Using the XmlSerializer Attributes

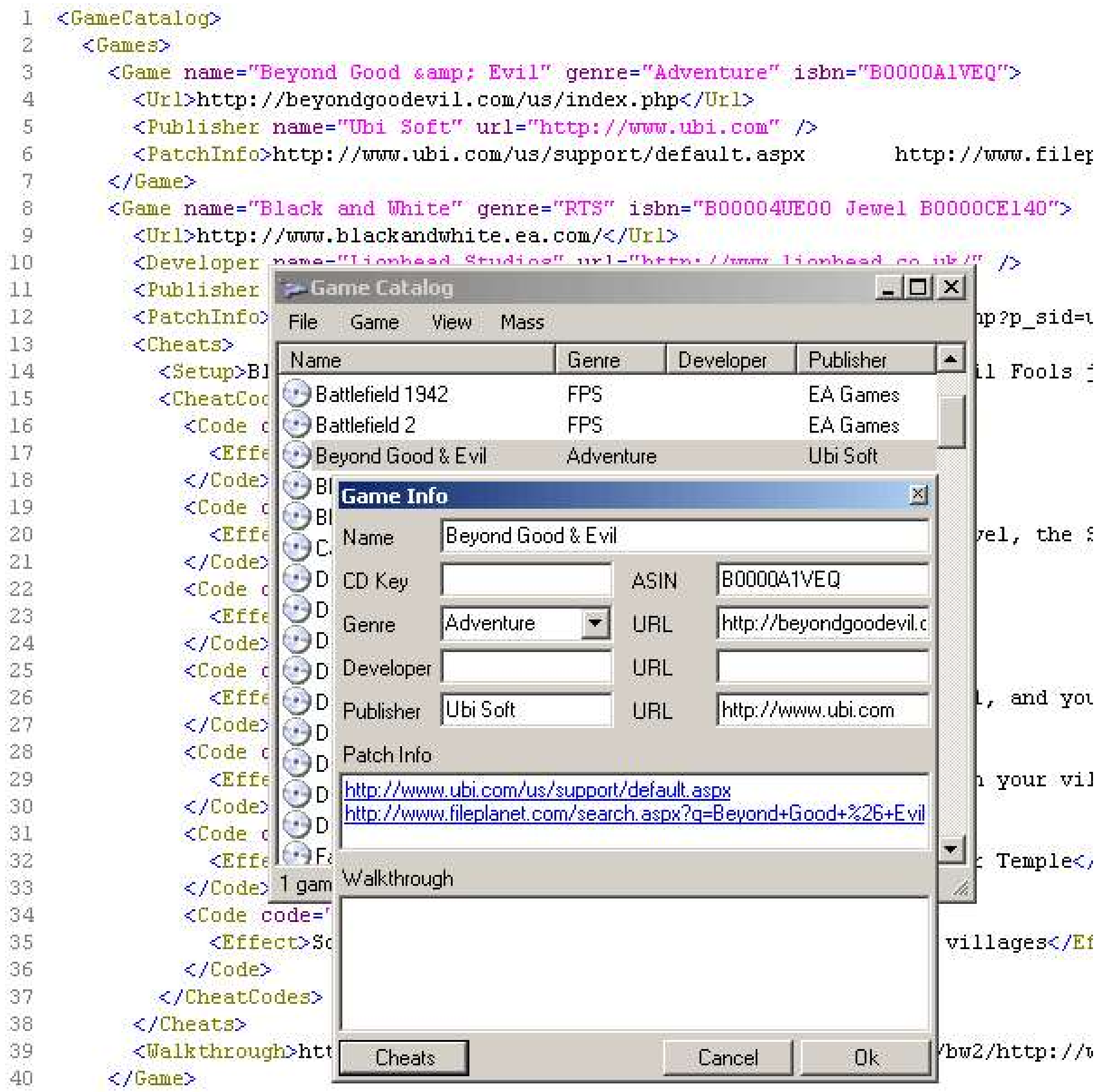
BoneSoft, 29 Jun 2006

★★★★★ 4.58 (50 votes) Rate:

How to serialize and de-serialize .NET objects and XML using the XmlSerializer and the serializer attributes.

Download source files - 155 Kb

Download demo project - 166 Kb



```
<GameCatalog>
  <Games>
    <Game name="Beyond Good &amp; Evil" genre="Adventure" isbn="B0000A1VEQ">
      <Url>http://beyondgoodevil.com/us/index.php</Url>
      <Publisher name="Ubi Soft" url="http://www.ubi.com" />
      <PatchInfo>http://www.ubi.com/us/support/default.aspx       http://www.filep
    </Game>
    <Game name="Black and White" genre="RTS" isbn="B00004UE00 Jewel B0000CE140">
      <Url>http://www.blackandwhite.ea.com/</Url>
      <Developer                                                       />
      <Publisher
      <PatchInfo>                                                          hp?p_sid=u
      <Cheats>
        <Setup>Bl                                                          il Fools :
        <CheatCod
          <Code c
            <Effe
          </Code>
          <Code c
            <Effe                                                          vel, the S
          </Code>
          <Code c
            <Effe
          </Code>
          <Code c
            <Effe                                                          l, and you
          </Code>
          <Code c
            <Effe                                                          n your vil
          </Code>
          <Code c
            <Effe                                                          : Temple</
          </Code>
          <Code code="
            <Effect>Sc                                                    villages</Ef
          </Code>
        </CheatCodes>
      </Cheats>
      <Walkthrough>htt                                                    /bw2/http://w
    </Game>
```

Game Catalog

File Game View Mass

Name	Genre	Developer	Publisher
Battlefield 1942	FPS		EA Games
Battlefield 2	FPS		EA Games
Beyond Good & Evil	Adventure		Ubi Soft

1 gam

Game Info

Name	Beyond Good & Evil		
CD Key		ASIN	B0000A1VEQ
Genre	Adventure	URL	http://beyondgoodevil.c
Developer		URL	
Publisher	Ubi Soft	URL	http://www.ubi.com

Patch Info

http://www.ubi.com/us/support/default.aspx
http://www.fileplanet.com/search.aspx?q=Beyond+Good+%26+Evil

Walkthrough

Introduction

The main purpose of this article is to discuss some of the uses for XML Serialization, and explore some of the basics of using the `XmlSerializer` and the attributes in the `System.Xml.Serialization` namespace. To keep this discussion basic, I will not talk about XML namespaces, or the attributes' more specific constructors. This article only aims to give a basic introduction to the serializer and attributes, and some links to further reading and helpful tools. However, one of the great things about this approach to working with XML, is that it takes very little work and you can benefit greatly from the basics alone. While a full exploration of the API is recommended, you can accomplish much with only a basic understanding.

Depending on your needs, there are several options available when working with XML. If you only need to read certain portions of your XML, using XPath queries with `XmlNode.SelectNodes()` and `XmlNode.SelectSingleNode()` is a reasonable option. But if you need to make your XML and code interchangeable, this is not a viable approach. To move from one to the other, both the code model and the XML structure must have the ability to represent the same information no matter how it's formatted. For this, you have three options... If you do not have a specific XML structure that you need to adhere to, you can use the `DataSet`'s `ReadXml()` and `WriteXml()` methods. If you have a specific XML structure or an XML Schema that you need to represent in code, you can use the *XSD.exe* tool, though it has some unpleasant side effects. However, if you have a class hierarchy that you need to represent in XML, your only two options are to use the the serialization attributes and/or the `XmlAttributeOverrides` class with the `XmlSerializer`. But keep in mind, that this last option can handle all of the previously mentioned scenarios. Actually, the *XSD.exe* tool uses this approach as well, it just generates the code from XML or XSD. The main problems with the XSD tool are the lack of customizability and the ugly code it produces.

About the Example Project

This article is a basic explaination of XML Serialization. The example is a small application, for cataloging game information, that uses the XML Serialization techniques described in this article to store its information. XML Serialization is a great alternative to databases, for many projects such as this that are small local applications. This example also shows some other interesting things such as using the `AxWebBrowser` control, XSLT rendering, and pre-VS2005 menu painting. Plus, it's a nice thing to have if you own a lot of games. :)

Using the XmlSerializer & Serialization Attributes

In discussing the `XmlSerializer` and the serializer attributes, we're talking about either modifying existing code to specify how its data will be serialized, or creating new code to produce a specific XML structure. For the most part, this only entails attaching attributes to the public fields or properties of classes, and in some situations to the class itself. But before we dive into the attributes, let's talk a little about the `XmlSerializer`...

The XmlSerializer

Given XML, the `XmlSerializer` can produce a graph of objects that hold the same instance data. This is known as XML de-serialization. For this to work, the classes of the objects must be structured to fit the XML supplied, or the attributes in the `System.Xml.Serialization` namespace must be used to specify how the object's properties map to XML entities.

The `XmlSerializer` can also produce XML when supplied with an instance of an XML Serializable object. This is known as XML Serialization. The criteria that determine if an object or a graph of objects is "XML Serializable" are fairly simple...

- In theory, it would be possible to serialize an interface, but impossible to determine what concrete type to de-serialize to. Because of this, the `XmlSerializer` cannot work directly with an interface reference.
- The `XmlSerializer` can only work with the public fields on your types, and or the public properties that supply both `get` and `set` accessors.
- The types of the properties of the type to be serialized must follow the previous rules.
- XML Serialization exceptions are notoriously hard to read at first glance, because they are almost always nested. Once you have a model built that you intend to use with XML serialization, it pays to use a pre-compiler to verify the serializability of your classes. One such tool is listed at the end of this article in the **Links and Tools** section.

The Attributes

With the use of the attributes in the `System.Xml.Serialization` namespace, you can, in effect, make your classes completely interchangeable with XML. The attributes basically allow you to specify if members of your class should be an attribute, or an element, or an array, and what those XML entities should be named in the XML.

XmlRootAttribute

The `XmlRootAttribute` attribute tells the serializer that the class that this attribute is attached to is the document root node. It also allows you to specify what the root node is named despite the class name. For example, the `RootClass` class is serialized to an `XmlDocRoot` root node:

Hide Copy Code

```
using System;
using System.Xml.Serialization;

namespace XmlEntities {
    [XmlRoot("XmlDocRoot")]
    public class RootClass {
    }
}
```

This will serialize to something similar to this...

Hide Copy Code

```
<XmlDocRoot />
```

XmlElementAttribute

The `XmlElementAttribute` attributes allows you to specify that a member should be serialized as an element and what the element should be named. Simple data (`int`, `string`, etc...) and complex data (objects with fields or properties) can be serialized to elements.

Hide Copy Code

```
using System;
using System.Xml.Serialization;

namespace XmlEntities {
    [XmlRoot("XmlDocRoot")]
    public class RootClass {
        private string element_description;

        [XmlElement("Description")]
        public string Description {
            get { return element_description; }
            set { element_description = value; }
        }
    }
}
```

This will serialize to something similar to this...

Hide Copy Code

```
<XmlDocRoot>
    <Description>text</Description>
</XmlDocRoot>
```

XmlAttributeAttribute

The `XmlAttributeAttribute` attribute allows you to specify that a member should be serialized as an attribute and what that attribute should be named. Only simple data can be used as an attribute because an attribute can only represent a single value.

Hide Copy Code

```
using System;
using System.Xml.Serialization;

namespace XmlEntities {
    [XmlRoot("XmlDocRoot")]
    public class RootClass {
        private int attribute_id;

        [XmlAttribute("id")]
        public int Id {
            get { return attribute_id; }
            set { attribute_id = value; }
        }
    }
}
```

This will serialize to something similar to this...

Hide Copy Code

```
<XmlDocRoot id="1" />
```

XmlTextAttribute

Using the `XmlTextAttribute` attribute specifies that the property it's attached to will be the text content of the parent node. This attribute can only be attached to one property of the class, for obvious reasons.

Hide Shrink Copy Code

```
using System;
using System.Xml.Serialization;

namespace XmlEntities {
    [XmlRoot("XmlDocRoot")]
    public class RootClass {
        private Description element_description;

        [XmlElement("Description")]
        public Description Description {
            get { return element_description; }
            set { element_description = value; }
        }
    }

    public class Description {
        private int attribute_id;
        private string element_text;

        [XmlAttribute("id")]
        public int Id {
            get { return attribute_id; }
            set { attribute_id = value; }
        }

        [XmlText()]
        public string Text {
            get { return element_text; }
            set { element_text = value; }
        }
    }
}
```

This will serialize to something similar to this...

Hide Copy Code

```
<XmlDocRoot>
    <Description id="1">text</Description>
</XmlDocRoot>
```

XmlIgnoreAttribute

Any public member that you do not wish to serialize, or that cannot be serialized, can be excluded from serialization by using the `XmlIgnoreAttribute` on the member. By using this, you insure that the member will not be included in serialization or de-serialization.

Hide Copy Code

```
using System;
using System.Xml.Serialization;

namespace XmlEntities {
    [XmlRoot("XmlDocRoot")]
    public class RootClass {
        private System.IO.Stream stream;

        [XmlIgnore()]
        public System.IO.Stream Stream {
            get { return stream; }
            set { stream = value; }
        }
    }
}
```

XmlArrayAttribute & XmlArrayItemAttribute

When serializing an array or collection, without the use of attributes, the default behaviour of the serializer is to append an element per member of the collection. The `XmlArrayAttribute` and `XmlArrayItemAttribute` attributes allow you to alter this behaviour. The `XmlArrayItemAttribute` attribute tells the serializer what the name and type of the elements of the collection should be. The `XmlArrayAttribute` specifies a parent element for the collection elements.

Hide Copy Code

```
using System;
using System.Xml.Serialization;

namespace XmlEntities {
    [XmlRoot("XmlDocRoot")]
    public class RootClass {
        private string[] element_list;

        [XmlArrayItem("ListItem", typeof(string))]
        [XmlArray("List")]
        public string[] List {
            get { return element_list; }
            set { element_list = value; }
        }
    }
}
```

This will serialize to something similar to this...

Hide Copy Code

```
<XmlDocRoot>
    <List>
        <ListItem>string</ListItem>
        <ListItem>string</ListItem>
    </List>
</XmlDocRoot>
```

XmlIncludeAttribute

The `XmlIncludeAttribute` attribute tells the serializer what types can be expected to extend a base type. When serializing to XML, members of the type that this attribute is attached to will be serialized based on the concrete type of the instance the property references. When de-serializing from XML, the attributes give the serializer a finite list of types that the instance data could fit. In the following example, the `XmlIncludeAttribute` lets the serializer know that wherever an `AbstractLogger` is referenced, that it could be an instance of `FileLogger`, `XmlLogger`, or `DataLogger`. Say, we have a class that has a property of type `AbstractLogger`... During de-serialization, the serializer will attempt to determine the type to de-serialize to, based on the structure of the XML, so that it can instantiate a concrete instance for the property. While serializing, if this property had an instance of `FileLogger`, the serializer will use the `FileLogger` class to determine how to serialize that property. The attributes just allow you to specify a finite list of the available types.

Hide Copy Code

```
using System;
using System.Xml.Serialization;

namespace XmlEntities {
    [XmlInclude(typeof(FileLogger))]
    [XmlInclude(typeof(XmlLogger))]
    [XmlInclude(typeof(DataLogger))]
    public abstract class AbstractLogger : ILogger {
        public abstract void Log(ILog log);
    }
}
```

Using the XmlSerializer

Now, we can talk about using the `XmlSerializer` to convert from XML to objects and vice versa... The following example class, `RootClassSerializer`, is an example of how to use the `XmlSerializer` class to read XML to objects and write objects to XML. A generic implementation is easy enough to produce, but if you need to support XML namespaces, the `XmlSerializer` will need them explicitly declared. In any event, the following is just an example of how to use the `XmlSerializer`.

Hide Shrink ▲ Copy Code

```
using System;
using System.IO;
using System.Xml;
using System.Xml.Serialization;
```

```
namespace XmlEntities {
    public class RootClassSerializer {
        private XmlSerializer s = null;
        private Type type = null;

        public RootClassSerializer() {
            this.type = typeof(RootClass);
            this.s = new XmlSerializer(this.type);
        }

        public RootClass Deserialize(string xml) {
            TextReader reader = new StringReader(xml);
            return Deserialize(reader);
        }

        public RootClass Deserialize(XmlDocument doc) {
            TextReader reader = new StringReader(doc.OuterXml);
            return Deserialize(reader);
        }

        public RootClass Deserialize(TextReader reader) {
            RootClass o = (RootClass)s.Deserialize(reader);
            reader.Close();
            return o;
        }

        public XmlDocument Serialize(RootClass rootclass) {
            string xml = StringSerialize(rootclass);
            XmlDocument doc = new XmlDocument();
            doc.PreserveWhitespace = true;
            doc.LoadXml(xml);
            return doc;
        }

        private string StringSerialize(RootClass rootclass) {
            TextWriter w = WriterSerialize(rootclass);
            string xml = w.ToString();
            w.Close();
            return xml.Trim();
        }

        private TextWriter WriterSerialize(RootClass rootclass) {
            TextWriter w = new StringWriter();
            this.s = new XmlSerializer(this.type);
            s.Serialize(w, rootclass);
            w.Flush();
            return w;
        }

        public static RootClass ReadFile(string file) {
            RootClassSerializer serializer = new RootClassSerializer();
            try {
                string xml = string.Empty;
                using (StreamReader reader = new StreamReader(file)) {
                    xml = reader.ReadToEnd();
                    reader.Close();
                }
                return serializer.Deserialize(xml);
            } catch {}
            return new RootClass();
        }

        public static bool WriteFile(string file, RootClass config) {
            bool ok = false;
            RootClassSerializer serializer = new RootClassSerializer();
            try {
                string xml = serializer.Serialize(config).OuterXml;
                using (StreamWriter writer = new StreamWriter(file, false)) {
                    writer.Write(xml.Trim());
                    writer.Flush();
                    writer.Close();
                }
                ok = true;
            } catch {}
            return ok;
        }
    }
}
```

Conclusion

The thing to take away from this discussion, is that using this approach to working with XML is extremely quick, easy, and effective. It takes very little work to setup existing models for serialization, there are great tools for building models that are serializable, and you don't have to write copious amounts of code to see if a node exists, and if it has data, and if it has the right kind of data, and if you can cast that data to a usable type, and blah... I'm getting a headache just thinking about it. And if you're not working with XML, maybe you should start, it has a lot to offer.

Links and Tools

- Top XML has an excellent article on the `XmlSerializer`.
- Skeleton Crew is a tool to help build XML serializable code from XML samples.
- Sells Brothers XML Pre-Compiler is a good tool for testing the serializability of a class or model that can tell you why a class isn't serializable.

License

Share

About the Author

BoneSoft

Software Developer (Senior) BoneSoft Software

United States

Follow this Member

I've been in software development for more than a decade now. Originally with ASP 2.0 and VB6. I worked in Japan for a year doing Java. And have been with C# ever since.

In 2005 I founded BoneSoft Software where I sell a small number of developer tools.

Group type: Organisation (No members)

Apply to join this group

You may also be interested in...

Public, Private, and Hybrid Cloud: What's the difference?

LINQ to CSV, a Type-Safe, dynamic High-Performance Approach

XmlSerializer and 'not expected' Inherited Types

NLP-Samurai@0.0.10 (Node.js-Demo)

A Deep XmlSerializer, Supporting Complex Classes, Enumerations, Structs, Collections, and Arrays

MVC Attributes

Comments and Discussions

Add a Comment or Question ? Email Alerts Search Comments

First Prev Next

Ever will be useful
Rafa Hernández 1-Dec-12 14:46

infinite cycle in PrintForm.cs and PrintForm2.cs
petr12345 10-May-12 7:53

What am I missing?
tpwright4423 1-Nov-10 13:48

Re: What am I missing?
BoneSoft 4-Nov-10 1:40

A nice and Straigtforward article
Damon88 10-Dec-09 23:51

About the use of XmlAttributeOverrides
Mehdi_S 5-Nov-08 19:18

Re: About the use of XmlAttributeOverrides
BoneSoft 5-Nov-08 23:25

Re: About the use of XmlAttributeOverrides
Mehdi_S 6-Nov-08 16:39

Re: About the use of XmlAttributeOverrides
Mehdi_S 6-Nov-08 18:53

Re: About the use of XmlAttributeOverrides
Mehdi_S 2-Feb-09 19:27

Re: About the use of XmlAttributeOverrides
BoneSoft 3-Feb-09 12:29

Re: About the use of XmlAttributeOverrides
Mehdi_S 8-Apr-09 20:12

Help with Serializing an Interface reference
TyrionTheImp 24-Apr-08 19:49

Re: Help with Serializing an Interface reference
The Limey 2-May-08 14:10

Re: Help with Serializing an Interface reference
BoneSoft 5-Nov-08 23:30

Great Article
VenDiddy 17-Apr-08 10:34

Re: Great Article
BoneSoft 17-Apr-08 12:48

Re: Great Article
The Limey 2-May-08 14:34

Is this available in C# 2.0?
SteveMets 22-Jan-08 4:32

Re: Is this available in C# 2.0?
BoneSoft 22-Jan-08 13:17

Re: Is this available in C# 2.0?
SteveMets 23-Jan-08 7:12

Re: Is this available in C# 2.0?
BoneSoft 23-Jan-08 9:01

Re: Is this available in C# 2.0?
SteveMets 23-Jan-08 22:19

Change order of xml-elements
crefcoti 24-May-06 23:13

Re: Change order of xml-elements
BoneSoft 7-Jun-06 4:06

Refresh 1 2 Next »

General News Suggestion Question Bug Answer Joke Praise Rant Admin

Use Ctrl+Left/Right to switch messages, Ctrl+Up/Down to switch threads, Ctrl+Shift+Left/Right to switch pages.

Permalink | Advertise | Privacy | Cookies | Terms of Use | Mobile
Web06 | 2.8.190214.1 | Last Updated 28 Jun 2006

Layout: fixed | fluid