Web Services Tools for iSeries Programmers

By Phil Coulthard and George Farr

Welcome back to our series of articles following the iSeries developer roadmap. Previous articles focused on the tools in WebSphere Development Studio Client for iSeries (WDSc) for developing new Web user interface applications. Now we will focus on the tools for creating Web services. Some of the functions we discuss are available only in the Advanced Edition of WDSc, which we highly recommend you take a look at.

What Are Web Services?

A Web service is an industry standard that enables invoking business logic over the Internet, passing in parameters, and receiving results. It is a request-response model. The caller does not know or care what language the business logic is written in or where it exists. Although many technologies have allowed for remote procedure calling over the years, the Web services standard is increasingly popular because it is fully open and has universal buy-in, with participation from both Microsoft and IBM.

Web services have many uses, including exposing services for business-to-business transactions (such as placing or checking the status of an order), public-to-business transactions (by publishing services in open catalogs), and an internal component model for defining reusable components within your company. You do not need to know much in order to expose your RPG or Cobol business logic as Web services.

Basically, a Web service is business logic that you can invoke remotely by sending an XML document to a Web service router or engine, such as the one supplied with WebSphere products. The XML document, or message, conforms to a Simple Object Access Protocol (SOAP) standard and identifies the service that is being called and its parameter data. At the "end point," a SOAP engine (really just a special servlet) receives the request; identifies, finds, and calls the requested service; and returns the result. This resulting response is also returned as a SOAP message. Generally, a generated class is supplied for client code to call a specific Web service, which knows how to build the SOAP XML document and how to send it over HTTP or some other transport.

A SOAP document or message describes one specific service call or response (including the data), and another XML-based language – Web Services Description Language (WSDL) – describes the services and their parameters and data types. WSDL also supplies binding information to produce SOAP messages that invoke the services they describe. Tools use WSDL to generate the client code that can invoke a Web service using SOAP.

One of the interesting things you can do with a Web service is publish it to a registry or catalog, so that others can discover it and call it. These registries use the Universal Description, Discovery, and Integration (UDDI) standard. In a UDDI registry, you register your company and your services, including access to their WSDL. Public registries are found at uddi.org/register.html, and you can have your own private register for testing or internal purposes. IBM supplies a public registry called the IBM UDDI Business Registry at ibm.com/software/solutions/webservices/uddi, and the WebSphere Studio family of products includes a built-in private registry.

Finally, there is a somewhat lighter alternative to a UDDI registry: a Web Service Inspection Language (WSIL) document. This XML file is generally published at your own Web site (as opposed to a central repository) or linked with a URL, and it identifies the Web services you want to expose. These Web services can be identified using WSDL or UDDI references.

What Are the Web Services Tools?

Now that we know some of the technology, let's talk about the tools in WDSc for creating Web services. The product contains not only the tools inherited from WebSphere Studio for Java developers but additional iSeries extensions for RPG programmers. We start with the former, and then describe the latter.

The inherited Web services development tools include:

- a built-in private UDDI registry, in addition to the built-in WebSphere Test Environment you need to test your Web services
- support for publishing a Web service to a UDDI registry
- a Web Services Explorer for exploring the services in a UDDI registry or WSIL document
- import, export, and editor support for WSDL and WSIL files
- a wizard to generate Java client code for calling an existing Web service, given the service's WSDL file
- wizards to generate a Web service, including its WSDL and Java client, from various sources including a simple Java bean, an Enterprise JavaBean (advanced edition only), a URL, DB2 XML Extender calls or stored procedures, and SQL queries
- a wizard to generate a "test client" Web page from a Web service, where you can enter input, submit the Web service call, and then see the result
- a TCP/IP monitor, which is very handy for seeing the SOAP messages

This list is not complete. WDSc has exceptional support for Web services, and indeed also for XML, because many Web services accept or return XML documents, although that is not required. Any Java, RPG, or Cobol non-interactive program or ILE procedure can be turned into a Web service.

Perhaps the most interesting wizard is the one for creating a Web service from a Java bean. The wizard takes any Java class ("program" in ILE-speak) that has a default constructor (no INZSR subroutine, for you RPG programmers) and methods (procedures), and generates the WSDL for the selected methods in that class, as well as the necessary code for creating a Web service (mainly code to serialize the class and its parameters).

Figure 1: Cheat sheet for creating a Web service from a Java bean

The other interesting wizard is the one that generates from WSDL a Java class client proxy for remotely invoking the service from Java. In fact, if you look under Help|Cheat Sheets|Create a Web service from a Java bean, you'll find a "cheat sheet" to help you walk through these steps (**Figure 1**).

So you need to focus on creating a Java class containing the methods you want to expose in your Web service. Where do you get these classes? Well, you could create them. If you have a database you want to retrieve data from, you could write a Java class with methods for accessing the data, using JDBC or the record access classes in the iSeries Toolbox for Java. Indeed, the data tooling in WDSc includes support to generate such a class using JDBC. In fact, a number of tools can generate Java classes to access existing resources, and as soon as you have any class with one or more interesting methods, you can quickly and easily produce a Web service out of it. Among these tools is one for creating a Java class from an RPG or Cobol program.

In addition to the RPG and Cobol development tools in the Remote System Explorer, WDSc provides a Program Call wizard, which generates a simple Java bean (again, a class with a default constructor) from a program or ILE procedure. Just as you do in the Web Interaction wizard we described in our previous article ("More Web Tools for iSeries Programmers," TUG Vol.21 No.3), in this wizard you describe the program or procedure name and its parameters, or you import a Program Call Markup Language file generated by the compiler.

Let's walk through creating a Web service to call the same program we created a Web interaction for in our last article. We assume you have the WSSLABXX library from that experience. Create a dynamic Web project – for example, MyWebServices – taking all the defaults. Right-click the project name in the Web perspective, select iSeries|Java on the left and Program Call Bean on the right, and click Next. Click Add Program, and enter Inquiry for the bean name, GETDATA for the program name, and WSSLABXX for the library name (**Figure 2**). Click OK.

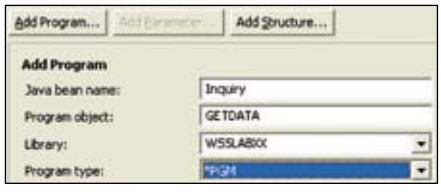

Figure 2: Specifying the program to call

This RPG program accepts one input parameter (customer ID) and updates two output parameters: an externally described data structure and a feedback field containing error information. Now we need to specify these parameters.

1. The first parameter is a seven-character customer ID that is input to the program. So enter CUSTNO for the parameter name, leave the data type as Character, enter 7 for the length, and select Input for the usage. Then click OK.

2. The second parameter is an externally described structure that is output from the program. Select the root (/) in the tree on the left, right-click it, and select Add Database Reference Structure. Expand your previously created iSeries connection, or if none exists, create one first (expand New Connection and fill in the information) and then expand it. Expand Work with Libraries and enter WSSLAB* for the library name. Click OK, and in the resulting list, expand library WSSLABXX and logical file CUSTOML3, and select the record format CUSTOM01 from the list. Click Add, and then Close.

3. You have added the structure to the tree, but now you need to define the parameter that uses the structure. Select Inquiry in the tree on the left, right-click it, and select Add Parameter. Enter CUSTDATA for the parameter name, and select Structure for the data type, CUSTOM01 for the structure name, and Output for the usage. Then click OK.

4. To define the final parameter, enter FEEDBACK for the parameter name, leave the data type as Character, specify 20 for the length and Output for the usage, and click OK.

At this point, you have defined the program that you want to call and each of its parameters, including whether each is read by the program (as input) or written to by the program (as output) or both (as both input and output). You can click Next.

On the "Create iSeries Program Call Java bean and PCML file" page, enter my.pkg for the package name. Ensure that the Services check box is selected, because this instructs the wizard to create the Java bean so that it is usable in a Web service. Click Next.

On the Configure Authentication page, enter your iSeries host name, your user ID, and your password. If you are using the Advanced Edition, you will notice that there is an option on this page to use a Java Connector Architecture (JCA) connector and JAAS for authentication. As you do more Web development, you might find

this a better way to do authentication. Click the Library list tab and add WSSLABXX to the initial library list. Click Finish to generate the Java bean.

In fact, a number of files are generated inside JavaResources. The key file is the name you gave the bean (Inquiry) followed by the word Services, or InquiryServices.java in our case. This is what you will feed into the Web services wizard. An input bean and an output bean are also generated; they group all the input-capable parameters and output-capable parameters into their own classes. Also, for each parameter that is a structure, a structure class is created. All of these classes are generated into the package you specified — my.pkg in our case.

You will notice that two other files were also generated: Inquiry.config and Inquiry.pcml. These files are needed by the generated Java at runtime. The Inquiry.pcml file records the program and parameter information, and the Inquiry.config file records the configuration information. The configuration file is new and quite interesting. This file identifies the iSeries host where the program is, the authentication information, and library list configuration information. The generated Java bean looks for this file at runtime to get this information. If it does not find the file, or for each value in the file that is not set, the bean looks in the web.xml file for this project, which is set using the iSeries Run Time Configuration Wizard (available on the toolbar in the Web perspective).

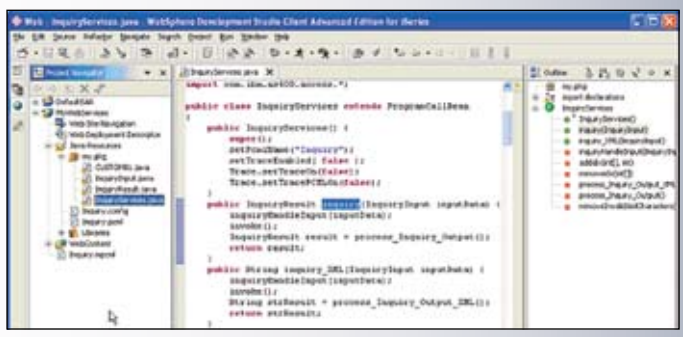

Figure 3: Files generated by Program Call Bean wizard

By externalizing this information, you can change it without changing your Java code. By using two files, you can specify information once to affect all the Web services in your project, and override those global settings for any Web service that you want to. The generated service bean also has methods that you can use to programmatically set this information.

Figure 3 shows all the generated files.

To use the generated classes, instantiate the InquiryInput class and call the setter methods in it, and then instantiate the InquiryServices class and call the inquiry method (with the same name as the bean) in it. This calls your program, passes the input parameters, and returns to you an instance of InquiryResult containing the output parameters, which you can query with the getter methods. There is also a method named inquiry_XML, which also takes an InquiryInput object, but returns the results as XML rather than an InquiryOutput object.

Here is a simple example of how you can write code to use the generated Java bean (minus error checking), if all you want is Java client code to invoke your RPG logic, compared to invoking it as a Web service (we will discuss this shortly):

```
import my.pkg.*;
public class TestMyService {
 public static void main(String[] args) {
 InquiryInput input = new InquiryInput();
 input.setCUSTNO("0010100");
 InquiryServices myService = new
 InquiryServices();
 InquiryResult output =
 myService.inquiry(input);
 // print the customer name from the
 // output datastructure...
 System.out.println("Resulting name: " +
 output.getCUSTDATA().getCUSTNA());
 }
}
```

The result of running this Java class is "Resulting name: Meridien Electronics Limited," plus some trace statements that the generated class emits.

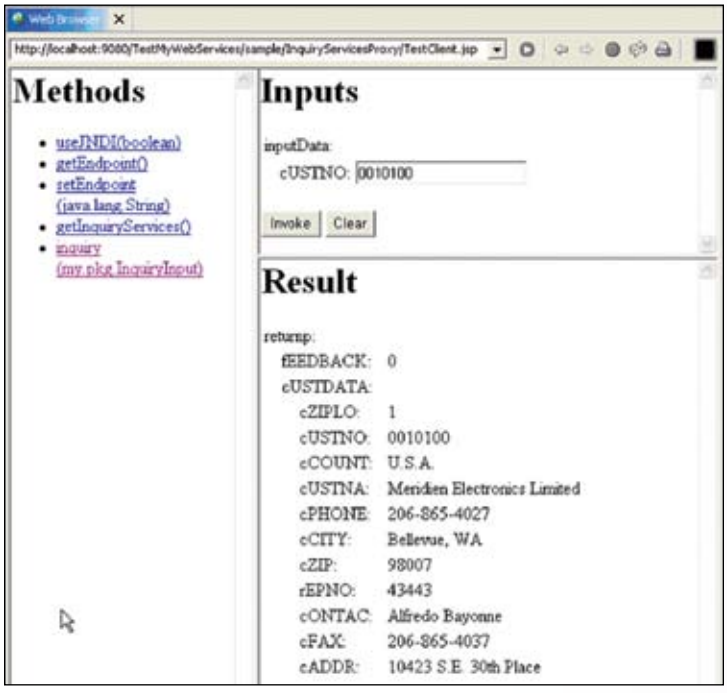

Figure 4: Generated test page for a Web service

Now we have a nice Java wrapper for an RPG or Cobol program. This wizard is easy to use (and even easier if you compile with the PGMINFO and INFOSTMF parameters to generate a program call markup language (PCML) file that you can subsequently import into the Program Call Bean wizard). What this does not give us, yet, is a Web service.

To create a Web service now, we simply follow the instructions in the cheat sheet (Help|Cheat Sheet), for creating a Web service from our Java bean, skipping the first three steps and then subsequently creating client Java code in another project. We will let you go through these steps on your own. On the Web Service Java Bean Identity page, be sure to select only the one method, which has the same name as the bean in our case — inquiry.

After going through the cheat sheet, your original project will be enhanced with generated classes to turn the Java bean into a Web service with a WSDL file, and you will have a new project containing client proxy code, generated from that WSDL file, and a sample servlet and Web page to test it (**Figure 4**). Your service project (the first one) can be deployed to WebSphere, wherever you decide to run it. You can probably ignore the second project, unless you want to send the client code to someone. Normally, however, they will instead get your WSDL file and generate the client code themselves. You can use the tools in WDSc to publish your WSDL file, either to a private or public UDDI registry, or to a WSIL file, which you then publish on your Web site.

Integrating with Integration Edition

Using WebSphere Studio Application Developer Integration Edition V5.1, you can create process flows with an easy-to-use-graphical designer, where each step in the flow is considered a "service." You can create a service from a Web Service, from a Java bean, or from a JCA connector. With the Program Call Bean wizard in WDSC, you can generate any of these from RPG or Cobol programs or ILE procedures. If you are using or considering using Integration Edition, you can import the output from the Program Call Bean wizard into a service project, using File|Import. To use the Java bean output, select the classes in the generated package (my.pkg in our case) and also the configuration and PCML files.

You also need to set the Java build path properties of your service project to include several external JAR files, where WDSC_HOME is where you installed WebSphere Development Studio Client on your system:

- {WDSC_HOME}\iseries\plugins\com.ibm.etools.iseries.webtools\lib\iwdtrt.jar
- {WDSC_HOME}\iseries\plugins\com.ibm.etools.iseries.webtools.ae\lib\idTokenRA.jar
- {WDSC_HOME}\iseries\plugins\com.ibm.etools.iseries.toolbox\runtime\jt400.jar

To create a service from your Java bean after it is imported, follow the steps in "Sample: Creating an enterprise service from a Java class (SOAP service)" in the Integration Edition documentation.

Toward an On-Demand World

That is all! It is very easy to create a Java bean wrapper from RPG or Cobol business logic, and to subsequently create a Web service from that wrapper, and even a Web servlet to test it. This is the component model for the future, and creating services are the first step in implementing Service Oriented Architecture (SOA), a key architectural goal in an on-demand world. RPG and Cobol programmers can participate fully in this world! TEG

Phil Coulthard *works at the IBM Toronto lab, where he is the lead architect for application development tools and languages on iSeries.*

George Farr *works at the IBM Toronto lab, where he is the technical development manager for the RPG and VisualAge for RPG languages, as well as the new RPG and Cobol tools in WDSc. Phil and George are frequent speakers at many conferences and user groups worldwide, and their books Java for RPG Programmers, 2nd Edition and Java for S/390 and AS/400 COBOL Programmers, are available from Penton and MC Press.*

This article was first published in iSeries News magazine.