Collection Serialization

ASP.NET Web Services

Mark A. Richman
February, 2004

Table of Contents

1 Introduction

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.

1.1 Collections Overview

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.

> **Note** 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.

1.2 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.

1.3 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".

1.4 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()
```

1.5 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*.

2 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.

Note **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.

Notices

Document Source

This document was written by:
Mark A. Richman
Empire Software, Inc.
7420 NW 70 Avenue, Parkland, FL 33067

Disclaimers

The following paragraph does not apply where such provisions are inconsistent with local law: EMPIRE SOFTWARE, INC. PROVIDES THIS PUBLICATION "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESSED OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. Some states do not allow disclaimer of expressed or implied warranties in certain transactions; therefore, this statement might not apply to you.

This publication could include technical inaccuracies or typographical errors. Changes are periodically made to the information herein; these changes will be incorporated in new editions of the publication. EMPIRE SOFTWARE, INC. might make improvements and/or changes in the product(s) and/or the program(s) described in this publication at any time.

This publication might contain reference to, or information about, EMPIRE SOFTWARE, INC. products (machines and programs), programming, or services that are not announced in your country. Such references or information must not be construed to mean that EMPIRE SOFTWARE, INC. intends to announce such EMPIRE SOFTWARE, INC. products, programming, or services in your country.

The examples used in this document are for instructional purposes only. Any names, persons, associations, company names, domain names, User IDs, email addresses, or other addresses contained in this document are not intended to represent actual persons, associations, company names, domain names, User IDs, email addresses, or other addresses and any that do are entirely coincidental.

If this document contains graphic representations of system functions, they are sourced from a development environment. These graphics represent only examples of actual plans, products or services. The user interface that you see might be the result of different branding schemes, Internet browsers, or products; therefore, the graphics on your screen might look different from the graphics in this document.

Trademarks

EMPIRE SOFTWARE, INC. and EMPIRE SOFTWARE, INC.-related product names are trademarks of EMPIRE SOFTWARE, INC. All other trademarks in this document are the property of their respective owners.

Contact Empire Software, Inc.

If you have comments or questions about the content in this document, please send an email message to the EmpireDocuments@empiresoftware.net and include any specific details.

Requests for technical information about EMPIRE SOFTWARE, INC. products should be made to EMPIRE SOFTWARE, INC. Marketing Representatives.

Document Conventions

The conventions used in this document are designed to be completely predictable and are used for the following specific purposes.

Conventions List

Typeface	Usage
Italic	Used to indicate the following: • The first mention of new terms in any information unit. For example: The *rudaplex* and the *strataguide* have been the modified for this model. • References to titles of books, chapters, headings, CDs, diskettes, or software programs. For example: Refer to *The Technical Manual* for technical term descriptions. • Variables that the user types. For example: Type the *User ID* in the User ID text box.
Bold	Used to indicate the following: • Exact text strings typed. For example: Type **ABCDEFG**. • Keyboard keys pressed. For example: **Press Ctrl+a**, then press **Enter**.
Blue Underline	Used to indicate linked email, IP, Network, or Web addresses. For example: Go to http://www.microsoft.com for more information about Microsoft products.
Cross-Reference	Used to indicate a reference to another part of the same document. The grey portion of the cross-reference is hot linked to the appropriate section of the document, followed by a page number, also hot-linked to the same portion of the document. For example: For more information about the Document Conventions, see the "Document Conventions" on page 8.
`Operating System Text`	Used to indicate text that appears in a shell session for an operating system. The displayed text pertains to operating system text only, not application elements. For example: Type `LIST MAIN FOLDER`. The screen displays the Main folder.

`Program Code`	Used to indicate code listings. For example:
	`{`
	`# do something;`
	`}`
	`# check to see if $user has the attrib 'atrib'`
	`if (hasKey($user_obj, 'atrib', $dbh) != 1)`
	`{`
	`print "User not Authorized to update!";25`
	`}`

Screen Element	Screen elements consist of anything that is displayed on screen (exclusive of the operating system). This includes toolbar menu items, drop-down lists and items in a drop-down list, buttons, or anything else a user sees on screen. For example: • From the Printer drop-down list, choose Local Printer. The Are You Sure? dialog box appears. Click OK. • User Not Authorized

Special Elements

These elements provide a variety of information ranging from warnings that the reader should not neglect to supplementary information and tips that will simply enhance their reading experience.

Note Used to point out helpful ideas, some not-so-obvious features, quick or alternate ways to get a particular job done, and techniques you might not discover by yourself. The **Tip List** special element is used when multiple tips are used.

Note: Used to highlight certain information for the reader. Generally, the Note element provides additional information on the current topic. The **Notes:** special element is used when multiple notes are required.

Important:
Used for information that is considered more pertinent to the reader than information presented in Note elements.

Caution:
Used as a hazard light in Empire Software, Inc. documents. Information included in a Caution element could save the reader from hours of lost work.

Revision History

This section provides document revision history.

Edition	Date	Technical Source	Technical Writer	Description
First	January, 2004	Mark A. Richman	Mark A. Richman	First Draft
First	February, 2004	Mark A. Richman	Mark A. Richman	Final Draft

Summary of Changes

Changes for First Edition

- Applied new styles