PINNACLE

XML Developer

XML Solutions for Client/Server, Web-Based, and Business-to-Business Systems

Connecting to Web Services

Zoran Zaev

DOWNLOAD

Implementing Web Services in different languages (and platforms) isn't difficult, as long as you have some familiarity with the particular language, the platform, and a general knowledge of XML and SOAP. However, making sure that the different systems can connect to each other can be more challenging. In this article, Zoran Zaev goes into more detail.

IN this article, I'll be covering different scenarios of systems using different languages, different platforms, and different object models. You'll see how they can all be brought to talk to each other. I'll start with Visual Basic and the scripting languages: JScript, VBScript, and Perl. In future articles, I'll continue this exercise with other language and platform combinations.

The reason for this quest is that Web Services are most useful when connecting systems based on different languages, operating systems, and/ or object models. These include, but aren't limited to, Microsoft's Component Object Model (COM), Common Object Model Request Broker Architecture (CORBA), or Java's Enterprise Java Bean (EJB) specification. When faced with situations where you need to connect these different systems, Web Services offer standard mechanisms. This isn't to say that you shouldn't use Web Services to connect systems based on the same programming language or object model. Similar systems can benefit from the inherent loose coupling of the Web Services model.

October 2001

Volume 2, Number 10

DOWNLOAD

Accompanying files available online at http://www.xmldevelopernewsletter.com

Loose coupling refers to systems where the components can easily bind at runtime without the need for additional steps (for instance, operating system registration). The opposite is a tightly bound system where the components of a system are bound together at compile time or even at design time with code that's dependent on the internals of each component. Loosely bound systems are easier to modify because components can be plugged in and out more easily, but they can be more difficult to develop.

In addition, using Web Services when building systems based on similar technologies might be beneficial if you plan or expect to have other systems connect to your system in the future and you don't want to limit yourself to systems that only run on your platform.

So, in this article, I'll create a simple Web Service with Microsoft's SOAP Toolkit 2 and Visual Basic 6. Then, I'll show you how to connect to this Web Service using Visual Basic 6, and some scripting languages, in particular JScript within ASP, VBScript within ASP, and Perl. In the next few months, I'll implement this same Web Service in different languages and try to access it from platforms other than the platform that the service is running on. Through these articles, you'll become familiar with what it takes to create a Web Service in various languages and have it interoperate with systems that aren't necessarily the same as the languages or platform used in the creation of the Web Service itself. In addition to discussing particular problems, I'll discuss some common interoperability issues and ways to get around them.

The Microsoft SOAP Toolkit 2 was reviewed by Wayne Wallace in the July 2001 issue of *XML Developer*. While it shouldn't be absolutely necessary, you might want to review that article before continuing with this one. In the August 2001 issue, Peter Vogel went further into the details of creating a Web Service using the Microsoft SOAP Toolkit 2, Visual Basic, and .NET ("Creating a Web Service"). So I'll only briefly cover the creation of the Web Service and spend much more time on building the various client implementations to focus on the interoperability issues.

The sample Web Service

My sample Web Service is a job submittal service, part of a sample Job Bank system. Partners will be allowed to submit new job postings to this Web Service. You can easily imagine that companies that host job banks could use this kind of a service to allow other companies to submit new job postings. Of course, this Job Bank is a sample application, and in a real-life scenario you'd have more complex Web Services (and likely more than one Web Service). Additionally, security would need to be applied, to control who's allowed to submit a new job posting.

The new job posting service, in my example, will consist of the following elements:

- companyID: The ID of the company submitting the job
- expireDate: The date when the posting ought to expire
- jobTitle: The short title for the job
- jobDesc: The text describing the job
- jobPay: The yearly compensation offered for this job

Creating the Web Service

For this article, I'll use the Microsoft SOAP Toolkit 2.0. If you want to create this version of the service, you should download and install the toolkit, if you haven't done that already (you can find it at http://msdn.microsoft.com/webservices/). The first step is to create the COM object that will serve the requests for job submittals and then expose this object to make it accessible as a Web Service.

I'll create the COM object using Visual Basic 6. The COM object will consist of one class and one function that can be called to add jobs to our Job Bank system. This function would look like this:

```
Public Function JobAdd( _
   ByVal companyID As Integer, _
   ByVal expireDate As Date, _
   ByVal jobTitle As String, _
   ByVal jobDesc As String, _
   salaryAmount As Double) As String
   If companyID = "1" Then
      JobAdd = "Your job with title: " & _
         jobTitle & " and description: " & _
         jobDesc & " was received " & _
         "successfully on " & Now() & _
         ". The salary requested is: " & _
         salaryAmount & " and the expiration" & _
         "date of this posting is: " & _
         expireDate & "."
   Else
       JobAdd = "Your company is not allowed" & _
       " to post jobs to our Job Bank."
   End If
End Function
```

This file is available in the Source Code file at www.xmldevelopernewsletter.com, or you can create it by using the code provided and compiling it with Visual Basic 6. If you download the COM object, you'll have to register it on the server by typing the following command at the command prompt within the directory where you have this file:

```
regsvr32 wsJobBank.dll
```

Within my code, I define the function that will be called to add new job postings and the parameters that will be passed in to the function. In the preceding code, I specified that all parameters are "ByVal." If I didn't do this, the parameters would have become both input and output parameters within the SOAP messages Generator. Typically, at this point, I'd have code to call other modules that would handle the data access and update. In order to make this service easier to install, I've omitted that code. I do check for the companyID, and if it's the company with an ID of "1," the caller will get a confirmation of a successful job posting (in production, you'd replace this with security related code).

After creating the COM object, I used the WSDL Generator utility provided with the SOAP Toolkit 2.0 to generate the WSDL and WSML file for the created COM object. The WSDL file describes the Web Service, and the WSML file (specific to Microsoft) helps the system map the Web Service to the COM object. Within the WSDL Generator wizard, I named the Web Service "wsJobBank" and provided the physical path to the wsJobBank.dll. Then, I selected the service that I'd like to expose—in this case, it's only one: the JobAdd service. As the listener URI, I provided "http://localhost/wsJobBank/." I used the 2001 version of the XSD (XML Schema), and I decided to use the ISAPI listener, which is recommended for most production-level situations. Finally, I created a virtual directory within the Web server (IIS, or Internet Information Server) with the name of "wsJobBank" pointing to the physical location where my WSDL and WSML files are located.

Testing the Web Service with a Visual Basic client

The easiest way to test a Web Service is to create a simple client within the same platform and language environment to attempt calling the Web Service. In this case, I created a simple Visual Basic 6 Standard EXE project. The code to access the Web Service is very short:

```
Private Sub txtExecute_Click()
Dim soapClient As MSSOAPLib.soapClient
   Set soapClient = New MSSOAPLib.soapClient

   soapClient.mssoapinit txtWSDL
   txtResult.Text = soapClient.JobAdd _
      (1, #12/31/2001 11:15:00 AM#, _
      "XML Project Manager", _
      "Must have 10 years of technical " & _
      "experience...", 80000)

   Set soapClient = Nothing
End Sub
```

Within the preceding code, I initiated the Web Service by supplying the WSDL file location to the mssoapinit method of Microsoft's soapClient object, which handles the communication with the Service. The soapClient object comes with Microsoft's SOAP Toolkit, version 2. I've purposefully omitted the second and third arguments to the mssoapinit method (the second parameter is the name of the Web Service, useful when more than one Web Service is present within the WSDL file, and the third argument is the location of the WSML file).

The first example of the issues involved appears here in the way that I passed the date parameter. You must have a way of submitting this value, so that the soapClient knows it's a date and not another type or a variant. I've enclosed the data in hash marks (#). I could also have used this:

```
CDate("12/31/2001 11:15:00 AM")
```

Once the soapClient knows that this is a date value, it will convert it into the standard ISO 8601 date format as required by SOAP (see www.iso.ch/markete/8601.pdf for more information). This is the format of the dateTime primitive datatype defined within the XML Schemas specification (see www.w3.org/TR/xmlschema-2/#dateTime). So the dateTime value, when sent in the SOAP message, will have the format YYYY-MM-DDThh:mm:ssZ. The "T" is the time separator, and the "Z" at the end indicates the Universal Time-Coordinated (UTC). If I didn't specify the time, Visual Basic would have stamped it with the value for midnight (00:00:00). In this example, my UTC was adjusted five hours ahead due to the time zone where I'm located. In the final SOAP message, my data parameter would look like this:

```
<expireDate>2001-12-31T16:15:00Z</expireDate>
```

It's important to pay attention to the datatypes used and the format used to send them across the wire. Datatype conversion is one of the most common interoperability challenges when connecting Web Services systems communicating across different languages or platforms.

If you have to update your Web Service component after you've initiated it, then you'll likely have to unload it from the Web server's memory (for example, by restarting the IIS service). The ISAPI for my service listener runs as an ISAPI extension to the WSDL file, and when the Web Service is called, the ISAPI listener loads the COM component in the memory space of the Web application. This memory space might or might not be the same as the memory space of the Web server, depending on the application protection level that's been set for the Web application directory. In IIS, the default is "Pooled," which has the component running in a common Web application space that's separate from the Web server itself.

Connecting from JScript and ASP

In the following code, you can see how to call the Web

Service from JScript running under ASP. Again, I've used the SoapClient, and I set the location of the WSDL file. The differences between this version and the Visual Basic client are primarily in setting the ServerHTTPRequest property of the SoapClient object to true in order to specify that you're getting the WSDL file via HTTP. If I'd decided to get the WSDL file from the local file system, then I wouldn't have had to set this property.

After initiating the soapClient object, I set a few HTML tags for the document, since it's going to be displayed in the browser, and then I called the JobAdd operation of the JobBank Web Service. Keep in mind that with ASP (in both JScript and VBScript), the parameters are passed as Variants, but the soapClient using the WSDL file properly converts them into a particular datatype (although the actual datatype isn't specified explicitly within the SOAP message). For example, the date is properly converted into the XML Schema format. Here's the JScript code:

```
<%@Language = "JScript"%>
<%
   var objSoapClient =
      Server.CreateObject ('MSSOAP.soapClient');
   var strWSDL =
      'http://localhost/wsJobBank/' +
      'wsJobBank.WSDL';

   objSoapClient.ClientProperty
      ('ServerHTTPRequest') = true;
   objSoapClient.mssoapinit (strWSDL);

   Response.Write ('<HTML><HEAD><TITLE>JobBank' +
      ':Add Jobs Via JScript' +
      '</TITLE></HEAD><BODY>The Response from ' +
      'the Web Service is: ' + '<BR><BR>');
   Response.Write (objSoapClient.JobAdd (1,
      '12/31/2001 11:15:00 AM',
      'XML Project Manager',
      'Must have 10 years of technical ' +
      'experience...', 80000));
   Response.Write ('</BODY></HTML>');

   objSoapClient = null;
%>
```

Connecting to the Web Service from the VBScript client within ASP

Connecting to the Job Bank Web Service via VBScript is very much the same as in the JScript example. I'm not going to comment on this code:

```
<%@Language = "VBScript"%>
<%
   Set objSoapClient = _
      Server.CreateObject ("MSSOAP.soapClient")
   strWSDL = "http://localhost/wsJobBank/" & _
      "wsJobBank.WSDL"

   objSoapClient.ClientProperty _
      ("ServerHTTPRequest") = True
   objSoapClient.mssoapinit (strWSDL)

   Response.Write ("<HTML><HEAD><TITLE>" & _
      "JobBank: Add Jobs Via VBScript" & _
      "</TITLE></HEAD><BODY>The Response " & _
      "from the Web Service is: " & "<BR><BR>")
   Response.Write (objSoapClient.JobAdd (1, _
      "12/31/2001 11:15:00 AM", _
      "XML Project Manager", _
      "Must have 10 years of technical " & _
      "experience...", 80000))
   Response.Write ("</BODY></HTML>")

   Set objSoapClient = Nothing
%>
```

In using this Job Bank Web Service, you're not limited to the COM-based languages, such as Visual Basic, or even JScript and VBScript (found in ASP or within the Windows Scripting Host environment). You can call this Web Service using any other language and/or platform. To demonstrate, I'll show you how to access this service with Perl.

Connecting from a Perl client

I used ActiveState's ActivePerl 5.6 (see http://aspn.activestate.com/ASPN/Downloads/ActivePerl/), but you should be able to use any version of Perl 5.004 or later. The SOAP implementation that I'll use is SOAP::Lite, a Perl implementation by Paul Kulchenko. You can find both Win32 and UNIX versions of SOAP::Lite at www.soaplite.com, where you'll also find all of the installation instructions, documentation, and so on. The code starts as follows:

```
use SOAP::Lite;
print "HTTP/1.0 201 Ok \n";
print "Content-type:text/html\n\n";
print "<HTML><HEAD><TITLE>JobBank:".
   "Add Jobs Via PERL";
print "</TITLE></HEAD><BODY>The Response ".
   "from the Web Service is: ";
print "<BR><BR>";
```

First, I simply set the HTTP headers, because I'd like to call this Perl script from within the browser instead of on the server as I did with ASP. Then, I set the beginning of the HTML document.

In the Perl version of the code, I call the SOAP::Lite object. I specify the namespace URI used (you can get it from the WSDL file) and the action HTTP header (this has to match to the SOAPAction attribute of the <soap:operation> element within the WSDL file). The proxy setting provides the address to the Web Service (this is the URL to the WSDL file, since I'm using the ISAPI listener) and points to port 8080. If the Web server that hosts the Service listens to the standard port 80, then you can omit this port value. Otherwise, change the value to the port number to which the Web site that hosts the Web Service is listening. For your testing, if you're going to change the port number from 8080, you ought to change this value within the WSDL file provided in the Source Code file. Within this file, this means modifying the location attribute of the <soap:address> element from 8080 to your own value. Keep in mind that the Trace Utility provided with the

Continues on page 14

Connecting to Web Services...

Continued from page 4

Microsoft SOAP Toolkit 2 requires you to specify a port number other than 80, such as 8080, in order to trace the actual SOAP messages that are sent between the client and the server.

Finally, I invoke the JobAdd operation on the Web Service and provide it with values for all of the parameters. In some situations, you might be able to point Perl to your WSDL file so that it will be able to figure out the types for all of your parameters (since Perl, in essence, is a typeless language). However, in this case, probably due to using the ISAPI listener, I had to manually specify the types and names for each one of the parameters, which made the code little longer than it would have been:

```
print SOAP::Lite
   ->uri ("http://tempuri.org/message/")
   ->on_action(sub { return
   '"http://tempuri.org/action/clsJobs.JobAdd"'})
   ->proxy ("http://localhost:8080/wsJobBank/".
   "wsJobBank.WSDL")
   ->JobAdd (SOAP::Data->type(short => 1)->
         name('companyID'),
      SOAP::Data->type(dateTime =>
         '2001-12-31T16:15:00Z')->
         name('expireDate'),
      SOAP::Data->type(string =>
         'XML Project Manager')->
         name('jobTitle'),
      SOAP::Data->type(string =>
         'Must have 10 years of technical '.
         'experience...')->name('jobDesc'),
      SOAP::Data->type(double => 80000)->
         name('salaryAmount'))
   ->result;
print "</BODY></HTML>";
```

You've now seen how to create a sample Web Service with the Microsoft's SOAP Toolkit 2 and Visual Basic 6. And you've seen how this service interoperates not only with Visual Basic clients, but also with scripting clients such as JScript, VBScript, and Perl. I pointed out some common issues around the data types being passed (such as the "date" datatype). In future articles on Web Services connectivity, I'll take this same Job Bank Web Service and port it to Java (among other languages) and call it using yet different languages and platforms. ▲

DOWNLOAD **INTEROP1.ZIP at www.xmldevelopernewsletter.com**

Zoran Zaev works as a senior Web solutions architect for Hitachi's solutions division. He enjoys helping others realize the potential of technology, and when he isn't working, he spends considerable time writing articles such as this one and books (for example, he co-authored *Professional XML Web Services*, from Wrox Press). Zoran's research interests include complex systems that often involve XML and highly distributed architectures, as well as the application of these concepts in newer areas such as biotechnology. When not programming or thinking of exciting system architectures, you'll find Zoran traveling, reading, and exploring various learning opportunities. zzaev@yahoo.com.