Getting started with WebSphere Enterprise Service Bus and WebSphere Integration Developer

Calum Byrom

January 20, 2006

This article introduces developers to the IBM WebSphere Enterprise Service Bus server and its accompanying tooling, WebSphere Integration Developer. This article describes how to: 1) develop a mediation flow providing a basic Web service; 2) develop an intermediate flow, to connect to this service, with more complex routing logic provided by several of the pre-built mediation functions offered by the tooling; 3) deploy and test these flows using both the tooling test facilities and a standalone JSP-based front-end.

Introduction

The IBM® WebSphere® Enterprise Service Bus (ESB) server and its associated tooling, WebSphere Integration Developer, are two new IBM products that enable you to improve connectivity between heterogeneous systems. In essence, WebSphere Integration Developer helps you develop a framework (mediation flows) with which to process incoming data (messages), adjusting and logging content as necessary, before routing appropriately to different destinations. You then use WebSphere ESB to run these flows, handling additional features such as security, performance, reliability, and transaction management. This article shows you how to:

• Develop a simple mediation to expose a Web service.
• Develop a more complex mediation to connect to this Web service, providing logic to filter, transform, and route incoming messages appropriately.
• Invoke the resulting mediation flows using both the WebSphere Integration Developer test client and a JSP Web-based front-end, and check the resulting output.

To start with, set up a simple Web service to which to connect mediation flows, by defining an interface and then using the tooling to auto-generate the necessary code, as described in the next section.

Creating a library interface

Web services are defined by interfaces that provide all the information required to invoke them, namely the type of required input and output parameters and the location of the Web service. Whilst you can import an existing, external Web service interface using the Web Services Explorer in WebSphere Integration Developer, this article will show you how to develop a Web service from scratch to demonstrate the full procedure.
Open WebSphere Integration Developer and close the welcome pane to enter the Business Integration perspective. If you happen to close this perspective, you can re-open it by selecting Window => Open Perspective => Other => Business Integration:

Figure 1.1. Business Integration perspective

To start, create a suitable project in which to store the Web service interface:

1. Select File => New => Other and choose Library from the wizard:
Figure 1.2. New Project wizard

2. Specify **StockQuoteLibrary** as the Library Name and click **Finish**.
Now to create the interface itself:

1. In the Business Integration Navigator on the left side of the tooling, expand the **StockQuoteLibrary** project and right-click **Interfaces => New => Interface**.
Figure 1.4. Creating a new interface

2. Provide the Name of `stockQuoteService` and click **Finish**.
The new interface will now open in the main display, although at this stage it will be largely empty since we have not yet defined anything. Every Web service has an effective function to achieve and we need to specify it in defining the interface. To keep things simple, we'll use a traditional "stock quote" service -- a service that accepts a string representing a stock name, and returns a float representing the price of that stock. We need to define this type of service as "two-way," since there are request and response messages involved:

1. Click **Add Request Response Operation**, the first icon to the right of Define Operations, to add a row to the interface table.
2. Change the default name in this row from `operation1` to `getQuote` and click **Add Input**, the third icon to the right of Define Operations.
3. Change the default name for the input parameter from `input1` to `inputGetQuote`.
4. Click in the **Type** column and select **New** to create a new type of business object to act as an input. In the Data Type Selection pop-up, click **New**:
5. In the New Business Object box that opens (Figure 1.7), give the name `StockName` and click Finish => OK:

**Figure 1.7. The New Business Object wizard**

6. The new `StockName` business object should now appear in the Type column. Right-click on it and select Open in Business Object Editor. A new editor opens where you can define this new business object in detail.
7. Click **Add an attribute to a business object** to the right of Business object and change the default value `attribute1` to `name`:

**Figure 1.8. StockName business object interface**

8. Close the panel, electing to save the changes; a simple business object, which just wraps a String value, has been defined. Back in the interface panel, complete a similar procedure to define an output parameter called `outputGetQuote`, with a new type of `StockQuote`, which wraps a float value with the name `price`.

9. Save the interface: Press **Ctrl-S** or close the editor window and click **Yes** when prompted.

You have fully defined the interface, which should look like Figure 1.9 below. Although interfaces can be defined and stored directly in the mediation projects in which they are used, by creating them in a separate library project such as this one, you make them easily available for reuse by other mediation projects.

**Figure 1.9. Finished StockQuote interface**

<table>
<thead>
<tr>
<th>Define Operation(s)</th>
<th>Define Operations and their corresponding parameters</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>GetQuote</strong></td>
<td></td>
</tr>
<tr>
<td><strong>Input(s)</strong></td>
<td><code>inputGetQuote</code></td>
</tr>
<tr>
<td><strong>Output(s)</strong></td>
<td><code>outputGetQuote</code></td>
</tr>
</tbody>
</table>

**Implementing a Web service**

Now that you have defined a Web service interface, you need to create a local implementation of this service, which can be accessed through a mediation flow. As with the interface, first create a suitable project to house this service:

1. Select **File => New => Other** and select **Mediation Module** from the wizard.
2. In the New Mediation Module pop-up, enter `StockQuoteService` as the Module Name and ensure that **get Runtime specified** is set to either an actual WebSphere ESB server or a WebSphere ESB server stub. Keep the default settings checked and click **Next**.
3. In the next panel, select the previously created `StockQuoteLibrary` and click **Finish**.

The Business Integration Navigator on the left-hand side of the tooling now displays the new project. Expand it and find the subcomponent `StockQuoteService`. It represents the assembly diagram for the mediation module, where the configuration of the high-level mediation components
(such as import and export connections, numbers and arrangement of mediation flows, and so on) are specified. To implement it:

1. Double-click on on this subcomponent. The Assembly Diagram Editor opens with a single mediation flow component, Mediation1. The important features of this editor are shown below:

   **Figure 2.1. Assembly diagram for the new StockQuoteService mediation module**

2. Click on the mediation flow component twice to change the name to StockQuoteServiceMediation. (The names specified in these diagrams are not actually used by the underlying code, but they make the flows easier to understand from a user perspective and simplify information output to system logs.)

3. From the icon menu on the left side of the panel, select the icon with a green arrow (as shown in Figure 2.1 above) to expand the menu. Then select the downward arrow icon, which represents an Export component. Click on the Assembly Diagram canvas to add an Export to the assembly diagram and rename it to StockQuoteServiceExport.

4. Click on the bottom icon, Wire in the icon menu and then click on StockQuoteServiceExport and StockQuoteServiceMediation to connect the two together. (You can create the same connection by clicking on the Export component and dragging the produced wire to the Mediation component before releasing the mouse button.)

5. A panel (Add Wire) is now displayed, explaining that the wiring action will add an interface matching that used by the mediation component to the export, thus exposing this interface for use by other modules. Select **Always create without prompt** and click **OK**.

6. In the next window, Add Interface, select the StockQuoteService interface defined earlier and click **OK**.

7. The Assembly Diagram Editor should now show a link between the Export and Mediation components. Note that there are matching interfaces specified on each component, represented by circles containing an "I." So far in this diagram, we have an Export, which exposes this mediation flow for use by other flows, and a Mediation component, which will contain the detail of the actual mediation flow. The last component required is a representation of the Web service to be invoked. To create it, click on the envelope icon, select the bottom icon with a "J" on it, and then click on the canvas.

8. In the same way as before, rename this component to StockQuoteService, and wire the mediation component up to it, selecting the StockQuoteLibrary interface as the reference
to add. This reference allows the Java component to be accessed by the linking mediation component, StockQuoteServiceMediation. It is displayed graphically in the editor as a box containing 1..1.

9. Finally, to tidy up things, right-click on the editor canvas and select Arrange Contents Automatically, then save the changes.

We now have the basics of a mediation flow, which should look like Figure 2.2 below. The blue exclamation marks signify that the underlying implementation of the components has not been completed.

Figure 2.2. Completed StockQuoteService assembly diagram

First we will implement the Java component:

1. Double-click on the Java component icon and select the default options. The Java class StockQuoteServiceImpl, based on the StockQuoteService library, will open automatically in a text editor. The class provides a basic skeleton of all the methods detailed in the interface. In our simple case, this means we only have a getQuote method, and a getMyService method to return an instance of the class.

2. Replace the getQuote method with the code featured in the section below. In addition to providing suitable output logging, it creates and populates a valid business object to output, using the Web service interface. The return value depends on the content of the input message, as discussed in detail below.

Listing 1. Code for the getQuote method in StockQuoteServiceImpl.java

```java
public DataObject getQuote(DataObject inputGetQuote) throws ServiceBusinessException {
    // retrieve the String value of the 'name' attribute
    System.out.println("StockQuoteServiceImpl - getQuote : Get name from inputGetQuote");
    String name = inputGetQuote.getString("name");
    System.out.println("StockQuoteServiceImpl - getQuote : name = " + name);

    // Now get the BO Factory to create the reply from
    System.out.println("StockQuoteServiceImpl - getQuote : Create BOFactory");
    ServiceManager serviceManager = new ServiceManager();
    BOFactory bofactory = (BOFactory) serviceManager.locateService("com/ibm/websphere/bo/BOFactory");

    // Create the output message structure - as specified by the StockQuoteService interface
    System.out.println("StockQuoteServiceImpl - getQuote : Create getQuoteResponse");
    DataObject getQuoteResponse = bofactory.createByElement(
        "http://StockQuoteLibrary/StockQuoteService",
        "getQuoteResponse");

    // Now create the Data Object to populate with the return value
    System.out.println("StockQuoteServiceImpl - getQuote : Create outputGetQuote");
    DataObject outputGetQuote = getQuoteResponse.createDataObject("outputGetQuote");

    // And finally, set the appropriate return value
    if (name.equals("IBMvetted")) {
        System.out.println("StockQuoteServiceImpl - getQuote : Set the price to 100.00f");
    }
```
outputGetQuote.setFloat("price", 100.00f);
} else if (name.equals("IBM")) {
    System.out.println("StockQuoteServiceImpl - getQuote : Accessed with not vetted input" + " - return old stockquote price, 80.00f");
    outputGetQuote.setFloat("price", 80.00f);
} else {
    System.out.println("StockQuoteServiceImpl - getQuote : Unrecognised stockquote name" + " - set price to 0.00f");
    outputGetQuote.setFloat("price", 0.00f);
}
return outputGetQuote;

3. WebSphere Integration Developer should highlight some errors in this new code, arising from class references it cannot resolve. To import these, right-click and select Source => Organise Imports.

Close and save the Java class and then implement the Mediation component by double-clicking on it and selecting the defaults. A two-panel Mediation Flow Editor opens. The top pane, the Operation Connections pane, displays the interfaces and references available to the mediation component. The bottom pane, the Mediation Flow pane, shows the actual mediation flow:

**Figure 2.3. Mediation Flow Editor**

![Mediation Flow Editor](image)

This mediation module enables the service, represented by the Java component, to be invoked, so the flow should be a simple, straight-through connection. To complete the implementation:

1. Click the `getQuote` operation in the interface box on the left in the top pane to open the flow in the Mediation Flow pane below. Initially, this flow contains two unconnected nodes, representing the points where messages will enter (`StockQuoteService_getQuote_Input`)
and exit (StockQuoteService_getQuote_InputResponse) the flow component. Because the StockQuoteServicePartner reference in the top pane is not connected to the interface, the flow can only directly return a message, and not pass it on for further processing by a later mediation component.

2. To give the mediation component access to the Java component defined earlier, join the interface and reference in the top panel by clicking and dragging, left to right, between the two getQuote operations. A new node, StockQuoteService_getQuote_Callout, is created in the bottom pane:

**Figure 2.4. Connecting up the interface and reference in the Mediation Flow Editor**

Now we create our first mediation flow, proper. Since the main purpose of this flow is to provide a connection to the Java component, we'll keep the contents simple. The icons in the menu on the left side of the Mediation Flow panel represent the set of mediation primitives available. These primitives offer pre-built units of functionality, as shown in Figure 2.5, to act as the basic building blocks of mediation flows:
Getting started with WebSphere Enterprise Service Bus and WebSphere Integration Developer

Figure 2.5. Palette of mediation primitives

1. Select the Message Logger mediation primitive (an envelope superimposed on a page) and click on the Mediation Flow canvas. A Message Logger primitive will be created in the flow (MessageLogger1), and its the job is to automatically write the contents of a message passing through it to a database, where it can be checked later (for details, see the section Checking the Message Logger default database).

2. Right-click on MessageLogger1 and select Show in Properties to open a view at the bottom of WebSphere Integration Developer where you can find further information and specify configuration details, as shown below in Figure 2.6. Of particular interest is the Details sub-view, which lets you specify different database details. Since the default Cloudscape database is created automatically, keep the initial settings:

   Figure 2.6. Examining the details of the Message Logger primitive

3. Notice the small boxes on the left and right sides of the mediation nodes in the editor canvas. These represent the input, output, and fail terminals that are used to provide connections to other mediation primitives and thus create useful mediation flows (to see which is which, hover the mouse pointer over them). Click on the output terminal of the Input node and drag the resulting wire to the input terminal of MessageLogger1 to join them together. Similarly, join MessageLogger1 to the Callout node. To handle error recovery, you can use the fail terminal
to specify a separate route to take should the Message Logger primitive throw an exception. To keep things simple for the time being, leave this terminal unconnected. The completed flow looks like this:

**Figure 2.7: The completed request mediation flow for StockQuoteService**

4. At the bottom of the Mediation Flow pane, click on **Response:getQuote** tab to switch to the canvas representing the mediation flow accessed following the invocation of the Web service -- the response flow.
5. Repeating the above process, add a Message Logger primitive and wire this to the CalloutResponse and InputResponse nodes:

**Figure 2.8. The completed response mediation flow for StockQuoteService**

6. Close the Mediation Flow editor, saving the changes and then saving the changes to the accompanying Assembly Diagram. You have now completed your first mediation module!

**Creating a more advanced mediation module**

The next step is to develop a more involved mediation module to connect to the module just created. This module will filter and adjust incoming messages, providing a different mediation routing and different result, depending on the contents of the message. The example here is based on a real-life scenario in which a single Web service may be connected in many different ways. Here we are amalgamating the stock quote service with additional business logic to simulate the imposition of a "level of service" according to the input message content:

1. Repeating the process used in the last section, create a Mediation Module named StockQuoteMediation, using the StockQuoteLibrary interface.
2. Open the Assembly Diagram Editor, add an Export component, and wire it to the mediation component, selecting the existing interface, StockQuoteService as before.
3. Since this mediation module is intended to provide connection to the module we've just defined, an import component is needed. You can add it using the side menu (icon with horizontal green arrow), but this requires additional, manual configuration to set up the connection to the StockQuoteService module. Instead, you can define this configuration automatically. To do so, expand the StockQuoteService module in the Business Integration
Navigator, then expand the underlying `StockQuoteService` folder to reveal this module's export component, `StockQuoteServiceExport`. Drag and drop it to the current mediation module Assembly Diagram Editor, selecting **Import with SCA Bindings**. An Import component will be added, providing a direct connection to the Export component in the `StockQuoteService` module:

**Figure 3.1. Creating an import component from an existing export component**

4. Wire up this new import, rename the contents of the editor appropriately, and save the changes:

**Figure 3.2. Completed assembly diagram for StockQuoteMediation**

Now, double-click on the mediation component, accept the offered default settings, and implement as follows:

1. Join the interface and reference operations as before, to allow connection to the `StockQuoteService` import component and thus the `StockQuoteService` module.
2. Add a Message Filter primitive (icon with envelope with two arrows) to the Mediation Flow Editor canvas. This primitive enables messages to be routed to different destinations according to their payload content. You can create extra output terminals and associate them with selection criteria:
1. Wire the output terminal of the Input node to the input of the filter primitive.
2. Right-click on the filter, select Add Output Terminal, and accept the default values in the pop-up box. A new terminal named match1 is added to the mediation primitive.
3. Right-click on the filter again and this time select Show In Properties and access the Details tab in the lower view. Your display should now look like this:

**Figure 3.3. StockQuoteMediation flow With Message Filter primitive added and Details displayed**

4. In this Details view, click Add and select Custom XPath.
5. In the XPath Expression Builder window, expand the body : getQuoteRequestMsg structure completely and click name : string to select it in the XPath Location: box.
6. Click Location value in the Condition: row and select self::node().
7. Click on Value, select String, and enter the string IBM.
8. The Full XPath Expression value at the bottom of the panel should now read /body/getQuote/inputGetQuote/name[self::node()="IBM"], as shown below in Figure 3.4. Click OK and verify that the Terminal name matches the new one you've just added (match1). Then click Finish:
9. Repeat steps 2 through 8 to add a new output terminal named match2, which this time matches to IBMold instead of IBM.
10. Wire this new terminal to the StockQuoteServicePartner Callout node.
11. The last thing to configure is the distribution mode: from the Distribution drop-down menu, select All to enable access to all output terminals with filters satisfied by an incoming message.

The final arrangement should look like this:
Figure 3.5. StockQuoteMediation flow with completed Message Filter primitive

As you’ve probably worked out, we now have a filter that checks the name value of an input message (assuming that the correct format, as specified by the interface, is being used). If it is equal to IBM or IBMold, then the specified terminal is passed the input message and subsequent connections are accessed.

3. Add a Message Logger primitive to the canvas and wire the default terminal of the Message Filter to the input of this. The default filter terminal is fired if none of the user-defined filters on the other terminals produce a valid match against an incoming message.

4. Add a Stop primitive (red square icon). Wire the output from the Message Logger primitive to it, as shown below in Figure 3.6. The Stop primitive lets a mediation flow halt cleanly. In this case, it is used to end any flows routed via the Message Filter default terminal, once their messages have been logged by the Message Logger primitive. In other words, the StockQuoteService module will be accessed only if the incoming message has a name with the value IBM.

Figure 3.6. StockQuoteMediation flow with Message Logger and Stop primitives added
5. Add an XSL Transformation primitive (icon with small and large page) to the canvas. This primitive lets you adjust the content of an incoming message. You can use it to append a string to the incoming stockquote name:
   1. Wire the match1 output of the Message Filter primitive to the input of the XSL Transformation primitive.
   2. Wire the XSL Transformation output terminal to the input to the Callout node.
   3. Right-click on the XSL primitive, select Show In Properties, and access the Details tab in the lower view.
   4. Keep the default Root: setting as body, which restricts the primitive to changing only the content of the message body.
5. Click New to start defining a mapping transform. In the New XSLT Mapping window, the input and output message types have been pre-defined, as shown below in Figure 3.7, because both the input and output terminals have already been connected, so the primitive knows what message type to expect and to pass on. If other message types are required, specify them by selecting directly from an existing interface via the Browse button. Accept the defaults by clicking Finish:

**Figure 3.7. XSLT Mapping wizard**

6. The XSL Transform Editor now opens, displaying the body structure of the incoming and outgoing messages. Expand the smo (Service Message Object -- the type of data object used within the mediation flows) elements fully in both messages:
7. Click on the smo root element of both messages, then right-click and select **Match Mapping**, which copies the contents of the input message to the output message to ensure they are the same.

8. Right-click on the **name [0..1]** element of the outgoing (Target) message and select **Define XSLT Function**.

9. In the pop-up window, select **String** and click **Next**.

10. From the drop-down menu, select **concat**, click on **Add**, and enter the string `'vetted'` (include the single quotes). This new function should now appear in the middle table:
11. Exit the transform editor, saving the changes. Back in the Mediation Flow Editor, this newly created mapping file should be displayed in the Properties view. The last required step is to turn this mapping file into a proper XSL file with which to enact the actual transform. Click **Regenerate XSL** and save the flow:
You have now configured this mediation primitive to append the string *vetted* to the input stockquote name value.

6. The last thing to do is add a Fail primitive (red exclamation mark icon -- you may need to click on the red square icon to reach it). This primitive is similar to the Stop primitive in that it provides a clean way to halt the flow. The difference lies in the treatment of any exception message passed to it: a Stop primitive will propagate any exceptions received up the stack, passing it back to the caller of the mediation, while a Fail primitive will cleanly swallow it. Note that this may not be the desired behaviour because it may end up hiding problems with a system. In this case, however, we use it to cover errors from the second Message Logger primitive (which may be thrown due to database access problems, for example). Since this provides a non-essential function, logging messages on a "dead-end" flow, it is appropriate to use the Fail primitive here.

7. The Request flow is now complete and should look like this:
Figure 3.11. Complete StockQuoteMediation flow

8. The last step to configure the mediation flow is to connect up the Response flow. Do this as before, including a Message Logger primitive. The result should look like the flow used in the StockQuoteService mediation, as shown above in Figure 2.7.

The new mediation module is now complete, offering three different mediation behaviours depending on the value of the input name String:

- **name=IBM**
  - The message is passed to the StockQuoteService module, after having the String vetted added to it, to give the name value IBMvetted.

- **name=IBMold**
  - The message is passed unchanged to the StockQuoteService module.

- **name=anything else**
  - The message is logged to the local Message Logger database and the flow is halted.

Looking at the listing of the service implementation, StockQuoteServiceImpl.java, we can understand this logic: Effectively, Route 1 provides a high-quality service, returning up-to-date IBM stock values. This service is limited to users of the StockQuoteMediation flow -- hence the need to append the vetted string. Route 2 provides an old, slightly out-of-date IBM stock value. Route 3 restricts the service to use only with IBM stock values. The real-time aspect of the Web service is not implemented, but this arrangement helps show what you can achieve using WebSphere ESB and WebSphere Integration Developer.

**Exposing the Web service**

Now that we have set up a mediation module providing a service, and a module providing flow logic to access this service, the last thing we need to create is something to invoke the complete mediation flow. In a business environment, this last stage is a bit artificial, because produced modules are likely to be accessed directly through their published interfaces. However, it is still instructive to demonstrate the complete flow being invoked -- and you will need to invoke the complete flow during testing.

There are several ways to invoke the StockQuoteService Web service that we’ve implemented. We will attempt two: the WebSphere Integration Developer Universal Test Environment (UTE), and a Web page running a JSP.
Setting up the test environment

Before you can test any of the modules developed here, you need to deploy and start them on a running instance of WebSphere ESB server (or WebSphere Process Server). The following steps assume that the server has already been installed:

1. In the lower WebSphere Integration Developer view, whilst in the Business Integration perspective, click the **Servers** tab.
2. Right-click in this new window and select **New => Server**.
3. In the New Server pop-up, select **IBM => WebSphere ESB Server V6** and then click **Next**.
4. Click **Browse** and locate the ESB installed on your system -- on Windows, the default location is C:\Program Files\IBM\WebSphere\ESB:

**Figure 4.1. New server wizard**

5. Click **Next** twice, accepting all default settings.
6. In the next panel, click **Add All** to add both of the projects developed here to the server, then click **Finish**. The server starts and these projects are published to it and automatically started.

7. Click the **Console** tab in the lower panel to check for errors during the server start-up listed on the output log from the server, which is located, by default, at `<ESB-install-directory>\profilesESB01\logs\server1\SystemOut.log`. If any exceptions are thrown during this process, restart the server by clicking on it in the Servers panel and selecting the **Restart the server** icon (two facing arrows). Starting the server and deploying the projects is likely to take a few minutes, so please be patient.

The server should now be running (indicated by a **started** status in the Servers view) and the projects deployed, ready for testing. One last thing before invoking the mediations -- turn on extra logging detail in the server to generate a detailed report on what is happening during the invocation:

1. Point a Web browser to `http://localhost:9060/ibm/console/` to take you to the log-in page of the WebSphere ESB administrative console -- the Web based GUI interface used to configure a running instance of WebSphere ESB server:

   **Figure 4.2. WebSphere ESB admin console log in panel**

   ![Welcome, please enter your information.](image)

   - **Welcome, please enter your information.**

   - **Log in**

   The User ID does not require a password, and does not need to be a User ID of a user in the local user registry. It is only used to track user specific changes to configuration data. Security is NOT enabled.

2. Click **Log in** and select **Server and Bus** from the Task filtering selector list, Then click **Apply**:
Figure 4.3. Selecting the task filter in the admin console

3. Expand the Troubleshooting menu in the left-side menu and select Logs and Trace.
4. In the new panel, select server1.
5. From here, you can adjust the size and number of historical copies to save of the various server log files produced. For the time being, we are only interested in adjusting the level of detail logged to the trace file, so click on Change Log Detail Level.
6. In the text box in this new page, enter *=info:com.ibm.ws.sibx.*=fine and click OK and save the changes.
7. To pick up these changes, restart the server: return to WebSphere Integration Developer, access the Servers view, and click on the Restart the server icon.

You have now configured the server to use a very detailed form of logging that prints out the full contents of a message passing through a mediation module at the entry and exit point of each component in the mediation flow, as we shall see in the next section. Whilst this is an excellent initial debugging tool, you would usually not use it for commercially deployed applications because of the performance overhead and verbose log files.

Invocation via the Universal Test Environment (UTE)

The UTE provides an interface to directly invoke a flow with a particular operation at a specific component in a mediation module:

1. In the Business Integration Navigator on the left, right-click on the StockQuoteMediation project and select Test => Test Module.
2. A new editor opens, displaying the various invocation options offered by the interface associated with the selected mediation module. Accept the default values and enter IBM as the name value, as shown in Figure 4.4 below. Click Continue to launch the test client.
Getting started with WebSphere Enterprise Service Bus and WebSphere Integration Developer

Figure 4.4. The initial UTE panel, with the IBM input value entered

3. In the Deployment Location pop-up, expand the WebSphere ESB Server V6 location, select the underlying localhost server, and then click Finish.

4. The client will now start to run -- first starting the server, then publishing the test client and using it to access the StockQuoteMediation module with the specified input parameter, IBM. As shown in Figure 4.5 below, the UTE displays the route taken by the message between mediation modules, together with the final result -- in this case, the expected price of 100.0:

Figure 4.5. UTE panel following invocation of getQuote operation with input IBM

5. Repeat the service invocation using the Invoke icon (first icon in the palette at top right of editor). Enter values such as IBMold and xxx to verify the expected results -- 80.0 and null, respectively.
Further checking of output logging
Whilst the previous section verified that the flows work as expected, using the UTE alone gives little detail about what is actually happening to messages during flow invocation. There are two ways to obtain this such information: checking the SystemOut log file, and checking the Message Logger default database.

Checking the SystemOut log file
As mentioned previously, WebSphere ESB outputs its system logging information by default to <ESB-install-directory>/profiles/ESB01/logs/server1/SystemOut.log. Open this file and examine the contents, concentrating on the end of the file where the output from the UTE test runs is stored. Thanks to the detailed logging specified in the previous section, "Setting up the test environment," the full messages will be printed at the entry and exit point of each mediation primitive. For example, Listing 2 below shows the information logged at the point of exit from the XSL Transformation primitive, with an input name value of IBM:

Listing 2. Sample content of the WebSphere ESB SystemOut.log file, illustrating a typical message content, with important features highlighted
[26/01/06 11:46:34:430 GMT] 00000058 SystemOut 0
+++++++++++++++++++++++++++++++++++++++++
ProcessMessage
Flow metadata:
  Port type: {http://StockQuoteLibrary/StockQuoteService}StockQuoteService
  Operation: getQuote --> invoked operation
  Interaction: Request --> direction of mediation flow
  Component: com.ibm.ws.sca.internal.scdl.impl.ManagedComponentImpl@2db851d4 (description: null)
    (displayName: StockQuoteMediationMediation) (name: StockQuoteMediationMediation)
  Node name: XSLTransformation1 --> current mediation primitive
  Node display name: XSLTransformation1
  Node type (class): com.ibm.ws.sibx.mediation.primitives.xslt.XSLTMediation
  Node properties: {XSLTransform=xslt/XSLTransformation1_req_1.xsl, root=/body, associatedXSL=xslt/XSLTransformation1_req_1.xsl, sibxMayChangeMessage=true, XMMMap=xslt/XSLTransformation1_req_1.xmx, validateInput=false}
  Terminal name: out
  Terminal type: Output --> current mediation terminal
  Message: com.ibm.ws.sibx.smobo.impl.ServiceMessageObjectImpl@9511ec
    { <?xml version="1.0" encoding="UTF-8"?>
      <smo:smo xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
        xmlns:service="http://StockQuoteLibrary/StockQuoteService"
        xmlns:smo="http://www.ibm.com/websphere/sibx/smo/v6.0.1">
        <context/>
        <headers/>
        <SMOHeader> --> message header information
          <MessageUUID>fe40906-0990-0000-0080-91d5c1b505a5</MessageUUID>
          <Version>
            <Version>6</Version>
            <Release>0</Release>
            <Modification>1c</Modification>
          </Version>
          <MessageType>Request</MessageType>
        </SMOHeader>
      </smo:smo>
    </body xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xmlns:service="http://StockQuoteLibrary/StockQuoteService"
    xmlns:smo="http://www.ibm.com/websphere/sibx/smo/v6.0.1">
As mentioned earlier, the same information is available from the WebSphere Integration Developer Console, but it has a small buffer size (which can be adjusted in the Preferences settings) and therefore the SystemOut.log file is usually used in problem diagnosis. Also, in addition to logging the content of the output messages, the log file contains any output logging from Java code run on the server and written to System.out. Check for the output generated by the service implementation, StockQuoteServiceImpl.java.

**Checking the Message Logger default database**

The mediation flows defined here use several Message Logger primitives, whose function is to write the contents of messages passing through to a specified database. In deploying these primitives, we used the default Cloudscape database. To access it:

1. Access the Servers view and stop the server, since whilst running, it holds a lock on the Cloudscape database.
2. Run `<ESB install directory>\cloudscape\bin\embedded\cview.bat` to access a GUI interface to the database.
3. Select **File => Open** and select `<ESB install directory>\profiles\ESB01\databases\EsbLogMedDB` in the browser window.
4. Expand **Tables** in the side menu and click **MSGLOG**, which is the default table used to store the message contents.
5. Click on the **Data** tab in the right-hand panel. It gives access to the full information logged, including when and where (both primitive and module name) the message was logged, as well as the message content itself. Since the table is quite small, use the **Text Editor** icon to display the full message information in another editor, as shown in Figure 4.6 below. Since we chose the default Message Logger settings, only the message body is logged in this database table. Compare it to the contents of SystemOut.log to clarify the difference:
Creating and running via a JSP

Now that we’ve demonstrated that the mediation modules work as expected, we can put together a Web based front-end to the service using a JSP. (If you need more information on JSP technology, see Related topics below.)

To start with, we need to add a reference to the mediation module to act as an input point for the JSP:

1. Open the Assembly Diagram Editor for the StockQuoteMediation module.
2. From the icon palette on the left side of the editor, expand the green arrow icon and select the blue arrow icon (Stand-alone References).
3. Add this component to the editor canvas and wire it to the StockQuoteMediationMediation component, choosing No in the Add Wire window. The editor should now look like this:
4. Save the changes and hover the mouse over the new reference name (the 1..1 icon) to display its name. The name (StockQuoteServicePartner) acts as the entry point to the flow and will be used by the JSP below.

Now that a valid reference has been set up to act as an entry point to the flow, define an appropriate JSP:

1. In the Business Integration Navigator, right-click on StockQuoteMediation and select New => Other.
2. In the New window, expand Web (check the Show All Wizards check box if this option is not initially available), click JSP File, and then click Next.
3. Click Browse and expand StockQuoteMediationWeb in the new window. Select the underlying WebContent folder, which is where the JSP file will be stored to make it available to Web browsers.
4. Back in the New JSP File pop-up window, enter invokeStockQuoteService as the JSP name:
5. Click **Finish**. WebSphere Integration Developer generates a basic skeleton for this file and opens it in a JSP editor, with three tabs below the main pane: Design, Source, and Preview:

**Figure 4.9. JSP Editor with basic code skeleton displayed**

```
<%! page language="java" contentType="text/html; charset=ISO-8859-1"
    pageEncoding="ISO-8859-1"%>

<%@ page language="java" contentType="text/html; charset=ISO-8859-1"
    pageEncoding="ISO-8859-1"%>

<%@ page language="java" contentType="text/html; charset=ISO-8859-1"
    pageEncoding="ISO-8859-1"%>

<%-- Place content here. --%>
```

6. Select the **Source** tab and replace the existing template with the following code:
Listing 3. Code for the invokeStockQuoteService JSP

```html
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=ISO-8859-1">
<meta name="GENERATOR" content="IBM Software Development Platform">
<link href="theme/Master.css" rel="stylesheet" type="text/css">
<title>invokeStockQuoteService.jsp</title>
</head>
<body>
<%page language="java" contentType="text/html; charset=ISO-8859-1"
pageEncoding="ISO-8859-1"%>
<%@ page import="com.ibm.websphere.sca.ServiceManager" %>
<%@ page import="com.ibm.websphere.sca.Service" %>
<%@ page import="commonj.sdo.DataObject" %>
<%@ page import="com.ibm.websphere.bo.BOFactory" %>
<%@ page import="com.ibm.websphere.sca.ServiceBusinessException;" %>

<h1 align="center">StockQuote Application</h1>
<form action="invokeStockQuoteService.jsp" method="get">
<table border="0" cellpadding="5" cellspacing="0">
<tbody>
<tr>
<td>Enter a StockQuote name (e.g. IBM): </td>
<td><input type="text" name="name" size="30" maxlength="30"></td>
</tr>
</tbody>
</table>
<p><input type="submit" name="Submit" value="Submit"></p>
</form>

<% System.out.println("Client accessed");
   System.out.println("invokeStockQuoteService.jsp : Check for name");
%
// retrieve name value input in HTML text box
String name = request.getParameter("name");
String ref = "StockQuoteServicePartner";
if (name != null) {
    System.out.println("invokeStockQuoteService.jsp :
                        Retrieved stockquote name: " + name + " from browser");
    // now attempt to invoke the StockQuote mediation flow
    try {
        ServiceManager serviceManager = new ServiceManager();
        Service service = (Service) serviceManager.locateService(ref);

        // Locate the BOFactory via its 'well known' location name
        // com/ibm/websphere/bo/BOFactory
        // this is used to generate a Business Object -- the data structure used by
        // the message passed to the mediation flow
        BOFactory bofactory = (BOFactory) serviceManager.locateService("com/ibm/websphere/bo/BOFactory");
```

```
// Use BOFactory with service interface to create a correctly formatted input message
System.out.println("invokeStockQuoteService.jsp : Get message getQuote");
DataObject getQuote =
    bofactory.createByElement("http://StockQuoteLibrary/StockQuoteService", "getQuote");

// Now get hold of necessary part of this input message and add the value input at browser
System.out.println("invokeStockQuoteService.jsp : Get input parameter inputGetQuote");
DataObject inputGetQuote = getQuote.createDataObject("inputGetQuote");

System.out.println("invokeStockQuoteService.jsp : Set inputGetQuote::name = " + name);
inputGetQuote.setString("name", name);

DataObject getQuoteResponse = null;
try {
    // Now invoke the operation 'getQuote' operation - note, the 'getQuoteResponse'
    // object represents the returned message
    System.out.println("invokeStockQuoteService.jsp : Invoke getQuote with " + name);
    getQuoteResponse = (DataObject)service.invoke("getQuote", inputGetQuote);

    // Retrieve the output, using the format specified in the interface
    System.out.println("invokeStockQuoteService.jsp : Retrieving the outputGetQuote");
    DataObject outputGetQuote = getQuoteResponse.getDataObject("outputGetQuote");

    if (outputGetQuote == null) {
        out.println("No message returned - this may be due to the " + "flow encountering a Stop or Fail mediation primitive.");
    } else {
        System.out.println("invokeStockQuoteService.jsp : Retrieving the price");
        float result = outputGetQuote.getFloat("price");
        System.out.println("invokeStockQuoteService.jsp : outputGetQuote::price = " + result);
        // Finally print the return value to the browser page
        out.println("<p>" + name + " stock price: " + result + "</p>");
    }
}
catch (ServiceBusinessException sbe) {
    out.println("invokeStockQuoteService.jsp : ServiceBusinessException caught - content as follows:");
    out.println("invokeStockQuoteService.jsp : " + sbe.getMessage());
}
} catch (Exception e) {
    System.out.println(e);
}
</BODY>
</HTML>

The ref variable in the code, used to look up the service to invoke, has been set to the value of the reference displayed in the Assembly Diagram for the StockQuoteMediation module, as shown in Figure 4.7. The logging out.println is directed to the Web browser where the page generated by the JSP is displayed, whereas the logging to System.out.println is directed to the WebSphere ESB SystemOut log.

7. Save the changes and close the JSP editor.

The JSP is stored in the StockQuoteMediation module, so you need to redeploy it to the server in order for the changes to be picked up: access the Servers view, right-click on the localhost server instance, and select Restart Project => StockQuoteMediation.
Once the project has been restarted, check the Console output for errors, open a Web browser, and enter the URL:

http://localhost:9080/StockQuoteMediationWeb/invokeStockQuoteService.jsp

It will load a Web page that looks like this:

**Figure 4.10. StockQuoteService JSP Web page, with results for IBM StockQuote input**

![StockQuote Application](image)

As in the UTE section, experiment with different input values for the StockQuote name and verify that the expected response is returned. In particular, enter a name outside of the expected values IBM and IBMold, and check the SystemOut logging to verify that the alternative mediation route has been taken.

**Conclusion**

In this article we have developed and deployed a Web-based service and mediation flow using WebSphere Enterprise Service Bus and WebSphere Integration Developer. In particular, we have:

1. Developed a simple Java-based Web service, providing a basic stockQuote service.
2. Developed a mediation flow to connect to this service, with logic to route requests to different destinations depending on their message payload.
3. Deployed these projects to a running instance of WebSphere ESB server and invoked them using both the provided test environment and a stand-alone JSP front-end.
4. Explored the available output logging facilities.

I hope that this fairly simple example has provided a useful introduction to the extensive capabilities of WebSphere Enterprise Service Bus and WebSphere Integration Developer.

**Acknowledgements**

The author would like to thank Chris Markes of IBM for his help in developing this article.
## Downloadable resources

<table>
<thead>
<tr>
<th>Description</th>
<th>Name</th>
<th>Size</th>
</tr>
</thead>
<tbody>
<tr>
<td>Code samples in zip format</td>
<td>dWProjectInterchangeFile.zip (HTTP</td>
<td>FTP )</td>
</tr>
</tbody>
</table>
Related topics

- WebSphere Enterprise Service Bus library
- WebSphere Integration Developer documentation in IBM Knowledge Center
- Introduction to JavaServer Pages Technology

© Copyright IBM Corporation 2006
Trademarks
(www.ibm.com/developerworks/ibm/trademarks/)