Collection Serialization in ASP.NET Web Services

Mark A. Richman

January 2004

Level of Difficulty: 1 2 3

Download the code for this article: Collections.zip (114KB)

SUMMARY

The most common Web Services implementations use simple parameters, such as strings and integers. However, in the wild, many systems will need to pass complex types such as DataSets and Collections. Although data structures such as these can solve many problems, they can also make interoperability more difficult. Some data structures on one Web Services platform may be different on another. XML data types defined in the XML Schema specification do not always match up perfectly with their corresponding types in the CLR. Here we will focus on a solution for serializing collections in ASP.NET Web Services.

Collections Overiew

Indexed collections are IList-implementing, distinguished by the fact that their contents can be retrieved via a zero-based numeric index, like an array. The System.Collections.ArrayList object is one example of an indexed collection.

Keyed collections are those that implement the IDictionary interface. They contain items that can be retrieved by an associated key value of some kind. The contents of IDictionary collections are also usually sorted in some fashion based on the key value and can be retrieved in sorted order by enumeration. The System.Collections.HashTable class implements the IDictionary interface. However, ASP.NET Web Services do not natively support the passing of IDictionary objects. If you need this functionality, you will need to either develop a custom keyed collection that can convert itself into a type that ASP.NET Web Services does support, or expose your methods in a more neutral form. Here, we choose the latter.

The good news is that Indigo will solve this problem by exporting valid XSD for any CLR construct, including Generics and IDictionary. The bad news is that we have to wait for Longhorn to get Indigo.

Passing an IList

Collections that can be converted into single dimensional arrays can be passed directly in a Web Service. For example, an ArrayList of type string serializes to the following XML:

```
<?xml version="1.0" encoding="utf-8" ?>
<ArrayOfAnyType
  xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <anyType xsi:type="xsd:string">MyString</anyType>
</ArrayOfAnyType>
```

Serializing an ArrayList referencing anything else than objects of type object raises an exception. That is the reason for the restrictions on passing collections in the XML Serialization docs.

Passing an IDictionary as a Multidimensional Array

This sounds like an obvious solution. However, XML Web services that are created with ASP.NET do not support multidimensional arrays. If you check out the Microsoft Knowledge Base article on the subject, you'll see this behavior is, in typical Microsoft fashion, "by design".

Passing an IDictionary as a Jagged Array

Having been left with no more convenient alternative, we can convert a name/value collection into a two-column jagged array in which one column contains the key and the other contains the value. It's pretty easy to write a two helper methods that convert between Hashtables and jagged arrays:

```
public object[][] ToJaggedArray(Hashtable ht)
{
    object[][] oo = new object[ht.Count][];
    int i = 0;

    foreach (object key in ht.Keys)
    {
        oo[i] = new object[] { key, ht[key] };
        i++;
    }
    return oo;
}

public Hashtable ToHashtable(object[][] oo)
{
    Hashtable ht = new Hashtable(oo.Length);

    foreach(object[] pair in oo)
    {
        object key = pair[0];
        object value = pair[1];
        ht[key] = value;
    }
    return ht;
}
```

To make use of these methods, simply change your WebMethod signatures from Hashtable/IDictionary to the jagged array. For example:

```
[WebMethod]
public Hashtable GetHashtable()
```

becomes

```
[WebMethod]
public object[][] GetHashtable()
```

Serializing an IDictionary Using XSD Types

Now that we have exposed our WebMethods using types that can be serialized as XML, ASP.NET can construct a proper SOAP envelope for our call:

```
<?xml version="1.0" encoding="utf-8"?>
<soap:Envelope xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
  <soap:Body>
    <GetHashtableResponse xmlns="http://markrichman.com/webservices/">
      <GetHashtableResult>
        <ArrayOfAnyType>
```

```
          <anyType />
          <anyType />
        </ArrayOfAnyType>
        <ArrayOfAnyType>
          <anyType />
          <anyType />
        </ArrayOfAnyType>
      </GetHashtableResult>
    </GetHashtableResponse>
  </soap:Body>
</soap:Envelope>
```

As you can see, the CLR object type is represented as the XSD type anyType.

Putting it all Together

Now that we have a framework for serializing our collections, we can implement a web service and client application to test it all out.

To paraphrase our web service definition, we expose two WebMethods of interest in CollectionsWS.asmx:

```
[WebMethod]
public object[][] GetHashtable()
{
    object[][] oo = helper.ToJaggedArray(this.ht);
    return oo;
}

[WebMethod]
public void SetHashtable(object[][] oo)
{
    this.ht = helper.ToHashtable(oo);
}
```

After setting a Web Reference to CollectionsWS.asmx in our client, we can invoke the service as follows:

```
[STAThread]
static void Main(string[] args)
{
    CollectionsHelper helper = new CollectionsHelper();
    Hashtable ht = new Hashtable();

    ht["Zero"]  = 0;
    ht["One"]   = 1;
    ht["Two"]   = 2;
    ht["Three"] = 3;
    ht["Four"]  = 4;

    object[][] oo = helper.ToJaggedArray(ht);
    MarkRichman.Collections.WebService.CollectionsWS ws
        = new MarkRichman.Collections.WebService.CollectionsWS();
    ws.SetHashtable(oo);
    oo = ws.GetHashtable();
}
```

Here, our Hashtable is cleanly serialized via SOAP and remains interoperable with non-ASP.NET clients, such as Apache Axis, SOAP::Lite, and webMethods Glue.

Mark A. Richman has extensive experience as an independent consultant and software developer. He specializes in large-scale distributed web applications. Mark demonstrates his technical expertise through engagements with both Fortune 500 corporations and small start-up firms. He frequently mentors software developers in object-oriented concepts and techniques, and is the author of several publications. Visit his website at http://www.markrichman.com.