THE XML FILES

Web Services and DataSets

Aaron Skonnard
Download the code for this article: XMLFiles0304.exe (279KB)

Programmers using Visual Basic® 6.0 have long bowed to the altar of the ADO recordset. It's probably the most commonly used data structure in Windows®-based applications today. The ADO.NET DataSet is poised to play a similar role in the realm of managed Windows-based applications.

When you need to extract information from a database, present it to the user, and apply updates, the DataSet is often the best fit. The DataSet has been used quite successfully for a variety of solutions targeting the Microsoft® .NET Framework. In the case of Web Services, the DataSet can cause interoperability issues.

If you use dynamic data structures like the DataSet it can be harder for developers who use other toolkits to consume your Web Service. Using typed DataSets in combination with slightly customized Web Services Description Language (WSDL) definitions, however, gets around this issue. Developers need to understand the issues surrounding all dynamic data structures (like the DataSet) before using them in Web Service interfaces.

Levels of Interoperability

The ability to process data regardless of where it comes from is the most primitive level of interoperability. I like to call this "data interoperability." XML 1.0 was specifically developed to facilitate data interoperability.

Data serialized into XML 1.0 is easy to extract and manipulate on any platform using any programming language for which an XML processor exists, which encompasses practically all languages today. Achieving data interoperability still requires you to write code against an XML API exposed by your XML processor, but you don't have to deal with the byte stream directly (see **Figure 1**).

Figure 1 Data Interoperability Through XML

Applying XML to distributed application technology (Web Services) simplifies interoperability between heterogeneous distributed systems. Web Services can be built using XML APIs directly. This approach gives you full control over

message processing, but it requires you to implement a great deal of common Web Service infrastructure, which can decrease productivity and introduce bugs that can lead to more interoperability problems.

Distributed technology vendors (like Microsoft) have been busy developing Web Service toolkits that provide this common infrastructure, helping reduce the XML API code required of the Web Service developer. The Microsoft ASP.NET WebMethod framework is one such toolkit that removes almost all traces of the lower-level System.Xml APIs. Web Service toolkits accomplish this by defining mappings between application types and XML Schema types, thereby automating the translation between XML documents and object instances at run time.

For example, consider the following WebMethod named CalculateMortgage that takes a MortgageInfo object as input and returns a MortgagePayments object to the caller:

```
[WebService(Namespace="http://example.org/mortgage")]
public class MortgageService : WebService
{
   [WebMethod]
   public MortgagePayments CalculateMortgage(
      MortgageInfo minfo)
   {
      ... // calculate mortgage
      return new MortgagePayments();
   }
}
```

Within this method, the developer is simply working with objects, but the [WebMethod] attribute tells the ASP.NET infrastructure to treat this method as a Web Service operation. So in this case the infrastructure automatically maps the MortgageInfo and MortagePayment classes to XML Schema type definitions.

Figure 2 shows the MortgageInfo and MortgagePayments C# class definitions used by the CalculateMortgage WebMethod, while **Figure 3** shows the XML Schema type definitions generated by the ASP.NET infrastructure. With these XML schema definitions in place, it's clear what the other side should expect in terms of XML.

Web Service toolkits make it easy to ignore XML since the Web Service developer deals only with objects. The toolkit on the other end of the wire, however, may also want to map the XML to objects. For example, a developer who uses the Java language might use something like Apache's Axis toolkit to consume this Web Service. To do so, he would use the WSDL2Java utility to automatically generate Java classes from the XML Schema type definitions (see **Figure 4**). This makes it possible for the Java developer to also work with objects without having to drop down to the lower-level Java XML APIs.

Now let's discuss the next level of interoperability—what I like to call "toolkit interoperability." Toolkit interoperability means that I can write Web Service code in one toolkit and easily consume it using another toolkit without dropping down to the lower-level XML APIs. There must be an equivalent object mapping or the other side will have to drop down and deal with XML directly (this is illustrated in **Figure 5**).

Figure 5 Toolkit Interoperability

Toolkit interoperability is harder to achieve due to the many differences in programming languages and type systems. The DataSet is one type that presents some interesting challenges to toolkit interoperability.

The DataSet Problem

Most Web Service toolkits do a good job mapping simple class definitions like the ones shown earlier. They begin to experience problems, however, when derivation and substitution come into play or when dynamic types are used, such as collections, hash tables, or DataSets. You should consider the following .NET Framework signature:

```
[WebMethod]
public DataSet GetAuthors()
{
    ... // return DataSet filled with authors
}
```

What should DataSet be mapped to in terms of XML Schema? The DataSet is a polymorphic type whose actual layout isn't determined until run time, after the DataSet has been filled with data.

The following XML Schema definition contains the element declaration for GetAuthorsResponse. Notice that GetAuthorsResponse contains an optional element named GetAuthorsResult, which contains two children: an XMLSchema schema element (s:schema), followed by a wildcard (s:any) indicating that any element from any namespace can be provided:

```
<s:element name="GetAuthorsResponse">
   <s:complexType>
      <s:sequence>
         <s:element minOccurs="0" maxOccurs="1"
            name="GetAuthorsResult">
            <s:complexType>
               <s:sequence>
                  <s:element ref="s:schema" />
                  <s:any />
...
</s:element>
```

This indicates that a schema will be provided at run time that will describe the XML that follows it. Here's what this would look like in the SOAP response:

```
<soap:Envelope
  xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/"
  xmlns:s="http://www.w3.org/2001/XMLSchema"
```

```
>
  <soap:Body>
    <GetAuthorsResponse
      xmlns="http://example.org/dataset-service">
      <GetAuthorsResult>
        <s:schema> *** schema goes here *** </s:schema>
        *** xml goes here ***
•••
</soap:Envelope>
```

Since the toolkit won't be able to determine the actual types used in the DataSet until run time, the next question is what the XML Schema definition (just shown) should map to in other environments like Apache's Axis toolkit.

Toolkit Bindings

At design time, when the Java developer runs WSDL2Java, there isn't enough information in the schema definition to do anything other than map the s:any element to java.lang.Object, as follows:

```
•••
public class GetAuthorsResult  implements java.io.Serializable {
    private java.lang.Object any;

    ... // omitted for brevity
}
```

As it turns out, Axis can't even get that far because it doesn't like the reference to the s:schema element that precedes s:any; it throws an exception and fails to produce anything in this case. If the schema reference didn't exist, however, the Java class would look something like the one shown previously.

Since neither the developer nor the toolkit have any additional type information (about the structure of the author information in this case), the only thing that makes sense is for the toolkit to provide a DOM tree in the field containing the parsed XML. Hence, in order for Java developers to consume this Web Service, they'll have to drop down and use their equivalent DOM API directly. The code in **Figure 6** shows you how to do this.

You will experience this issue with any toolkit that doesn't have a special case for DataSets in the .NET Framework. You'll even have to deal with the DOM API directly using the Microsoft SOAP Toolkit for COM developers.

According to the definitions I provided earlier there is still data interoperability in this situation (it's just XML), but a level of toolkit interoperability is lost since the consuming developer is now required to deal with the raw XML. If I know I'm returning a collection of author records with a well-defined format, there are a few ways I could make it easier for users of other toolkits to process my data without touching the XML APIs directly.

Avoiding DataSet

Probably the easiest way around this is to not use DataSets in your Web Service interfaces. For example, in this case you could define a class that models author information and declare the GetAuthors method as follows:

```
public class Author
{
```

```
    public string id;
    public string au_fname;
    public string au_lname;
    •••
}

[WebMethod]
public Author[] GetAuthors()
{
    DataSet authorsDS = GetAuthorsFromDB();
    ... // walk through DataSet and generate
        // Author array to return
}
```

Here I'm manually mapping a DataSet to the static type Author (since I'm probably only using the DataSet here to retrieve the data from the database, it probably makes more sense to use IDataReader directly since it would provide a more efficient solution). In this case there would be a natural mapping between the Author class and the equivalent XML Schema definition, which also map nicely back to an equivalent Java class (like the CalculateMortgage example shown earlier).

With something like this in place, the Axis developer would be able to write code without touching the XML APIs directly (see **Figure 7**). Another way to accomplish this would be to generate a typed DataSet from an XML Schema definition and use it instead of the generic DataSet.

Typed DataSets

Typed DataSets are classes that derive from DataSets and expose strongly typed members that depict a specific view of the data. Unlike generic DataSets, typed DataSets are bound to a specific structure, an XML Schema definition, at design time. You can automatically generate typed DataSets from XML Schema definitions using xsd.exe or the built-in designers in Visual Studio® .NET.

Figure 8 contains an XML Schema definition that describes the author records I intend to return from my Web Service. After running this schema through xsd.exe using the /dataset switch (or using the Visual Studio .NET designer), I'll have a new class named AuthorSet that derives from DataSet, as shown here:

```
[Serializable()]
public class AuthorSet : DataSet {

    private authorsDataTable tableauthors;

    ... // strongly typed accessors
}
```

The infrastructure knows what the schema definition should be for this typed DataSet, so it shouldn't have to provide a schema at run time anymore (although it still does). Now I can create a new WebMethod that returns an AuthorSet object:

```
[WebMethod]
public AuthorSet GetAuthorsAsTypedDataSet()
{
    AuthorSet aus = new AuthorSet();
    sqlDataAdapter1.Fill(aus);
    return aus;
}
```

Now the infrastructure has enough information to inform clients at design time about the structure of the author records in the returned set. When the ASP.NET infrastructure generates the WSDL and XML Schema definitions, however, it imports the typed DataSet's XML Schema definition and uses a constrained wildcard, as shown in **Figure 9**.

Since there is no longer a reference to the s:schema element, WSDL2Java is able to generate Java classes successfully. And since the import statements give it access to the XML Schema definition for AuthorSet, it can generate equivalent Java classes that represent the author information. Axis, however, still maps the wildcard (s: any) to java.lang.Object, which will contain a DOM tree at run time. This gets me further than before, but still leaves the client working with the DOM API directly.

Customizing the Schema

Since you know the wildcard slot is always going to contain an AuthorSet, you might be tempted to just modify the schema definition to reference the AuthorSet element instead of the wildcard. The wildcard, however, is actually a placeholder for the generic DataSet schema (which doesn't make much sense since the wildcard is constrained to the http://example.org/dataset namespace). In other words, when you execute the WebMethod, the GetAuthorsAsTypeDataSetResult element will contain a schema element followed by a diffgram element, and the diffgram element will contain the AuthorSet element.

But since you know the structure ahead of time, you don't need to provide a run-time schema or a diffgram element; you can simply return an AuthorSet element. The way to do this is to define a new WebMethod that doesn't return a DataSet, but rather an XmlNode. Then if you return an XmlDataDocument that wraps the typed DataSet object, only the AuthorSet element is returned to the client, without the run-time schema or diffgram elements:

```
[WebMethod]
public XmlNode GetAuthorsAsXml()
{
    AuthorSet aus = new AuthorSet();
    sqlDataAdapter1.Fill(aus);
    return new XmlDataDocument(aus);
}
```

XmlDataDocument is inefficient since it roughly doubles the size of the original DataSet. Since it's transient, it may be acceptable.

Specifying that the method returns an XmlNode will also result in a wildcard in the generated schema, but now you

can replace the wildcard with a reference to the AuthorSet element, as shown here:

```
•••
<s:element name="GetAuthorsAsXmlResponse">
  <s:complexType>
    <s:sequence>
      <s:element minOccurs="0" maxOccurs="1"
        name="GetAuthorsAsXmlResult">
        <s:complexType>
          <s:sequence>
            <s:element ref="x:AuthorSet" />
          </s:sequence>
        </s:complexType>
•••
```

You'll have to make sure clients no longer retrieve the auto-generated WSDL file once you start down this path. Instead you'll want to specify a custom WSDL file location by using the WebServiceBindingAttribute (see the Location property) or by providing a static WSDL file on your server and disabling the automatic WSDL generation. You can disable the automatic WSDL generation by adding the following section to your web.config file:

```
<webServices>
  <protocols>
    <remove name="Documentation" />
  </protocols>
</webServices>
```

Now the Web Service can still work with DataSets and other clients can process the results as an array of objects. **Figure 10** shows what an Axis client would look like in this case.

If this seems like a big hassle just to use DataSets, you can always avoid them in your Web Service interfaces or simply require your clients to deal with the XML directly. It all comes down to what you want other toolkits to see.

The Ultimate Solution

This problem could eventually go away if the industry defined a standard XML Schema definition for representing database resultsets and all toolkits properly supported it. Then toolkits could map their native resultset types to the standardized schema. The same holds true for other dynamic types like collections, hash tables, lists, queues, and so on. Standardizing XML Schema definitions for such types would have the effect of promoting toolkit interoperability across a wider range of extremely useful and commonly used data structures.

If this were a reality, you could return a generic DataSet (like the first GetAuthors example) and the Axis toolkit could map it to a JDBC WebRowSet (or something equivalent). Then, the Java developer could bind the WebRowSet to a UI component, allow the user to modify the data, and pass an updated set back to the server—all with very minimal code. The updated set could then be deserialized back into a DataSet on the server in order to apply the changes to the data.

This solution makes the most sense if you think about why you would want to use a DataSet in a Web Service.

DataSets are typically used when you need to present data to the user in a generic fashion (usually in a grid), allow the user to make changes, and apply updates back to the database. The techniques described here don't support this pattern (since you don't get data binding or diffgram support with DOM trees or arrays), so it's a lot like pounding a square peg into a round hole.

Of course, standardizing anything in this industry is much easier said than done. Don't hold your breath waiting for such standards to become codified and supported throughout today's toolkits.

A .NET Framework Workaround

You've probably seen trade show demos in the past where the presenter returned a DataSet to a .NET Framework client and somehow it magically became a DataSet on the other end. This works because the Web Service infrastructure in the .NET Framework looks for a special flag in the runtime-provided XML Schema definition called IsDataSet. For example, in the case of the GetAuthorsAsTypedDataSet example, the AuthorSet element declaration will look like this in the runtime-provided schema:

```
•••
<xs:element name="AuthorSet" msdata:IsDataSet="true">
•••
```

The IsDataSet flag is the "wink" to the client, telling it to deserialize the XML back into a DataSet. Obviously, this only works for .NET Framework clients or other toolkits that have provided this hardcoded special case (there are none that I know of today). The code in **Figure 11** shows how a Visual Basic® .NET client could invoke both GetAuthors and GetAuthorsAsTypedDataSet and easily map the returned DataSet to a DataGrid (on a Windows Form).

It would be just as easy across all toolkits and a variety of database APIs if there was a standardized schema for database resultsets that everyone supported. Ironically, once you have something like this in place, it's actually more tedious to work with an array of Author objects when your goal is to present the data to the user for generic manipulation.

Wrapping It Up

XML and Web Services provide basic data interoperability today. If you're willing to work with XML APIs, there isn't a Web Service in the world that you can't consume. If, however, you'd rather tuck the XML APIs under the rug and never deal with them again, be careful with dynamic types like the DataSet. Using the generic DataSet in Web Service interfaces will force developers using other toolkits to deal with the XML directly. Using typed DataSets in combination with slightly customized WSDL definitions gets around this issue by exposing the results as a simple array of objects. The ultimate solution, however, would be to standardize an XML Schema definition for representing database resultsets that could be supported across all toolkits.

Send your questions and comments for Aaron to xmlfiles@microsoft.com.

Aaron Skonnard is an instructor/researcher at DevelopMentor, where he develops the XML and Web Service-related curriculum. Aaron coauthored *Essential XML Quick Reference* (Addison-Wesley, 2001) and *Essential XML* (Addison-Wesley, 2000). Reach him at http://staff.develop.com/aarons.

From the April 2003 issue of MSDN Magazine.
Get it at your local newsstand, or better yet, *subscribe*.

Figure 2 Class Definitions

```
public class MortgageInfo
{
   public double amount;
   public double years;
   public double interest;
   public double annualTax;
   public double annualInsurance;
}

public class MortgagePayments
{
   public double MonthlyPI;
   public double MonthlyTax;
   public double MonthlyInsurance;
   public double MonthlyTotal;
}
```

Figure 3 XML Schema Type Definitions

```
•••
<s:complexType name="MortgageInfo">
   <s:sequence>
     <s:element name="amount" type="s:double" />
     <s:element name="years" type="s:double" />
     <s:element name="interest" type="s:double" />
     <s:element name="annualTax" type="s:double" />
     <s:element name="annualInsurance" type="s:double" />
   </s:sequence>
</s:complexType>

<s:complexType name="MortgagePayments">
   <s:sequence>
     <s:element name="MonthlyPI" type="s:double" />
     <s:element name="MonthlyTax" type="s:double" />
     <s:element name="MonthlyInsurance" type="s:double" />
     <s:element name="MonthlyTotal" type="s:double" />
   </s:sequence>
</s:complexType>
•••
```

Figure 4 Apache Axis-generated Java Class Definitions

```
/**
 * MortgageInfo.java
```

```
 *
 * This file was auto-generated from WSDL
 * by the Apache Axis WSDL2Java emitter.
 */

public class MortgageInfo  implements java.io.Serializable {
    private double amount;
    private double years;
    private double interest;
    private double annualTax;
    private double annualInsurance;

    public MortgageInfo() {
    }

    public double getAmount() {
        return amount;
    }

    public void setAmount(double amount) {
        this.amount = amount;
    }
        ... // remainder ommited for brevity
}

/**
 * MortgagePayments.java
 *
 * This file was auto-generated from WSDL
 * by the Apache Axis WSDL2Java emitter.
 */

package mortgage_tools;

public class MortgagePayments  implements
  java.io.Serializable {
    private double monthlyPI;
    private double monthlyTax;
    private double monthlyInsurance;
    private double monthlyTotal;

    public MortgagePayments() {
    }

    public double getMonthlyPI() {
        return monthlyPI;
    }

    public void setMonthlyPI(double monthlyPI) {
        this.monthlyPI = monthlyPI;
    }
        ... // remainder omitted for brevity
}
```

Figure 6 Axis Client Program (Using XML APIs)

```
import java.io.*;
import org.example.*;
import org.w3c.dom.*;
import org.apache.xerces.dom.*;
import org.apache.axis.message.*;

public class AuthorsClient {
   static void main(String[] args) throws Exception
   {
       DataSetServiceSoapStub stub =
           new DataSetServiceSoapStub(new java.net.URL(
           "http://localhost/ds/ds.asmx"), null);

       GetAuthorsAsTypedDataSetResult result =
         stub.getAuthors();
       Object any = result.getAny();
       // it's really a DOM tree
       Element docElement = (Element)any;

       NodeList authors = docElement.getElementsByTagNameNS(
           "http://example.org/dataset", "authors");
       for (int i=0; i<authors.getLength(); i++)
       {
           Element authorsElem = (Element)authors.item(i);
           ElementImpl fnameElem =
              (ElementImpl)authorsElem.getElementsByTagNameNS(
              "http://example.org/dataset","au_fname").item(0);
           System.out.println(fnameElem.getTextContent());
       }
   }
}
```

Figure 7 Axis Client (No XML APIs)

```
import java.io.*;
import org.example.*;

public class AuthorsClient {
   static void main(String[] args) throws Exception
   {
      DataSetServiceSoapStub stub =
          new DataSetServiceSoapStub(new
          java.net.URL("http://localhost/ds/ds.asmx"), null);
      Author[] authors = stub.getAuthors();
      for (int i=0; i<authors.length; i++)
         System.out.println(authors[i].getAu_Fname());
   }
}
```

Figure 8 Typed DataSet XML Schema Definition (AuthorSet.xsd)

```
<xs:schema id="AuthorSet"
  targetNamespace="http://example.org/dataset"
  xmlns="http://example.org/dataset"
```

```
  xmlns:xs="http://www.w3.org/2001/XMLSchema"
  elementFormDefault="qualified"
  ...
>
  <xs:element name="AuthorSet" ...>
    <xs:complexType>
      <xs:choice maxOccurs="unbounded">
        <xs:element name="authors">
          <xs:complexType>
            <xs:sequence>
              <xs:element name="au_id" type="xs:string" />
              <xs:element name="au_lname" type="xs:string" />
              <xs:element name="au_fname" type="xs:string" />
              ...
            </xs:sequence>
          </xs:complexType>
        </xs:element>
      </xs:choice>
    </xs:complexType>
    ...
  </xs:element>
</xs:schema>
```

Figure 9 GetAuthorsAsTypedDataSet XML Schema Definition

```
<definitions
  xmlns:s="http://www.w3.org/2001/XMLSchema"
  targetNamespace="http://example.org/dataset-service"
  xmlns="http://schemas.xmlsoap.org/wsdl/"
  ...
>
  <import namespace="http://example.org/dataset"
   location="http://localhost/ds/ds.asmx?schema=AuthorSet" />
  <types>
    <s:schema
     elementFormDefault="qualified"
     targetNamespace="http://example.org/dataset-service"
    >
      <s:import namespace="http://example.org/dataset" />
      ...
      <s:element name="GetAuthorsAsTypedDataSetResponse">
        <s:complexType>
          <s:sequence>
            <s:element minOccurs="0" maxOccurs="1"
             name="GetAuthorsAsTypedDataSetResult">
              <s:complexType>
                <s:sequence>
                  <s:any
                   namespace="http://example.org/dataset" />
                </s:sequence>
              </s:complexType>
            </s:element>
          </s:sequence>
        </s:complexType>
      </s:element>
```

```
      ...
    </s:schema>
    ...
</definitions>
```

Figure 10 Axis Client from Typed DataSet

```
import org.example.*;

public class AuthorsClient {
   static void main(String[] args) throws Exception
   {
      DataSetServiceSoapStub stub =
        new DataSetServiceSoapStub(new java.net.URL
        ("http://localhost/ds/ds.asmx"), null);
      GetAuthorsAsTypedDataSetResult res =
        stub.getAuthorsAsTypedDataSet();
      AuthorSet aset = res.getAuthorSet();
      Author[] authors = aset.getAuthors();
      for (int i=0; i<authors.length; i++)
         System.out.println(authors[i].getAu_Fname());
   }
}
```

Figure 11 Mapping DataSet to a DataGrid

```
Private Sub Button1_Click(ByVal sender As System.Object, _
  ByVal e As System.EventArgs) Handles Button1.Click
    Dim svc As DataSetService = New DataSetService()
    Dim ds As DataSet = svc.GetAuthors()
    DataGrid1.DataSource = ds.Tables.Item(0)
End Sub

Private Sub Button2_Click(ByVal sender As System.Object, _
  ByVal e As System.EventArgs) Handles Button2.Click
    Dim svc As DataSetService = New DataSetService()
    Dim auset As AuthorSet = svc.GetAuthorsAsTypedDataSet()
    DataGrid1.DataSource = auset.authors
End Sub
```