classes, to parse and write an XML stream. For more information, see [XML Documents and Data](http://msdn.microsoft.com/en-us/library/2bcctyt8(VS.85).aspx). These classes allow you to parse any XML stream. In contrast, use the **XmlSerializer** when the XML stream is expected to conform to a known XML Schema.

Attributes control the XML stream generated by the **XmlSerializer** class, allowing you to set the XML namespace, element name, attribute name, and so on, of the XML stream. For more information about these attributes and how they control XML serialization, see [Controlling XML Serialization Using Attributes](http://msdn.microsoft.com/en-us/library/2baksw0z(VS.85).aspx). For a table of those attributes that are used to control the generated XML, see [Attributes That Control XML Serialization](http://msdn.microsoft.com/en-us/library/83y7df3e(VS.85).aspx).

The **XmlSerializer** class can further serialize an object and generate an encoded SOAP XML stream. The generated XML adheres to section 5 of the World Wide Web Consortium document titled "Simple Object Access Protocol (SOAP) 1.1." For more information about this process, see [How to: Serialize an Object as a SOAP-Encoded XML Stream](http://msdn.microsoft.com/en-us/library/d5wt2he6(VS.85).aspx). For a table of the attributes that control the generated XML, see [Attributes That Control Encoded SOAP Serialization](http://msdn.microsoft.com/en-us/library/b29kkt2s(VS.85).aspx).

The **XmlSerializer** class generates the SOAP messages created by, and passed to, XML Web services. To control the SOAP messages, you can apply attributes to the classes, return values, parameters, and fields found in an XML Web service file (.asmx). You can use both the attributes listed in "Attributes That Control XML Serialization" and "Attributes That Control Encoded SOAP Serialization" because an XML Web service can use either the literal or encoded SOAP style. For more information about using attributes to control the XML generated by an XML Web service, see [XML Serialization with XML Web Services](http://msdn.microsoft.com/en-us/library/564k8ys4(VS.85).aspx). For more information about SOAP and XML Web services, see [Customizing SOAP Message Formatting](http://msdn.microsoft.com/en-us/library/dkwy2d72(VS.85).aspx).

# **Security Considerations for XmlSerializer Applications**

When creating an application that uses the **XmlSerializer**, you should be aware of the following items and their implications:

* The **XmlSerializer** creates C# (.cs) files and compiles them into .dll files in the directory named by the TEMP environment variable; serialization occurs with those DLLs.

|  |
| --- |
| **NoteNote** |
| These serialization assemblies can be generated in advance and signed by using the SGen.exe tool. This does not work for the server side of Web services. In other words, it is only for client-side use, and for manual serialization. |

* The code and the DLLs are vulnerable to a malicious process at the time of creation and compilation. When using a computer running Microsoft Windows NT 4.0 or later, it might be possible for two or more users to share the temp directory. Sharing a temp directory is dangerous if the two accounts have different security privileges, and the higher-privilege account runs an application using the **XmlSerializer**. In this case, one user can breach the computer's security by replacing either the .cs or .dll file that is compiled. To eliminate this concern, always be sure that each account on the computer has its own profile. By default, the TEMP environment variable points to a different directory for each account.
* If a malicious user sends a continuous stream of XML data to a Web server (a denial of service attack), then the **XmlSerializer** continues to process the data until the computer runs low on resources.

This kind of attack is eliminated if you are using a computer running Internet Information Services (IIS), and your application is running within IIS. IIS features a gate that does not process streams longer than a set amount (the default is 4 KB). If you create an application that does not use IIS and deserializes with the **XmlSerializer**, you should implement a similar gate that prevents a denial of service attack.

* The **XmlSerializer** serializes data and runs any code using any type given to it.

There are two ways in which a malicious object presents a threat. It could run malicious code, or it could inject malicious code into the C# file created by the **XmlSerializer**. In the first case, if a malicious object tries to run a destructive procedure, code access security helps prevent any damage from being done. In the second case, there is a theoretical possibility that a malicious object may somehow inject code into the C# file created by the **XmlSerializer**. Although this issue has been examined thoroughly, and such an attack is considered unlikely, you should take the precaution of never serializing data with an unknown and untrusted type.

* Serialized sensitive data might be vulnerable.

After the **XmlSerializer** has serialized data, it can be stored as an XML file or other data store. If your data store is available to other processes, or is visible on an intranet or the Internet, the data can be stolen and used maliciously. For example, if you create an application that serializes orders that include credit card numbers, the data is highly sensitive. To help prevent this, always protect the store for your data and take steps to keep it private.

# **Serialization of a Simple Class**

The following code example shows a simple class with a public field.

Visual Basic

[[](javascript:CopyCode('ctl00_rs1_mainContentContainer_ctl12VisualBasic');)Copy Code](javascript:CopyCode('ctl00_rs1_mainContentContainer_ctl12VisualBasic');)

Public Class OrderForm

Public OrderDate As DateTime

End Class

C#

[[](javascript:CopyCode('ctl00_rs1_mainContentContainer_ctl13CSharp');)Copy Code](javascript:CopyCode('ctl00_rs1_mainContentContainer_ctl13CSharp');)

public class OrderForm

{

public DateTime OrderDate;

}

When an instance of this class is serialized, it might resemble the following.

[[](javascript:CopyCode('ctl00_rs1_mainContentContainer_ctl14other');)Copy Code](javascript:CopyCode('ctl00_rs1_mainContentContainer_ctl14other');)

<OrderForm>

<OrderDate>12/12/01</OrderDate>

</OrderForm>

For more examples of serialization, see [Examples of XML Serialization](http://msdn.microsoft.com/en-us/library/58a18dwa(VS.85).aspx).

# **Items That Can Be Serialized**

The following items can be serialized using the **XmLSerializer** class:

* Public read/write properties and fields of public classes
* Classes that implement **ICollection** or **IEnumerable**

|  |
| --- |
| **NoteNote** |
| Only collections are serialized, not public properties. |

* **XmlElement** objects
* **XmlNode** objects
* **DataSet** objects

For more information about serializing or deserializing objects, see [How to: Serialize an Object](http://msdn.microsoft.com/en-us/library/szzyf24s(VS.85).aspx) and [How to: Deserialize an Object](http://msdn.microsoft.com/en-us/library/fa420a9y(VS.85).aspx).

# **Advantages of Using XML Serialization**

The **XmlSerializer** class gives you complete and flexible control when you serialize an object as XML. If you are creating an XML Web service, you can apply attributes that control serialization to classes and members to ensure that the XML output conforms to a specific schema.

For example, **XmlSerializer** enables you to:

* Specify whether a field or property should be encoded as an attribute or an element.
* Specify an XML namespace to use.
* Specify the name of an element or attribute if a field or property name is inappropriate.

Another advantage of XML serialization is that you have no constraints on the applications you develop, as long as the XML stream that is generated conforms to a given schema. Imagine a schema that is used to describe books. It features a title, author, publisher, and ISBN number element. You can develop an application that processes the XML data in any way you want, for example, as a book order, or as an inventory of books. In either case, the only requirement is that the XML stream conforms to the specified XML Schema definition language (XSD) schema.

# **XML Serialization Considerations**

The following should be considered when using the **XmlSerializer** class:

* The Sgen.exe tool is expressly designed to generate serialization assemblies for optimum performance.
* The serialized data contains only the data itself and the structure of your classes. Type identity and assembly information are not included.
* Only public properties and fields can be serialized. Properties must have public accessors (get and set methods). If you need to serialize non-public data, use the **BinaryFormatter** class rather than XML serialization.
* A class must have a default constructor to be serialized by **XmlSerializer**.
* Methods cannot be serialized.
* **XmlSerializer** can process classes that implement **IEnumerable** or **ICollection** differently if they meet certain requirements, as follows.

A class that implements **IEnumerable** must implement a public **Add** method that takes a single parameter. The **Add** method's parameter must be consistent (polymorphic) with the type returned from the **IEnumerator.Current** property returned from the **GetEnumerator** method.

A class that implements **ICollection** in addition to **IEnumerable** (such as **CollectionBase**) must have a public **Item** indexed property (an indexer in C#) that takes an integer, and it must have a public **Count** property of type integer. The parameter passed to the **Add** method must be the same type as that returned from the **Item** property, or one of that type's bases.

For classes implementing **ICollection**, values to be serialized are retrieved from the indexed **Item** property rather than by calling **GetEnumerator**. Also, public fields and properties are not serialized, with the exception of public fields that return another collection class (one that implements **ICollection**). For an example, see [Examples of XML Serialization](http://msdn.microsoft.com/en-us/library/58a18dwa(VS.85).aspx).

# **XSD Data Type Mapping**

The World Wide Web Consortium (www.w3.org) document titled "XML Schema Part 2: Datatypes" specifies the simple data types that are allowed in an XML Schema definition language (XSD) schema. For many of these (for example, **int** and **decimal**), there is a corresponding data type in the .NET Framework. However, some XML data types do not have a corresponding data type in the .NET Framework (for example, the **NMTOKEN** data type). In such cases, if you use the XML Schema Definition tool ([XML Schema Definition Tool (Xsd.exe)](http://msdn.microsoft.com/en-us/library/x6c1kb0s(VS.85).aspx)) to generate classes from a schema, an appropriate attribute is applied to a member of type string, and its **DataType** property is set to the XML data type name. For example, if a schema contains an element named "MyToken" with the XML data type **NMTOKEN**, the generated class might contain a member as shown in the following example.

Visual Basic

[[](javascript:CopyCode('ctl00_rs1_mainContentContainer_ctl20VisualBasic');)Copy Code](javascript:CopyCode('ctl00_rs1_mainContentContainer_ctl20VisualBasic');)

<XmlElement(DataType:="NMTOKEN")> \_

Public MyToken As String

C#

[[](javascript:CopyCode('ctl00_rs1_mainContentContainer_ctl21CSharp');)Copy Code](javascript:CopyCode('ctl00_rs1_mainContentContainer_ctl21CSharp');)

[XmlElement(DataType = "NMTOKEN")]

public string MyToken;

Similarly, if you are creating a class that must conform to a specific XML Schema (XSD), you should apply the appropriate attribute and set its **DataType** property to the desired XML data type name.

For a complete list of type mappings, see the **DataType** property for any of the following attribute classes:

* [SoapAttributeAttribute](http://msdn.microsoft.com/en-us/library/system.xml.serialization.soapattributeattribute(VS.85).aspx)
* [SoapElementAttribute](http://msdn.microsoft.com/en-us/library/system.xml.serialization.soapelementattribute(VS.85).aspx)
* [XmlArrayItemAttribute](http://msdn.microsoft.com/en-us/library/system.xml.serialization.xmlarrayitemattribute(VS.85).aspx)
* [XmlAttributeAttribute](http://msdn.microsoft.com/en-us/library/system.xml.serialization.xmlattributeattribute(VS.85).aspx)
* [XmlElementAttribute](http://msdn.microsoft.com/en-us/library/system.xml.serialization.xmlelementattribute(VS.85).aspx)
* [XmlRootAttribute](http://msdn.microsoft.com/en-us/library/system.xml.serialization.xmlrootattribute(VS.85).aspx)