Using the SQL integration service with WebSphere Lombardi Edition V7.2 and WebSphere Application Server V7

Danh Ngoc Phi

October 12, 2011

About this tutorial

WebSphere Lombardi Edition V7.2 (Author Environment) provides an Integrated Service to connect to the database management system (DBMS) that was specified during installation time – Microsoft® SQL server, DB2®, or Oracle®. This tutorial illustrates how to manipulate data in the DBMS using WebSphere Lombardi Edition (hereafter called Lombardi Edition) by using the Java® Naming and Directory Interface (JNDI) in WebSphere Application Server. In this tutorial, we use DB2 Version 9.7 Fix Pack 1.

Objectives

In this tutorial, you will learn how to:

- Work with the Authoring Environment in WebSphere Lombardi Edition.
- Create a Toolkit, a re-used library that can be used across numerous process applications.
- Create the following variables: input, output, private, and complex structure.
- Create an integrated service, interacting with the database by using the SQL integration.
- Use the parameter in the SQL integration.
- Debug a process application.
- Create a Human Service.
- Integrate the Human Service with the integration service.
- Create a database and table and configure access authority in DB2 for testing.
- Create a custom data source that you can use in Lombardi Edition.
• Test the custom data source.

**Prerequisites**


**System requirements**

You need WebSphere Lombardi Edition V7.2 running on Windows® XP or Windows 7 with access to DB2 V9.7.1.

**Duration**

This tutorial takes about 6 hours to complete.

**Creating a toolkit in the Authoring Environment**

A toolkit is a collection of library items that you can reuse in a variety of process applications, as shown in Figure 1.

**Figure 1. Toolkit and process application**

1. Login into the Lombardi Authoring Environment (Author Envir) with tw_admin user.
2. Create a new toolkit in the Lombardi Authoring Environment, as shown in Figure 2.

**Figure 2. Create a Toolkit in Lombardi Edition**

3. Fill in the pop up dialog as shown in Figure 3 and click the **Create** button.
Figure 3. Create a New Toolkit dialog

4. Open the Designer perspective, as shown in Figure 4.

Figure 4. Open the Designer

5. Create an Integration Service to connect to DB2 by clicking on the Integration Service item, as shown in Figure 5. You need to have a connection to DB2 to extract the data from DB2. Therefore, you need to have an Integration Service to do it.

Figure 5. Create a new Integration Service

6. Fill in the popup dialog as shown in Figure 6, then click the Finish button.
7. Design the workflow for the new Integration Service.
   On the right Common pallet, drag two components (Server Scriptlet and Server Script), which run on the server side, into the diagram as shown in Figure 7.

Figure 7. Component diagram of Retrieve Policy Details

8. Connect with the SQL Execute Statement.
   In the Author Environment, the SQL integration services are Java-based integrations that bind to a specific method in the teamworks.SQLConnector Java class. This means that you have many built-in services in the System Data toolkit to interact with the database. In this tutorial, we use the SQL Execute Statement.
   Under Toolkits, expand System Data, and select All. Then type SQL Execute to filter the components. Drag the SQL Execute Statement into the diagram, as shown in Figure 8a.
9. Wire all the components together and set reasonable names for the components:
   a. To set the name for **Untitled1**, click on **Untitled 1**. In the Properties panel, select **Step**, and type **Set Query** for the **name** property.
   b. Set the name for **Untitled2** as **Set Parameter**. The wired components look similar to Figure 8b.

**Figure 8a. Available Integration Services**

**Figure 8b. Wired components**

At this stage, we will have a brief overview about the workflow of the new toolkit. In the next step, you will store the data. Click the **Save** (or **Ctrl + S**) button to save the new toolkit.

**Creating variables for the toolkit**

1. Select the **Variables** tab and click on the **Add Input** button to add an input.
   The Default data type is "String". Check the **Has Default** check box and enter **PO_1234_56** as the default value, as shown in Figure 9.

**Figure 9. Variables tab**
2. Click on the **Add Output** button to add the output data. The integration service is retrieving data from the database. To add the output to return the data from the service, do the following steps:

   a. Set the name for the output as `policyDetails`, as shown in Figure 10.
   b. In the Variable Type section at the bottom of the screen, click the **New** button.

**Figure 10. Create a new data type**

![Create a new data type](image1.png)

`policyDetails` contains user-defined data such as the policy number, policy issue date, and so on. In the Author Environment, there is no similar data type, therefore you need to create a Complex Structure Type.

c. Enter `PolicyDetails` as the name for the new Complex Structure Type and click **Finish** (see Figure 11) to display an assist window for the new data type.

**Figure 11. New Variable type dialog**

![New Variable type dialog](image2.png)
d. In the Parameter section, click the Add button and fill in the Name property as PolicyNum. The default data type is "String", so you do not need to select it.

e. Click the Add button in the Parameter section, and fill in the Name property as PolicyIssueDate, as shown in Figure 12. Click the Select button to select data type.

**Figure 12. Define parameters**

![Define parameters](image1)

**Figure 13. Select Date as date type**

![Select Date as date type](image2)

f. Add new parameters for this complex data and click Add as shown in Figure 14.

**Figure 14. Complex Variable window**

![Complex Variable window](image3)

g. Click Save and press the Close button to navigate to the Retrieve Policy Details toolkit.
You need an SQL string for the SQL integration to execute. You may have the following questions:

- Where is this SQL string defined? Is it inside this integration service or can it be dynamically set as an input?
- Does it contain any parameter? If yes, what kind of parameters?
- What sort of data is returned after executing this SQL? What data types are returned (a list, a string, a number, and so on)?
- Where do you store the return data for the SQL execution?

Depending on your requirements, you will have different answers and these answers affect the way you design the toolkit and application. For this tutorial, you need to define the SQL string inside the toolkit. The return value is a PolicyDetails type. This is the reason you need to have a private variable to store the SQL string, a private variable to store a parameter, and a private variable to store the returned result list of the SQL string. We will discuss further in the next section.

3. Add the following private variables:
   a. Add a private variable named SQL with the data type as String.
   b. Add a private variable named params with the data type as SQL Parameter. Check Is List.
   c. Add a private variable named policyDetailsList with the data type as PolicyDetails Check Is List (see Figure 15).
   d. Click Save.

**Figure 15. Total variables**

Implementing the components of the Retrieve Policy Details Integration Service

1. Implement the Set Query component:
   a. Click on the Set Query component (Figure 16) and click on Implementation.
Using the SQL integration service with WebSphere Lombardi Edition V7.2 and WebSphere Application Server V7

Figure 16. Component diagram of Retrieve Policy Details

b. In the Script section, enter the following SQL string:

```sql
SELECT POLICYNUM as POLICYNUM, POLICYISSUEDATE as POLICYISSUEDATE, POLICYSTATUS as POLICYSTATUS, POLICYCOMMENCEMENTDATE as POLICYCOMMENCEMENTDATE, POLICYMATURITYDATE as POLICYMATURITYDATE, PROPOSERNAME as PROPOSERNAME, FIRSTLIFEASSURED as FIRSTLIFEASSURED, FIRSTLIFEASSUREDNAME as FIRSTLIFEASSUREDNAME, GENDERFIRSTLIFEASSURED as GENDERFIRSTLIFEASSURED, SECONDLIFEASSURED as SECONDLIFEASSURED, SECONDLIFEASSUREDNAME as SECONDLIFEASSUREDNAME, GENDERSECONDLIFEASSURED as GENDERSECONDLIFEASSURED, VIP as VIP, CLAIMNUMBER as CLAIMNUMBER, CLAIMSTATUS as CLAIMSTATUS FROM DB2ADMIN.POLICYDETAILS WHERE POLICYNUM=?
```

c. In the Binding section, click on the Select button to select the variable to store the SQL string into a private variable, and then select the private sql variable (Figure 17).

Figure 17. Implementation of Set Query component

2. Implement the Set Parameter component:
   a. Select the Set Parameter component and click on its Implementation property (Figure 18).
   b. In the Script section, enter the following script (see Figure 18):

```javascript
tw.local.params = new tw.object.listOf.SQLParameter();
tw.local.params[0] = new tw.object.SQLParameter();
tw.local.params[0].value=tw.local.policyNum;
tw.local.params[0].type="VARCHAR";
```

Figure 18. Implementation of Set Parameter component

c. Implement the SQL Execute Statement component.
   d. You have already defined the SQL string and kept it in a private variable named sql. For this component, you need to pass the SQL string into this component as one of the
e. Click the SQL Execute Statement component and select the Data Mapping property.

f. Click the mapping button to map the SQL string as shown in Figure 19.

Figure 19. Mapping the data for the SQL Execute Statement component

<table>
<thead>
<tr>
<th>SQL Execute Statement</th>
<th>Data Mapping</th>
</tr>
</thead>
<tbody>
<tr>
<td>Start</td>
<td>Set Query</td>
</tr>
</tbody>
</table>

- Uncheck the "Use default" checkbox of parameters to edit its content. Click the mapping button to map the data to the private variable named `params`.
- Uncheck the "Use default" check of returnType, and enter `PolicyDetails` as its value as shown in Figure 20.
- Edit `dataSourceName` and insert `jdbc/sampleDB` as its value.
- For the `results(ANY)` field of the mapping output section, choose `policyDetailsList` as the output.

Figure 20. Full mapping data of the SQL Execute Statement component

### Debugging a process application

Assume that you have successfully configured the JNDI `jdbc/sampleDB` in WebSphere Application Server and the database is set up already. You will now debug the Retrieve Policy Details integration service to test whether the new service functions properly.
If you have not set up the database, see the Create the database and configure the database authority section.

If you have not created the JNDI, see the Set up the data source (JNDI) in WebSphere Application Server section.

**Figure 21. Component diagram of Retrieve Policy Details**

1. Click on the debug button of this service (Figure 21). In the web browser, you will see a new window specified for debugging. In the browser, there are two buttons, **Step** and **Run**. The **Step** button is used to debug every step in the service (Figure 22). By contrast, the **Run** button is purely an execution to the next break point or to the end of the work flow.

**Figure 22. Debug on the web browser**

2. Click on the **Step** button in the web browser to examine all the parameters and values of the service.

3. Go back to your Lombardi Authoring Environment to look over the token and data inside each component. The Lombardi Authoring Environment navigates to the Inspector perspective. You can see the token, indicating the current flow of the process. In the Inspector view, you can examine more details and the data flow of the current service.
   a. In the **Execution State** panel, click on **Server Script** to display the variables of the current Flow Object.
   b. In the **Variables** panel, click on **policyNum** to retrieve its value, as shown in Figure 23.
4. In the web browser, click the **Step** button. In Lombardi, the token moves to the **SQL Execute Statement** component.

5. In the web browser, click the **Step** button, **not** the **Step Over** button, to debug inside the **SQL Execute Statement** component. In Lombardi Edition, the token moves into this component as shown in Figure 24.

**Figure 24. Token**

6. In the web browser, click the **Step** button. You can see the returned results similar to what is shown in Figure 25.
Using the SQL integration service with WebSphere Lombardi Edition V7.2 and WebSphere Application Server V7

Figure 25. Debug in web browser

![WebSphere Lombardi Edition Service Debug](image)

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>deleteSourceName</td>
<td>String</td>
<td>jobbs/sampleDB</td>
</tr>
<tr>
<td>maxOccurs</td>
<td>Integer</td>
<td>1</td>
</tr>
<tr>
<td>parameters</td>
<td>SQLParameter[]</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>&lt;object type=&quot;SQLParameter&quot;&gt;</td>
</tr>
<tr>
<td></td>
<td></td>
<td>&lt;/object&gt;</td>
</tr>
</tbody>
</table>

**Figure 26. Server Script**

7. In the web browser, click the **Step** button to debug the next flow. Click the **Step** button to finish debugging.

By going through these steps, you can understand and analyze the data flow behind the scenes and monitor the token flow among the activities and flow objects.

**Refining the output of the Integration Service**

As you can see, the results in the debugging mode are in a list. However, the SQL Integration Service retrieves the PolicyDetails data by the policyNum, which is a primary key of the PolicyDetails table. Hence, the output of the service is a single record, not a list.

1. Drag the **Server Script** into the diagram, between the **SQL Execute Statement** and the **end** (Figure 26).

**Figure 26. Server Script**
2. Rename the new component as `Refine Output` and wave the new component (Figure 27).

**Figure 27. New diagram of the integration services**

![Diagram](image)

3. Click on the **End** line. Uncheck **Show End State**, as shown in Figure 28.

**Figure 28. Modify the connection line**

![Behavior settings](image)

4. Click the **Refine Output** Server Script, select **Implementation**, and put the script shown in Listing 1 into its **Script** field (see Figure 29). This script is used to assign the first record in the return result list to the output variable.

**Listing 1. Code of the Refine Output**

```java
if(tw.local.policyDetailsList.listLength > 0){
    tw.local.policyDetails = tw.local.policyDetailsList[0];
}
```

**Figure 29. Implementation of the Refine Output**

![Script implementation](image)

---

**Designing a Human Service (Coach) to capture the return data from the database integration service**

You can design a Human Service in a process application. However, in this tutorial we have chosen to design the Human Service inside this toolkit because we need to re-use this Human service in many process applications. Putting it into your toolkit or process application is up to your specific requirements.

1. In the **Designer** view, click the + button of the **User Interface**, and select **Human Service** as shown in Figure 30.
Figure 30. Location of Human Service

2. Enter Select Policy in the new popup dialog and click the Finish button.
3. In the Select Policy service, switch to the Variables tab as shown in Figure 31.

Figure 31. Variables tab

4. Click the Add Private button, and put policyDetails as its name and select PolicyDetails as its variable type. Check Has Default as shown in Figure 32.

Figure 32. Check Has Default values

5. Go back to the Diagram view and drag a Coach into the diagram, as shown in Figure 33.
6. Click **Save** and double-click on the new Coach to navigate into the **Coach** view (Figure 34).

**Figure 34. Design a Coach**

7. Drag the **policyDetails** variable in the **Variables** panel into the Coach (Figure 35).

**Figure 35. Variables into a Coach**

8. Select the **Policy Details** panel. In the **Properties** panel, select **Section** and increase the **# of Columns** field to **2**, as shown in Figure 36.

**Figure 36. Refine the Policy Details’ properties**

9. Drag the **Button Group** in the **Controls** panel right below the **Policy Num** and re-arrange the GUIs.

10. Select the new button (**Button**), select **Presentation**, and change the Label of this button into **Search**, as shown in Figure 37.
Figure 37. Add label for a button

<table>
<thead>
<tr>
<th>Button Details</th>
</tr>
</thead>
<tbody>
<tr>
<td>Button Id: ButtonGroup1.Button0</td>
</tr>
<tr>
<td>Label: search</td>
</tr>
<tr>
<td>Is Cancel:</td>
</tr>
<tr>
<td>Create flow:</td>
</tr>
</tbody>
</table>

Validation Script

11. Drag the **OK** button into the **Policy Details** section, select the **Section Title**, and press the **Delete** button. Figure 38 shows the outlook of the new Coach.

**Figure 38. Outlook of the Coach**

12. Go back to the **Diagram** view and wave all the objects together (Figure 39).

**Figure 39. Diagram of Select Policy**

13. Select **Coach** and change its name to **Main GUI**, as shown in Figure 40.

**Figure 40. Change name of the new Coach**

14. Select **Implementation** and drag the **Retrieve Policy Details** Integration Service into the Human Service's diagram, as shown in Figure 41.
15. Select the **Sequence Flow** button and draw the diagram as shown in Figure 42.

**Figure 42. Diagram of Select Policy**

16. Select **Retrieve Policy Details**. In the **Data Mapping**, map the input to `tw.local.policyDetails.policyNum` and output to `tw.local.policyDetails`, as shown in Figure 43.

**Figure 43. Mapping the data**

17. Press the **Save** button and click the **Run** button to test the product. Click the **Search** button as shown in Figure 44.

**Figure 44. Run in web browser 1**

After clicking the **Search** button, you see the result as shown in Figure 45.
Creating the database and configure the database authority

Depending on the DB2 frontend, there are numerous ways to create a dedicated database for business interactions. This tutorial does not cover all the details to set up a database. However, it does cover the following high-level steps:

1. Create a database name called **SAMPLE**.
2. Ensure the security of the **SAMPLE** database by granting access to appropriate users as shown in Figure 46. In this tutorial, we chose **DB2ADMIN** as the granted user.
3. Create a table and populate the initial data into the new table:
   a. Create a table as follows:

```
CREATE TABLE DB2ADMIN.POLICYDETAILS ( POLICYNUM VARCHAR (10) NOT NULL,
    POLICYISSUEDATE DATE, POLICYSTATUS VARCHAR (10), POLICYCOMMENCEMENTDATE
    DATE, POLICYMATURITYDATE DATE, PROPOSERNAME VARCHAR (10),
    FIRSTLIFEASSURED VARCHAR (10), FIRSTLIFEASSUREDNAME VARCHAR (10),
    GENDERFIRSTLIFEASSURED VARCHAR (10), SECONDLIFEASSURED VARCHAR (10),
    SECONDLIFEASSUREDNAME VARCHAR (10), GENDERSECONDLIFEASSURED VARCHAR (10),
    VIP CHARACTER (1), CLAIMNUMBER INTEGER, CLAIMSTATUS VARCHAR (10),
    CONSTRAINT CC13975979827Z2 PRIMARY KEY ( POLICYNUM) ) ;
```
b. Populate the data as follows:

```
INSERT INTO SYSIBM.POLICYDETAILS (POLICYNUM, POLICYSTATUS, PROPOSERNAME, FIRSTLIFEASSURED, FIRSTLIFEASSUREDNAME, GENDERFIRSTLIFEASSURED, SECONDLIFEASSURED, SECONDLIFEASSUREDNAME, GENDERSECONDLIFEASSURED, VIP, CLAIMNUMBER, CLAIMSTATUS)
VALUES('PO_1234_56', 'VALID', 'DANH_PROP', 'FIRST_DANH', '1ST AS_N', 'MALE', '2ND_ASS', '2ND_AS_NA', 'MALE', 'Y', 1234, 'VALID');
```

---

**Set up the data source (JNDI) in WebSphere Application Server**

1. The assumptions are:
   a. DB2 is running and you have created a sample database to test with it. In this case, you are using the sample database that comes with the DB2 product. If the sample database does not exist, you may run the "DB2 First Steps" to create it.
   b. The WebSphere Application Server V7.0 – ProcessCenter01 service is started.
2. In a web browser, bring up the WebSphere Administrative Console. For example, `https://localhost:9043/ibm/console/logon.jsp`. The port number may be different in your environment. You can look in the `<WLE_HOME>/AppServer/profiles/Lombardi/logs/ProcessCenter01_Service.log` file to determine the port number.
3. Log in as `tw_user` as shown in Figure 47. The password is `tw_user`.

   **Figure 47. WebSphere Administrative console login**

4. In the navigation pane, go to **Resources > JDBC > Data sources**.
5. In All scopes, you see five data sources that were created during the product installation. One is the default "Datasource", and the other four are related to Lombardi Teamworks.
6. You will create a new data source that points to the sample database:
   a. Change the Scope to **Cell=LombardiCell01**.
   b. Click **New**.
   c. Enter a data source name, such as **SampleDB Data Source**.
   d. Enter a JNDI name, such as `jdbc/sampleDB` (Figure 48) and click **Next**.
e. You may choose to create a new Java™ Database Connectivity (JDBC) provider. Since
one already exists for DB2, select an existing JDBC provider.
f. Click **Select an existing JDBC Provider**. From the drop down list, select **TeamWorks Process Server DB2 JDBC Driver** as shown in Figure 49 and click **Next**.

**Figure 49. Select the JDBC provider**

g. For the driver type, choose **4**.
h. Enter the database name as **SAMPLE**.
i. Enter the database server name as **localhost**.
j. If the port number is not filled in, enter the database port number as shown in Figure 50. The default port number for DB2 is **50000**.
k. Click **Next**.

**Note:** In production environments, we recommend that you use a fully qualified host name rather than a localhost.
Figure 50. Database properties

I. Choose the **LombardiCell01/processdblogon** alias for all (Figure 51), except the mapping-configuration alias.

m. Click **Next**.

**Note**: Typically, you create a J2C authentication alias. Since you are simply using a different database table in the same DB2 instance, you can use the existing J2C aliases.

Figure 51. Security settings

n. On the Summary screen, double-check the values and click **Finish**, as shown in Figure 52.
Figure 52. Data source creation summary

![Data source creation summary diagram](image)

- o. Save to the master configuration and restart the server.
- p. Select the newly created data source and click the **Test connection** button. Make sure the connection is successful. If there are errors, fix them before proceeding.

7. We need to add user and password to this JNDI. In the data source page, click on the link `Sample Data Source Name`, as shown in Figure 53.

Figure 53. Data sources

![Data sources](image)

8. In the new window, click on **Custom properties** as shown in Figure 54.
9. Click on the **New** button as shown in Figure 55.

**Figure 55. Create a new property in Custom properties**

10. Type `user` in the Name text field (Figure 56). Fill your DB2 user value in the Value text field. We used `db2admin` in this tutorial. Click **Apply**.

**Figure 56. Create a new property**
After this step, you have one property with a DB2 user name.

11. You need to create another property containing the password as well (Figure 57).

**Figure 57. Create another new property**

![Property Creation Screen](image)

12. Click on **OK**.

13. Double check the overall **Custom properties** and click **Save**. The new properties are shown in Figure 58.

**Figure 58. New properties are displayed**

<table>
<thead>
<tr>
<th></th>
<th></th>
<th>UNIK</th>
</tr>
</thead>
<tbody>
<tr>
<td>⚫</td>
<td>user</td>
<td>db2admin</td>
</tr>
<tr>
<td>⚫</td>
<td>password</td>
<td>***</td>
</tr>
</tbody>
</table>

14. Test the connection and you see a message similar to the one shown in Figure 59.

**Figure 59. Testing the connection**

![Connection Test Result](image)

Now you have a JNDI ready to be retrieved.
Conclusion

In this tutorial, you learned how to create an integration service. This is critical when interacting with a database in any enterprise system. You also learned how to use a Human Service and toolkit, debug an application, and control a business data flow.

© Copyright IBM Corporation 2011
Trademarks
(www.ibm.com/developerworks/ibm/trademarks/)