**XML Serialization in the .NET Framework**

Dare Obasanjo  
Microsoft Corporation

January 23, 2003

**Summary:** Dare Obasanjo discusses how XML serialization lets you process strongly typed XML within the .NET Framework while supporting W3C standards and improving interoperability. A FAQ is also included in this article. (11 printed pages)

[Download the xml01202003\_sample.exe](http://download.microsoft.com/download/a/8/e/a8e628e5-6f7a-4adb-8f31-73fd5b3dbe2c/xml01202003_sample.exe).

**The Story So Far**

In my previous columns, [Things to Know and Avoid When Querying XML Documents with XPath](http://msdn.microsoft.com/en-us/library/ms950786.aspx) and [Working with Namespaces in XML Schema](http://msdn.microsoft.com/en-us/library/ms950796.aspx), I mentioned creating an XML format for tracking the books in my personal library. While working with this format, I have explored various aspects of W3C recommendations, such as [XPath](http://www.w3.org/tr/xpath) and [XML Schema](http://www.w3.org/tr/xmlschema-1/). In this month's article, I'll explore using the XML serialization technology in the .NET Framework with my XML format, and answer some of the commonly asked questions about .NET Framework-based XML serialization. Come along, it should be an interesting ride.

**Overview of XML Serialization in the .NET Framework**

The primary purpose of XML serialization in the .NET Framework is to enable the conversion of XML documents and streams to common language runtime objects and vice versa. Serialization of XML to common language runtime objects enables one to convert XML documents into a form where they are easier to process using conventional programming languages. On the other hand, serialization of objects to XML facilitates persisting or transporting the *state* of such objects in an open, standards compliant and platform agnostic manner.

XML serialization in the .NET Framework supports serializing objects as either XML that conforms to a specified [W3C XML Schema Definition (XSD) schema](http://www.w3.org/tr/xmlschema-1/) or that is conformant to the serialization format defined in section five of the [SOAP specification](http://www.w3.org/tr/soap/). During XML serialization, only the public properties and fields of an object are serialized. Also, type fidelity is not always preserved during XML serialization. This means that if, for instance, 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. However, this means that objects serialized using the XML serialization in the .NET Framework can be shipped from one machine to the other without requiring that the original type be present on the target machine or that the XML is even processed using the .NET Framework. XML serialization of objects is a useful mechanism for those who want to provide or consume data using platform agnostic technologies such as XML and SOAP.

XML documents converted to objects by the XML serialization process are strongly typed. Data type information is associated with the elements and attributes in an XML document through a schema written in the W3C XML Schema Definition (XSD) Language. The data type information in the schema allows the [XmlSerializer](http://msdn.microsoft.com/en-us/library/system.xml.serialization.xmlserializer.aspx) to convert XML documents to strongly typed classes.

For more information about XML serialization in the .NET Framework, read the SDK documentation topic entitled XML and SOAP Serialization.

**The Book Inventory Application**

In my previous articles, I created an XML document that listed all my books and described their availability in my personal library. Upon reflection, I decided that I'd prefer a GUI interface for viewing and manipulating the document instead of editing the raw XML file in a text editor. The first step I took in creating this application was to look at the classes in the [System.Windows.Forms namespace](http://msdn.microsoft.com/en-us/library/system.windows.forms.aspx) to see if any could satisfy my needs out of the box. The [DataGrid class](http://msdn.microsoft.com/en-us/library/system.windows.forms.datagrid.aspx) looked promising.

The description of potential data sources for the **DataGrid** class included single dimensional arrays, which struck a chord because I imagined that a sequential listing of books is something that could be mapped to an array by XML serialization. I decided to give this a try by converting the schema shown below to a C# class.

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

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

<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"

targetNamespace="urn:xmlns:25hoursaday-com:my-bookshelf"

xmlns:bk="urn:xmlns:25hoursaday-com:my-bookshelf"

elementFormDefault="qualified">

<xs:element name="books">

<xs:complexType>

<xs:sequence>

<xs:element name="book" type="bk:bookType"

maxOccurs="unbounded" />

</xs:sequence>

</xs:complexType>

</xs:element>

<xs:complexType name="bookType">

<xs:sequence>

<xs:element name="title" type="xs:string" />

<xs:element name="author" type="xs:string" />

<xs:element name="publication-date" type="xs:date" />

</xs:sequence>

<xs:attribute name="publisher" type="xs:string" />

<xs:attribute name="on-loan" type="xs:string" />

</xs:complexType>

</xs:schema>

The .NET Framework SDK provides the [XML Schema Definition Tool xsd.exe](http://msdn.microsoft.com/en-us/library/x6c1kb0s.aspx) that can be used to convert XSD schemas to C# classes. I converted my schema file to C# source file issuing the following commands on the command line:

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

xsd.exe /c books.xsd

The generated C# class is decorated with attributes that provide information on how the XmlSerializer converts the class to XML. The [XmlRootAttribute](http://msdn.microsoft.com/en-us/library/system.xml.serialization.xmlrootattribute.aspx), [XmlElementAttribute](http://msdn.microsoft.com/en-us/library/aa335740.aspx), and [XmlAttributeAttribute](http://msdn.microsoft.com/en-us/library/system.xml.serialization.xmlattributeattribute.aspx) attributes are used to specify which classes, fields, or properties become the root, element, and attribute nodes in the generated XML. The class generated from the schema is shown below.

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

using System.Xml.Serialization;

/// <remarks/>

[System.Xml.Serialization.XmlTypeAttribute(Namespace=

"urn:xmlns:25hoursaday-com:my-bookshelf")]

[System.Xml.Serialization.XmlRootAttribute("books",

Namespace="urn:xmlns:25hoursaday-com:my-bookshelf", IsNullable=false)]

public class books {

/// <remarks/>

[System.Xml.Serialization.XmlElementAttribute("book")]

public bookType[] book;

}

/// <remarks/>

[System.Xml.Serialization.XmlTypeAttribute(Namespace=

"urn:xmlns:25hoursaday-com:my-bookshelf")]

public class bookType {

/// <remarks/>

public string title;

/// <remarks/>

public string author;

/// <remarks/>

[System.Xml.Serialization.XmlElementAttribute("publication-date",

DataType="date")]

public System.DateTime publicationdate;

/// <remarks/>

[System.Xml.Serialization.XmlAttributeAttribute()]

public string publisher;

/// <remarks/>

[System.Xml.Serialization.XmlAttributeAttribute("on-loan")]

public string onloan;

}

**Note**The attribute that annotates the **publicationdate** field has a **DataType** property. There is no type in the .NET Framework that matches the type xs:date completely. The closest match is System.DateTime, which stores date and time data. Specifying the **DataType** property as a "date" ensures that the XmlSerializer will only serialize the date part of the **DateTime** object.

The XmlSerializer mapped the multiple book elements of type bookType in my schema to an array of bookType objects named book. I thought I was ready to bind my classes to the DataGrid until I noticed a section of the documentation on Data Sources for the DataGrid Control which pointed out that the objects in an array must have **public properties** if they are to be bound to a DataGrid. Since the XSD tool created public fields, I had to fix my classes by making the fields private and exposing them through properties instead. Then, I also had to move the XML serialization specific attributes from the fields to the properties because the XmlSerializer only checks for attributes on public fields and properties. Below is the altered bookType class.

[Copy Code](javascript:CopyCode('ctl00_MTCS_main_ctl20_code');" \o "Copy Code)

/// <remarks/>

[System.Xml.Serialization.XmlTypeAttribute(Namespace=

"urn:xmlns:25hoursaday-com:my-bookshelf")]

public class bookType

{

/// <remarks/>

private string \_title;

public string title{

get{ return \_title; }

set { \_title = value; }

}

/// <remarks/>

private string \_author;

public string author

{

get{ return \_author; }

set { \_author = value; }

}

/// <remarks/>

private System.DateTime \_publicationdate;

[System.Xml.Serialization.XmlElementAttribute("publication-date",

DataType="date")]

public System.DateTime publicationdate

{

get{ return \_publicationdate; }

set { \_publicationdate = value; }

}

private string \_publisher;

/// <remarks/>

[System.Xml.Serialization.XmlAttributeAttribute()]

public string publisher

{

get{ return \_publisher; }

set { \_publisher = value; }

}

private string \_onloan;

[System.Xml.Serialization.XmlAttributeAttribute("on-loan")]

public string onloan

{

get{ return \_onloan; }

set { \_onloan = value; }

}

}

With the aforementioned changes, it is now possible to bind my list of books as XML to a **DataGrid**. Firing up the Visual Studio® .NET forms designer, I quickly dragged and dropped a **DataGrid** onto a form, along with a couple of buttons for navigation purposes. The final step was to add some code to ensure that the **DataGrid** was bound to my XML once the form was loaded. The Form\_Load method in my class is shown below.

[Copy Code](javascript:CopyCode('ctl00_MTCS_main_ctl21_code');" \o "Copy Code)

private void Form1\_Load(object sender, System.EventArgs e)

{

try{

TextReader reader = new StreamReader("books.xml");

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

myBooks = (books)serializer.Deserialize(reader);

reader.Close();

//currency manager used for cursoring through book array in UI

currencyManager =

(CurrencyManager)dataGrid1.BindingContext[myBooks.book];

dataGrid1.DataSource= myBooks.book;

}catch(XmlException xe){

MessageBox.Show (xe.Message, "XML Parse Error",

MessageBoxButtons.OK, MessageBoxIcon.Error);

}catch(InvalidOperationException ioe){

MessageBox.Show (ioe.InnerException.Message, "XML

Serialization Error",

MessageBoxButtons.OK, MessageBoxIcon.Error);

}

}

And that's all it takes. Shown below is the XML document I plan to edit with my application.

[Copy Code](javascript:CopyCode('ctl00_MTCS_main_ctl22_code');" \o "Copy Code)

<books xmlns="urn:xmlns:25hoursaday-com:my-bookshelf" >

<book publisher="QUE">

<title>XML By Example</title>

<author>Benoit Marchal</author>

<publication-date>1999-12-31</publication-date>

</book>

<book publisher="Addison Wesley" on-loan="Dmitri">

<title>Essential C++</title>

<author>Stanley Lippman</author>

<publication-date>2000-10-31</publication-date>

</book>

<book publisher="WROX">

<title>XSLT Programmer's Reference</title>

<author>Michael Kay</author>

<publication-date>2001-04-30</publication-date>

</book>

<book publisher="Addison Wesley" on-loan="Sanjay">

<title>Mythical Man Month</title>

<author>Frederick Brooks</author>

<publication-date>1995-06-30</publication-date>

</book>

<book publisher="Apress">

<title>Programmer's Introduction to C#</title>

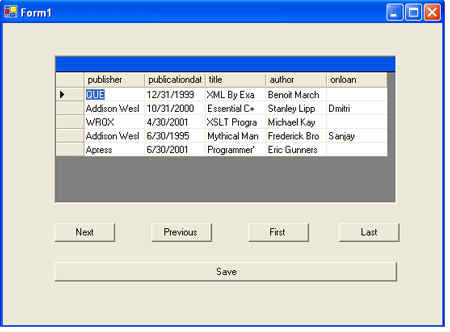
<author>Eric Gunnerson</author>

<publication-date>2001-06-30</publication-date>

</book>

</books>

Figure 1 below shows a DataGrid bound to the above XML document in my Book Inventory application .



**Figure 1. The Book Inventory application**

It should be noted that although my application allows me to edit the contents of my XML document bound to the DataGrid, I cannot add or delete new books to my inventory file. The inability to add modify the number of books through the DataGrid is a limitation imposed on it when bound to arrays. This issue does not exist when bound to other data sources.

For the purpose of completeness, I should highlight an alternative approach to solving my data binding problem. I could have loaded the XML document and schema into a [DataSet](http://msdn.microsoft.com/en-us/library/system.data.dataset.aspx) using the [ReadXml()](http://msdn.microsoft.com/en-us/library/system.data.dataset.readxml.aspx) and [ReadXmlSchema()](http://msdn.microsoft.com/en-us/library/system.data.dataset.readxmlschema.aspx) methods respectively, then bound that to the DataGrid. Descriptions of how to use this alternate approach are available in the .NET SDK documentation in the section entitled Generating DataSet Relational Structure from XML Schema (XSD). A code example using the **DataSet** class instead of XML serialization is also included in the download file for the article.

**XML Serialization Frequently Asked Questions**

The following is a list of questions commonly asked on the [microsoft.public.dotnet.xml newsgroup](http://msdn.microsoft.com/newsgroups/loadframes.asp?icp=msdn&slcid=us&newsgroup=microsoft.public.dotnet.xml) about XML serialization in the .NET Framework.

**Q: Why can't I serialize .NET Framework classes like exceptions and fonts?**

**A:** The XmlSerializer is primarily designed with two goals in mind: XML data binding to XSD compliant data structures and operation without any special code access privileges. These two goals work against the XmlSerializer as a general-purpose object persistence solution for some kinds of objects.

General purpose serialization may require accessing private fields, by-passing the framework's standard object construction process, and so on, which in turn requires special privileges. The **SoapFormatter** from the **System.Runtime.Serialization.Formatters.Soap** namespace provides an alternative that is not subject to these restrictions, but requires full trust to operate. It also produces an XML format, a generation of which is customizable using the attributes in the **System.Runtime.Remoting.Metadata** namespace.

The **BinaryFormatter** from the **System.Runtime.Serialization.Formatters.Binary** namespace can also be used as a mechanism to provide simple object persistence and transport for situations where XML serialization does not meet your needs.

**Q: How can I serialize classes that were not designed for XML serialization if I do not want to use the SoapFormatter?**

**A:** You can design special wrapper classes that expose or hide fields and properties from the XmlSerializer.

**Q: How do I serialize collections of objects?**

**A:** The XmlSerializer throws an exception when the collection contains types that were not declared to the constructor of the XmlSerializer. You can:

1. Declare the types to the serializer by passing in a **Type[]** with the types to expect within the collection.

**OR**

1. Implement a strongly-typed collection derived from **System.Collections.CollectionBase** with an indexer matching the **Add()** method.

**Q: Why aren't all properties of collection classes serialized?**

**A:** The XmlSerializer only serializes the elements in the collection when it detects either the **IEnumerable** or the **ICollection** interface. This behavior is by design. The only work around is to re-factor the custom collection into two classes, one of which exposes the properties including one of the pure collection types.

**Q: Why can't I serialize hashtables?**

**A:** The XmlSerializer cannot process classes implementing the **IDictionary** interface. This was partly due to schedule constraints and partly due to the fact that a hashtable does not have a counterpart in the XSD type system. The only solution is to implement a custom hashtable that does not implement the **IDictionary** interface.

**Q: Why do exceptions thrown by the XmlSerializer not contain any details about the error?**

**A:** They do contain all the information, but it's stored in the **InnerException** property of the exception thrown, which is usually an InvalidOperationException. In general, one should always call **ToString()** on caught exceptions to get the full details of the exception.

**Q: What aspects of W3C XML Schema are not supported by the XmlSerializer during conversion of schemas to classes?**

**A:** The XmlSerializer does not support the following:

* Any of the simple type restriction facets besides enumeration.
* Namespace based wildcards.
* Identity constraints.
* Substitution groups.
* Blocked elements or types.

**Q: Why doesn't XSD.exe support the schemaLocation attribute on imports and includes?**

**A:** The W3C XML Schema recommendation describes this attribute as a hint, which can be ignored by processors that can use alternate means to locate schemas. XSD.exe only uses schemas that are specified through the command line to convert schema A.xsd, which imports schema B.xsd.

[Copy Code](javascript:CopyCode('ctl00_MTCS_main_ctl28_code');" \o "Copy Code)

xsd.exe /c A.xsd B.xsd

Also, the wsdl.exe application, considered a sister application to xsd.exe, can be downloaded from the Web. If you do this and use the wsdl.exe, you would follow schemaLocation hints in imports and includes.

**Q: What are the differences between the XSD to runtime type mapping used by XML serialization and those used by ADO.NET or XSD validation?**

**A:** The mappings from W3C XML Schema types to runtime types used by the dataset and schema validation are described in the Data Type Support between XML Schema (XSD) Types and .NET Framework Types topic.

This mapping is different from those used when XML serialization attributes are specified on the fields and properties of a class. Each of the XML serialization attributes has its mapping from objects to W3C XML Schema types defined in the description for the **DataType** property for that class. This includes the descriptions of the [SoapAttributeAttribute.DataType property](http://msdn.microsoft.com/en-us/library/system.xml.serialization.soapattributeattribute.datatype.aspx) , [SoapElementAttribute.DataType property](http://msdn.microsoft.com/en-us/library/system.xml.serialization.soapelementattribute.datatype.aspx), [XmlArrayItemAttribute.DataType property](http://msdn.microsoft.com/en-us/library/system.xml.serialization.xmlarrayitemattribute.datatype.aspx), [XmlAttributeAttribute.DataType property](http://msdn.microsoft.com/en-us/library/system.xml.serialization.xmlattributeattribute.datatype.aspx), [XmlElementAttribute.DataType property](http://msdn.microsoft.com/en-us/library/system.xml.serialization.xmlelementattribute.datatype.aspx), and [XmlRootAttribute.DataType property](http://msdn.microsoft.com/en-us/library/system.xml.serialization.xmlrootattribute.datatype.aspx).

A major difference is that the Gregorian dates, such as xs:gMonth and xs:gYear, are mapped to strings by XML serialization, whereas validation maps them to **DateTime** objects. Another difference is that the DTD-based list types, such as IDREFS and NMTOKENS, are mapped to strings by XML serialization while validation maps them to arrays of strings. The xs:duration type is also mapped differently in XML serialization compared to schema validation. XML serialization maps them to strings, while validation maps them to **TimeSpan** objects. The xs:integer type is specified as a number with no upper or lower bound on its size. For this reason, neither XML serialization nor validation map it to the **System.Int32** type. Instead, XML serialization maps the xs:integer to a string while validation maps it to the **Decimal** type that is much larger than any of the integer types in the .NET Framework.

**Conclusion**

The features delivered by XML serialization in the .NET Framework provide the ability to process strongly typed XML seamlessly within the .NET Framework. Strongly typed XML documents can leverage the benefits of the .NET Framework, such as data binding to Windows and ASP.NET Forms, a rich collections framework, and a choice of programming languages. All of this is done while taking care to support W3C standards like XML 1.0 and XML Schema, meaning that there is no cost to the interoperability of XML when attempting to transfer or persist such strongly typed documents.

**Acknowledgements**

I'd like to thank Doug Purdy and Stefan Pharies from the XML serialization team, and [Christoph Schittko](http://groups.google.com/groups?&q=%22christoph%20schittko%22) for their help in reviewing and providing content for this article.

**Dare Obasanjo** is a member of Microsoft's WebData team, which among other things develops the components within the System.Xml and System.Data namespace of the .NET Framework, Microsoft XML Core Services (MSXML), and Microsoft Data Access Components (MDAC).

Feel free to post any questions or comments about this article on the [Extreme XML message board](http://www.gotdotnet.com/community/messageboard/messageboard.aspx?id=207) on GotDotNet.