Invoking web services using RPG proxy

Setting up and generating web service RPG proxy

By Nadir Amra, IBM Software Engineer
Published 01 June 2011

Abstract: This article demonstrates how you can invoke a web service from an RPG application using a new feature of the integrated web services client for ILE - the ability to generate an RPG proxy (or stub).

Introduction

For a few years now, RPG programmers have had the ability to invoke web services using C stubs emitted by the `wsdl2ws.sh` tool. However, the process of using the C stubs in an RPG application is cumbersome. A programmer is required to perform the following steps:

1. Map C structures to RPG structures.
2. Add RPG prototypes for service interface functions and operations.

Depending on the complexity of the WSDL file, step one can potentially be a nightmare.

And say you get through the steps described above, you still must deal with pointers and worry about memory leaks!

Well, RPG programmers rejoice. IBM has enhanced the `wsdl2ws.sh` tool so that a user can generate RPG stubs. This article provides a quick example of how to create a web services client application written in ILE RPG using the RPG stub code generated by the `wsdl2ws.sh` tool.

Prerequisites

Software

Table 1 lists the PTFs that are needed for each of the supported releases of the IBM i operating system.

Table 1. Software prerequisites

IBM i release	PTFs
i 7.1	SI42767
i 6.1 i 5.4	SI42766 SI42765

Assumptions

Since the example relies on the `ConvertTemp` service, you may want to create a web services server. You do not need to deploy a web service; we will be using the sample web service that gets deployed automatically when a server is created.

Note: The installation directory for web services client for ILE is `/QIBM/ProdData/OS/WebServices/V1/client`. In this article the installation directory is shown as `<install_dir>` .

Creating an RPG application that uses RPG stub code

To develop a Web services client application, the following steps should be followed:

1. Generate the client Web service stubs using the `wsdl2ws.sh` command.
2. Build the client application.
3. Run the client application.

The following sections will discuss each of these steps. For illustrative purposes we will be using the sample code that is shipped as part of the product in directories `<install_dir>/samples/ConvertTemp`. The files we will be using are listed in Table 2 and will be referenced throughout this article.

Table 2. Files used in example

File	Description
ConvertTemp.wsdl	WSDL file.
ConvertTempClientWSDL2RPG.RPGLE	Client implementation code written in RPG.

The WSDL file shown in Listing 1 is for a temperature conversion service, which converts a temperature from Fahrenheit to Celsius. Two operations are defined:

- `converttemp`
- `converttemp_XML`

The `converttemp` operation returns the temperature in Celsius, while the `converttemp_XML` operation returns the results as an XML document. Note that Listing 1 provides a partial view of the WSDL for this service, showing only the portions involved in the `converttemp` operation, which is the operation that we will be using in the RPG application.

Listing 1. WSDL definition

```
<?xml version="1.0" encoding="UTF-8"?>
<wsdl:definitions xmlns:wsdl=http://schemas.xmlsoap.org/wsdl/
                  xmlns:axis2=http://converttemp.wsbeans.iseries
                  xmlns:ns1="http://org.apache.axis2/xsd"
                  xmlns:wsaw="http://www.w3.org/2006/05/addressing/wsdl"
                  xmlns:http="http://schemas.xmlsoap.org/wsdl/http/"
                  xmlns:ns0="http://converttemp.wsbeans.iseries/xsd"
                  xmlns:xs="http://www.w3.org/2001/XMLSchema"
                  xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
                  xmlns:mime="http://schemas.xmlsoap.org/wsdl/mime/"
                  xmlns:soap12="http://schemas.xmlsoap.org/wsdl/soap12/"
                  targetNamespace="http://converttemp.wsbeans.iseries">
  <wsdl:types>
    <xs:schema xmlns:ns="http://converttemp.wsbeans.iseries/xsd"
               attributeFormDefault="qualified" elementFormDefault="qualified"
               targetNamespace="http://converttemp.wsbeans.iseries/xsd">
      ...
      <xs:complexType name="CONVERTTEMPInput">
        <xs:sequence>
          <xs:element minOccurs="0" name="_TEMPIN" nillable="true" type="xs:string"/>
        </xs:sequence>
      </xs:complexType>
      ...
```

```
    <xs:element name="converttemp">
      <xs:complexType>
        <xs:sequence>
          <xs:element minOccurs="0" name="param0" nillable="true"
                      type="ns:CONVERTTEMPInput"/>
        </xs:sequence>
      </xs:complexType>
    </xs:element>
    <xs:element name="converttempResponse">
      <xs:complexType>
        <xs:sequence>
          <xs:element minOccurs="0" name="return" nillable="true"
                      type="ns:CONVERTTEMPResult"/>
        </xs:sequence>
      </xs:complexType>
    </xs:element>
    <xs:complexType name="CONVERTTEMPResult">
      <xs:sequence>
        <xs:element minOccurs="0" name="_TEMPOUT" nillable="true"
                    type="xs:string"/>
      </xs:sequence>
    </xs:complexType>
  </xs:schema>
</wsdl:types>
...
<wsdl:message name="converttempRequest">
  <wsdl:part name="parameters" element="ns0:converttemp"/>
</wsdl:message>
<wsdl:message name="converttempResponse">
  <wsdl:part name="parameters" element="ns0:converttempResponse"/>
</wsdl:message>
<wsdl:portType name="ConvertTempPortType">
  <wsdl:operation name="converttemp_XML">
    <wsdl:input message="axis2:converttemp_XMLRequest"
                wsaw:Action="urn:converttemp_XML"/>
    <wsdl:output message="axis2:converttemp_XMLResponse"
                 wsaw:Action="urn:converttemp_XMLResponse"/>
  </wsdl:operation>
  <wsdl:operation name="converttemp">
    <wsdl:input message="axis2:converttempRequest" wsaw:Action="urn:converttemp"/>
    <wsdl:output message="axis2:converttempResponse"
                 wsaw:Action="urn:converttempResponse"/>
  </wsdl:operation>
</wsdl:portType>
<wsdl:binding name="ConvertTempSOAP11Binding" type="axis2:ConvertTempPortType">
  <soap:binding transport="http://schemas.xmlsoap.org/soap/http" style="document"/>
  <wsdl:operation name="converttemp_XML">
    <soap:operation soapAction="urn:converttemp_XML" style="document"/>
    <wsdl:input>
      <soap:body use="literal"/>
    </wsdl:input>
    <wsdl:output>
      <soap:body use="literal"/>
    </wsdl:output>
  </wsdl:operation>
  <wsdl:operation name="converttemp">
    <soap:operation soapAction="urn:converttemp" style="document"/>
    <wsdl:input>
      <soap:body use="literal"/>
    </wsdl:input>
    <wsdl:output>
      <soap:body use="literal"/>
    </wsdl:output>
  </wsdl:operation>
</wsdl:binding>
<wsdl:service name="ConvertTemp">
  <wsdl:port name="ConvertTempSOAP11port_http"
             binding="axis2:ConvertTempSOAP11Binding">
    <soap:address location="http://localhost:10022/web/services/ConvertTemp"/>
  </wsdl:port>
</wsdl:service>
```

```
</wsdl:definitions>
```

Step 1. Create the client stub code

Before you can create a web service client application, you must first generate the RPG client stub using the `wsdl2ws.sh` tool. The `wsdl2ws.sh` tool uses the WSDL file that is passed to it, and any associated XSD files referenced in the WSDL file, to create client stubs.

To generate the client stub from the WSDL source file, complete the following steps.

1. Create a library called CVTTEMP in which the program objects will be stored by issuing the CL command `CRTLIB` from the CL command line as follows:

   ```
   CRTLIB CVTTEMP
   ```
2. Start a Qshell session by issuing the `QSH` CL command from the CL command line.
3. Run the `wsdl2ws.sh` tool to generate the client RPG stub code as shown in following example:

   ```
   <install_dir>/bin/wsdl2ws.sh -o/convtemp/RPG -lrpg
   -s/qsys.lib/cvttemp.lib/wsrpg.srvpgm
   <install_dir>/samples/ConvertTemp/ConvertTemp.wsdl
   ```

If you examine the command, you see that we are indicating to the `wsdl2ws.sh` tool that RPG stub code should be generated and stored in directory `/convtemp/RPG`, and that a service program, `/qsys.lib/cvttemp.lib/wsrpg.srvpgm`, should be created using the generated stub code.

The files generated by the `wsdl2ws.sh` tool is shown in Listing 2:

Listing 2. Generated stub files

```
ConvertTempPortType_util.rpgle
ConvertTempPortType_util.rpgleinc
ConvertTempPortType_xsdtypes.rpgleinc
ConvertTempPortType.c
ConvertTempPortType.cl
ConvertTempPortType.h
ConvertTempPortType.rpgle
ConvertTempPortType.rpgleinc
CONVERTTEMPInput.c
CONVERTTEMPInput.h
CONVERTTEMPResult.c
CONVERTTEMPResult.h
```

Note that in addition to the RPG stub code being generated, C stub code is also generated since the RPG stub code is built on top of the C stub code. For all practical purposes, the C stub code can be ignored.

Here is a description of each RPG file that is generated:

- ConvertTempPortType_util.rpgle – RPG utility routines.
- ConvertTempPortType_util.rpgleinc – RPG utility routines include.
- ConvertTempPortType_xsdtypes.rpgleinc – standard data types include.
- ConvertTempPortType.rpgle – RPG web service implementation code.
- ConvertTempPortType.rpgleinc – RPG web service include.

From an RPG programmer perspective, the only files you would need to look at are the `ConvertTempPortType.rpgleinc` and `ConvertTempPortType_xsdtypes.rpgleinc` files. The `ConvertTempPortType.rpgleinc` file defines the RPG functions to create and destroy a web service interface object, in addition to the web service operations. Also defined in the file are any types that are needed by the web service operations. The `ConvertTempPortType_xsdtypes.rpgleinc` file defines all the primitive types and various constants.

Listing 3 shows the `ConvertTempPortType.rpgleinc` file.

Listing 3. ConvertTempPortType.rpgleinc file

```
 * ********************************************************************
 * ********************************************************************
 * D A T A   T Y P E S
 * ********************************************************************
 * ********************************************************************
D CONVERTTEMPInput_t...
D                 DS                  qualified based(Template)
D isNil_CONVERTTEMPInput_t...
D                                1n
D TEMPIN                              likeds(xsd_string)

D CONVERTTEMPResult_t...
D                 DS                  qualified based(Template)
D isNil_CONVERTTEMPResult_t...
D                                1n
D TEMPOUT                             likeds(xsd_string)


 * ********************************************************************
 * ********************************************************************
 * P R O T O T Y P E S
 * ********************************************************************
 * ********************************************************************


 * ********************************************************************
 * WEB SERVICE CLIENT STUB PROTOTYPES
 * ********************************************************************

 * ************************************************************
 * RPG Call : stub_create_ConvertTempPortType
 * ************************************************************
D stub_create_ConvertTempPortType...
D                 PR             1N   extproc('stub_create_ConvertTempPo+
D                                     rtType@')
D this                                likeds(This_t)

 * ************************************************************
 * RPG Call : stub_destroy_ConvertTempPortType
 * ************************************************************
D stub_destroy_ConvertTempPortType...
```

```
D                 PR             1N   extproc('stub_destroy_ConvertTempP+
D                                     ortType@')
D this                                likeds(This_t)

 * ****************************************************************
 * WEB SERVICE OPERATION PROTOTYPES
 * ****************************************************************

 * ************************************************************
 * RPG call : stub_op_converttemp_XML
 * ************************************************************
D stub_op_converttemp_XML...
D                 PR             1N   extproc('converttemp_XML@')
D this                                likeds(This_t)
D Value0                              likeds(CONVERTTEMPInput_t)
D out                                 likeds(xsd_string)

 * ************************************************************
 * RPG call : stub_op_converttemp
 * ************************************************************
D stub_op_converttemp...
D                 PR             1N   extproc('converttemp@')
D this                                likeds(This_t)
D Value0                              likeds(CONVERTTEMPInput_t)
D out                                 likeds(CONVERTTEMPResult_t)
```

As you examine Listing 3, note the following points:

- `ConvertTemp.wsdl` has only one service called `ConvertTemp`.
- The service only has one port type called `ConvertTempPortType`.
- The `ConvertTempPortType` port type has two operations called `converttemp` and `converttemp_XML`. The corresponding RPG stub operations (defined in the generated `ConvertTempPortType.rpgleinc` include file) are `stub_op_converttemp()` and `stub_op_converttemp_XML()`.
- The web service is called `ConvertTempPortType`. So to get an instance of the web service you would call the `stub_create_ConvertTempPortType` () function. The handle that is returned by the function should then be used when calling the web service operation. To destroy the web service instance, you would call the `stub_destroy_ConvertTempPortType` () function. (Both these functions are defined in the generated `ConvertTempPortType.rpgleinc` include file.)

Finally, there is also the file `ConvertTempPortType.cl` that is also generated. This file is a CL source file that has the CL commands needed to recreate the service program containing the stub code. You can copy this source file to a source physical file and create a CL program.

Step 2. Build the client application

After the client stubs have been generated, the stubs can be used to create a web service client application.

Listing 4 shows the RPG client application using the created RPG stubs to invoke the `converttemp` web service operation.

Listing 4. RPG application using generated RPG stubs

```
h DFTNAME(CVTTEMP)
  *
  /copy ConvertTempPortType.rpgleinc

 d OutputText      s             50
 d WsStub          ds                   likeds(This_t)
 d Input           ds                   likeds(CONVERTTEMPInput_t)
 d Result          ds                   likeds(CONVERTTEMPResult_t)

  *---------------------------------------------------------------------
  * Program entry point. The input parameter is a character field
  * representing the temperature in Fahrenheit.
  *---------------------------------------------------------------------
 C     *ENTRY        PLIST
 C                   PARM                    TEMPIN           32
  *---------------------------------------------------------------------
  * Web service logic. The code will attempt to invoke a Web
  * service in order to convert temperature in Fahrenheit to Celsius
  * and then display the results.
  *---------------------------------------------------------------------

  /free
     // Get a Web service stub. The host and port for the endpoint may need
     // to be changed to match host and port of Web service. Or you can pass
     // blanks and endpoint in the WSDL file will be used.
     clear WsStub;
     WsStub.endpoint = 'http://localhost:10000/web/services/ConvertTemp';

     clear input;
     Input.TEMPIN.value = %trim(TEMPIN);

     if (stub_create_ConvertTempPortType(WsStub) = *ON);

       // Invoke the ConvertTemp Web service operation.
       if (stub_op_ConvertTemp(WsStub:Input:Result) = *ON);
          OutputText = Input.TEMPIN.value + ' Fahrenheit is '
                     + Result.TEMPOUT.value + ' Celsius.';
       else;
          OutputText = WsStub.excString;
       endif;

       // Display results.
       dsply  OutputText;

       // Destroy Web service stubs.
       stub_destroy_ConvertTempPortType(WsStub);
     endif;

     *INLR=*ON;
  /end-free
```

To build the client application, complete the following steps.

1. Change the current working directory to the location of the RPG stub code. Issue the following command from the CL command line:

   ```
   cd '/convtemp/RPG'
   ```
2. Copy the sample RPG code the uses the generated stub code from the product samples directory to the current working directory by issuing the following command from the CL command line:

```
COPY OBJ('<install_dir>/samples/ConvertTemp/ConvertTempClientWSDL2RPG.RPGLE')
TODIR('/convtemp/RPG')
```

3. If you have created a web services server then change the server name and port number of the endpoint in the file copied in the previous step to match the server and port of the `ConvertTemp` web service installed in the server. The endpoint in the file is:

```
'http://localhost:10000/web/services/ConvertTemp'
```

4. Build the client application by using the following commands from the CL command line:

```
CRTRPGMOD MODULE(CVTTEMP/CNVRTTEMP)
SRCSTMF('/convtemp/RPG/ConvertTempClientWSDL2RPG.rpgle')

CRTPGM PGM(CVTTEMP/CNVRTTEMP)
MODULE(CVTTEMP/CNVRTTEMP)
BNDSRVPGM(QSYSDIR/QAXIS10CC CVTTEMP/WSRPG)
```

Step 3. Run the client application

When you have finished coding and building the web service client application, run and test the client application. Run the client application by issuing the following from the CL command line (in this example we want to find out what 5 degrees Fahrenheit is in Celsius):

```
CALL CVTTEMP/CNVRTTEMP  '5'
```

Check that the client application shows the Celsius representation of the Fahrenheit value used above. The example screen shot below in Figure 1 shows the client application run from the command line (note that to see the result you need to press "F10=Include detailed messages" function key):

Figure 1. Calling client application

Summary

The ability to generate RPG web service stubs is a quantum leap from when RPG programmers had to figure out how to use the C web service stubs. So, what are you waiting for?

Resources

- For everything about the integrated Web services support on IBM i see the product web page.