**XML serialization**

* 03/30/2017
* 10 minutes to read
  + [[](https://github.com/dotnet/docs/blob/master/docs/standard/serialization/introducing-xml-serialization.md)](https://github.com/dotnet/docs/blob/master/docs/standard/serialization/introducing-xml-serialization.md" \o "13 Contributors)

* + [[](https://github.com/dotnet/docs/blob/master/docs/standard/serialization/introducing-xml-serialization.md)](https://github.com/dotnet/docs/blob/master/docs/standard/serialization/introducing-xml-serialization.md" \o "13 Contributors)

* + [[](https://github.com/dotnet/docs/blob/master/docs/standard/serialization/introducing-xml-serialization.md)](https://github.com/dotnet/docs/blob/master/docs/standard/serialization/introducing-xml-serialization.md" \o "13 Contributors)

* + [[](https://github.com/dotnet/docs/blob/master/docs/standard/serialization/introducing-xml-serialization.md)](https://github.com/dotnet/docs/blob/master/docs/standard/serialization/introducing-xml-serialization.md" \o "13 Contributors)

* + [[](https://github.com/dotnet/docs/blob/master/docs/standard/serialization/introducing-xml-serialization.md)](https://github.com/dotnet/docs/blob/master/docs/standard/serialization/introducing-xml-serialization.md" \o "13 Contributors)

* + [+8](https://github.com/dotnet/docs/blob/master/docs/standard/serialization/introducing-xml-serialization.md" \o "13 Contributors)

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 is deserialized into an object of the same type.

**Note**

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 [**DataContractSerializer**](https://docs.microsoft.com/en-us/dotnet/api/system.runtime.serialization.datacontractserializer) instead of XML serialization.

The central class in XML serialization is the [XmlSerializer](https://docs.microsoft.com/en-us/dotnet/api/system.xml.serialization.xmlserializer) class, and the most important methods in this class are the **Serialize** and **Deserialize** methods. The [XmlSerializer](https://docs.microsoft.com/en-us/dotnet/api/system.xml.serialization.xmlserializer) 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)](https://docs.microsoft.com/en-us/dotnet/standard/serialization/xml-serializer-generator-tool-sgen-exe) 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 (W3C) [XML Schema definition language (XSD) 1.0 recommendation](https://www.w3.org/TR/xslt). 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](https://docs.microsoft.com/en-us/dotnet/standard/data/xml/). 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](https://docs.microsoft.com/en-us/dotnet/standard/serialization/controlling-xml-serialization-using-attributes). For a table of those attributes that are used to control the generated XML, see [Attributes That Control XML Serialization](https://docs.microsoft.com/en-us/dotnet/standard/serialization/attributes-that-control-xml-serialization).

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](https://docs.microsoft.com/en-us/dotnet/standard/serialization/how-to-serialize-an-object-as-a-soap-encoded-xml-stream). For a table of the attributes that control the generated XML, see [Attributes That Control Encoded SOAP Serialization](https://docs.microsoft.com/en-us/dotnet/standard/serialization/attributes-that-control-encoded-soap-serialization).

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](https://docs.microsoft.com/en-us/dotnet/standard/serialization/xml-serialization-with-xml-web-services). For more information about SOAP and XML Web services, see [Customizing SOAP Message Formatting](https://docs.microsoft.com/en-us/previous-versions/dotnet/netframework-4.0/dkwy2d72(v=vs.100)).

**Security Considerations for XmlSerializer Applications**

When creating an application that uses the **XmlSerializer**, 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.

**Note**

These serialization assemblies can be generated in advance and signed by using the SGen.exe tool. This does not work a server of Web services. In other words, it is only for client 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 basic class with a public field.

C#Copy

public class OrderForm

{

public DateTime OrderDate;

}

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

XMLCopy

<OrderForm>

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

</OrderForm>

For more examples of serialization, see [Examples of XML Serialization](https://docs.microsoft.com/en-us/dotnet/standard/serialization/examples-of-xml-serialization).

**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**.

**Note**

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](https://docs.microsoft.com/en-us/dotnet/standard/serialization/how-to-serialize-an-object) and [How to: Deserialize an Object](https://docs.microsoft.com/en-us/dotnet/standard/serialization/how-to-deserialize-an-object).

**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 must serialize non-public data, use the [DataContractSerializer](https://docs.microsoft.com/en-us/dotnet/api/system.runtime.serialization.datacontractserializer) class rather than XML serialization.
* A class must have a parameterless 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 that implement **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](https://docs.microsoft.com/en-us/dotnet/standard/serialization/examples-of-xml-serialization).

**XSD Data Type Mapping**

The W3C document titled [XML Schema Part 2: Datatypes](https://www.w3.org/TR/xmlschema-2/) 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)](https://docs.microsoft.com/en-us/dotnet/standard/serialization/xml-schema-definition-tool-xsd-exe)) 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.

C#Copy

[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](https://docs.microsoft.com/en-us/dotnet/api/system.xml.serialization.soapattributeattribute)
* [SoapElementAttribute](https://docs.microsoft.com/en-us/dotnet/api/system.xml.serialization.soapelementattribute)
* [XmlArrayItemAttribute](https://docs.microsoft.com/en-us/dotnet/api/system.xml.serialization.xmlarrayitemattribute)
* [XmlAttributeAttribute](https://docs.microsoft.com/en-us/dotnet/api/system.xml.serialization.xmlattributeattribute)
* [XmlElementAttribute](https://docs.microsoft.com/en-us/dotnet/api/system.xml.serialization.xmlelementattribute)
* [XmlRootAttribute](https://docs.microsoft.com/en-us/dotnet/api/system.xml.serialization.xmlrootattribute)

**See also**

* [XmlSerializer](https://docs.microsoft.com/en-us/dotnet/api/system.xml.serialization.xmlserializer)
* [DataContractSerializer](https://docs.microsoft.com/en-us/dotnet/api/system.runtime.serialization.datacontractserializer)
* [FileStream](https://docs.microsoft.com/en-us/dotnet/api/system.io.filestream)
* [XML and SOAP Serialization](https://docs.microsoft.com/en-us/dotnet/standard/serialization/xml-and-soap-serialization)
* [Binary Serialization](https://docs.microsoft.com/en-us/dotnet/standard/serialization/binary-serialization)
* [Serialization](https://docs.microsoft.com/en-us/dotnet/standard/serialization/)
* [XmlSerializer](https://docs.microsoft.com/en-us/dotnet/api/system.xml.serialization.xmlserializer)
* [Examples of XML Serialization](https://docs.microsoft.com/en-us/dotnet/standard/serialization/examples-of-xml-serialization)
* [How to: Serialize an Object](https://docs.microsoft.com/en-us/dotnet/standard/serialization/how-to-serialize-an-object)
* [How to: Deserialize an Object](https://docs.microsoft.com/en-us/dotnet/standard/serialization/how-to-deserialize-an-object)