Transform protocols and route messages through an ESB

Modify your Web service using WebSphere service integration technologies

Skill Level: Advanced

Tendai Chinoda (tendai@us.ibm.com)
Advisory Software Engineer, WebSphere Competency Center
IBM

13 Sep 2005

Learn how to deploy a Simple Object Access Protocol/Java™ Messaging Service (SOAP/JMS) stateless session bean Web service implementation in IBM WebSphere® Application Server Version 6.0 (Application Server) and provide access to SOAP/JMS using configured Service Integration Bus (SIB) messaging resources. IBM Advisory Software Engineer Tendai Chinoda also demonstrates how to configure Service Integration Bus Web services (SIBWS) inbound and outbound services to provide protocol transformation and routing of SOAP/HTTP service requests to the same SOAP/JMS target service.

Section 1. Before you start

Learn what you can get from this series and how to get the most from this tutorial.

About this series

In a previous tutorial Develop a multi-transport, accessible Web service in Rational Application Developer Version 6.0, we demonstrated the development of a multi-transport, accessible Web service implementation in Rational® Application Developer Version 6.0. The implementation was then deployed and tested in WebSphere® Application Server Version 6.0 (Application Server) with SOAP (Simple Object Access Protocol)/Java™ Messaging Service (JMS) and SOAP/HTTP access to the Web service implementation. In this tutorial, you'll modify the solution
provided in the above referenced tutorial by removing the need for a separate Web router module for SOAP/HTTP access and demonstrate the routing and protocol transformation capabilities of Service Integration technologies in Application Server Version 6.0.

About this tutorial

In this tutorial, the accessibility of your Web service implementation using SOAP/JMS leverages the Service Integration Bus (SIB) messaging resources to direct SOAP/JMS services requests to the Bank application. The SOAP/HTTP service requests utilize the Service Integration Bus Web services (SIBWS) routing and protocol transformation capabilities (inbound or outbound service configurations) to direct SOAP/HTTP service requests to the target Bank application Web service.

In summary, the scope of this tutorial is to:

- Demonstrate the messaging capabilities of the SIB.
- Enable forwarding of SOAP/JMS requests to a target endpoint.
- Demonstrate the routing and protocol transformation capabilities of the SIBWS.
- Enable routing and protocol transformation of SOAP/HTTP requests to a SOAP/JMS target endpoint.
- Deploy and test the described solution in Application Server Version 6.0 with SOAP/JMS and SOAP/HTTP access to the Web service implementation, that is:
  - SOAP/JMS transport: using the SIB (messaging resources)
  - SOAP/HTTP transport: using the SIBWS (inbound or outbound services)

Prerequisites

To run the examples in this tutorial, you need Application Server Version 6.0 Network Deployment. The tutorial assumes knowledge of Java 2 Platform, Enterprise Edition (J2EE) 1.4 and Web service technologies.

Section 2. Background: Overview of the Enterprise Service Bus

An Enterprise Service Bus (ESB) is an architectural construct and middleware
infrastructure component that supports Service-Oriented Architectures (SOA). Within the WebSphere Business Integration (WBI) Reference Architecture, the ESB is positioned to provide interconnectivity services. Run-time implementations of an ESB might be realized using the following IBM products:

- WebSphere MQ
- WebSphere Message Broker/Event Broker
- Web Services Gateway
- Application Server Version 6.0

When realizing an ESB implementation with Application Server Version 6.0, the service integration technologies play a primary role in facilitating application connectivity. In particular, the Application Server Version 6.0 SIB for messaging and the SIBWS for routing and protocol transformation. SIBWS also provides other interconnectivity services of the ESB, such as gateway and security services.

A detailed discussion on the WBI Reference Architecture, ESB implementations, and the Application Server Version 6.0 service integration technologies is beyond the scope of this tutorial (see the Resources).

Section 3. About the simple Bank application

The example used in this tutorial is a simplified banking J2EE application (BankApp.ear), consisting of an EJB module (BankAppEJB.jar) with a stateless session bean Web service implementation. The stateless session bean serves as a session facade to the accounting application. The J2EE application also consists of a second EJB module (BankAppEJBJMSRouter.jar) for routing the SOAP/JMS request to the Bank application Web service implementation.

The BankAppEJB module consists of four main classes:

- Account.java: Java bean representation of an account record.
- Accounts.java: Interface for access to and manipulation of account records.
- AccountsImpl.java: Implementation of the Accounts interface, allowing the creation, updating, and retrieval of static account objects stored in a static hashtable called existingAccounts. The existingAccounts variable is prepopulated with four default accounts.
- AccountsSessionFacadeBean: Stateless session bean facade to the Accounts interface.
The BankAppEJBJMSRouter module consists of a single message-driven bean (MDB), WebServicesJMSRouter. The implementation class of the bean, com.ibm.ws.webservices.engine.transport.jms.JMSListenerMDB, is provided by Application Server Version 6.0. The MDB processes incoming messages, invokes the Web service, and sends a reply message where necessary.

See Develop a multi-transport, accessible Web service in Rational Application Developer Version 6.0 for the steps to create a stateless session bean Web services implementation and the corresponding SOAP/JMS EJB Router module.

The final deployed solution, shown in Figure 1, exposes the banking application as an accessible Web service using SOAP/JMS and SOAP/HTTP.

**Figure 1. Proposed solution**
Section 4. Implement the solution

To begin, download the **BankApp.ear** file (see Downloads) and follow the steps as outlined below:

- Create an Application Server Version 6.0 profile
- Configure the Application Server Version 6.0 SIB and messaging resources
- Deploy the Bank application Web service implementation
- Complete the installation of SIBWS
- Configure the Application Server Version 6.0 SIBWS inbound and outbound services
- Test the Web service client for both transports

Section 5. Implement the solution -- Create an Application Server Version 6.0 profile

Create an Application Server Version 6.0 profile

For the purposes of this tutorial, you'll create a new Application Server profile. See the Resources section for more information on Application Server Version 6.0 profiles.

This section shows how to create a profile using the Application Server Profile Creator wizard. In order to do this:

1. Run the `pctWindows` command from `<was_home>/bin/ProfileCreator`.

2. Click **Next** on the Welcome page.

3. On the Profile Type Selection page:
   - Select the **Create an Application Server** profile.
   - Click **Next**.
4. On the Profile Name page:
   - Enter **SimpleBankAppServer** as the Profile name.
   - Click **Next**.
   - Note: It is important that, where specified, you use the default values outlined in the following steps, as the port numbers used here correspond to the sample application provided in this tutorial. However, be sure to uncheck **Make this profile the default** if you do not want this profile to be used as the default profile.

5. Leave the **Profile Directory** as the default value provided and click **Next**.

6. On the Node and Host Names page:
   - Enter an appropriate **Node Name** and set the **Host Name** to the host name for your machine. (In most cases, the defaults are acceptable.)

7. On the Port value assignments page:
   - Select the **Unique Port Numbers**, as shown in Figure 2. (Use of the default values is necessary for successful completion of this tutorial.)
   - Click **Next**.

**Figure 2. SimpleBankAppServer profile port assignments**
1. On the Windows service definition:
   - Unselect the Run the Application Server Process as a Windows service. You'll start and stop the server from the workspace.
   - Click Next.

2. Click Next on the Profile Summary page to start the creation of the new profile. (This process might take several minutes.)

3. On the following screen, uncheck Launch the First steps console and click Finish.

This completes the creation of the Application Server Version 6.0 profile you'll use to deploy and test your solution.
Section 6. Implement the solution -- Configure the Application Server Version 6.0 SIB and messaging resources

Configure the Application Server Version 6.0 SIB and messaging resources

As introduced earlier in the tutorial, the ESB is positioned to provide connectivity services within the WBI Reference Architecture. The Application Server Version 6.0 SIB facilitates messaging within an ESB implementation.

In this section, you'll learn how to create the SIB messaging resources needed to enable SOAP/JMS accessibility to your Bank application Web service implementation.

In summary, you'll perform the following tasks:

- Create the SIB and add the SimpleBankAppServer as a bus member.
- Create the SIB Queue destination.
- Create the JMS resources.
- Create the JMS queue connection factory resources for use by the application.
  - Create the JMS queue resource definitions used by the MDB.
  - Associate JMS queue with SIB destination.
  - Create the JMS activation specification resource used by the MDB.
  - Associate JMS activation specification with JMS queue.

The following steps show how to create and configure the SIB messaging resources for the Bank application Web service implementation.


   - From the command line, enter:
     ```bash
     <was_home>/profiles/SimpleBankAppServer/bin/startServer.bat|sh server1
     ```

   Note: If the server starts with errors, they need to be resolved prior to continuing. Check the server logs for more details:
If the cause of the failure is not obvious from the information in the logs, then check the WebSphere documentation for more information on troubleshooting failed server startups.

2. Access the Administrative console for the server

   • From a browser, access the Administrative console URL http://localhost:9060/ibm/console.
   • Enter a **userid** and click on **Login**.

3. Create the SIB

   • Expand **Service Integration** and click on **Buses** and then **New** to create a new SIB.
   • Enter **SimpleBankAppSIBus** as the name of the new bus.
   • Click on **Apply**. This will enable the ability to enter the additional properties. The resulting new bus configuration is shown in Figure 3.

**Figure 3. SIB configuration settings**
4. Add SimpleBankAppServer as a bus member on the SimpleBankAppSIBus

- In the SIB configuration frame, click on **Additional Properties > Bus members**.
- Click **Add**.
- Verify that the Server entry points to your_node:server1.
- Click **Next** and then **Finish**.
• Save configuration changes.

Note: By adding a server as a bus member, a messaging engine is automatically created on that server. The messaging engine will be started on restart of the server.

5. Create a SIB queue destination

• Select Service Integration > Buses and click on SimpleBankAppSIBus.

• In the SIB configuration frame, click on Additional Properties > Destinations and then New to create a new destination.

• In the Create new destination frame, ensure that Queue is selected and click on Next.

• Enter SIMPLEBANKAPP.INCOMING.QUEUE as the Destination Identifier.

• Click Next.

• Click Next on the following page and then Finish.

• Save configuration changes.

The SIB queue destination settings are shown in Figure 4.

Figure 4. Create SIB queue destination settings

6. Create the JMS queue connection factory resources

• Expand Resources > JMS Providers and click on Default Messaging.

• In the Default Messaging Provider frame under Connection Factories, click on JMS queue connection factory and then New to create a new Queue connection factory.
- Enter SimpleBankAppIncomingQCF as the QCF Name.
- Enter jms/SimpleBankAppIncomingQCF as the JNDI Name.
- In the Bus name pull down, select SimpleBankAppSIBus.
- Accept all other defaults and click OK.

The resulting SimpleBankAppIncomingQCF JMS queue connection factory configuration settings are shown on the right in Figure 5.

- Click New to create the second Connection Factory.
- Enter WebServicesReplyQCF as the QCF Name.
- Enter jms/WebServicesReplyQCF as the JNDI Name.
- In the Bus name pull down, select SimpleBankAppSIBus.
- Accept all other defaults and click OK.
- Save configuration changes.

Note: The JNDI name, “jms/WebServicesReplyQCF”, is reserved for the JMS Web services reply queue connection factory. A temporary outgoing reply queue is implicitly created and then destroyed whenever a response is required.

7. Create the JMS queue resource and associate resource to the SIB queue destination

- Expand Resources > JMS Providers and click on Default Messaging.
- In the Default Messaging Provider frame under Destinations, click on JMS Queue and then New to create a new queue.
- Enter SimpleBankAppQueue as the Queue name.
- Enter jms/SimpleBankAppQueue as the JNDI Name.
- In the Bus Name pull down, select SimpleBankAppSIBus.
- In the Queue Name pull down, select SIMPLEBANKAPP.INCOMING.QUEUE.
- Click OK.
- Save configuration changes.

The resulting JMS queue configuration settings are shown on the left in Figure 5.
8. Create the JMS activation specification

- Expand Resources and click on Resource Adapters.
- In the Resource Adapters frame, click on SIB JMS Resource Adapter.
- In the SIB JMS Resource Adapter frame under Additional Properties, click on J2C Activation specifications and then New to create a new activation specification.
- Enter SimpleBankAppActivationSpec as the Name.
- Enter jms/SimpleBankAppActivationSpec as the JNDI Name.
- In the Destination JNDI name, select Select an existing destination JNDI name.
- In the Administered Objects pull down, select jms/SimpleBankAppQueue.
- Click on Apply.

Figure 6 shows the resulting JMS activation specification configuration settings.

Figure 6. JMS activation specification configuration settings
9. Associate the activation specification to the SimpleBankAppSIBus

- In the **J2C activation specification** configuration frame for the **SimpleBankAppActivationSpec**, click on **Additional Properties > J2C activation specification custom properties**.
- Click on **busName** and enter **SimpleBankAppSIBus** as the value for the bus name.
- Click **OK**.
- Save configuration changes.

10. Verify that configuration changes were successful

- Log out from the console and close the Administration console browser.
- Stop the server. From the command line, enter: 
  ```
  <was_home>/profiles/SimpleBankAppServer/bin/stopServer.bat|sh server1.
  ```
- Restart the server. From the command line, enter: 
  ```
  <was_home>/profiles/SimpleBankAppServer/bin/startServer.bat|sh server1.
  ```
• Check the server startup messages for errors.
• From a browser, access the Administrative console (http://localhost:9060.ibm/console).
• Enter a userid and click on Login.
• Expand Servers and click on Application Servers.
• Click on server1.
  • Under Server messaging click on Messaging Engines.

If the status of <your_node>.server1-SimpleBankAppSIBus is started, then you have successfully configured the Application Server Version 6.0 for SOAP/JMS. If the server starts with errors, they need to be resolved prior to continuing. Check the server logs for more details and verify your configuration is as per the steps outlined above.

    <was_home>/profiles/SimpleBankAppServer/logs/server1/SystemOut.log

    <was_home>/profiles/SimpleBankAppServer/logs/server1/SystemErr.log

If the cause of the failure is not obvious from the information in the logs, then check the WebSphere documentation for more information on troubleshooting failed server startups.

Section 7. Implement the solution -- Deploy the Bank application Web service implementation

Deploy the Bank application Web service implementation

In this section, you'll deploy the Bank application Web service implementation.

1. Access the Administrative console.
   • The Application Server should have been started with no errors in the console view. Right-click on the Server.
   • From a browser, access the Administrative console (http://localhost:9060.ibm/console).
   • Enter a userid and click on Login.
2. Download the BankApp.ear file to a location in your environment (see the Downloads section).

3. Expand Applications and click on Install New Application.

4. Click on Browse and locate the downloaded BankApp.ear file and click Next.

5. Click Next.

6. In the Install New Application frame, ensure that Deploy enterprise beans is selected (default) and click Next.

7. Click Next on Steps 2 and 3.

8. On Step 4, "Provide listener bindings for MDB", ensure that the binding Activation specification is selected and the JNDI name is set to "jms/SimpleBankAppActivationSpec" for the WebServicesJMSRouter MDB. Click Next.

9. Click on Step 9 to proceed, as the defaults for the remaining steps are sufficient, and click Finish.

10. Verify that there are no error messages from the install.

11. Click on Save Master configuration and save configuration changes.

12. Start the Application.

   • Expand Applications and click on Enterprise Applications.
   • Select the BankAppEAR and click on Start.
   • Check the server logs to ensure the application started without errors.

Section 8. Implement the solution -- Complete installation of SIBWS

Complete installation of SIBWS

In order to use the SIBWS technologies to provide SOAP/HTTP access to the Bank application Web service implementation, you'll need to install the necessary SIBWS components. For the purposes of this tutorial, you'll need to install the following components:
- **SDO Repository** -- used by the SIBWS to store WSDL definitions, such as schema types.
- **SIBWS Resource Adapter** -- used by the SIBWS to invoke Web services at outbound ports.
- **SIBWS application and endpoint listener**
  - The SIBWS application needs to be installed. It enables configuration and access of Web services through the SIB.
  - Endpoint listeners are the receivers of requests for inbound services. For the purposes of this tutorial, you'll need to install a SOAP/HTTP listener for your Bank application Web service implementation. You'll not need a SOAP/JMS endpoint listener, as the SOAP/JMS requests will be handled by the SIB and not need to be routed using the SIBWS inbound/outbound services.

1. **Install the SDO repository**

   - From the command line, enter the following:
     - `cd <was_home>/profiles/SimpleBankAppServer/bin/
     - `wsadmin.bat|sh -conntype SOAP -port 8880 -f <was_home>/bin/installSdoRepository.jacl –createDb`
   - Ensure the following messages are displayed:

     ...  
     ...  
     CWSJO0022I: Saving configuration.  
     CWSJO0023I: SDO repository installation completed successfully.

2. **Install the SIBWS resource adapter**

   - From the command line, enter the following:
     - `cd <was_home>/profiles/SimpleBankAppServer/bin/
     - `wsadmin.bat|sh -f <was_home>/util/sibwsInstall.jacl INSTALL_RA -installRoot <was_home> -nodeName <node_name>`
   - Ensure the following messages are displayed:

     ...  
     CWSWS5056I: Installing resource adapter : SIB_RA  
     CWSWS5059I: Creating activation specification : SIBWS_OUTBOUND_MDB  
     CWSWS5058I: The resource adapter has started successfully : SIB_RA  
     CWSWS5060I: Saving configuration

3. **Install the SIBWS and endpoint listener applications**
• From the command line, enter the following to install the SIBWS application:
  
  • cd <was_home>/profiles/SimpleBankAppServer/bin/
  • wsadmin.bat|sh -f <was_home>/util/sibwsInstall.jacl
    INSTALL -installRoot<was_home> -serverName server1
    -nodeName <node_name>
  
  • Ensure the following messages are displayed:

<table>
<thead>
<tr>
<th>Message</th>
<th>Details</th>
</tr>
</thead>
<tbody>
<tr>
<td>CWSWS5050I</td>
<td>Installing application : sibws.&lt;node_name&gt;.server1</td>
</tr>
<tr>
<td>CWSWS5051I</td>
<td>The application is being started : sibws.&lt;node_name&gt;.server1 server1 &lt;node_name&gt;</td>
</tr>
<tr>
<td>CWSWS5055I</td>
<td>The application has started : sibws.&lt;node_name&gt;.server1</td>
</tr>
</tbody>
</table>

• To install the endpoint listener for SOAP over HTTP bindings from the command line, enter the following:

  • wsadmin.bat|sh -f <was_home>/util/sibwsInstall.jacl
    INSTALL_HTTP -installRoot <was_home> -serverName server1 -nodeName <node_name>

  • Ensure the following messages are displayed:

<table>
<thead>
<tr>
<th>Message</th>
<th>Details</th>
</tr>
</thead>
<tbody>
<tr>
<td>CWSWS5050I</td>
<td>Installing application : sibwshhttp1.&lt;node_name&gt;.server1</td>
</tr>
<tr>
<td>CWSWS5051I</td>
<td>The application is being started : sibwshhttp1.&lt;node_name&gt;.server1 server1 &lt;node_name&gt;</td>
</tr>
<tr>
<td>CWSWS5055I</td>
<td>The application has started : sibwshhttp1.&lt;node_name&gt;.server1</td>
</tr>
</tbody>
</table>

If you successfully install and start all applications, the installation of the SIBWS applications needed for this tutorial is complete.
Section 9. Implement the solution -- Configure the Application Server Version 6.0 SIBWS inbound and outbound services

Configure the Application Server Version 6.0 SIBWS inbound and outbound services

As introduced earlier in the tutorial, the ESB is positioned to provide connectivity services within the WBI Reference Architecture. The Application Server Version 6.0 SIBWS facilitates routing and protocol transformation of Web service requests within an ESB implementation.

In this section, you'll learn how to configure the SIBWS inbound and outbound services needed to enable routing and protocol transformation of SOAP/HTTP requests to your SOAP/JMS accessible Bank application Web service implementation.

In summary, you'll perform the following tasks:

- Configure the endpoint listener used for SOAP/HTTP requests.
- Create and configure outbound services used for protocol transformation.
- Create and configure inbound services used to route SOAP/HTTP requests to the outbound service.

1. Configure the endpoint listener

   - From the Administrative console, expand **Servers > Application servers** and click on **server1**.
   - Under **Additional Properties**, click on **Endpoint Listeners** and click **New**.
   - Enter **SOAPHTTPChannel1** as the **Name**.
   - Enter **http://<your_host>:9080/wsgwsoaphttp1** as the **URL root**.
   - Enter **http://<your_host>:9080/sibws** as the **WSDL serving HTTP URL**.
   - Click **Apply**.
   - Under **Additional Properties**, click on **Connection Properties**.
   - Click **New**, select **SimpleBankAppSIBus**, and click **OK**.

See Figure 7 for the resulting endpoint listener configuration.
2. Create a new outbound service

The outbound service you'll create will provide access using an outbound port to the SOAP/JMS accessible Bank application Web service implementation. The port destination configured on the outbound port will be associated with the SOAP/JMS transport binding in the AccountSessionFacade.wsdl file. The corresponding WSDL port defined is shown in Listing 1:

Listing 1. WSDL port binding

```xml
<wsdl:port binding="impl:AccountSessionFacadeJMSSoapBinding"
    name="AccountSessionFacadeJMS">
  <wsdlsoap:address
    location="jms:/queue?destination=jms/SimpleBankAppQueue&
    connectionFactory=jms/SimpleBankAppIncomingQCF&
    targetService=AccountSessionFacadeJMS&
    jndiProviderURL=corbaloc:iiop:localhost:2809/NameServiceServerRoot"/>
</wsdl:port>
```

- Download the AccountSessionFacade.wsdl file for the Bank application Web service implementation.
- From the Administrative console, expand Service Integration and click on Buses.
- Click on SimpleBankAppSIBus.
- Under Additional Properties, click on Outbound Services and then New.
- Enter the location of the downloaded WSDL file, "<your_location>/AccountSessionFacade.wsdl", as shown in Figure 8.
• Click Next.

• Select \${http://ejb.bankapp.simple.ibm.com}AccountSessionFacadeService as the Service, as shown in Figure 9.

Figure 9. Outbound service -- Select service

• Click Next.

• Select the AccountSessionFacadeJMS as the port and click Next.

• Enter BankAppOutboundService as the Outbound Service name.

• Enter http://ejb.bankapp.simple.ibm.com:BankAppService as the Service Destination name, as shown in Figure 10.

Figure 10. Outbound service -- Service name and destinations
• Click Next.

• Ensure that AccountSessionFacadeJMS is associated to a bus member <your_node>:server1 and click Finish.

3. Create a new inbound service

The SOAP/HTTP requests received at the SOAPHTTPChannel1 endpoint listener will be forwarded to the inbound service using an inbound port. The inbound service created and configured in this section will route the SOAP/HTTP Web service requests to the http://ejb.bankapp.simple.ibm.com:BankAppService service destination configured on the outbound service. The outbound service configured in the previous step will perform the protocol transformation of the SOAP/HTTP requests coming in from the inbound service to SOAP/JMS for the target Bank application Web service implementation.

• From the Administrative console navigation pane, expand Service Integration and click on Buses.

• Click on SimpleBankAppSI.Bus.

• Under Additional Properties, click on Inbound Services and then New.


• Select URL as the Template WSDL location type.

• Enter the location of the downloaded WSDL file <your_location>\AccountSessionFacade.wsdl for the Template WSDL location.

Figure 11 shows the resulting inbound service template WSDL location configuration and service selection.

Figure 11. Inbound service -- Service selection and template WSDL location
• Click **Next**.

• Select **${http://ejb.bankapp.simple.ibm.com}AccountSessionFacadeService** as the **Service** and click **Next**.

• Enter **BankAppInboundService** as the **Inbound service name**. The endpoint listener entry is prepopulated and will forward requests to the inbound service. **Figure 12** shows the resulting inbound service name and endpoint listener configuration.

**Figure 12. Inbound service -- Name and endpoint listener**

• Click **Next**.

• Click **Finish** on Step 4, as you'll not use a UDDI registry to publish the WSDL.

• Click on the new **BankAppInboundService**.

• Under Additional Properties click on **Inbound Ports**.

• Click on the default Inbound Port created for the
BankAppInboundService.

- Set the name of the Inbound Port to SOAPHTTPChannel1InboundPort.
- Click OK.
- Save configuration changes.

NOTE: At this point, you could choose to export the WSDL for the inbound service for client development. This has already been done for you and the J2EE application client, BankAppWSClient.ear, and is available for download.

Section 10. Implement the solution -- Test the Web service client for both transports

Test the Web service client for both transports

In this section, you'll run the J2EE application client, BankAppWSJMSHTTPClient.java, packaged in the BankAppWSClient.ear to test the deployed solution.

Test client overview

- Developed using Rational Application Developer Version 6.0 after generating the necessary proxy classes from the corresponding WSDL files described below:

  - SOAP/JMS (deployed target service): AccountSessionFacade.wsdl, the WSDL file corresponding to the target Bank application Web service implementation
  - SOAP/HTTP (configured inbound service -- endpoint listener and inbound port):
    - BankAppInboundServiceService.wsdl
    - BankAppInboundServicePortTypes.wsdl
    - BankAppInboundServiceBindings.wsdl
  - The steps necessary to export the WSDL files from the Administrative console are listed below:
    - Expand Service Integration and select Buses.
    - Select SimpleBankAppSIBus.
    - Under Additional Properties, click on Inbound Services.
• Click on **BankAppInboundService** and then **Publish WSDL files to the .zip file**.
• Click on **BankAppInboundService.zip** to download WSDL.
• Accepts either of the following command line arguments:

  **JMS** -- executes the method
  BankAppJMSHTTPTestClient.testJMS(), which validates functionality by simply creating a new account.

  **HTTP** -- executes the method
  BankAppJMSHTTPTestClient.testHTTP() -- validates functionality by simply updating an existing account.

  **BOTH** -- executes both of the above methods.

• Source code for the BankAppWSJMSHTTPClient.java, as shown in **Listing 2**.

**Listing 2. Source code for the BankAppWSJMSHTTPClient.java**

```java
package com.ibm.simple.bankapp.j2ee.client;
import java.rmi.RemoteException;
import javax.xml.rpc.ServiceException;
import com.ibm.simple.bankapp.ejb.*;
import com.ibm.www.*;
/**
 * BankAppWSClientJ2EE BankAppJMSHTTPTestClient.java
 */
public class BankAppJMSHTTPTestClient {
    public static AccountSessionFacade accountSF = null;
    public static void main(String[] args) {
        if (args.length < 1) {
            printUsage();
            return;
        }
        if (args[0].equalsIgnoreCase("BOTH")) {
            testJMS();
            testHTTP();
        } else if (args[0].equalsIgnoreCase("JMS")) {
            testJMS();
        } else if (args[0].equalsIgnoreCase("HTTP")) {
            testHTTP();
        } else {
            printUsage();
        }
    }

    private static void testJMS() {
        AccountSessionFacadeServiceLocator sl = new AccountSessionFacadeServiceLocator();
        try {
            // use Service locator to obtain SOAP/JMS Stub
            accountSF = sl.getAccountSessionFacadeJMS();
            System.out.println("\nAttempting to invoke Bank App using SOAP/JMS Endpoint");
            System.out.println("JMS Endpoint::");
            sl.getAccountSessionFacadeJMSAddress());
        }
    }
```
System.out.println("Binding Stub:: " + accountSF.getClass().getName());
System.out.println("Invoking createAccount");
System.out.println(accountSF.createAccount("555-55-5555", 100.0f,false));
catch (ServiceException e) {
e.printStackTrace();
catch (RemoteException e1) {
e1.printStackTrace();
}

private static void testHTTP(){
BankAppInboundServiceLocator sl = new BankAppInboundServiceLocator();
try{
//use Servicelocator to obtain SOAP/HTTP Stub
accountSF = sl.getSOAPHTTPChannel1InboundPort();
System.out.println("Attempting to invoke Bank App using SOAP/HTTP Endpoint");
System.out.println("HTTP Endpoint:: " + sl.getSOAPHTTPChannel1InboundPortAddress());
System.out.println("Binding Stub :: " + accountSF.getClass().getName());
Integer acctnum = new Integer(4);
System.out.println("Invoking updateAccount where acctnum = " + acctnum);
System.out.println(accountSF.updateAccountBalance(acctnum,500.0f));
System.out.println("Invoking getAccountBalance where acctnum = " + acctnum);
System.out.println("New Balance = " + accountSF.getAccountBalance(acctnum));
catch (ServiceException e) {
e.printStackTrace();
catch (RemoteException e1) {
e1.printStackTrace();
}
}

private static void printUsage(){
System.out.println("USAGE::");
System.out.println("launchClient [-profileName SimpleBankAppServer]
<location>/BankAppWSClient JMS|HTTP|BOTH");
System.out.println("-profileName is optional");
}

To run the provided BankAppJMSHTTPTestClient code, follow the steps below:

1. Download the BankAppWSClient.ear file.
2. Run the client to verify SOAP/JMS and/or SOAP/HTTP access to the Bank application Web service implementation
   - To test SOAP/JMS, enter the following at the command line:
     <was_home>\bin\launchClient C:\BankAppTest\BankAppWSClient.ear
     JMS:
   - To test SOAP/HTTP, enter the following at the command line:
To test both SOAP/JMS and SOAP/HTTP, enter the following at the command line:

```
<was_home>/bin/launchClient C:\BankAppTest\BankAppWSClient.ear

HTTP
```

Sample output from a successful run using the BOTH command line option is shown in Listing 3.

### Listing 3. Sample output from a successful run

```
IBM WebSphere Application Server, Release 6.0
J2EE Application Client Tool
Copyright IBM Corp., 1997-2004
WSCL0012I: Processing command line arguments.
WSCL0013I: Initializing the J2EE Application Client Environment.
WSCL0035I: Initialization of the J2EE Application Client Environment has completed.
WSCL0014I: Invoking the Application Client class
com.ibm.simple.bankapp.j2ee.client.BankAppJMSHTTPTestClient

Attempting to invoke Bank App using SOAP/JMS Endpoint
JMS Endpoint::
jms:/queue?destination=jms/SimpleBankAppQueue&
  connectionFactory=jms/SimpleBankAppIncomingQCF&
  targetService=AccountSessionFacadeJMS&
  jndiProviderURL=corbaloc:iiop:localhost:2809/NameServiceServerRoot
Binding Stub :: com.ibm.simple.bankapp.ejb.AccountSessionFacadeJMSSoapBindingStub
Invoking createAccount
Account created new account number is 5

IBM WebSphere Application Server, Release 6.0
J2EE Application Client Tool
Copyright IBM Corp., 1997-2004
WSCL0012I: Processing command line arguments.
WSCL0013I: Initializing the J2EE Application Client Environment.
WSCL0035I: Initialization of the J2EE Application Client Environment has completed.
WSCL0014I: Invoking the Application Client class
com.ibm.simple.bankapp.j2ee.client.BankAppJMSHTTPTestClient

Attempting to invoke Bank App using SOAP/HTTP Endpoint
HTTP Endpoint::
http://localhost:9080/wsgwsoaphttp1/soaphttpengine/
SimpleBankAppSIBus/BankAppInboundService/SOAPHTTPChannel1InboundPort
Binding Stub :: com.ibm.www.SOAPHTTPChannel1InboundPortBindingStub
Invoking updateAccount where acctnum = 4
Account Updated: New Account Balance = 600.0
Invoking getAccountBalance where acctnum = 4
New Balance = 600.0
```

You have successfully tested your Bank application Web service implementation and the ability to invoke the Web service using the SOAP/JMS and SOAP/HTTP transport mechanisms.
Section 11. Troubleshooting

Problem

During deployment of the application on Windows, you receive the following error:

`java.io.IOException: URI length is greater than Windows limit of 259 characters`

Solution

Change the location used by Application Server for temp files during application deployment.

- From the Administrative console navigation pane, expand Servers and click on Application Servers.
- Click on server1.
- Under Additional Properties, click on Java Virtual Machine (JVM).
- Enter "-Dworkspace.user.root=<new_location>" in the Generic JVM arguments field.
- Save Server Configuration.
- Restart Server.

Issue

When attempting to run the J2EE application test client, you receive the following errors:

**Listing 4. Possible errors running the J2EE application test client**

```
C:\temp\Ear-Files>c:\websphere\appserver6\bin\launchClient
C:\temp\Ear-Files\BankApp\WSClient.ear BOTH
IBM WebSphere Application Server, Release 6.0
J2EE Application Client Tool
Copyright IBM Corp., 1997-2004
WSCL0012I: Processing command line arguments.
WSCL0013I: Initializing the J2EE Application Client Environment.
WSCL0035I: Initialization of the J2EE Application Client Environment has completed.
```
WSCL0014I: Invoking the Application Client class
com.ibm.simple.bankapp.j2ee.client.BankAppJMSHTTPTestClient

Attempting to invoke Bank App using SOAP/JMS Endpoint
JMS Endpoint:
  jms:/queue?destination=jms/SimpleBankAppQueue&
  connectionFactory=jms/SimpleBankAppIncomingQCF&
  targetService=AccountSessionFacadeJMS&
  jndiProviderURL=corbaloc:iiop:localhost:2809/NameServiceServerRoot
Binding Stub::
  com.ibm.simple.bankapp.ejb.AccountSessionFacadeJMSSoapBindingStub

Invoking createAccount
[7/11/05 15:28:26:240 EDT] 0000000d E UOW=3-7c64e7aa-5298439:blueberry
source=com.ibm.ws.webservices.engine.transport.jms.JMSSender org=IBM
prod=WebSphere component=Application Server thread={P=97377:O=0:CT}

WSWS3016E: Caught JMSException: javax.jms.JMSException: CWSIA0241E:
An exception was received during the call to the method
com.ibm.websphere.sib.exception.SIResourceException: CWSIT0006E:
It is not possible to contact a messaging engine in bus
SimpleBankAppSIBus..
WSWS3017E: Linked Exception:
com.ibm.websphere.sib.exception.SIResourceException: CWSIT0006E:
It is not possible to contact a messaging engine in bus
SimpleBankAppSIBus.
WebServicesFault
  faultCode: JMS
  faultString: WSWS3016E: Caught JMSException: javax.jms.JMSException:
  CWSIA0241E: An exception was received during the call to the method
  JmsManagedConnectionFactoryImpl.createConnection:
  com.ibm.websphere.sib.exception.SIResourceException:
  ...
  ...
C:\temp\Ear-Files>

Result

Ensure that the SIB_ENDPOINT_ADDRESS for the server is set to default 7276.

- From the Administrative console navigation pane, expand Servers and click on Application Servers.
- Click on server1.
- Under Communications, expand Ports.
- Verify that port is set to 7276.

If the value is not set to 7276, there are two options to resolve this problem.

1. Change the SIB_ENDPOINT_ADDRESS port to default 7276.
   - Under Communications, click on Ports.
   - Click on SIB_ENDPOINT_ADDRESS and enter the default value of 7276 for the Port.
   - Save configuration changes.
• Restart Server.

2. Add a provider endpoint for the JMS queue connection factory SimpleBankAppIncomingQCF.
   • Expand Resources > JMS Providers and click on Default Messaging.
   • In the default messaging provider frame under Connection Factories, click on JMS queue connection factory.
   • Click on SimpleBankAppIncomingQCF.
   • In the provider endpoints section, enter "your_host:<SIB_ENDPOINT_ADDRESS>:BootstrapBasicMessaging". For example, "localhost:7277:BootstrapBasicMessaging". You must make sure that this entry is correct.

Note: If you set up the server to use the default SIB_ENDPOINT_ADDRESS of 7276, then you do not have to enter a provider endpoint. In the case where the defaults were not used when creating your profile, the provider endpoint entry is required (and should be as above) in order for the Application client to have access to resources in the SIB. See the following article for help with configuring a connection to a non-default bootstrap server.

Section 12. Conclusion

This tutorial demonstrated how to deploy a SOAP/JMS stateless session bean Web service implementation in Application Server Version 6.0 and provided SOAP/JMS accessibility using configured SIB messaging resources. It also illustrated how to configure SIBWS inbound and outbound services to provide routing and protocol transformation of SOAP/HTTP service requests to the same SOAP/JMS target service. The overall solution demonstrated an ESB implementation with Application Server Version 6.0 SIB and SIBWS technologies facilitating application connectivity by providing messaging, routing, and protocol transformation services.
## Downloads

<table>
<thead>
<tr>
<th>Description</th>
<th>Name</th>
<th>Size</th>
<th>Download method</th>
</tr>
</thead>
<tbody>
<tr>
<td>EAR and WSDL files for the Bank application</td>
<td>ws-transform-esb.zip</td>
<td>58 KB</td>
<td>HTTP</td>
</tr>
</tbody>
</table>

**Information about download methods**
Resources

Learn

• Implement the continuation of the Bank Application Web services example: Configure secure Web services communications through an ESB.

• For those new to the technology, the following reference article is a good starting point: Enterprise Service Bus Capabilities of WebSphere Business Integration Server Foundation V5.1.

• To learn more about the features, architecture, and benefits of the Enterprise Service Bus, see WebSphere Software -- The enterprise service bus.

• You can also find information on ESB in the WebSphere Application Server InfoCenter -- Introduction to Service Integration.

• The IBM Redbooks Patterns: Implementing an SOA using an Enterprise Service Bus and Patterns: SOA with an Enterprise Service Bus in WebSphere Application Server V6 are excellent sources of information on implementing an SOA.

• WebSphere Version 6 Web Services Handbook Development and Deployment is a comprehensive resource for Web Services developers.

• Find information on creating profiles in WebSphere Application Server at the WebSphere Application Server Version 6.0 Information Center.

• See the article Accessing an EJB Web service using SOAP over JMS or SOAP over HTTP with WebSphere Studio to learn how to use WebSphere Studio to make a Web service accessible using either SOAP or HTTP or SOAP or JMS.

• The IBM developerWorks team hosts hundreds of technical briefings around the world which you can attend at no charge.

• Standards roadmap -- understand the impact and importance of standards and specifications for the development of SOA and Web services.

• SOA and Web services -- hosts hundreds of informative articles and introductory, intermediate, and advanced tutorials on how to develop Web services applications.

Get products and technologies

• See WebSphere Application Server for information on purchasing WebSphere.

• Get your hands on application development tools and middleware products from DB2®, Lotus®, Rational®, Tivoli®, and WebSphere®. You can download evaluation versions of the products at no charge, or select the Linux™ or Windows® version of developerWorks' Software Evaluation Kit.

Discuss

• developerWorks blogs: Get involved in the developerWorks community.
About the author

Tendai Chinoda
Tendai Chinoda is an Advisory Software Engineer at the IBM Business Partner Technical Enablement - WebSphere Competency Center. He provides enablement services to premier ISVs and Business Partners for the IBM WebSphere family of products. Tendai is an IBM Certified for On Demand Business - Solution Designer, an IBM Certified Solution Developer - Web Services Development with WebSphere Studio V5, a Sun Certified Programmer for the Java 2 Platform 1.2, and holds other WebSphere product certifications. He received his Bachelors degree in Electrical Engineering at Tennessee State University, Nashville TN and completed his Masters in Computer Science and Engineering from the Pennsylvania State University, University Park, PA. You can contact him at tendai@us.ibm.com.