SUN SEEBEYOND

eWAY™ ADAPTER FOR DB2 CONNECT USER'S GUIDE

Release 5.1.2

Part Number: 819-7373-10

Version 20061006100629

Contents

Chapter 1

Introducing the DB2 Connect eWay

Welcome to the DB2 Connect eWay Intelligent Adapter User's Guide. This document includes information about installing, configuring, and using the Sun Java Composite Application Suite (CAPS) DB2 Connect eWay Intelligent Adapter, referred to as the DB2 Connect eWay throughout this guide.

What's in This Chapter

1.1 About DB2 Connect

DB2 Connect, IBM's database management system, makes host data directly available to Personal Computers and LAN-based workstations. It connects desktop and palm-top applications to mainframe and minicomputer host databases, leveraging enterprise information no matter where it is.

DB2 Connect provides the application enablement and communication infrastructure for connecting Web, Windows, UNIX, Linux, OS/2 and mobile applications to S/390 and AS/400 data. It also enables secure access to legacy data through intranets, extranets or the public Internet, allowing you to build new Internet applications and extend existing applications.

DB2 Connect is included in many of the DB2 UDB products, providing extensive application programming tools for developing client-server and web applications using industry standard APIs such as ODBC, ADO, OLE DB, JDBC, SQLJ, DB2 CLI and Embedded SQL.

1.2 About the DB2 Connect eWay

The eWay enables eGate Integrator Projects to exchange data with external DB2 databases. This document describes how to install and configure the eWay.

Note: *The DB2 Connect eWay connects to DB2 via the IBM DB2 Connect driver, which is NOT packaged with the eWay. The product must be separately installed and configured.*

1.3 What's New in This Release

This 5.1.2 version release provides general maintenance fixes for the DB2 Connect Intelligent Adapter. The DB2 Connect Intelligent Adapter also includes the following changes and new features:

New for Version 5.1.0

- **Version Control**: An enhanced version control system allows you to effectively manage changes to the eWay components.
- **Manual Connection** Management: Establishing a connection can now be performed automatically (configured as a property) or manually (using OTD methods from the Java Collaboration).
- **Multiple Drag-and-Drop Component Mapping from the Deployment Editor**: The Deployment Editor now allows you to select multiple components from the Editor's component pane, and drop them into your Environment component.
- **Support for Runtime LDAP Configuration**: eWay configuration properties now support LDAP key values.
- **MDB Pool Size Support**: Provides greater flow control (throttling) by specifying the maximum and minimum MDB pool size.
- **Connection Retry Support**: Allows you to specify the number of attempts to reconnect, and the interval between retry attempts, in the event of a connection failure.
- **Editable OTD Support**: An existing OTD can be edited and saved using the OTD Wizard. This allows you to make minor changes to an OTD without having to completely recreate the OTD from scratch. The OTD is then rebuilt, saved, and then relaunched back to the same Java Collaboration or BPEL.
- **Connectivity Map Generator**: Generates and links your Project's Connectivity Map components using a Collaboration or Business Process.

Many of these features are documented further in the *Sun SeeBeyond eGate Integrator User's Guide* or the *Sun SeeBeyond eGate Integrator System Administrator Guide*.

1.4 About This Document

This guide explains how to install, configure, and operate the Sun Java Composite Application Platform Suite DB2 eWay Intelligent Adapter, referred to as the DB2 Connect eWay throughout this guide.

1.4.1 What's in This Document

This document includes the following chapters:

- Chapter 1**"Introducing the DB2 Connect eWay":** Provides an overview description of the product as well as high-level information about this document.
- Chapter 2**"Installing the DB2 Connect eWay"**: Describes the system requirements and provides instructions for installing the DB2 Connect eWay.
- Chapter 3**"Configuring the DB2 Connect eWay"**: Provides instructions for configuring the eWay to communicate with your legacy systems.
- Chapter 4 **"Using the eWay Database Wizard"**: Provides information about .sag files and using the DB2 Connect wizard.
- Chapter 5 **"Using DB2 Connect Operations"**: Provides database operations used to access the DB2 Connect database through activities in BPEL, or through methods called from a JCD Collaboration.
- Chapter 6**"Implementing the DB2 Connect Sample Projects"**: Provides instructions for installing and running the sample Projects.
- Appendix A **"Common DataType Conversions"**: Provides conversions between the DB2 Connect and OTD/Java datatypes and the value or size of the data element that can be used.

1.4.2 Scope

This document describes the process of installing, configuring, and running the DB2 Connect eWay. This document does not cover the Java methods exposed by this eWay. For information on the Java methods, download and view the DB2 Connect eWay Javadoc files from the Enterprise Manager.

1.4.3 Intended Audience

This guide is intended for experienced computer users who have the responsibility of helping to set up and maintain a fully functioning Java Composite Application Platform Suite system. This person must also understand any operating systems on which the Java Composite Application Platform Suite will be installed (Windows and UNIX), and must be thoroughly familiar with Windows-style GUI operations.

1.4.4 Text Conventions

The following conventions are observed throughout this document.

Table 1 Text Conventions

Text Convention	Used For	Examples
Bold	Names of buttons, files, icons, parameters, variables, methods, menus, and objects	▪ Click **OK**. ▪ On the **File** menu, click **Exit**. ▪ Select the **eGate.sar** file.
`Monospaced`	Command line arguments, code samples; variables are shown in ***bold italic***	`java -jar` ***`filename`***`.jar`
Blue bold	Hypertext links within document	See **Text Conventions** on page 10
Blue underlined	Hypertext links for Web addresses (URLs) or email addresses	http://www.sun.com

1.4.5 Screenshots

Depending on what products you have installed, and how they are configured, the screenshots in this document may differ from what you see on your system.

1.5 Related Documents

The following Sun documents provide additional information about the Java Composite Application Platform Suite:

- *eGate Integrator User's Guide*
- *Sun Java Composite Application Platform Suite Installation Guide*

1.6 Sun Microsystems, Inc. Web Site

The Sun Microsystems web site is your best source for up-to-the-minute product news and technical support information. The site's URL is:

http://www.sun.com

1.7 Documentation Feedback

We appreciate your feedback. Please send any comments or suggestions regarding this document to:

CAPS_docsfeedback@sun.com

Chapter 2

Installing the DB2 Connect eWay

This chapter describes how to install the DB2 Connect eWay.

What's in This Chapter

2.1 Before You Install

Open and review the **Readme.txt** file for the DB2 Connect eWay for any additional information or requirements, prior to installation. The **Readme.txt file** is located on the installation CD-ROM.

2.2 Installing the DB2 Connect eWay

The Enterprise Manager, a web-based application, is used to select and upload eWays and add-on files during the installation process. The following section describes how to install the components required for this eWay.

Refer to the readme for the latest information on:

- Supported Operating Systems
- System Requirements
- External System Requirements

When the Repository is running on a UNIX operating system, the eWays are loaded from the Enterprise Manager running on a Windows platform connected to the Repository server using Internet Explorer.

Note: *The correct environment setting is needed when using the type 2 jdbc driver on a Unix platform.*

2.2.1 Installing the DB2 Connect eWay on an eGate Supported System

After you have installed Core Products, do the following:

1. From the Sun Java Composite Application Platform Suite Installer, click on the **Click to install additional products** link (on the Administration tab).
2. Expand the **eWay** option.
3. From **Select Sun Java Composite Application Platform Suite Products to Install**, select the products for your **Sun Java Composite Application Platform Suite** and include the following:
   - **FileeWay** (the File eWay is used by most sample Projects)
   - **DB2ConnecteWay**
4. Once you have selected all of your products, click **Next** in the top-right or bottom-right corner of the **Select Sun Java Composite Application Platform Suite Products to Install** box.
5. From the **Selecting Files to Install** box, locate and select your first product's SAR file. Once you have selected the SAR file, click **Next**. Follow this procedure for each of your products. The Installing Files window appears after the last SAR file has been selected.
6. From the **Installing Files** window, review the product list. If it is correct, Click **Install Products**. The Enterprise Manager starts the installation.
7. When your product's installation is completed, click on the prompt, "**When installation completes, click here to continue**."

To upload the Sun SeeBeyond eWay™ Adapter for DB2 User's Guide, Help file, Javadoc, Readme, and sample Projects, do the following:

A. Expand the **Documentation** option.

B. Select **DB2ConnecteWayDocs.**

C. Click **Next** in the top-right or bottom-right corner of the **Select Sun Java Composite Application Platform Suite Products to Install** box.

Adding the eWay to an Existing Sun Java Composite Application Platform Suite Installation

It is possible to add the eWay to an existing Sun Java Composite Application Platform Suite installation.

Steps required to add an eWay to an Existing CAPS installation include:

1. Complete steps 1 through 6 on **Installing the DB2 Connect eWay on an eGate Supported System** on page 13.
2. Open the Enterprise Designer and select **Update Center** from the Tools menu. The Update Center Wizard appears.
3. For Step 1 of the wizard, simply click **Next**.

4 For Step 2 of the wizard, click the **Add All** button to move all installable files to the **Include in Install** field, then click **Next**.

5 For Step 3 of the wizard, wait for the modules to download, then click **Next**.

6 The wizard's Step 4 window displays the installed modules. Review the installed modules and click **Finish.**

When prompted, restart the IDE (Integrated Development Environment) to complete the installation.

2.3 After Installation

After installing the eWay, you will need to create a .jar file depending on what type driver you are using.

2.3.1 Configuring JAR Files for Enterprise Designer

Configuring the Type 2 Driver for the Legacy Driver

After installation, make sure that the following three DB2 Connect related paths are included in your System's PATH:

- C:\PROGRA~1\IBM\SQLLIB\BIN
- C:\PROGRA~1\IBM\SQLLIB\FUNCTION
- C:\PROGRA~1\IBM\SQLLIB\SAMPLES\REPL

Note: *The Type 2 driver does not support updatable result sets. To do **Insert** and **Update** you must use a prepared statement*

After installing the eWay, you will need to copy the **db2java.zip** file and load it, **before it can perform its intended functions. To create the .jar file, do the following:**

1 Locate the **db2java.zip** file in your DB2 Connect installation directory.

2 Copy the jar file to: `c:JavaCAPS51\edesigner\lib\ext`

3 Copy a valid license .jar file to the same location.

4 Restart Enterprise Designer if it's currently active.

Note: *Your **db2java.jar** file must be the same version as the DB2 Connect you are running; otherwise errors will occur because of incompatibility.*

Configuring the Type 4 Driver for the Universal Driver

After installing the eWay, you will need to copy the **db2jcc.jar** file and load it, **before it can perform its intended functions.**

1 Locate the **db2jcc.jar** file in your DB2 Connect installation directory.

2 Copy the zip file to: `c:JavaCAPS51\edesigner\lib\ext`

3. Copy a valid license .jar file to the same location.
4. Restart Enterprise Designer if it's currently active.

Note: *Your **db2jcc.zip** file must be the same version as the DB2 Connect you are running; otherwise errors will occur because of incompatibility.*

2.3.2 Configuring JAR Files for the Logical Host

When using **local:** as the URL value in the Environment properties, .jar files must be installed prior to running a DB2 Connect eWay Project.

Note: *The <egate_logicalhost>:\\is\domains\domain1\lib location only appears after running the createdomain script (i.e. createdomain.bat).*

Configuring the Type 2 Driver

1. Copy the Legacy driver, **db2java.zip,** to the following location:

   ```
   <egate_logicalhost>:\\is\domains\domain1\lib.
   ```
2. **Change the extension from .zip to .jar to create db2java.jar**.
3. Restart the Logical Host if it's currently active.

Configuring UNIX for the DB2 Connect eWay

Configuring the Logical Host on UNIX to use the DB2 Connect Type 2 JDBC Driver:

1. Copy the **db2java.zip** file to the following location:

   <egate_logicalhost>:\\is\domains\domain1\lib.
2. **Change the extension from .zip to .jar to create db2java.jar**.
3. Source the db2profile shell script in directory /export/home/db2inst1/sqllib/ prior to initiating the Logical Host.

   Where /export/home/ is the directory that the DB2 Instance was installed and created in.

   Where db2inst1 is the name of the DB2 Instance.

   Example:

   cd /export/home/db2inst1/sqllib/

   .db2profile

Note: *AIX Customers running the 32 bit version of the Logical Host with the 64 bit version of DB2 Connect should note that the current default LIBPATH for the 64 bit version of DB2 Connect on AIX (i.e. LIBPATH=/opt/CA/SharedComponents/lib:/export/home/db2inst1/sqllib/lib) points to the 64 bit version of the DB2 Connect Shared Libraries. If you are running the 32 bit version of the Logical Host, you must specify the 32 bit version of the DB2 Connect Shared Libraries in the LIBPATH on AIX prior to starting up the Logical Host.*

Example:

export LIBPATH=/opt/CA/SharedComponents/lib:/export/home/db2inst1/sqllib/lib32

4 Restart the Logical Host if it's currently active.

2.3.3 Extracting the Sample Projects and Javadocs

The DB2 Connect eWay includes sample Projects and Javadocs. The sample Projects are designed to provide you with a basic understanding of how certain database operations are performed using the eWay, while Javadocs provide a list of classes and methods exposed in the eWay.

Steps to extract the Javadoc include:

1. Click the Documentation tab of the Suite Installer, then click the Add-ons tab.
2. Click the DB2 Connect eWay Intelligent Adapter link. Documentation for the DB2 Connect eWay appears in the right pane.
3. Click the icon next to **Javadoc** and extract the ZIP file.
4. Open the index.html file to view the Javadoc.

Steps to extract the Sample Projects include:

1. Click the Documentation tab of the Suite Installer, then click the Add-ons tab.
2. Click the DB2 Connect eWay Intelligent Adapter link. Documentation for the DB2 Connect eWay appears in the right pane.
3. Click the icon next to **Sample Projects** and extract the ZIP file. Note that the **DB2Connect_eWay_Sample.zip** file contains two additional ZIP files for each sample Project.

   Refer to **Importing a Sample Project** on page 64 for instructions on importing the sample Project into your repository via the Enterprise Designer.

2.4 ICAN 5.0 Project Migration Procedures

This section describes how to transfer your current ICAN 5.0 Projects to the *Sun Java Composite Application Platform Suite* 5.1.2. To migrate your ICAN 5.0 Projects to the *Sun Java Composite Application Platform Suite* 5.1.2, do the following:

Export the Project

1. Before you export your Projects, save your current ICAN 5.0 Projects to your Repository.
2. From the Project Explorer, right-click your Project and select **Export** from the shortcut menu. The Export Manager appears.

3 Select the Project that you want to export in the left pane of the Export Manager and move it to the Selected Projects field by clicking the **Add to Select Items** (arrow) button, or click **All** to include all of your Projects.

4 In the same manner, select the Environment that you want to export in the left pane of the Export Manager and move it to the Selected Environments field by clicking the **Add to Select Items** (arrow) button, or click **All** to include all of your Environments.

5 Browse to select a destination for your Project ZIP file and enter a name for your Project in the **ZIP file** field.

6 Click **Export** to create the Project ZIP file in the selected destination.

Install Sun Java Composite Application Platform Suite 5.1.2

7 Install the Composite Application Platform Suite 5.1.2, including all eWays, libraries, and other components used by your ICAN 5.0 Projects.

8 Start the Composite Application Platform Suite 5.1.2 Enterprise Designer.

Import the Project

9 From the Composite Application Platform Suite 5.1.2 Enterprise Designer's Project Explorer tree, right-click the Repository and select **Import Project** from the shortcut menu. The Import Manager appears.

10 Browse to and select your exported Project file.

11 Click **Import**. A warning message, **"Missing APIs from Target Repository**," may appear at this time. This occurs because various product APIs were installed on the ICAN 5.0 Repository when the Project was created, that are not installed on the Composite Application Platform Suite 5.1.2 Repository. These APIs may or may not apply to your Projects. You can ignore this message if you have already installed all of the components that correspond to your Projects. Click **Continue** to resume the Project import.

12 Close the Import Manager after the Project is successfully imported.

Deploy the Project

13 A new Deployment Profile must be created for each of your imported Projects. When a Project is exported, the Project's components are automatically *"checked in"* to Version Control to write-protected each component. These protected components appear in the Explorer tree with a red padlock in the bottom-left corner of each icon. Before you can deploy the imported Project, the Project's components must first be *"checked out"* of Version Control from both the Project Explorer and the Environment Explorer. To *"check out"* all of the Project's components, do the following:

A From the Project Explorer, right-click the Project and select **Version Control > Check Out** from the shortcut menu. The Version Control - Check Out dialog box appears.

B Select **Recurse Project** to specify all components, and click **OK**.

C. Select the Environment Explorer tab, and from the Environment Explorer, right-click the Project's Environment and select **Version Control > Check Out** from the shortcut menu.

D. Select **Recurse Environment** to specify all components, and click **OK**.

14. If your imported Project includes File eWay External Systems in the Environment, the Project's Environment must be reconfigured prior to deploying the Project. To reconfigure your Environment, do the following:

    A. The properties file for the File External System now includes both inbound and outbound properties. If your Environment includes both inbound and outbound File External Systems, these can now be combined. Delete all but one of the File External Systems.

    B. From the Environment Explorer tree, right-click your remaining File External System, and select **Properties** from the shortcut menu. The Properties Editor appears.

    C. Set the inbound and outbound directory values, and click **OK**.

15. Deploy your Projects.

2.4.1 Database Migration Issues

Projects imported from previous ICAN versions can potentially display different results, depending on whether the 5.0.x Java Collaboration Definition (JCD) included multiple (insert/update/delete) operations. This only affects non-XA transactions. If you are using an XA transaction, then you can skip this section.

Example:

In 5.0.x, five new records are to be inserted into a table. If one of the records fails to insert (like having a duplicate key), all five records will not be inserted.

In 5.1.2, five new records are to be inserted into a table. If one of the records fails to insert (like having a duplicate key), the other four records will be inserted.

In order to achieve the same result as in 5.0.x ICAN versions, you can choose one of the methods below.

1. In the Connectivity Map, delete the link to the database external application, then reconnect the link and select XA.
2. Fill in the XA property for the database external system under the Environment.
3. Rebuild the Project.

If you want to keep the non-XA transaction, the following changes are required:

1. Add a **setAutoCommit()** method and set it to "false" before inserting any records. If you are using a while loop, then add this method before entering the loop.
2. Add a **Commit()** method after inserting all the records. If you are using a while loop, add this method after exiting the loop.
3. Add a Try/Catch Block. The Try Block contains your current JCD code.

4. Add The **rollback()** method to the Catch Block. This method rolls back any changes that might have taken place prior to the failure.
5. Rebuild the Project.

JCD Sample Code (new code in BOLD):

```
public class jcdAutoCommitManual
{
public com.stc.codegen.logger.Logger logger;
public com.stc.codegen.alerter.Alerter alerter;
public com.stc.codegen.util.CollaborationContext collabContext;
public com.stc.codegen.util.TypeConverter typeConverter;
public void receive( com.stc.connector.appconn.file.FileTextMessage
input, uniqueTable.UniqueTableOTD UniqueTable_1 )
        throws Throwable
    {
        try {
            UniqueTable_1.setAutoCommit( false );
            UniqueTable_1.getTEST().insert();
            UniqueTable_1.getTEST().setFIRSTNAME( "Rex" );
            UniqueTable_1.getTEST().setLASTNAME( "Chan1" );
            UniqueTable_1.getTEST().insertRow();
            UniqueTable_1.getTEST().setFIRSTNAME( "Rex" );
            UniqueTable_1.getTEST().setLASTNAME( "Chan2" );
            UniqueTable_1.getTEST().insertRow();
            UniqueTable_1.getTEST().setFIRSTNAME( "Rex" );
            UniqueTable_1.getTEST().setLASTNAME( "Chan1" );
            UniqueTable_1.getTEST().insertRow();
            UniqueTable_1.commit();
        } catch ( java.sql.SQLException sqle ) {
            UniqueTable_1.rollback();
        }
    }

}
```

2.5 Installing Enterprise Manager eWay Plug-Ins

The **Sun SeeBeyond Enterprise Manager** is a Web-based interface that allows you to monitor and manage your Composite Application Platform Suite applications. The Enterprise Manager requires an eWay specific “plug-in” for each different eWay you install. These plug-ins enable the Enterprise Manager to target specific alert codes for each eWay type, as well as to start and stop the inbound eWays.

The *Sun Java Composite Application Platform Suite Installation Guide* describes how to install Enterprise Manager. The *Sun SeeBeyond eGate Integrator System Administration Guide* describes how to monitor servers, Services, logs, and alerts using the Enterprise Manager and the command-line client.

The **eWay Enterprise Manager plug-ins** are available from the **List of Components to Download** under the Sun Java Composite Application Platform Suite Installer’s **DOWNLOADS** tab.

There are two ways to add the eWay Enterprise Manager plug-ins:

- From the **Sun SeeBeyond Enterprise Manager**

- From the **Sun Java Composite Application Platform Suite Installer**

To add plug-ins from the Enterprise Manager

1. From the **Enterprise Manager**'s Explorer toolbar, click **configuration**.
2. Click the **Web Applications Manager** tab, go to the **Auto-Install from Repository** tab, and connect to your Repository.
3. Select the application plug-ins you require, and click **Install**. The application plug-ins are installed and deployed.

To add plug-ins from the Sun Java Composite Application Platform Suite Installer

1. From the **Sun Java Composite Application Platform Suite Installer's Download tab**, select the Plug-Ins you require and save them to a temporary directory.
2. From the **Enterprise Manager**'s Explorer toolbar, click **configuration**.
3. Click the **Web Applications Manager** tab and go to the **Manage Applications** sub-tab.
4. Browse for and select the WAR file for the application plug-in that you downloaded, and click **Deploy**. The plug-in is installed and deployed.

Viewing Alert Codes

You can view and delete alerts using the Enterprise Manager. An alert is triggered when a specified condition occurs in a Project component. The purpose of the alert is to warn the administrator or user that a condition has occurred.

To View the eWay Alert Codes

1. Add the eWay Enterprise Manager plug-in for this eWay.
2. From the **Enterprise Manager**'s Explorer toolbar, click **configuration**.
3. Click the **Web Applications Manager** tab and go to the **Manage Alert Codes** tab.
4. Browse for and select the Alert Properties File for the application plug-in that you added. The Alert Properties Files are located in the **alertcodes** folder of your Sun Java Composite Application Platform Suite installation directory.
5. Click **Deploy**. The available alert codes for your application are displayed under **Results**. A listing of available alert codes is displayed in Table 2.

Table 2 Alert Codes for the DB2 Connect eWay

Alert Code\Description	Description Details	User Actions
DBCOMMON-CONNECT-FAILED000001=Failed to connect to database {0} on host {1}. Reason: The Pooled connection could not be allocated: [{2}]	Occurs during the initial database connection.	▪ Database is down; start your database. ▪ External configuration information is invalid. You may need to verify the following: ♦ Server name ♦ Database name ♦ User ♦ Password ♦ Port
DBCOMMON-CONNECT-FAILED000002=Operation failed because of a database connection error. Reason: [{0}]	Occurs while retrieving a connection from the database or the pool.	▪ Verify that the database has not terminated with unexpected errors.
DBCOMMON-CONNECT-FAILED000005=Connection handle not usable. Reason:[{0}]	The connection in the pool is stale and is not usable.	▪ Probably a database restart occurred causing the connection to be stale, retry the operation after the database is up.
DBCOMMON-XARESOURCE-FAILED000001=Unable to get XAResource for the database. Reason: [{0}]	Could not obtain XAResource for the connection.	▪ Check if the database supports XA and has been configured for Distributed Transactions.
DBCOMMON-XACONNECT-FAILED000001=Failed to connect to database {0} on host {1}. The XA connection could not be allocated: Reason [{2}]	Occurs during the initial database connection.	▪ Check if the database is configured for XA and if the database is running. ▪ External configuration information is invalid. You may need to verify the following: ♦ Server name ♦ Database name ♦ User ♦ Password ♦ Port
DBCOMMON-XASTART-FAILED000001=Unable to perform XAStart for the connection. Reason: [{0}]	A connection error has occurred which caused XASTART to fail.	▪ Check if the database is running, and that there are no network issues.
DBCOMMON-XAEND-FAILED000001=XAEnd failed. Reason: [{0}]	Error occurred during commit on XA connection.	▪ Look for the detailed error mentioned in the alert for the appropriate action.

Alert Code\Description	Description Details	User Actions
DBCOMMON-CANNOT-GET-ISOLATION-LEVEL=Unable to get isolationLevel for the transaction. Reason: [{0}]	Could not read transaction isolation information of the connection.	▪ Transaction isolation is one of the following constants: ♦ Connection.TRANSACTION_READ_UNCOMMITTED ♦ Connection.TRANSACTION_READ_COMMITTED ♦ Connection.TRANSACTION_REPEATABLE_READ ♦ Connection.TRANSACTION_SERIALIZABLE ♦ Connection.TRANSACTION_NONE

For information on Managing and Monitoring alert codes and logs, see the *Sun SeeBeyond eGate Integrator System Administration Guide.*

Chapter 3

Configuring the DB2 Connect eWay

This chapter describes how to set the properties of the DB2 Connect eWay.

What's in This Chapter

3.1 Creating and Configuring the DB2 Connect eWay

All eWays contain a set of parameters with properties that are unique to that eWay type. The DB2 Connect eWay properties are modified from these locations:

- **Connectivity Map**: These parameters most commonly apply to a specific component eWay, and may vary from other eWays (of the same type) in the Project.
- **Environment Explorer**: These parameters are commonly global, applying to all eWays (of the same type) in the Project. The saved properties are shared by all eWays in the DB2 Connect External System window.
- **Collaboration or Business Process**: DB2 Connect eWay properties may also be set from your Collaboration or Business process, in which case the settings will override the corresponding properties in the eWay's configuration file. Any properties that are not overridden retain their configured default settings.

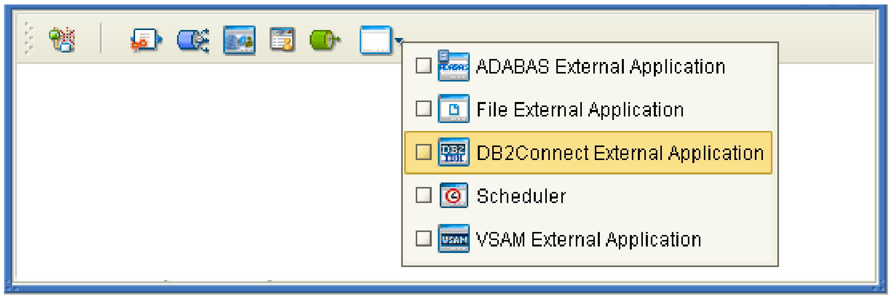
3.1.1 Selecting DB2 Connect as the External Application

To create a DB2 Connect eWay you must first create a DB2 Connect External Application in your Connectivity Map. DB2 Connect eWays are located between a DB2 Connect External Application and a Service. Services are containers for Collaborations, Business Processes, eTL processes, and so forth.

To create the DB2 Connect External Application

1. From the Connectivity Map toolbar, click the **External Applications** icon.
2. Select the **DB2 Connect External Application** from the menu (see Figure 1). The selected DB2 Connect External Application icon now appears on the Connectivity Map toolbar.

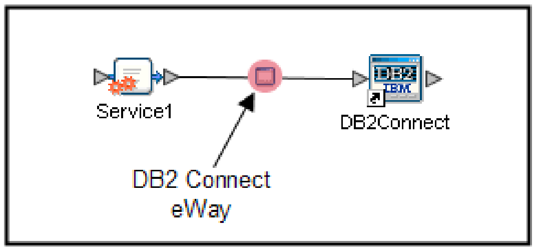
Figure 1 External Application menu


ADABAS External Application
File External Application
DB2Connect External Application
Scheduler
VSAM External Application

3 Drag the new **DB2 Connect External Application** from the toolbar onto the Connectivity Map canvas. This icon now represents an external DB2 Connect system.

From the Connectivity Map, you can associate (bind) the External Application to the Service to establish an eWay (see Figure 2).

Figure 2 eWay Location.



When DB2 Connect is selected as the External Application, it automatically applies the default DB2 Connect eWay properties, provided by the OTD, to the eWay that connects it with the Service. These properties can then be or modified for your specific system using the **Properties Editor**.

3.1.2 Configuring the DB2 Connect eWay Properties

A Project's eWay properties can be modified after the eWay has been established in the Connectivity Map and the Environment has been created.

Configuring the DB2 Connect eWay (Connectivity Map) Properties

1 From the **Connectivity Map**, double click the eWay icon located in the link between the associated External Application and the Service.

2 The eWay **Properties Editor** appears with a template containing the DB2 Connect eWay Connectivity Map properties. Make any necessary changes to the property values and click **OK** to save the settings.

Configuring the DB2 Connect eWay (Environment Explorer) Properties

1 From the **Environment Explorer** tree, right-click the DB2 Connect External System. Select **Properties** from the shortcut menu. The **Properties Editor** opens with the DB2 Connect eWay Environment properties.

2 Make any necessary changes to the Environment property values, and click **OK** to save the settings.

3.1.3 Using the DB2 eWay Properties Editor

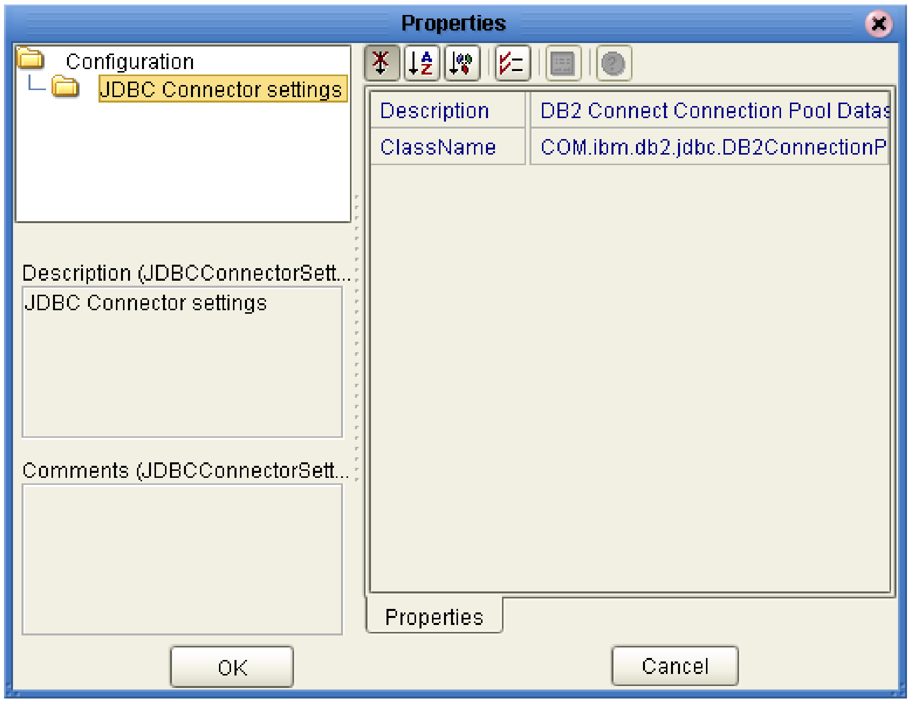
Modifications to the eWay configuration properties are made from the DB2 Connect eWay **Properties Editor**.

Modifying the Default eWay Configuration Properties

1 From the Connectivity Map or the Environment Explorer, open the Properties Editor to the DB2 Connect eWay default properties.

2 From the upper-right pane of the Properties Editor, select a subdirectory of the configuration directory. The parameters contained in that subdirectory are now displayed in the Properties pane of the Properties Editor. For example, if you click on the **connector** subdirectory, the editable **connector** parameters are displayed in the right pane (see Figure 3).

Figure 3 Properties Editor -- DB2 Connect eWay Properties



Description (JDBCConnectorSett...
JDBC Connector settings

Comments (JDBCConnectorSett...

Description	DB2 Connect Connection Pool Datas
ClassName	COM.ibm.db2.jdbc.DB2ConnectionP

3 Click on any property field to make it editable. For example, click on the **Description** property to edit the description value. If a property value is true/false or multiple choice, the field displays a submenu of property options.

4 Click on the ellipsis (...) in the properties field to open a separate configuration dialog box. This is helpful for large values that cannot be fully displayed in the parameter's property field. Enter the property value in the dialog box and click **OK**. The value is now displayed in the property field.

5. A description of each property is displayed in the **Description** pane when that property is selected. This provides a brief explanation of the required settings or options.
6. The **Comments** pane provides an area to record notes and information regarding the currently selected property. These comments are saved when you close the editor.
7. After modifying the configuration properties, click **OK** to close the Properties Editor and save your changes.

3.2 DB2 Connect eWay Connectivity Map Properties

The DB2 Connect eWay configuration parameters, accessed from the Connectivity Map, are organized into the following sections:

- **Outbound DB2 Connect eWay Connectivity Map Properties** on page 26
- **Inbound DB2 Connect eWay Connectivity Map Properties** on page 29

3.2.1 Outbound DB2 Connect eWay Connectivity Map Properties

The Outbound configuration parameters, accessed from the Connectivity Map, are organized into the following sections:

- **Outbound DB2 Connect Type 4 eWay - JDBC Connector Settings** on page 27
- **Outbound DB2 Connect XA eWay - JDBC Connector Settings** on page 27
- **Outbound DB2 Connect eWay - JDBC Connector Settings** on page 27
- **Outbound DB2 Connect non-Transactional Type 4 eWay - JDBC Connector Settings** on page 28
- **Outbound DB2 Connect Type 4 XA eWay - JDBC Connector Settings** on page 28
- **Outbound DB2 Connect non-Transactional eWay - JDBC Connector Settings** on page 29

Outbound DB2 Connect Type 4 eWay - JDBC Connector Settings

The **JDBC Connector Settings** section in the **Outbound DB2 Connect Type 4 eWay** Connectivity Map properties contains the top-level parameters displayed in Table 3.

Table 3 Outbound DB2 Connector Type 4 eWay - JDBC Connector Settings

Name	Description	Required Value
Description	Enter a description for the database.	A valid string. The default is **DB2 Connect Type 4 Connection Pool Datasource.**
ClassName	Specify the Java class in the JDBC driver that is used to implement the ConnectionPoolDataSource interface for a Type 4 driver.	A valid class name. The default is **com.ibm.db2.jcc.DB2ConnectionPoolDataSource**.

Outbound DB2 Connect XA eWay - JDBC Connector Settings

The **JDBC Connector Settings** section in the **Outbound DB2 Connect XA eWay** Connectivity Map properties contains the top-level parameters displayed in Table 4.

Table 4 Outbound DB2 Connect XA eWay - JDBC Connector Settings

Name	Description	Required Value
Description	Enter a description for the database.	A valid string. The default is **DB2 Connect XA Datasource**.
ClassName	Specify the Java class in the JDBC driver that is used to implement the ConnectionPoolDataSource interface for a XA driver.	A valid class name. The default is **COM.ibm.db2.jdbc.DB2XADataSource.**

Outbound DB2 Connect eWay - JDBC Connector Settings

The **JDBC Connector Settings** section in the **Outbound DB2 Connect eWay** Connectivity Map properties contains the top-level parameters displayed in Table 5.

Table 5 Outbound DB2 Connect eWay - JDBC Connector Settings

Name	Description	Required Value
Description	Enter a description for the database.	A valid string. The default is **DB2 Connect Connection Pool Datasource**.

Table 5 Outbound DB2 Connect eWay - JDBC Connector Settings

Name	Description	Required Value
ClassName	Specify the Java class in the JDBC driver that is used to implement the ConnectionPoolDataSource interface.	A valid class name. The default is **COM.ibm.db2.jdbc.DB2ConnectionPoolDataSource**.

Outbound DB2 Connect non-Transactional Type 4 eWay - JDBC Connector Settings

The **JDBC Connector Settings** section in the **Outbound DB2 Connect non-Transactional Type 4 eWay** Connectivity Map properties contains the top-level parameters displayed in Table 3.

Table 6 Outbound DB2 Connect non-Transactional Type 4 eWay - JDBC Connector Settings

Name	Description	Required Value
Description	Enter a description for the database.	A valid string. The default is **DB2 Connect Type 4 Connection Pool Datasource**
ClassName	Specify the Java class in the JDBC driver that is used to implement the ConnectionPoolDataSource interface for a non-transactional Type 4driver.	A valid class name. The default is **com.ibm.db2.jcc.DB2ConnectionPoolDataSource**.

Outbound DB2 Connect Type 4 XA eWay - JDBC Connector Settings

The **JDBC Connector Settings** section in the **Outbound DB2 Connect Type 4 XA eWay** Connectivity Map properties contains the top-level parameters displayed in Table 7.

Table 7 Outbound DB2 Connect Type 4 XA eWay - JDBC Connector Settings

Name	Description	Required Value
Description	Enter a description for the database.	A valid string. The default is **DB2 Connect Type 4 XA Datasource**.
ClassName	Specify the Java class in the JDBC driver that is used to implement the ConnectionPoolDataSource interface for a Type 4 XA driver.	A valid class name. The default is **com.ibm.db2.jcc.DB2XADataSource**.

Outbound DB2 Connect non-Transactional eWay - JDBC Connector Settings

The **JDBC Connector Settings** section in the **Outbound DB2 Connect non-Transactional eWay** Connectivity Map properties contains the top-level parameters displayed in Table 8.

Table 8 Outbound DB2 Connect non-Transactional eWay - JDBC Connector Settings

Name	Description	Required Value
Description	Enter a description for the database.	A valid string. The default is **DB2 Connect non-Transactional Connection Pool Datasource**.
ClassName	Specify the Java class in the JDBC driver that is used to implement the ConnectionPoolDataSource interface for a non-transactional driver.	A valid class name. The default is **COM.ibm.db2.jdbc.DB2ConnectionPoolDataSource**.

3.2.2 Inbound DB2 Connect eWay Connectivity Map Properties

The Inbound configuration parameters, accessed from the Connectivity Map, are organized into the following section:

- **Inbound DB2 Connect Type 4 eWay - Parameter Settings** on page 29

Inbound DB2 Connect Type 4 eWay - Parameter Settings

The **Parameter Settings** section of the **Inbound DB2 Connect Type 4 eWay** Connectivity Map properties contains the top-level parameters displayed in Table 9.

Table 9 Inbound DB2 Connect eWay Type 4

Name	Description	Required Value
Pollmilliseconds	Polling interval in milliseconds.	A valid numeric value. The default is **5000**.

Table 9 Inbound DB2 Connect eWay Type 4

Name	Description	Required Value
PreparedStatement	The Prepared Statement used for polling against the database.	The Prepared Statement must be the same Statement you created using the Database OTD Wizard. Only a SELECT Statement is allowed. Additionally, no place holders should be used. This is a SQL statement that cannot contain any input data (i.e. you cannot use "?" in the Prepared Query).

Inbound DB2 Connect eWay - Parameter Settings

The **Parameter Settings** section of the **Inbound DB2 Connect eWay** Connectivity Map properties contains the top-level parameters displayed in Table 10.

Table 10 Inbound DB2 Connect eWay Type 4

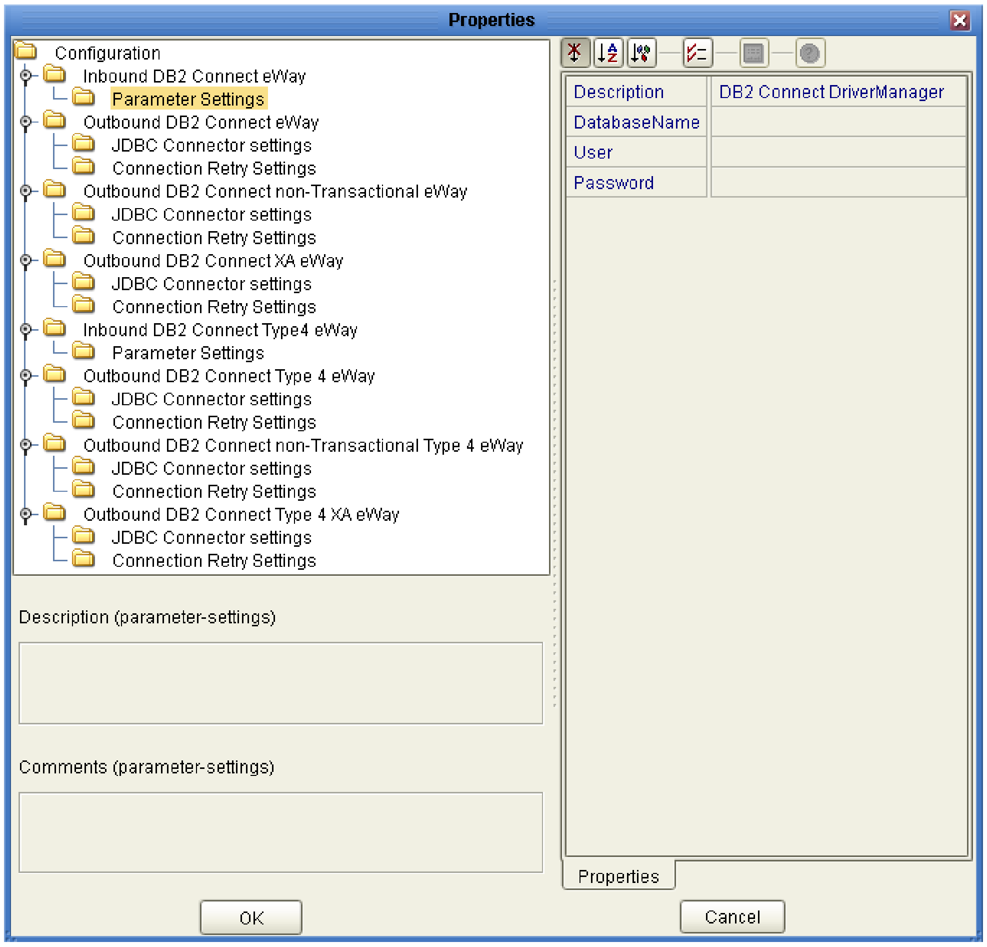
Name	Description	Required Value
Pollmilliseconds	Polling interval in milliseconds.	A valid numeric value. The default is **5000**.
PreparedStatement	The Prepared Statement used for polling against the database.	The Prepared Statement must be the same Statement you created using the Database OTD Wizard. Only a SELECT Statement is allowed. Additionally, no place holders should be used. This is a SQL statement that cannot contain any input data (i.e. you cannot use "?" in the Prepared Query).

3.3 DB2 Connect eWay Environment Explorer Properties

The DB2 Connect eWay configuration parameters, accessed from the Environment Explorer tree, are organized into the following sections (as displayed in **Figure 4 on page 32**:

- **Inbound DB2 Connect eWay - Parameter Settings** on page 32
- **Outbound DB2 Connect eWay - JDBC Connector Settings** on page 33
- **Outbound DB2 Connect eWay - Connection Retry Settings** on page 34
- **Outbound DB2 Connect non-Transactional eWay - JDBC Connector Settings** on page 34
- **Outbound DB2 Connect non-Transactional eWay - Connection Retry Settings** on page 35
- **Outbound DB2 Connect XA eWay - JDBC Connector Settings** on page 36
- **Outbound DB2 Connect XA eWay - Connection Retry Settings** on page 37
- **Inbound DB2 Connect Type 4 eWay - Parameter Settings** on page 37
- **Outbound DB2 Connect Type 4 eWay - JDBC Connector Settings** on page 38
- **Outbound DB2 Connect Type 4 eWay - Connection Retry Settings** on page 39
- **Outbound DB2 Connect non-Transactional Type 4 eWay - JDBC Connector Settings** on page 40
- **Outbound DB2 Connect non-Transactional Type 4 eWay - Connection Retry Settings** on page 41
- **Outbound DB2 Connect Type 4 XA eWay - JDBC Connector Settings** on page 42
- **Outbound DB2 Connect Type 4 XA eWay - Connection Retry Settings** on page 43

Figure 4 DB2 Connect eWay Environment Configuration


Properties
Configuration
Inbound DB2 Connect eWay
Parameter Settings
Outbound DB2 Connect eWay
JDBC Connector settings
Connection Retry Settings
Outbound DB2 Connect non-Transactional eWay
JDBC Connector settings
Connection Retry Settings
Outbound DB2 Connect XA eWay
JDBC Connector settings
Connection Retry Settings
Inbound DB2 Connect Type4 eWay
Parameter Settings
Outbound DB2 Connect Type 4 eWay
JDBC Connector settings
Connection Retry Settings
Outbound DB2 Connect non-Transactional Type 4 eWay
JDBC Connector settings
Connection Retry Settings
Outbound DB2 Connect Type 4 XA eWay
JDBC Connector settings
Connection Retry Settings
Description (parameter-settings)
Comments (parameter-settings)
Properties
OK
Cancel

Description	DB2 Connect DriverManager
DatabaseName	
User	
Password	

Inbound DB2 Connect eWay - Parameter Settings

The **Inbound DB2 Connect eWay > Parameter Settings** section of the DB2 Connect Environment properties, contains the top-level parameters displayed in Table 11.

Table 11 Inbound DB2 Connect eWay - Parameter Settings

Name	Description	Required Value
Description	Enter a description for the database.	A valid string. The default is **DB2 Connect DriverManager**.
DatabaseName	Specifies the name of the database instance used on the Server.	Any valid string.
User	Specifies the user name that the eWay uses to connect to the database.	Any valid string.
Password	Specifies the password used to access the database.	Any valid string.

Outbound DB2 Connect eWay - JDBC Connector Settings

The **Outbound DB2 Connect eWay > JDBC Connector Settings** section of the DB2 Connect Environment properties, contains the top-level parameters displayed in Table 12.

Table 12 Outbound DB2 Connect eWay - JDBC Connector Settings

Name	Description	Required Value
Description	Enter a description for the database.	A valid string. The default value is **DB2 Connect Connection Pool Datasource**.
DatabaseName	Specifies the name of the database instance used on the Server.	Any valid string.
User	Specifies the user name that the eWay uses to connect to the database.	Any valid string.
Password	Specifies the password used to access the database.	Any valid string.
DriverProperties	Use the JDBC driver that is shipped with this eWay. Often times the DataSource implementation needs to execute additional methods to assure a connection. You must identify the additional methods in the Driver Properties.	Any valid delimiter. Valid delimiters are: “<method-name-1>#<param-1>#<param-2>##.........<param-n>##<method-name-2>#<param-1>#<param-2>#........<param-n>##......##”. **For example**: To obtain a DB2 Connect trace, give the method a string for the email “settraceLevel#-1##settraceFile#c:/temp/db2trace.txt##”
Delimiter	This is the delimiter character to be used in the DriverProperties prompt.	The default is **#**.
MinPoolSize	The minimum number of physical connections the pool keeps available at all times. 0 (zero) indicates that there are no physical connections in the pool and new connections are created as needed.	A valid numeric value. The default is **0**.
MaxPoolSize	The maximum number of physical connections the pool keeps available at all times. 0 (zero) indicates that there is no maximum.	A valid numeric value. The default is **10**.
MaxIdleTime	The maximum number of seconds that a physical connection may remain unused before it is closed. 0 (zero) indicates that there is no limit.	A valid numeric value. The default is **0**.

Outbound DB2 Connect eWay - Connection Retry Settings

The **Outbound DB2 Connect eWay > Connection Retry Settings** section of the DB2 Connect Environment properties, contains the top-level parameters displayed in Table 13.

Table 13 Outbound DB2 Connect eWay - Connection Retry Settings

Name	Description	Required Value
ConnectionRetries	Specifies the number of retries to establish a connection with the DB2 Connect database upon a failure to acquire one.	an integer indicating the number of attempts allowed to establish a connection. The configured default is **0**.
ConnectionRetry Interval	Specifies the milliseconds of pause before each attempt to access the database. This setting is used in conjunction with the **Connection Retries** setting. For example: In the event that the eWay cannot connect to the Database, the eWay will try to reconnect to the database 10 times in 5 seconds intervals when the Connection Retries is 10 and the Connection Retry Interval is 5000.	An integer indicating the configured length of the time (in milliseconds) before each reattempt to access the destination file. The configured default is **1000** (1 second).

Outbound DB2 Connect non-Transactional eWay - JDBC Connector Settings

The **Outbound DB2 Connect non-Transactional eWay > JDBC Connector Settings** section of the DB2 Connect Environment properties, contains the top-level parameters displayed in Table 14.

Table 14 Outbound DB2 Connect non-Transactional eWay - JDBC Connector Settings

Name	Description	Required Value
Description	Enter a description for the database.	A valid string. The default value is **DB2 Connect non-Transactional Connection Pool Datasource**.
DatabaseName	Specifies the name of the database instance used on the Server.	Any valid string.
User	Specifies the user name that the eWay uses to connect to the database.	Any valid string.
Password	Specifies the password used to access the database.	Any valid string.

Table 14 Outbound DB2 Connect non-Transactional eWay - JDBC Connector Settings

Name	Description	Required Value
DriverProperties	Use the JDBC driver that is shipped with this eWay. Often times the DataSource implementation needs to execute additional methods to assure a connection. You must identify the additional methods in the Driver Properties.	Any valid delimiter. Valid delimiters are: “<method-name-1>#<param-1>#<param-2>##.........<param-n>##<method-name-2>#<param-1>#<param-2>#........<param-n>##......##”. **For example**: To obtain a DB2 Connect trace, give the method a string for the email “settraceLevel#-1##settraceFile#c:/temp/db2trace.txt##”
Delimiter	This is the delimiter character to be used in the DriverProperties prompt.	The default is **#**.
MinPoolSize	The minimum number of physical connections the pool keeps available at all times. 0 (zero) indicates that there are no physical connections in the pool and new connections are created as needed.	A valid numeric value. The default is **0**.
MaxPoolSize	The maximum number of physical connections the pool keeps available at all times. 0 (zero) indicates that there is no maximum.	A valid numeric value. The default is **10**.
MaxIdleTime	The maximum number of seconds that a physical connection may remain unused before it is closed. 0 (zero) indicates that there is no limit.	A valid numeric value. The default is **0**.

Outbound DB2 Connect non-Transactional eWay - Connection Retry Settings

The **Outbound DB2 Connect non-Transactional eWay > Connection Retry Settings** section of the DB2 Connect Environment properties, contains the top-level parameters displayed in Table 15.

Table 15 Outbound DB2 Connect non-Transactional eWay - Connection Retry Settings

Name	Description	Required Value
ConnectionRetries	Specifies the number of retries to establish a connection with the DB2 Connect database upon a failure to acquire one.	an integer indicating the number of attempts allowed to establish a connection. The configured default is **0**.

Table 15 Outbound DB2 Connect non-Transactional eWay - Connection Retry Settings

Name	Description	Required Value
ConnectionRetry Interval	Specifies the milliseconds of pause before each attempt to access the database. This setting is used in conjunction with the **Connection Retries** setting. For example: In the event that the eWay cannot connect to the Database, the eWay will try to reconnect to the database 10 times in 5 seconds intervals when the Connection Retries is 10 and the Connection Retry Interval is 5000.	An integer indicating the configured length of the time (in milliseconds) before each reattempt to access the destination file. The configured default is **1000** (1 second).

Outbound DB2 Connect XA eWay - JDBC Connector Settings

The **Outbound DB2 Connect XA eWay > JDBC Connector Settings** section of the DB2 Connect Environment properties, contains the top-level parameters displayed in Table 16.

Table 16 Outbound DB2 Connect XA eWay - JDBC Connector Settings

Name	Description	Required Value
Description	Enter a description for the database.	A valid string. The default value is **DB2 Connect XA Datasource**.
DatabaseName	Specifies the name of the database instance used on the Server.	Any valid string.
User	Specifies the user name that the eWay uses to connect to the database.	Any valid string.
Password	Specifies the password used to access the database.	Any valid string.
DriverProperties	Use the JDBC driver that is shipped with this eWay. Often times the DataSource implementation needs to execute additional methods to assure a connection. You must identify the additional methods in the Driver Properties.	Any valid delimiter. Valid delimiters are: “<method-name-1>#<param-1>#<param-2>##.........<param-n>##<method-name-2>#<param-1>#<param-2>#........<param-n>##......##”. **For example**: To obtain a DB2 Connect trace, give the method a string for the email “settraceLevel#-1##settraceFile#c:/temp/db2trace.txt##”
Delimiter	This is the delimiter character to be used in the DriverProperties prompt.	The default is **#**.

Table 16 Outbound DB2 Connect XA eWay - JDBC Connector Settings (Continued)

Name	Description	Required Value
MinPoolSize	The minimum number of physical connections the pool keeps available at all times. 0 (zero) indicates that there are no physical connections in the pool and new connections are created as needed.	A valid numeric value. The default is **0**.
MaxPoolSize	The maximum number of physical connections the pool keeps available at all times. 0 (zero) indicates that there is no maximum.	A valid numeric value. The default is **10**.
MaxIdleTime	The maximum number of seconds that a physical connection may remain unused before it is closed. 0 (zero) indicates that there is no limit.	A valid numeric value. The default is **0**.

Outbound DB2 Connect XA eWay - Connection Retry Settings

The **Outbound DB2 Connect XA eWay > Connection Retry Settings** section of the DB2 Connect Environment properties, contains the top-level parameters displayed in Table 17.

Table 17 Outbound DB2 Connect XA eWay - Connection Retry Settings

Name	Description	Required Value
ConnectionRetries	Specifies the number of retries to establish a connection with the DB2 Connect database upon a failure to acquire one.	an integer indicating the number of attempts allowed to establish a connection. The configured default is **0**.
ConnectionRetry Interval	Specifies the milliseconds of pause before each attempt to access the database. This setting is used in conjunction with the **Connection Retries** setting. For example: In the event that the eWay cannot connect to the Database, the eWay will try to reconnect to the database 10 times in 5 seconds intervals when the Connection Retries is 10 and the Connection Retry Interval is 5000.	An integer indicating the configured length of the time (in milliseconds) before each reattempt to access the destination file. The configured default is **1000** (1 second).

Inbound DB2 Connect Type 4 eWay - Parameter Settings

The **Inbound DB2 Connect Type 4 eWay > Parameter Settings** section of the DB2 Connect Environment properties, contains the top-level parameters displayed in Table 18.

Table 18 Inbound DB2 Connect Type 4 eWay - Parameter Settings

Name	Description	Required Value
Description	Enter a description for the database.	A valid string. The default is **DB2 Connect DriverManager**.
ServerName	Specifies the host name of the external database server.	Any valid string.
PortNumber	Specifies the I/O port number on which the server is listening for connection requests.	A valid port number. The default is **446**.
DatabaseName	Specifies the name of the database instance used on the Server.	Any valid string.
User	Specifies the user name that the eWay uses to connect to the database.	Any valid string.
Password	Specifies the password used to access the database.	Any valid string.

Outbound DB2 Connect Type 4 eWay - JDBC Connector Settings

The **Outbound DB2 Connect Type 4 eWay > JDBC Connector Settings** section of the DB2 Connect Environment properties, contains the top-level parameters displayed in Table 19.

Table 19 Outbound DB2 Connect Type 4 eWay - JDBC Connector Settings

Name	Description	Required Value
Description	Enter a description for the database.	A valid string. The default value is **DB2 Connect Type 4 Connection Pool Datasource**.
ServerName	Specifies the host name of the external database server.	Any valid string.
PortNumber	Specifies the I/O port number on which the server is listening for connection requests.	A valid port number. The default is **446**.
DatabaseName	Specifies the name of the database instance used on the Server.	Any valid string.
User	Specifies the user name that the eWay uses to connect to the database.	Any valid string.
Password	Specifies the password used to access the database.	Any valid string.

Table 19 Outbound DB2 Connect Type 4 eWay - JDBC Connector Settings (Continued)

Name	Description	Required Value
DriverProperties	Use the JDBC driver that is shipped with this eWay. Often times the DataSource implementation needs to execute additional methods to assure a connection. You must identify the additional methods in the Driver Properties.	Any valid delimiter. Valid delimiters are: "<method-name-1>#<param-1>#<param-2>##.........<param-n>##<method-name-2>#<param-1>#<param-2>#........<param-n>##......##". **For example**: To obtain a DB2 Connect trace, give the method a string for the email "settraceLevel#-1##settraceFile#c:/temp/db2trace.txt##"
Delimiter	This is the delimiter character to be used in the DriverProperties prompt.	The default is **#**.
MinPoolSize	The minimum number of physical connections the pool keeps available at all times. 0 (zero) indicates that there are no physical connections in the pool and new connections are created as needed.	A valid numeric value. The default is **0**.
MaxPoolSize	The maximum number of physical connections the pool keeps available at all times. 0 (zero) indicates that there is no maximum.	A valid numeric value. The default is **10**.
MaxIdleTime	The maximum number of seconds that a physical connection may remain unused before it is closed. 0 (zero) indicates that there is no limit.	A valid numeric value. The default is **0**.

Outbound DB2 Connect Type 4 eWay - Connection Retry Settings

The **Outbound DB2 Connect Type 4 eWay > Connection Retry Settings** section of the DB2 Connect Environment properties, contains the top-level parameters displayed in Table 20.

Table 20 Outbound DB2 Connect Type 4 eWay - Connection Retry Settings

Name	Description	Required Value
ConnectionRetries	Specifies the number of retries to establish a connection with the DB2 Connect database upon a failure to acquire one.	an integer indicating the number of attempts allowed to establish a connection. The configured default is **0**.

Table 20 Outbound DB2 Connect Type 4 eWay - Connection Retry Settings (Continued)

Name	Description	Required Value
ConnectionRetry Interval	Specifies the milliseconds of pause before each attempt to access the database. This setting is used in conjunction with the **Connection Retries** setting. For example: In the event that the eWay cannot connect to the Database, the eWay will try to reconnect to the database 10 times in 5 seconds intervals when the Connection Retries is 10 and the Connection Retry Interval is 5000.	An integer indicating the configured length of the time (in milliseconds) before each reattempt to access the destination file. The configured default is **1000** (1 second).

Outbound DB2 Connect non-Transactional Type 4 eWay - JDBC Connector Settings

The **Outbound DB2 Connect non-Transactional Type 4 eWay > JDBC Connector Settings** section of the DB2 Connect Environment properties, contains the top-level parameters displayed in Table 21.

Table 21 Outbound DB2 Connect non-Transactional Type 4 eWay - JDBC Connector Settings

Name	Description	Required Value
Description	Enter a description for the database.	A valid string. The default value is **DB2 Connect Type 4 Connection Pool Datasource**.
ServerName	Specifies the host name of the external database server.	Any valid string.
PortNumber	Specifies the I/O port number on which the server is listening for connection requests.	A valid port number. The default is **446**.
DatabaseName	Specifies the name of the database instance used on the Server.	Any valid string.
User	Specifies the user name that the eWay uses to connect to the database.	Any valid string.
Password	Specifies the password used to access the database.	Any valid string.

Table 21 Outbound DB2 Connect non-Transactional Type 4 eWay - JDBC Connector Settings

Name	Description	Required Value
DriverProperties	Use the JDBC driver that is shipped with this eWay. Often times the DataSource implementation needs to execute additional methods to assure a connection. You must identify the additional methods in the Driver Properties.	Any valid delimiter. Valid delimiters are: "<method-name-1>#<param-1>#<param-2>##.........<param-n>##<method-name-2>#<param-1>#<param-2>#........<param-n>##......##". **For example**: To obtain a DB2 Connect trace, give the method a string for the email "settraceLevel#-1##settraceFile#c:/temp/db2trace.txt##"
Delimiter	This is the delimiter character to be used in the DriverProperties prompt.	The default is **#**.
MinPoolSize	The minimum number of physical connections the pool keeps available at all times. 0 (zero) indicates that there are no physical connections in the pool and new connections are created as needed.	A valid numeric value. The default is **0**.
MaxPoolSize	The maximum number of physical connections the pool keeps available at all times. 0 (zero) indicates that there is no maximum.	A valid numeric value. The default is **10**.
MaxIdleTime	The maximum number of seconds that a physical connection may remain unused before it is closed. 0 (zero) indicates that there is no limit.	A valid numeric value. The default is **0**.

Outbound DB2 Connect non-Transactional Type 4 eWay - Connection Retry Settings

The **Outbound DB2 Connect non-Transactional Type 4 eWay > Connection Retry Settings** section of the DB2 Connect Environment properties, contains the top-level parameters displayed in Table 22.

Table 22 Outbound DB2 Connect non-Transactional Type 4 eWay - Connection Retry Settings

Name	Description	Required Value
ConnectionRetries	Specifies the number of retries to establish a connection with the DB2 Connect database upon a failure to acquire one.	an integer indicating the number of attempts allowed to establish a connection. The configured default is **0**.

Table 22 Outbound DB2 Connect non-Transactional Type 4 eWay - Connection Retry Settings (Continued)

Name	Description	Required Value
ConnectionRetry Interval	Specifies the milliseconds of pause before each attempt to access the database. This setting is used in conjunction with the **Connection Retries** setting. For example: In the event that the eWay cannot connect to the Database, the eWay will try to reconnect to the database 10 times in 5 seconds intervals when the Connection Retries is 10 and the Connection Retry Interval is 5000.	An integer indicating the configured length of the time (in milliseconds) before each reattempt to access the destination file. The configured default is **1000** (1 second).

Outbound DB2 Connect Type 4 XA eWay - JDBC Connector Settings

The **Outbound DB2 Connect Type 4 XA eWay > JDBC Connector Settings** section of the DB2 Connect Environment properties, contains the top-level parameters displayed in Table 23.

Table 23 Outbound DB2 Connect Type 4 XA eWay - JDBC Connector Settings

Name	Description	Required Value
Description	Enter a description for the database.	A valid string. The default value is **DB2 Connect Type 4 XA Datasource**.
ServerName	Specifies the host name of the external database server.	Any valid string.
PortNumber	Specifies the I/O port number on which the server is listening for connection requests.	A valid port number. The default is **446**.
DatabaseName	Specifies the name of the database instance used on the Server.	Any valid string.
User	Specifies the user name that the eWay uses to connect to the database.	Any valid string.
Password	Specifies the password used to access the database.	Any valid string.

Table 23 Outbound DB2 Connect Type 4 XA eWay - JDBC Connector Settings (Continued)

Name	Description	Required Value
DriverProperties	Use the JDBC driver that is shipped with this eWay. Often times the DataSource implementation needs to execute additional methods to assure a connection. You must identify the additional methods in the Driver Properties.	Any valid delimiter. Valid delimiters are: "<method-name-1>#<param-1>#<param-2>##.........<param-n>##<method-name-2>#<param-1>#<param-2>#........<param-n>##......##". **For example**: To obtain a DB2 Connect trace, give the method a string for the email "settraceLevel#-1##settraceFile#c:/temp/db2trace.txt##"
Delimiter	This is the delimiter character to be used in the DriverProperties prompt.	The default is **#**.
MinPoolSize	The minimum number of physical connections the pool keeps available at all times. 0 (zero) indicates that there are no physical connections in the pool and new connections are created as needed.	A valid numeric value. The default is **0**.
MaxPoolSize	The maximum number of physical connections the pool keeps available at all times. 0 (zero) indicates that there is no maximum.	A valid numeric value. The default is **10**.
MaxIdleTime	The maximum number of seconds that a physical connection may remain unused before it is closed. 0 (zero) indicates that there is no limit.	A valid numeric value. The default is **0**.

Outbound DB2 Connect Type 4 XA eWay - Connection Retry Settings

The **Outbound DB2 Connect Type 4 XA eWay > Connection Retry Settings** section of the DB2 Connect Environment properties, contains the top-level parameters displayed in Table 24.

Table 24 Outbound DB2 Connect Type 4 XA eWay - Connection Retry Settings

Name	Description	Required Value
ConnectionRetries	Specifies the number of retries to establish a connection with the DB2 Connect database upon a failure to acquire one.	an integer indicating the number of attempts allowed to establish a connection. The configured default is **0**.

Table 24 Outbound DB2 Connect Type 4 XA eWay - Connection Retry Settings (Continued)

Name	Description	Required Value
ConnectionRetry Interval	Specifies the milliseconds of pause before each attempt to access the database. This setting is used in conjunction with the **Connection Retries** setting. For example: In the event that the eWay cannot connect to the Database, the eWay will try to reconnect to the database 10 times in 5 seconds intervals when the Connection Retries is 10 and the Connection Retry Interval is 5000.	An integer indicating the configured length of the time (in milliseconds) before each reattempt to access the destination file. The configured default is **1000** (1 second).

Chapter 4

Using the eWay Database Wizard

This chapter describes how to use the DB2 Connect eWay Database Wizard to build OTD's.

What's in This Chapter

4.1 Using the Database OTD Wizard

The Database OTD Wizard generates OTDs by connecting to external data sources and creating corresponding Object Type Definitions. The OTD Wizard can create OTDs based on any combination of Tables and Stored Procedures or Prepared SQL Statements.

Field nodes are added to the OTD based on the Tables in the external data source. Java method and parameter nodes are added to provide the appropriate JDBC functionality. For more information about the Java methods, refer to your JDBC developer's reference.

Note: *Database OTDs are not messagable. For more information on messagable OTDs, see the eGate Integrator User's Guide.*

4.2 Steps to Create a New DB2 Connect OTD

The following steps are required to create a new OTD for the DB2 Intelligent Adapter eWay.

- **Select Wizard Type** on page 46
- **Connect to Database** on page 46
- **Select Database Objects** on page 48
- **Select Table/Views/Aliases** on page 49
- **Select Procedures** on page 51

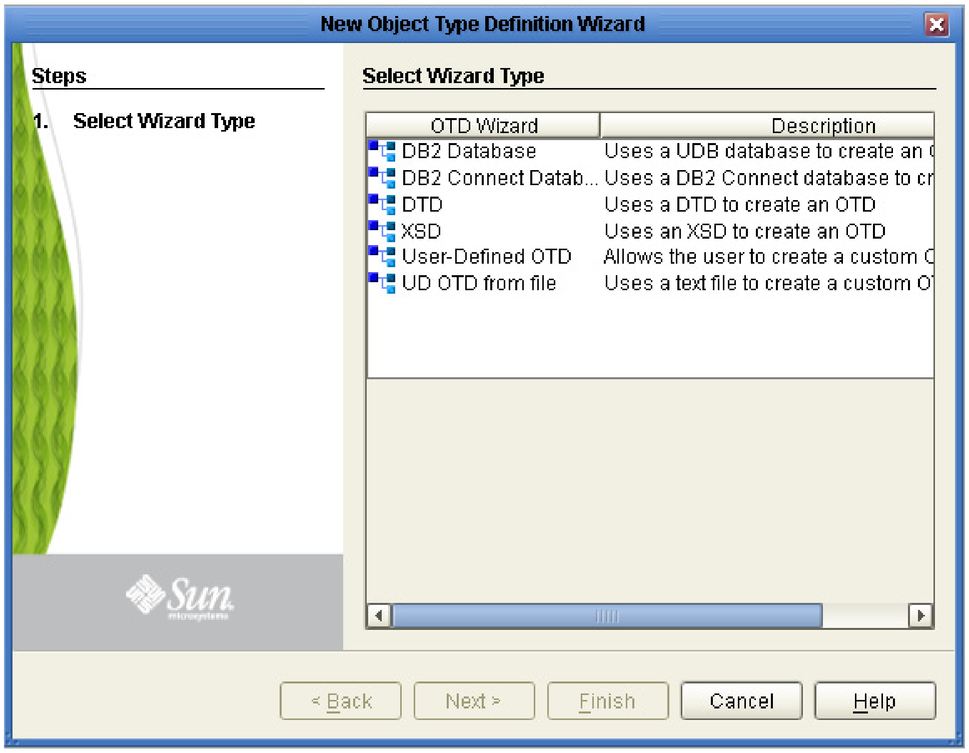
- **Add Prepared Statements** on page 55
- **Specify the OTD Name** on page 57
- **Review Selections** on page 58

4.2.1 Select Wizard Type

On the Enterprise Explorer, right click on the project and select **Create an Object Type Definition** from the shortcut menu.

From the OTD Wizard Selection window, select the **DB2 Connect Database** and click **Next** (See Figure 5).

Figure 5 OTD Wizard Selection

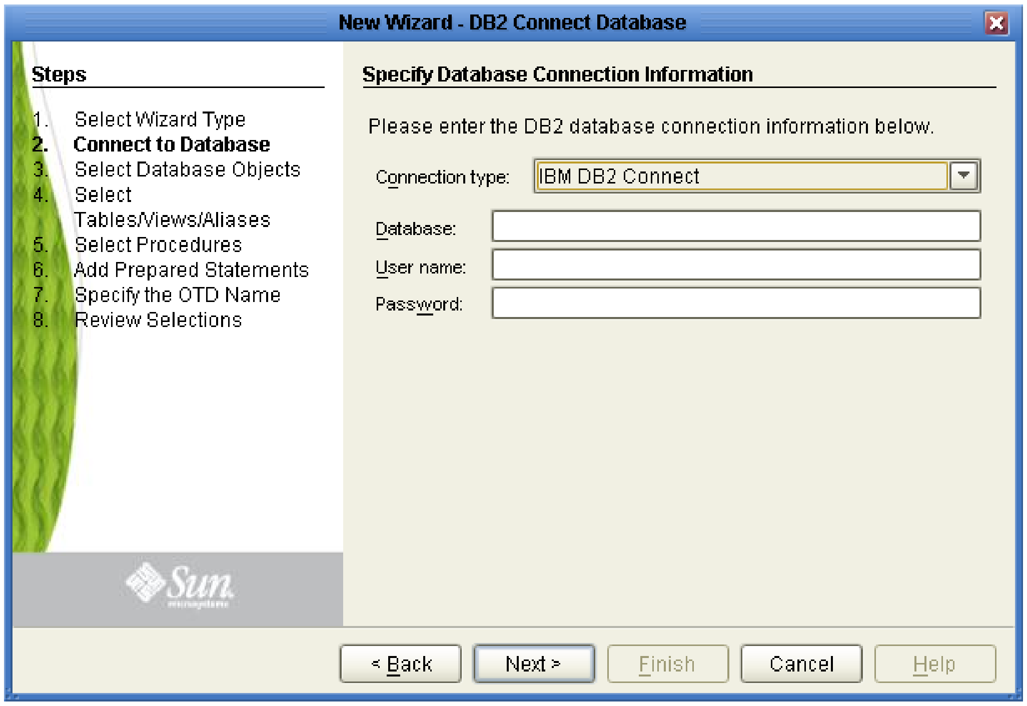


OTD Wizard	Description
DB2 Database	Uses a UDB database to create an (
DB2 Connect Datab...	Uses a DB2 Connect database to cr
DTD	Uses a DTD to create an OTD
XSD	Uses an XSD to create an OTD
User-Defined OTD	Allows the user to create a custom C
UD OTD from file	Uses a text file to create a custom O

4.2.2 Connect to Database

1. Select the **Connection type** using the drop-down list (see Figure 6). The resulting Connection Information fields displayed will depend on your selection (i.e. selecting a Type 2 or Type 4 connector).
2. Specify the applicable connection information (depending on the Connection type) for your database including:

Figure 6 Type 2 Database Connection Information



Specify Database Connection Information

Please enter the DB2 database connection information below.

Connection type: IBM DB2 Connect

Database:

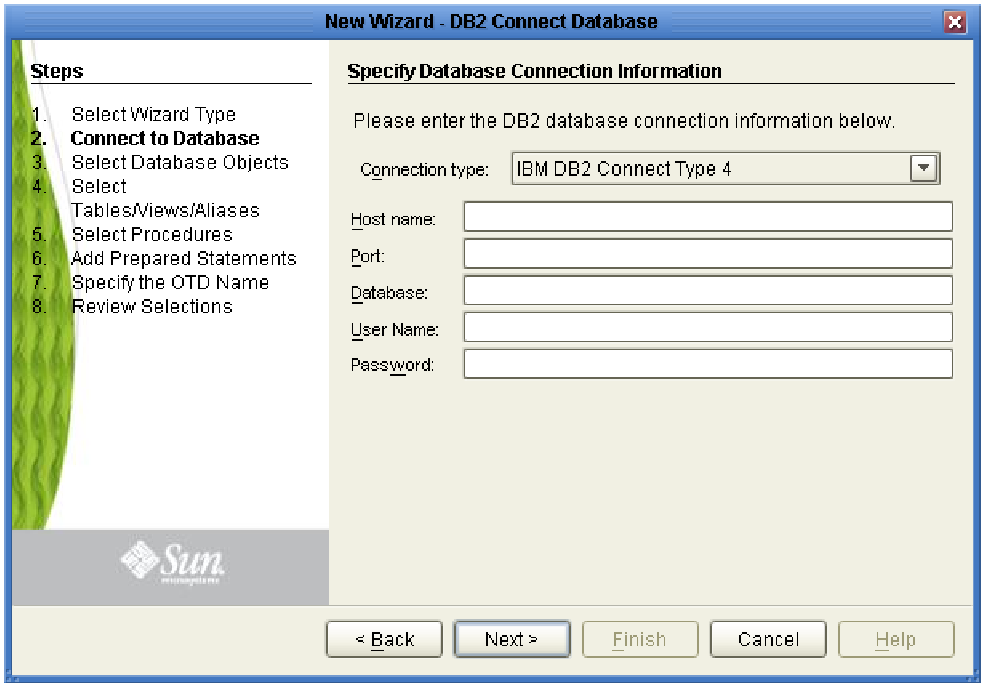
User name:

Password:

For the Type 2 Connection (shown above):

- **Database** - The name of the database instance.
- **User Name** - The user name that the eWay uses to connect to the database.
- **Password** - The password used to access the database.

Figure 7 Type 4 Database Connection Information



New Wizard - DB2 Connect Database

Steps

1. Select Wizard Type
2. **Connect to Database**
3. Select Database Objects
4. Select Tables/Views/Aliases
5. Select Procedures
6. Add Prepared Statements
7. Specify the OTD Name
8. Review Selections

Specify Database Connection Information

Please enter the DB2 database connection information below.

Connection type: IBM DB2 Connect Type 4

Host name:

Port:

Database:

User Name:

Password:

For the Type 4 Connection(shown above):

- **Host Name** - The server where DB2 Connect resides.
- **Port** - The port number of DB2 Connect.
- **Database** - The name of the database instance dor the DB2 running on Windows/ Unix. The name of the locator for the DB2 running in z/OS or AS/400.
- **User Name** - The user name that the eWay uses to connect to the database.
- **Password** - The password used to access the database.

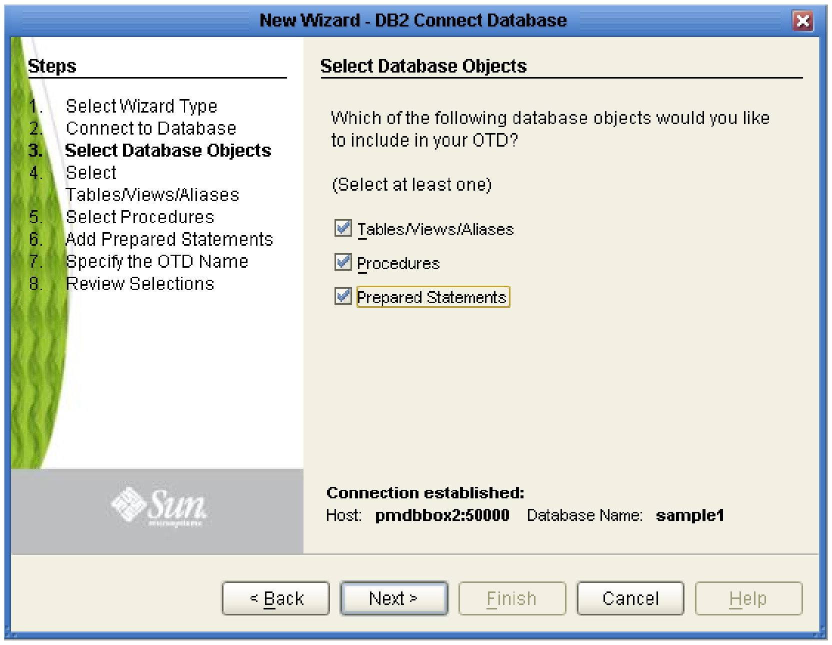
3 Click **Next**. The Select Database Objects window appears.

4.2.3 Select Database Objects

1 When selecting Database Objects, you can select any combination of **Tables**, **Views**, **Procedures**, or **Prepared Statements** you would like to include in the .otd file. Click **Next** to continue. See Figure 8.

Note: *Views are read-only and are for informational purposes only.*

Figure 8 Select Database Objects



Which of the following database objects would you like to include in your OTD?

(Select at least one)

[✓] Tables/Views/Aliases
[✓] Procedures
[✓] Prepared Statements

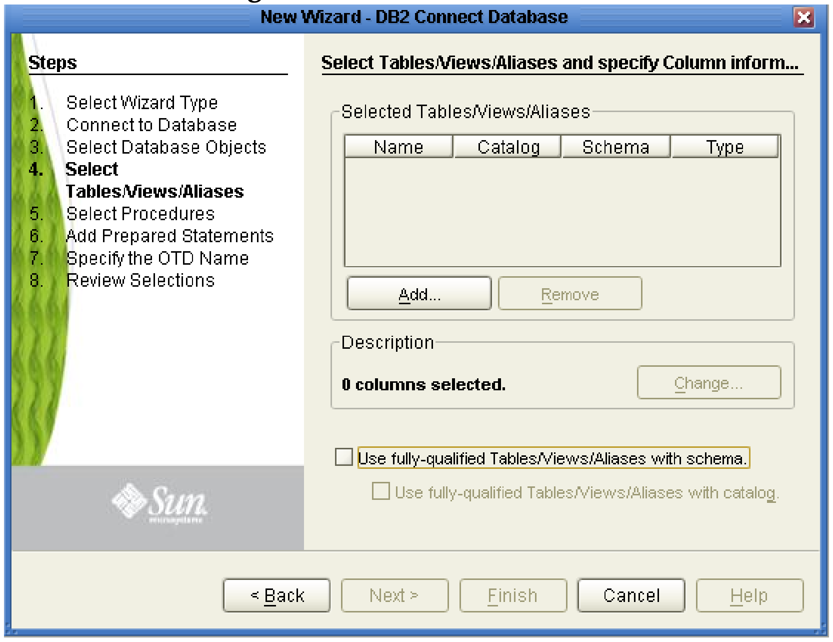
Connection established:
Host: **pmdbbox2:50000** Database Name: **sample1**

2 Click **Next** to continue. The Select Tables/Views/Aliases window appears.

4.2.4 Select Table/Views/Aliases

1 In the **Select Tables/Views** window, click **Add**. See Figure 9.

Figure 9 Select Tables/Views


New Wizard - DB2 Connect Database
Steps
1. Select Wizard Type
2. Connect to Database
3. Select Database Objects
4. Select Tables/Views/Aliases
5. Select Procedures
6. Add Prepared Statements
7. Specify the OTD Name
8. Review Selections
Select Tables/Views/Aliases and specify Column inform...
Selected Tables/Views/Aliases
Name
Catalog
Schema
Type
Add...
Remove
Description
Change...
< Back
Next >
Finish
Cancel
Help

0 columns selected.

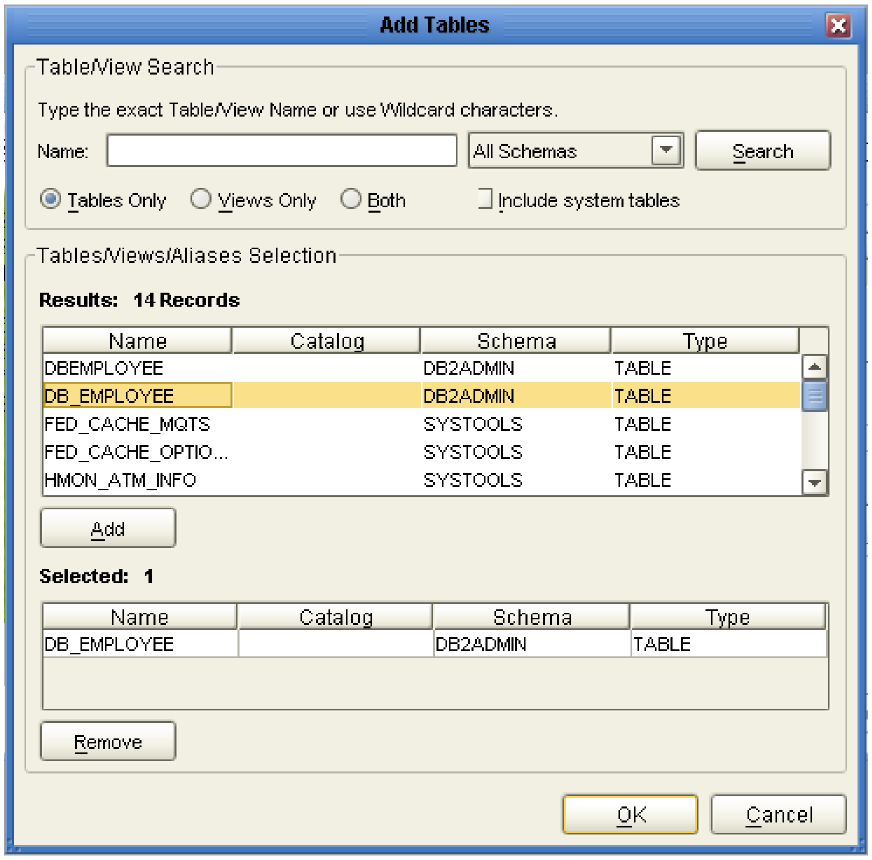
[ ] Use fully-qualified Tables/Views/Aliases with schema.
[ ] Use fully-qualified Tables/Views/Aliases with catalog.

2 In the **Add Tables** window, select the type of criteria to be used for your search, consisting of table data, view only data, or both. You can include system tables in your search by selecting the checkbox.

3 From the **Table/View** Name drop down list, select the location of your database table and click **Search** (see Figure 10). You can search for Table/View Names by entering a table name. The use of wildcard characters of '?', and '*' as part of your Table/View name search allow for greater search capabilities. For example, "AB?CD" or "AB*CD".

4 Select the table of choice and click **OK**. The table selected is added to the Selected window (see Figure 10).

Figure 10 Selected Tables/Views window with a table selected



Results: 14 Records

Name	Catalog	Schema	Type
DBEMPLOYEE		DB2ADMIN	TABLE
DB_EMPLOYEE		DB2ADMIN	TABLE
FED_CACHE_MQTS		SYSTOOLS	TABLE
FED_CACHE_OPTIO...		SYSTOOLS	TABLE
HMON_ATM_INFO		SYSTOOLS	TABLE

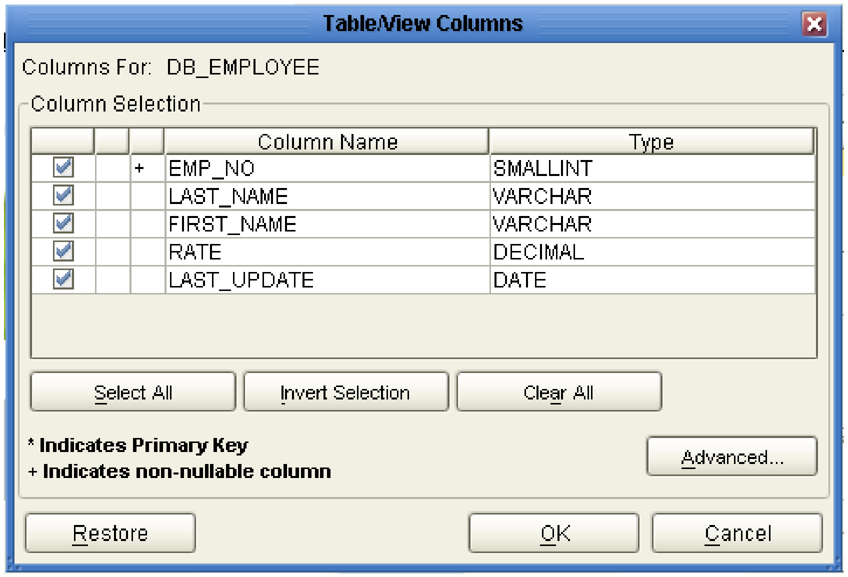
Selected: 1

Name	Catalog	Schema	Type
DB_EMPLOYEE		DB2ADMIN	TABLE

5 On the **Selected Tables/Views** window, review the table(s) you have selected. To make changes to the selected Table or View, click **Change**. If you do not wish to make any additional changes, click **Next** to continue.

6 If you clicked **Change** on the Selected **Tables/Views** window, you can select or deselect your table columns on the **Table/View Columns** window. You can also change the data type for each table by highlighting the data type and selecting a different one from the drop down (see Figure 11).

Figure 11 Tables/Views Columns



Table/View Columns

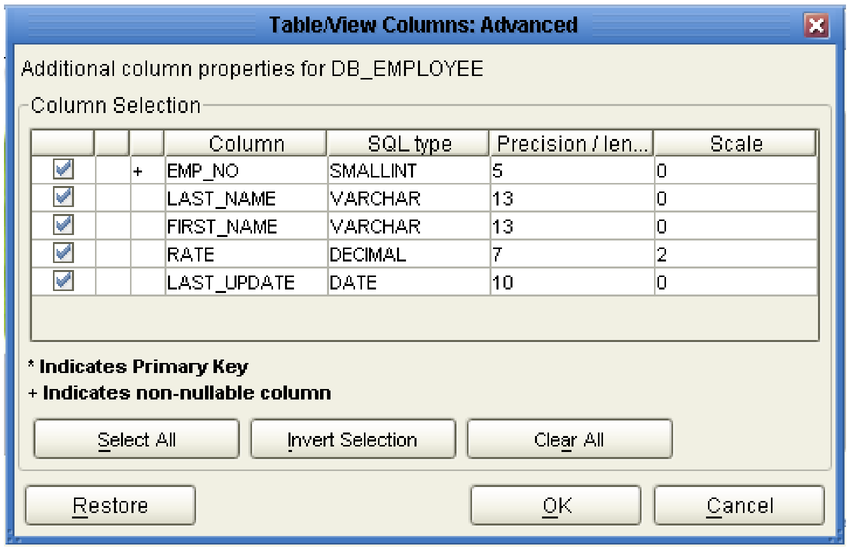
Columns For: DB_EMPLOYEE

Column Selection

			Column Name	Type
✓		+	EMP_NO	SMALLINT
✓			LAST_NAME	VARCHAR
✓			FIRST_NAME	VARCHAR
✓			RATE	DECIMAL
✓			LAST_UPDATE	DATE

7 Click **Advanced** to change the data type, precision/length, or scale. In general, do not change the precision/length or the scale. Once you have finished your table choices, click **OK** (see Figure 12).

Figure 12 Table/View Columns — Advanced



Table/View Columns: Advanced

Additional column properties for DB_EMPLOYEE

Column Selection

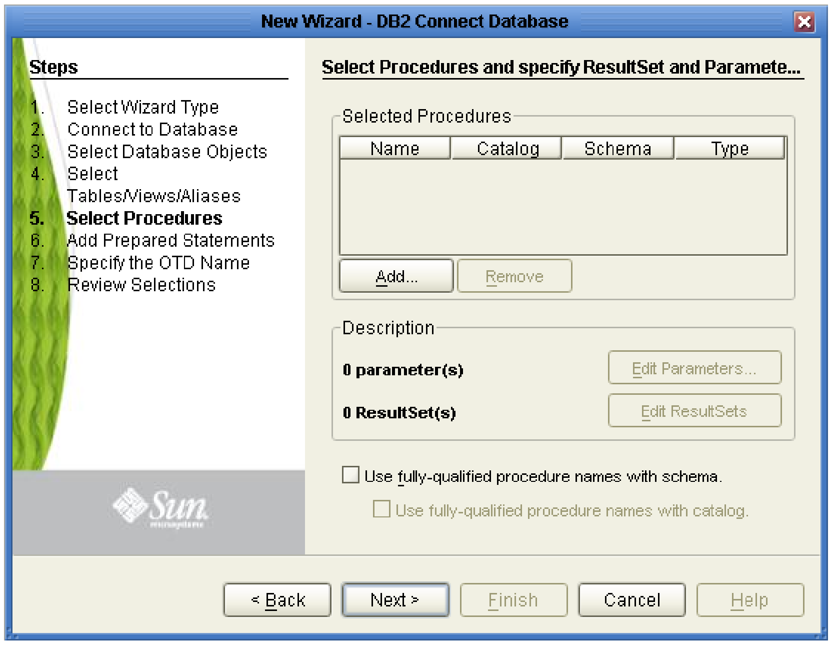
			Column	SQL type	Precision / len...	Scale
✓		+	EMP_NO	SMALLINT	5	0
✓			LAST_NAME	VARCHAR	13	0
✓			FIRST_NAME	VARCHAR	13	0
✓			RATE	DECIMAL	7	2
✓			LAST_UPDATE	DATE	10	0

8 When using Prepared Statement packages, select **Use fully qualified table/view names in the generated Java code**.

4.2.5 Select Procedures

1 On the **Select Procedures and specify Resultset and Parameter Information** window, click **Add.**

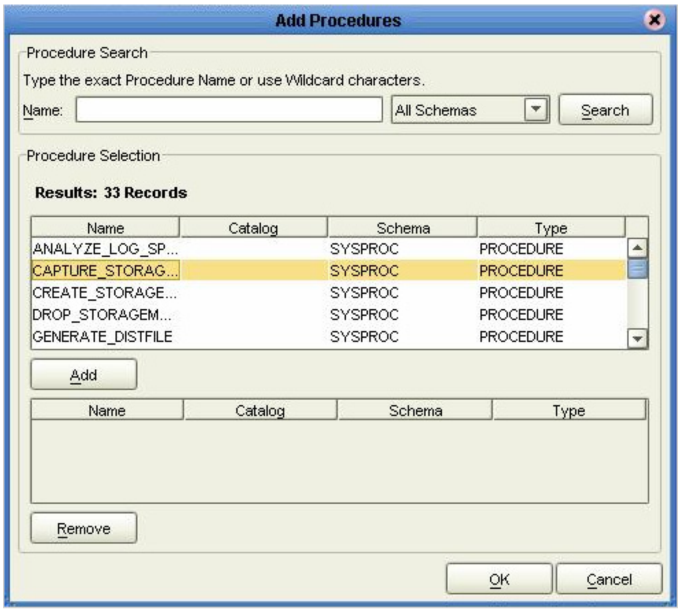
Figure 13 Select Procedures and specify Resultset and Parameter Information


New Wizard - DB2 Connect Database
Steps
1. Select Wizard Type
2. Connect to Database
3. Select Database Objects
4. Select Tables/Views/Aliases
5. Select Procedures
6. Add Prepared Statements
7. Specify the OTD Name
8. Review Selections
Select Procedures and specify ResultSet and Paramete...
Selected Procedures
Name
Catalog
Schema
Type
Add...
Remove
Description
0 parameter(s)
0 ResultSet(s)
Edit Parameters...
Edit ResultSets
Use fully-qualified procedure names with schema.
Use fully-qualified procedure names with catalog.
< Back
Next >
Finish
Cancel
Help

2 On the **Select Procedures** window, enter the name of a Procedure or select a table from the drop down list. Click **Search**. Wildcard characters can also be used.

3 In the resulting **Procedure Selection** list box, select a Procedure. Click **OK**.

Figure 14 Add Procedures

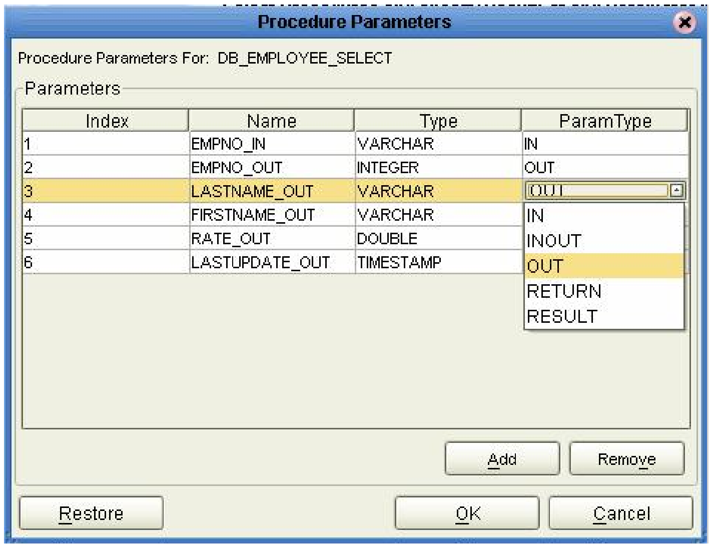


Results: 33 Records

Name	Catalog	Schema	Type
ANALYZE_LOG_SP...		SYSPROC	PROCEDURE
CAPTURE_STORAG...		SYSPROC	PROCEDURE
CREATE_STORAGE...		SYSPROC	PROCEDURE
DROP_STORAGEM...		SYSPROC	PROCEDURE
GENERATE_DISTFILE		SYSPROC	PROCEDURE

4 On the **Select Procedures and specify Resultset and Parameter Information** window click **Edit Parameters** to make any changes to the selected Procedure. See Figure 15.

Figure 15 Procedure Parameters



Procedure Parameters For: DB_EMPLOYEE_SELECT

Parameters

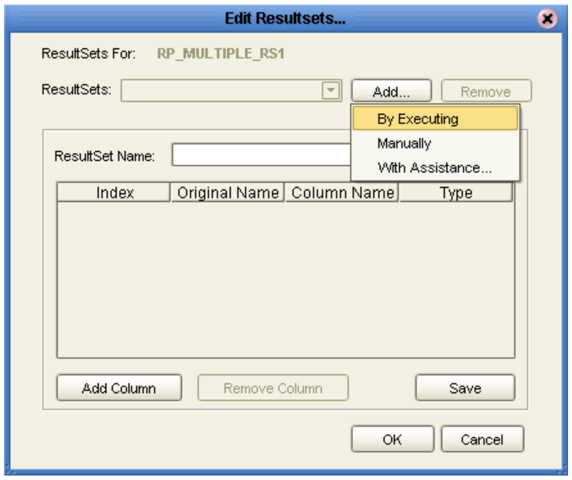
Index	Name	Type	ParamType
1	EMPNO_IN	VARCHAR	IN
2	EMPNO_OUT	INTEGER	OUT
3	LASTNAME_OUT	VARCHAR	OUT
4	FIRSTNAME_OUT	VARCHAR	
5	RATE_OUT	DOUBLE	
6	LASTUPDATE_OUT	TIMESTAMP	

5 To restore the data type, click **Restore**. When finished, click **OK**.

6 To select how you would like the OTD to generate the nodes for the Resultset click **Edit Resultsets**.

7 Click **Add** to add the type of Resultset node you would like to generate (see Figure 16.

Figure 16 Edit Resultset



The DBWizard provides three different ways to generate the ResultSet nodes of a Stored Procedure. They are the "**By Executing**", "**Manually**", and "**With Assistance**" modes.

By Executing Mode	"By Executing" mode executes the specified Stored Procedure with default values to generate the ResultSet(s). Depending on the business logic of the Stored Procedure, zero or more ResultSets can be returned from the execution. In the case that there are multiple ResultSets and "By Executing" mode does not return all ResultSets, one should use the other modes to generate the ResultSet nodes.
With Assistance Mode	"With Assistance" mode allows users to specify a query and execute it to generate the ResultSet node. To facilitate this operation, the DBWizard tries to retrieve the content of the specified Stored Procedure and display it. However, content retrieval is not supported by all types of Stored Procedures. We can roughly classify Stored Procedures into two types: SQL and external. SQL Stored Procedures are created using CREATE PROCEDURE SQL statements while external Stored Procedures are created using host languages (e.g. Java). Since external Stored Procedures do not store their execution plans in the database, content retrieval is impossible. When using "Assist" mode, highlight the execute statement up to and including the table name(s) before executing the query.
Manually Mode	"Manually" mode is the most flexible way to generate the result set nodes. It allows users to specify the node name, original column name and data type manually. One drawback of this method is that users need to know the original column names and data types. This is not always possible. For example, the column name of 3*C in this query. `SELECT A, B, 3*C FROM table T` is generated by the database. In this case, "With Assistance" mode is a better choice. If you modify the ResultSet generated by the "Execute" mode of the Database Wizard you need to make sure the indexes match the Stored Procedure. This assures your ResultSet indexes are preserved.

8 On the **Select Procedures and specify Resultset and Parameter Information** window click **Next** to continue.

4.2.6 Add Prepared Statements

A Prepared Statement OTD represents a SQL statement that has been compiled. Fields in the OTD correspond to the input values that users need to provide.

Prepared statements can be used to perform insert, update, delete and query operations. A prepared statement uses a question mark (?) as a place holder for input. For example: insert into EMP_TAB (Age, Name, Dept No) values (?, ?, ?)

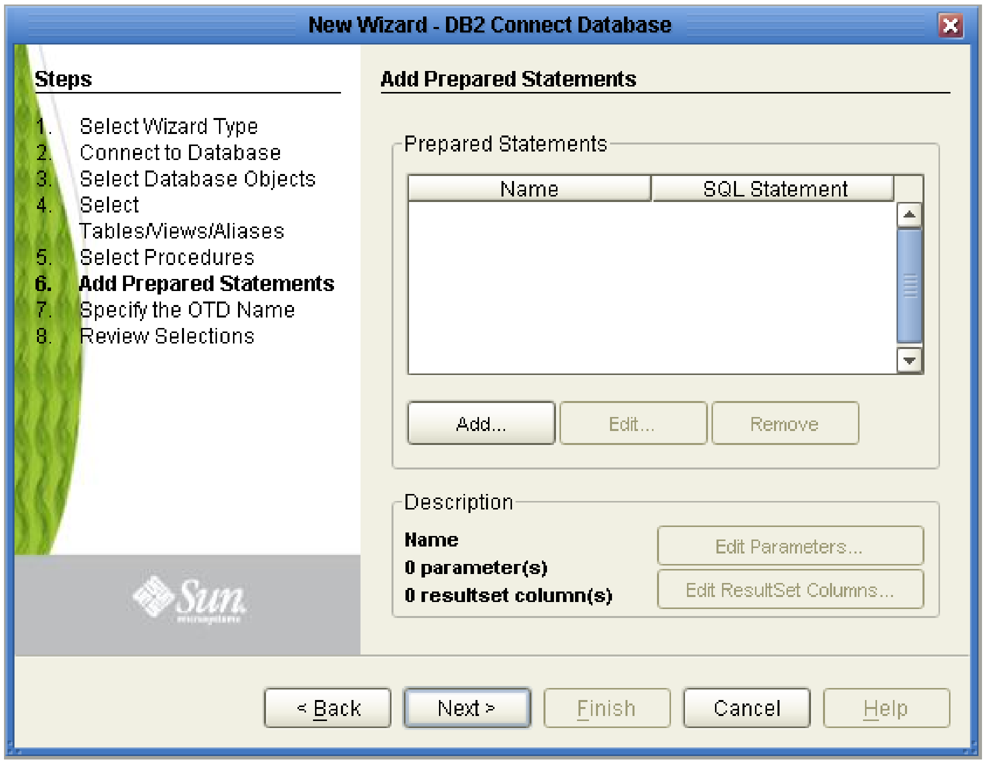
To execute a prepared statement, set the input parameters and call **executeUpdate()** and specify the input values if any.

Note: *When using a Prepared Statement, the 'ResultsAvailable()' method will always return true. Although this method is available, you should not use it with a 'while' loop. Doing so would result in an infinite loop at runtime and will stop all of the system's CPU. If it is used, it should only be used with the 'if' statement.*

You can process a resultset by looping through the next() method. For more information, see **The Query (Select) Operation** *on page 52.*

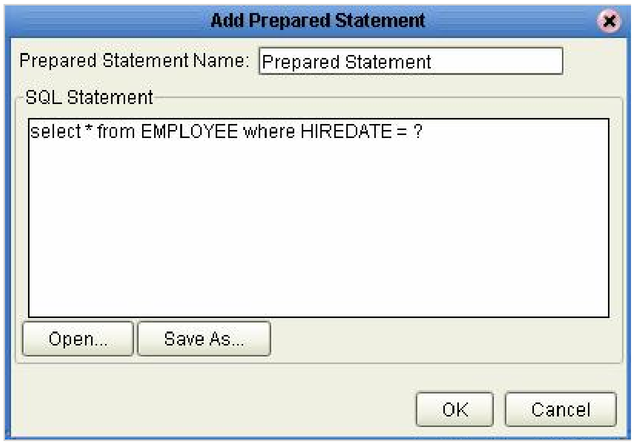
1 On the **Add Prepared Statements** window, click **Add**.

Figure 17 Prepared Statement



2 Enter the name of a Prepared Statement or create a SQL statement by clicking in the SQL Statement window. When finished creating the statement, click **Save As** giving the statement a name. This name will appear as a node in the OTD. Click **OK**. See Figure 18.

Figure 18 Prepared SQL Statement



Prepared Statement Name: Prepared Statement

SQL Statement

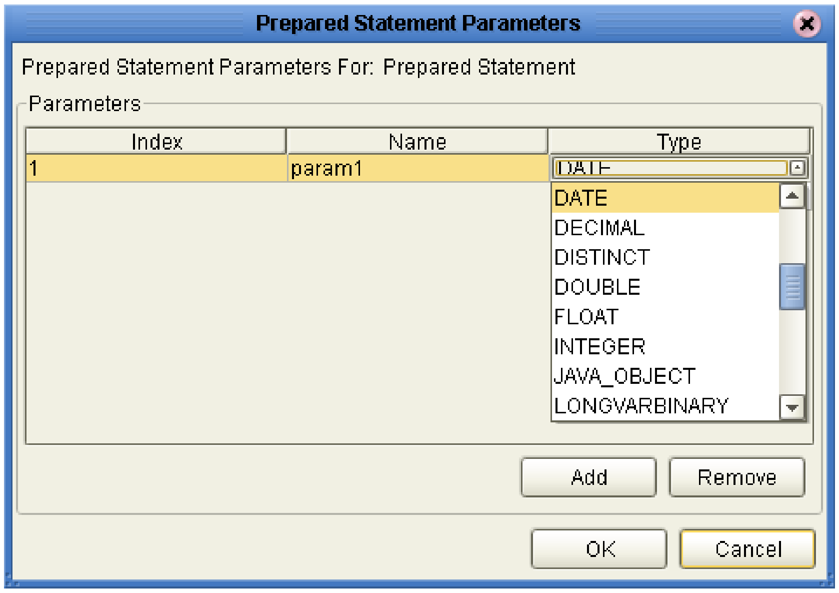
select * from EMPLOYEE where HIREDATE = ?

3 On the **Add Prepared Statement** window, the name you assigned to the Prepared Statement appears. To edit the parameters, click **Edit Parameters**. You can change the data type by clicking in the **Type** field and selecting a different type from the list.

Note: *When doing a Prepared Statement with two or more tables, where multiple tables have the same column name, you must put the table name qualifier in the Prepared Statement to build the OTD.*

4 Click **Add** if you want to add additional parameters to the Statement or highlight a row and click **Remove** to remove it. Click **OK**. See Figure 19.

Figure 19 Edit the Prepared Statement Parameters



Prepared Statement Parameters For: Prepared Statement

Parameters

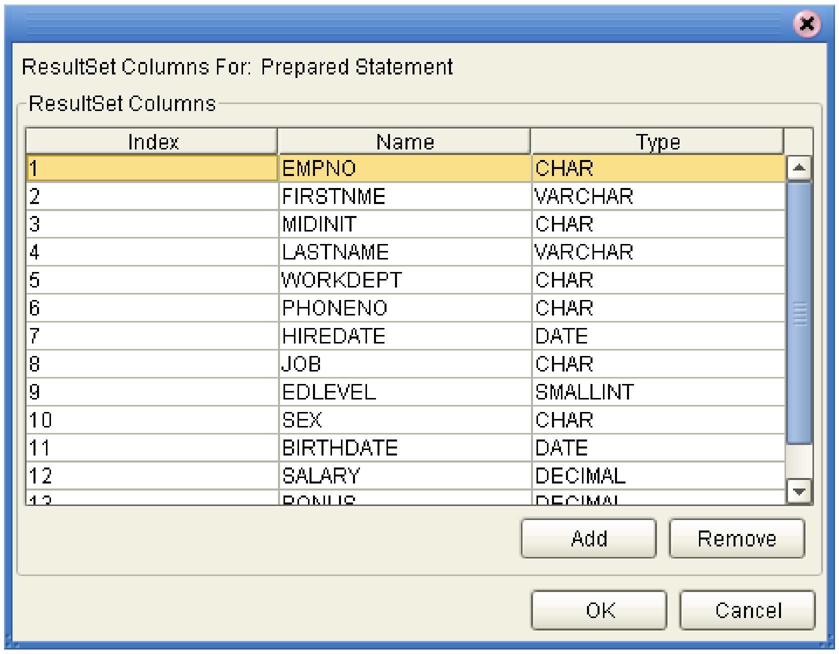
Index	Name	Type
1	param1	DATE

5 To edit the Resultset Columns, click **Edit Resultset Columns**. Both the Name and Type are editable but it is recommend you do not change the Name. Doing so will

cause a loss of integrity between the Resultset and the Database. Click **OK** (see Figure 20).

Note: *The OTD Wizard fails to create OTDs with complex prepared statements that use the same column name in different tables. This problem is resolved by modifying the SQL statement to use column name aliases.*

Figure 20 ResultSet Columns



ResultSet Columns For: Prepared Statement

ResultSet Columns

Index	Name	Type
1	EMPNO	CHAR
2	FIRSTNME	VARCHAR
3	MIDINIT	CHAR
4	LASTNAME	VARCHAR
5	WORKDEPT	CHAR
6	PHONENO	CHAR
7	HIREDATE	DATE
8	JOB	CHAR
9	EDLEVEL	SMALLINT
10	SEX	CHAR
11	BIRTHDATE	DATE
12	SALARY	DECIMAL
13	BONUS	DECIMAL

6 On the **Add Prepared Statements** window, click **OK**.

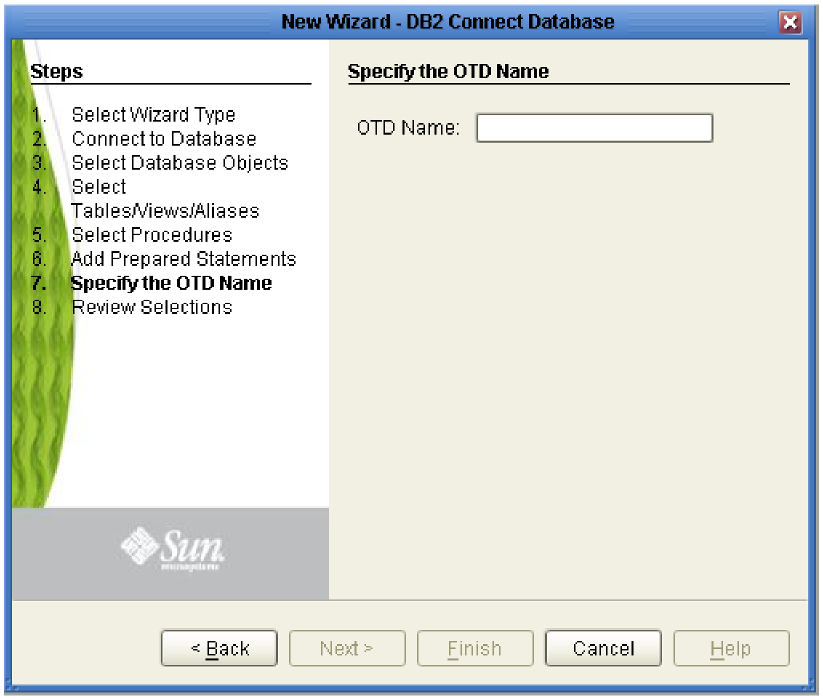
4.2.7 Specify the OTD Name

Specify the name that your OTD will display in the Enterprise Designer Project Explorer.

Steps Required to Specify the OTD Name:

1 Enter a name for the OTD. The OTD contains the selected tables and the package name of the generated classes (see Figure 21).

Figure 21 Naming an OTD



2 Click **Next** to continue. The **Review your Selections** window appears.

4.2.8 Review Selections

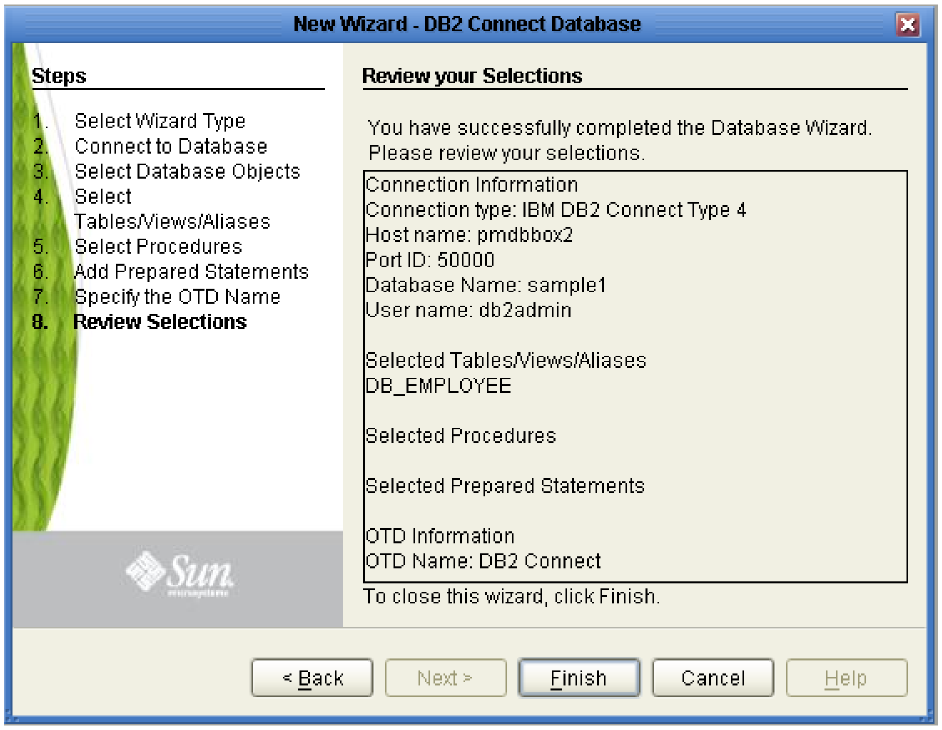
Review the selections made for the new OTD.

Steps Required to Review Your OTD Selections:

1 View the summary of the OTD. If you find you have made a mistake, click **Back** and correct the information.

2 If you are satisfied with the OTD information, click **Finish** to begin generating the OTD (see Figure 22).

Figure 22 Database Wizard - Summary



Review your Selections

You have successfully completed the Database Wizard.
Please review your selections.

Connection Information
Connection type: IBM DB2 Connect Type 4
Host name: pmdbbox2
Port ID: 50000
Database Name: sample1
User name: db2admin

Selected Tables/Views/Aliases
DB_EMPLOYEE

Selected Procedures

Selected Prepared Statements

OTD Information
OTD Name: DB2 Connect

To close this wizard, click Finish.

The resulting OTD appears on the Enterprise Designer's canvas.

4.3 Editing Existing OTDs

A single OTD can consist of many Database objects. They can be a mixture of **Tables**, **Prepared Statements** and **Stored Procedures**. By using the Database OTD Wizard, the OTD Edit feature allows you to:

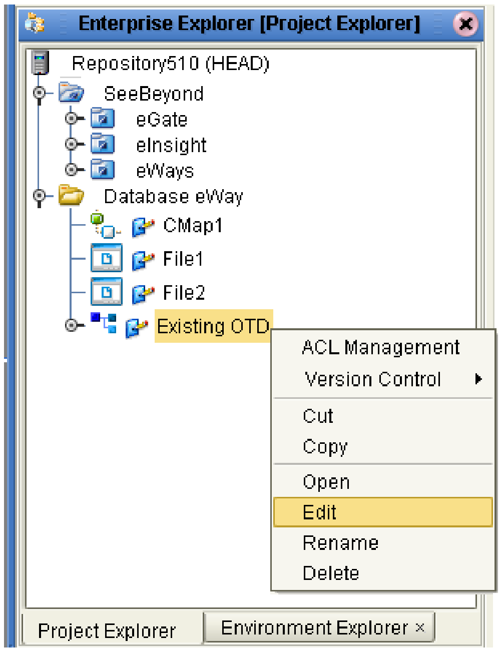
- Add or Remove **Table/Views**.
- Change data types by selecting a different one from a list.
- Add or Remove columns from a **Table** object.
- Add or Remove **Prepared Statement** objects.
- Edit **Prepared Statement** objects.
- Add or Remove **Stored Procedure** objects.
- Edit **Stored Procedure Resultsets**.

To Edit an Existing OTD

When a minor change is needed for an existing OTD, there is no need to rebuild it from scratch; instead, you can edit the OTD. To edit an OTD, complete the following steps:

1 In the Enterprise Explorer, right-click on the OTD. From the submenu, click **Edit** (see Figure 23). The Database Connection Information Wizard opens.

Figure 23 OTD Edit Menu Item



2 Connect to the database by entering the applicable information in the wizard. Once the connection is established, the Database Wizard opens, allowing you to make modifications to the OTD.

3 Once you have completed editing the OTD, click the **Finish** button to save the changes.

Caution: *Once the OTD has been edited, you must verify that the changes are reflected in the Collaboration so that no errors occur at runtime. For example, if during the edit process, you delete a database object that is included in a Collaboration, the Collaboration could fail at activation or run-time.*

When editing an OTD, you can connect to another instance of the database under the following conditions:

- The same version of the database should be used unless the newer version is compatible with the older version.
- Tables in the database must be defined with the same definition.
- The stored procedures must be identical.
- For tables/stored procedures built with 'qualified-name', the schema name for the tables/stored procedures must be identical in both database instances.

Chapter 5

Using DB2 Connect Operations

The database operations used in the DB2 Connect eWay are used to access the DB2 Connect database. Database operations are either accessed through Activities in BPEL, or through methods called from a JCD Collaboration.

What's in This Chapter

5.1 DB2 Connect eWay Database Operations (BPEL)

The DB2 Connect eWay uses a number operations to query the DB2 Connect database. Within a BPEL business process, the DB2 Connect eWay uses BPEL Activities to perform basic outbound database operations, including:

- Insert
- Update
- Delete
- SelectOne
- SelectMultiple
- SelectAll

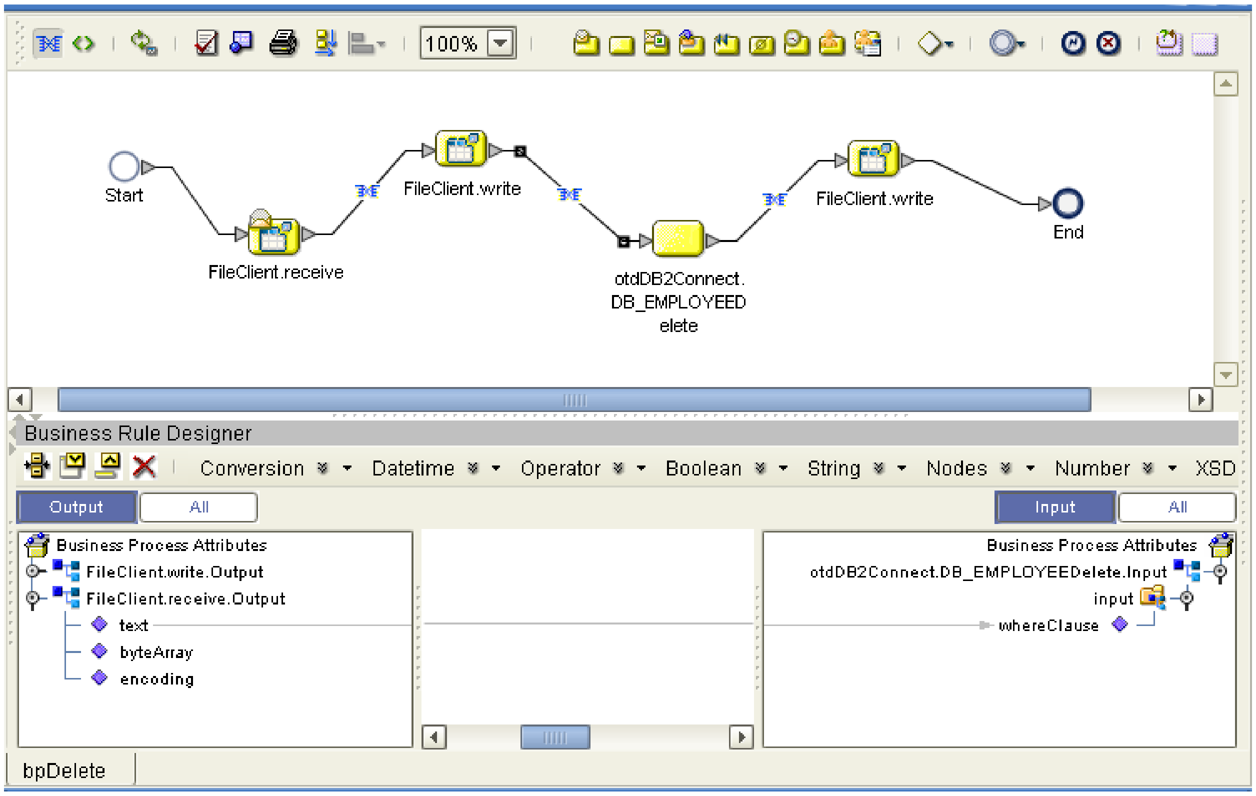
In addition to these outbound operations, the DB2 Connect eWay also employs the inbound Activity **ReceiveOne** within a Prepared Statement OTD.

5.1.1 Activity Input and Output

The Sun SeeBeyond Enterprise Designer – Business Rules Designer includes Input and Output columns to map and transform data between Activities displayed on the Business Process Canvas.

Figure 24 displays the business rules between the **FileClient.write** and **otdDB2 Connect.Db_employeeDelete** Activities. In this example, the **whereClause** appears on the Input side.

Figure 24 Input and Output Between Activities



The following table lists the expected Input and Output of each database operation Activity.

Table 25 DB2 Connect Operations

eInsight Operation	Activity Input	Activity Output
SelectAll	where() clause (optional)	Returns all rows that fit the condition of the where() clause.
SelectMultiple	number of rows where() clause (optional)	Returns the number of rows specified that fit the condition of the where() clause, and the number of rows to be returned.
SelectOne	where() clause (optional)	Returns the first row that fits the condition of the where() clause.
Insert	definition of new item to be inserted	Returns status.
Update	where() clause	Returns status.
Delete	where() clause	Returns status.

5.2 DB2 Connect eWay Database Operations (JCD)

The same database operations are also used in the JCD, but appear as methods to call from the Collaboration.

Tables, Views, and Stored Procedures are manipulated through OTDs. Methods to call include:

- update(*String sWhere*)
- updateRow()
- delete(*String sWhere*)
- deleteRow()
- select(*String where*)

Note: *Refer to the Javadoc for a full description of methods included in the DB2 Connect eWay.*

5.2.1 The Table

A table OTD represents a database table. It consists of fields and methods. Fields correspond to the columns of a table while methods are the operations that you can apply to the OTD. This allows you to perform query, update, insert, and delete SQL operations in a table. The ability to update via a resultset is called "Updatable Resultset", which is a feature supported by this eWay if the Type 4 Universal driver is used (for alternate methods for the Type 2 Legacy driver refer to **Prepared Statement** on page 66).

Note: *The DB2 Connect Universal Driver only supports Updatable Resultsets for Update and Delete. The Insert operation is not supported. You can use a Prepared Statement to perform the Insert operation.*

By default, the Table OTD has UpdatableConcurrency and ScrollTypeForwardOnly. Normally you do not have to change the default setting.

The type of result returned by the select() method can be specified using:

- SetConcurrencytoUpdatable
- SetConcurrencytoReadOnly
- SetScrollTypetoForwardOnly
- SetScrollTypetoScrollSensitive
- SetScrollTypetoInsensitive

The Query (Select) Operation

To perform a query operation on a table

1 Execute the **select()** method with the "**where**" clause specified if necessary.

Note: *The content of the **input.getText()** file may contain null, meaning it will not have a "**where**" clause or it can contain a "**where**" clause such as **empno > 50**.*

2. Loop through the ResultSet using the **next()** method.
3. Process the return record within a **while()** loop.

For example:

```
package prjDB2Connect_JCDjcdALL;


public class jcdTableSelect
{

    public com.stc.codegen.logger.Logger logger;

    public com.stc.codegen.alerter.Alerter alerter;

    public com.stc.codegen.util.CollaborationContext
collabContext;

    public com.stc.codegen.util.TypeConverter typeConverter;

    public void receive(
com.stc.connector.appconn.file.FileTextMessage input,
dtd.otdOutputDTD1325973702.DB_Employee otdOutputDTD_DB_Employee_1,
otdDB2Connect.OtdDB2ConnectOTD otdDB2Connect_1,
com.stc.connector.appconn.file.FileApplication FileClient_1 )
        throws Throwable
    {
        FileClient_1.setText( "Selectiong records from db_employee
table via Table Select........" );
        FileClient_1.write();
        otdDB2Connect_1.getDb_employee().select( input.getText() );
        while (otdDB2Connect_1.getDb_employee().next()) {
            otdOutputDTD_DB_Employee_1.setEmpNo(
typeConverter.shortToString(
otdDB2Connect_1.getDb_employee().getEMP_NO(), "#", false, "" ) );
            otdOutputDTD_DB_Employee_1.setLastname(
otdDB2Connect_1.getDb_employee().getLAST_NAME() );
            otdOutputDTD_DB_Employee_1.setFirstname(
otdDB2Connect_1.getDb_employee().getFIRST_NAME() );
            otdOutputDTD_DB_Employee_1.setRate(
otdDB2Connect_1.getDb_employee().getRATE().toString() );
            otdOutputDTD_DB_Employee_1.setLastDate(
typeConverter.dateToString(
otdDB2Connect_1.getDb_employee().getLAST_UPDATE(), "yyyy-MM-dd
hh:mm:ss", false, "" ) );
            FileClient_1.setText(
otdOutputDTD_DB_Employee_1.marshalToString() );
            FileClient_1.write();
        }
        FileClient_1.setText( "Table Select Done." );
        FileClient_1.write();
    }

}
```

The Update Operation

To perform an update operation on a table

1. Execute the **update()** method.

Note: *The content of the **input.getText()** file may contain null, meaning it will not have a "**where**" clause or it can contain a "**where**" clause such as **empno > 50**.*

2. Using a while loop together with **next()**, move to the row that you want to update.
3. Assign updating value(s) to the fields of the table OTD
4. Update the row by calling **updateRow()**.

```
package prjDB2Connect_JCDjcdALL;


public class jcdUpdate
{

public com.stc.codegen.logger.Logger logger;
public com.stc.codegen.alerter.Alerter alerter;
public com.stc.codegen.util.CollaborationContext collabContext;
public com.stc.codegen.util.TypeConverter typeConverter;

    public void receive(
com.stc.connector.appconn.file.FileTextMessage input,
otdDB2Connect.OtdDB2ConnectOTD otdDB2Connect_1,
com.stc.connector.appconn.file.FileApplication FileClient_1 )
        throws Throwable
    {
        FileClient_1.setText( "Updating the Rate and Last_update
fields .. " );
        FileClient_1.write();
        otdDB2Connect_1.getDb_employee().update( input.getText() );
        while (otdDB2Connect_1.getDb_employee().next()) {
         otdDB2Connect_1.getDb_employee().setLAST_NAME( "Krishna" );
         otdDB2Connect_1.getDb_employee().setFIRST_NAME( "Kishore" );
         otdDB2Connect_1.getDb_employee().updateRow();
        }
        FileClient_1.setText( "Update Done." );
        FileClient_1.write();
    }

}
```

The Delete Operation

To perform a delete operation on a table

1. Execute the **delete()** method.

Note: *The content of the **input.getText()** file may contain null, meaning it will not have a "**where**" clause or it can contain a "**where**" clause such as **empno > 50**.*

In this example DELETE an employee.

```
package prjDB2Connect_JCDjcdALL;
```

```
public class jcdDelete
{
public com.stc.codegen.logger.Logger logger;
public com.stc.codegen.alerter.Alerter alerter;
public com.stc.codegen.util.CollaborationContext collabContext;
public com.stc.codegen.util.TypeConverter typeConverter;

    public void receive(
com.stc.connector.appconn.file.FileTextMessage input,
otdDB2Connect.OtdDB2ConnectOTD otdDB2Connect_1,
com.stc.connector.appconn.file.FileApplication FileClient_1 )
        throws Throwable
    {
        FileClient_1.setText( "Deleting record..........." );
        FileClient_1.write();
       otdDB2Connect_1.getDb_employee().delete( input.getText() );
        FileClient_1.setText( "Delete Done." );
        FileClient_1.write();
    }
}
```

5.2.2 Prepared Statement

A Prepared Statement is a SQL statement which can also contain parameter marker as input holder.

Example: select * from table1 where col1 > ?

This statement selects all the columns from a table called table1 if column col1 is greater than a certain value. The value will be supplied during runtime.

Note: *The DB2 Connect Universal Driver only supports Updatable Resultsets for Update and Delete. The Insert operation is not supported. You can use a Prepared Statement to perform the Insert operation.*

The Insert Operation

To perform an insert operation using Prepared Statement

1 Assign values to input fields.

2 Execute the executeUpdate().

This example inserts employee records. The Prepared Statement looks like this:

```
Insert into DB_EMPLOYEE values (?, ?, ?, ?, ?)
```

Note: *If you don't want to insert values into all columns, your insert statement should look like this:*

```
Insert into DB_EMPLOYEE (col1, col2) values (?, ?)

public class jcdPsInsert
{
    public com.stc.codegen.logger.Logger logger;

    public com.stc.codegen.alerter.Alerter alerter;
    public com.stc.codegen.util.CollaborationContext
collabContext;

    public com.stc.codegen.util.TypeConverter typeConverter;

    public void receive(
com.stc.connector.appconn.file.FileTextMessage input,
otdDB2Connect.OtdDB2ConnectOTD otdDB2Connect_1,
com.stc.connector.appconn.file.FileApplication FileClient_1,
dtd.otdInputDTD_654315252.DBemployees otdInputDTD_DBemployees_1,
dtd.otdOutputDTD1750519912.DBemployee otdOutputDTD_DBemployee_1 )

        throws Throwable

    {

        FileClient_1.setText( "Inserting records into db_employee
table using Prepared Statement....." );

        FileClient_1.write();

        otdInputDTD_DBemployees_1.unmarshalFromString(
input.getText() );

        for (int i1 = 0; i1 <
otdInputDTD_DBemployees_1.countX_sequence_A(); i1 += 1) {

            otdDB2Connect_1.getInsert_ps().setEMP_NO(
typeConverter.stringToShort(
otdInputDTD_DBemployees_1.getX_sequence_A( i1 ).getEmpNo(), "#",
false, 0 ) );

            otdDB2Connect_1.getInsert_ps().setLAST_NAME(
otdInputDTD_DBemployees_1.getX_sequence_A( i1 ).getLastname() );

            otdDB2Connect_1.getInsert_ps().setFIRST_NAME(
otdInputDTD_DBemployees_1.getX_sequence_A( i1 ).getFirstname() );

            otdDB2Connect_1.getInsert_ps().setRATE( new
java.math.BigDecimal( otdInputDTD_DBemployees_1.getX_sequence_A(
i1 ).getRate() ) );

            otdDB2Connect_1.getInsert_ps().setLAST_UPDATE(
typeConverter.stringToSQLDate(
otdInputDTD_DBemployees_1.getX_sequence_A( i1 ).getLastDate(),
"yyyy-MM-dd hh:mm:ss", false, "" ) );

            otdDB2Connect_1.getInsert_ps().executeUpdate();
        }

        FileClient_1.setText( "Insert Done......" );

        FileClient_1.write();

    }

}
```

The Update Operation

To perform an update operation using Prepared Statement

1 Assign value to input field.

2 Execute the executeUpdate() .

This example updates employee records which matches the where clause. The Prepared Statement looks like this:

Update DB_EMPLOYEE set rate = 19 where EMP_NO = ?

Note: *The content of the input.getText() file must contain the input value to substitute the parameter marker ?*

```
package prjDB2Connect_JCDjcdALL;

public class jcdPsUpdate
{

    public com.stc.codegen.logger.Logger logger;

    public com.stc.codegen.alerter.Alerter alerter;

    public com.stc.codegen.util.CollaborationContext
collabContext;

    public com.stc.codegen.util.TypeConverter typeConverter;

    public void receive(
com.stc.connector.appconn.file.FileTextMessage input,
otdDB2Connect.OtdDB2ConnectOTD otdDB2Connect_1,
com.stc.connector.appconn.file.FileApplication FileClient_1 )

        throws Throwable
    {

        FileClient_1.setText( "Update the Rate and Last_update
fields using Prepared Statement.. " );

        FileClient_1.write();

        otdDB2Connect_1.getUpdate_ps().setEmp_no(
typeConverter.stringToShort( input.getText(), "#", false, 0 ) );

        otdDB2Connect_1.getUpdate_ps().executeUpdate();

        FileClient_1.setText( "Done Update." );

        FileClient_1.write();
    }

}
```

5.2.3 The Stored Procedure

A Stored Procedure OTD represents a database stored procedure. Fields correspond to the arguments of a stored procedure while methods are the operations that you can apply to the OTD. It allows you to execute a stored procedure. Remember that while in the Collaboration Editor you can drag and drop nodes from the OTD into the Collaboration Editor.

Executing Stored Procedures

The OTD represents the Stored Procedure “LookUpGlobal” with two parameters, an inbound parameter (INLOCALID) and an outbound parameter (OUTGLOBALPRODUCTID). These inbound and outbound parameters are generated by the DataBase Wizard and are represented in the resulting OTD as nodes. Within the Transformation Designer, you can drag values from the input parameters, execute the call, collect data, and drag the values to the output parameters.

Below are the steps for executing the Stored Procedure:

1. Specify the input values.
2. Execute the Stored Procedure.
3. Retrieve the output parameters if any.

For example:

```
package Storedprocedure;

public class sp_jce
{

    public com.stc.codegen.logger.Logger logger;

    public com.stc.codegen.alerter.Alerter alerter;

    public void receive(
com.stc.connector.appconn.file.FileTextMessage
input,com.stc.connector.appconn.file.FileApplication
FileClient_1,employeedb.Db_employee
employeedb_with_top_db_employee_1,insert_DB.Insert_DBOTD insert_DB_1
)
    throws Throwable
    {

        employeedb_with_top_db_employee_1.unmarshalFromString(
input.getText() );

        insert_DB_1.getInsert_new_employee().setEmployee_no(
java.lang.Integer.parseInt(
employeedb_with_top_db_employee_1.getEmployee_no() ) );

        insert_DB_1.getInsert_new_employee().setEmployee_Lname(
employeedb_with_top_db_employee_1.getEmployee_lname() );

        insert_DB_1.getInsert_new_employee().setEmployee_Fname(
employeedb_with_top_db_employee_1.getEmployee_fname() );

        insert_DB_1.getInsert_new_employee().setRate(
java.lang.Float.parseFloat(
employeedb_with_top_db_employee_1.getRate() ) );

        insert_DB_1.getInsert_new_employee().setUpdate_date(
java.sql.Timestamp.valueOf(
employeedb_with_top_db_employee_1.getUpdate_date() ) );

        insert_DB_1.getInsert_new_employee().execute();

        insert_DB_1.commit();

        FileClient_1.setText( "procedure executed" );
```

```
                FileClient_1.write();
            }

        }
```

Manipulating the ResultSet and Update Count Returned by Stored Procedure

For Stored Procedures that return ResultSets and Update Count, the following methods are provided to manipulate the ResultSet:

- enableResultSetOnly
- enableUpdateCountsOnly
- enableResultSetandUpdateCounts
- resultsAvailable
- next
- getUpdateCount
- available

DB2 Connect stored procedures do not return records as ResultSets, instead, the records are returned through output reference cursor parameters. Reference Cursor parameters are essentially ResultSets.

The **resultsAvailable()** method, added to the PreparedStatementAgent class, simplifies the whole process of determining whether any results, be it Update Counts or ResultSets, are available after a stored procedure has been executed. Although JDBC provides three methods (**getMoreResults()**, **getUpdateCount()**, and **getResultSet()**) to access the results of a stored procedure call, the information returned from these methods can be quite confusing to the inexperienced Java JDBC programmer and they also differ between vendors. You can simply call **resultsAvailable()** and if Boolean true is returned, you can expect either a valid Update Count when **getUpdateCount()** is called and/or the next ResultSet has been retrieved and made available to one of the ResultSet nodes defined for the Stored Procedure OTD, when that node's **available()** method returns true.

Frequently, Update Counts information that is returned from a Stored Procedures is insignificant. You should process returned ResultSet information and avoid looping through all of the Update Counts. The following three methods control exactly what information should be returned from a stored procedure call. The **enableResultSetsOnly()** method, added to the PreparedStatement Agent class allows only ResultSets to be returned and thus every **resultsAvailable()** called only returns Boolean true if a ResultSet is available. Likewise, the **enableUpdateCountsOnly()** causes **resultsAvailable()** to return true only if an Update Count is available. The default case of **enableResultsetsAndUpdateCount()** method allows both ResultSets and Update Counts to be returned.

Collaboration usability for a stored procedure ResultSet

The Column data of the ResultSets can be dragged-and-dropped from their nodes to the Business Rules. Below is a code snippet that can be generated by the Collaboration Editor:

```
while (getSPIn().getSpS_multi().resultsAvailable())
{
if (getSPIn().getSpS_multi().getUpdateCount() > 0)
  {
    System.err.println("Updated
"+getSPIn().getSpS_multi().getUpdateCount()+" rows");
  }

  if (getSPIn().getSpS_multi().getNormRS().available())
  {
    while (getSPIn().getSpS_multi().getNormRS().next())
    {
      System.err.println("Customer Id   =
"+getSPIn().getSpS_multi().getNormRS().getCustomerId());
      System.err.println("Customer Name =
"+getSPIn().getSpS_multi().getNormRS().getCustomerName());
      System.err.println();
    }
    System.err.println("===");
  }
  else if (getSPIn().getSpS_multi().getDbEmployee().available())
  {
    while (getSPIn().getSpS_multi().getDbEmployee().next())
    {
      System.err.println("EMPNO    =
"+getSPIn().getSpS_multi().getDbEmployee().getEMPNO());
      System.err.println("ENAME    =
"+getSPIn().getSpS_multi().getDbEmployee().getENAME());
      System.err.println("JOB      =
"+getSPIn().getSpS_multi().getDbEmployee().getJOB());
      System.err.println("MGR      =
"+getSPIn().getSpS_multi().getDbEmployee().getMGR());
      System.err.println("HIREDATE =
"+getSPIn().getSpS_multi().getDbEmployee().getHIREDATE());
      System.err.println("SAL      =
"+getSPIn().getSpS_multi().getDbEmployee().getSAL());
      System.err.println("COMM     =
"+getSPIn().getSpS_multi().getDbEmployee().getCOMM());
      System.err.println("DEPTNO   =
"+getSPIn().getSpS_multi().getDbEmployee().getDEPTNO());
      System.err.println();
    }
    System.err.println("===");
  }
}
```

Note: **resultsAvailable()** *and* **available()** *cannot be indiscriminately called because each time they move ResultSet pointers to the appropriate locations.*

After calling "**resultsAvailable()**", the next result (if available) can be either a **ResultSet** or an **UpdateCount** if the default "**enableResultSetsAndUpdateCount()**" was used.

Because of limitations imposed by some DBMSs, it is recommended that for maximum portability, all of the results in a ResultSet object should be retrieved before OUT parameters are retrieved. Therefore, you should retrieve all ResultSet(s) and Update Counts first followed by retrieving the OUT type parameters and return values.

The following list includes specific ResultSet behavior that you may encounter:

- The method **resultsAvailable()** implicitly calls **getMoreResults()** when it is called more than once. You should not call both methods in your java code. Doing so may result in skipped data from one of the ResultSets when more than one ResultSet is present.
- The methods **available()** and **getResultSet()** can not be used in conjunction with multiple ResultSets being open at the same time. Attempting to open more the one ResultSet at the same time closes the previous ResultSet. The recommended working pattern is:
    - Open one Result Set (ResultSet_1) and work with the data until you have completed your modifications and updates. Open ResultSet_2, (ResultSet_1 is now closed) and modify. When you have completed your work in ResultSet_2, open any additional ResultSets or close ResultSet_2.
- If you modify the ResultSet generated by the Execute mode of the Database Wizard, you need to assure the indexes match the stored procedure. By doing this, your ResultSet indexes are preserved.
- Generally, getMoreResults does not need to be called. It is needed if you do not want to use our enhanced methods and you want to follow the traditional JDBC calls on your own.

The DBWizard Assistant expects the column names to be in English when creating a ResultSet.

Prepared Statement

A Prepared Statement OTD represents a SQL statement that has been compiled. Fields in the OTD correspond to the input values that users need to provide.

Prepared statements can be used to perform insert, update, delete and query operations. A prepared statement uses a question mark (?) as a place holder for input. For example:

```
insert into EMP_TAB(Age, Name, Dept No) value(?, ?, ?)
```

To execute a prepared statement, set the input parameters and call **executeUpdate()** and specify the input values if any.

```
getPrepStatement().getPreparedStatementTest().setAge(23);
getPrepStatement().getPreparedStatementTest().setName('Peter Pan');
getPrepStatement().getPreparedStatementTest().setDeptNo(6);
getPrepStatement().getPreparedStatementTest().executeUpdate();
```

Batch Operations

To achieve better performance, consider using a bulk insert if you have to insert many records. This is the “Add Batch” capability. The only modification required is to include the **addBatch()** method for each SQL operation and then the **executeBatch()** call to submit the batch to the database server. Batch operations apply only to Prepared Statements.

```
getPrepStatement().getPreparedStatementTest().setAge(23);
getPrepStatement().getPreparedStatementTest().setName(‘Peter Pan’);
getPrepStatement().getPreparedStatementTest().setDeptNo(6);
getPrepStatement().getPreparedStatementTest().addBatch();

getPrepStatement().getPreparedStatementTest().setAge(45);
getPrepStatement().getPreparedStatementTest().setName(‘Harrison
Ford’);
getPrepStatement().getPreparedStatementTest().setDeptNo(7);
getPrepStatement().getPreparedStatementTest().addBatch();
getPrepStatement().getPreparedStatementTest().executeBatch();
```

Chapter 6

Implementing the DB2 Connect Sample Projects

This chapter provides an introduction to the DB2 Connect eWay components, and information on how these components are created and implemented in a Sun Java Composite Application Suite Project.

It is assumed that the reader understands the basics of creating a Project using the Sun SeeBeyond Enterprise Designer. For more information on creating an eGate Project, see the *eGate Tutorial* and the *eGate Integrator User's Guide*.

What's in This Chapter

6.1 About the DB2 Connect eWay Sample Projects

The DB2 Connect eWay **DB2Connect_eWay_Sample.zip** file contains two sample Projects that provide basic instruction on using DB2 Connect operations in the Java Collaboration Definition (JCD), or the Business Process Execution Language (BPEL) Projects.

- **prjDB2Connect_JCD:** demonstrates how to select, insert, update, and delete data from a DB2 Connect database using JCDs.
- **prjDB2Connect_BPEL:** demonstrates how to select, insert, update, and delete data from a DB2 Connect database using a BPEL business process.

Note: *These samples are only valid if you use the Type 4 driver, since the Type 2 driver does not support updateable result sets. To do **Insert\Update\Delete** you must use a prepared statement*

Both the **prjDB2Connect_JCD** and **prjDB2Connect_BPEL** sample Projects demonstrate how to:

- Select employee records from the database using a prepared statement.
- Select employee records from the db_employee table.
- Insert employee records data into the db_employee table.
- Update an employee record in the db_employee table.
- Delete an employee record in the db_employee table.

In addition to sample Projects, the **DB2Connect510_SAMPLE_projects.zip** file also includes six sample input trigger files and ten sample output files (five per sample).

Sample input files include:

- TriggerDelete.in.~in
- TriggerInsert.in.~in
- TriggerBpInsert.in.~in
- TriggerPsSelect.in.~in
- TriggerTableSelect.in.~in
- TriggerUpdate.in.~in
- TriggerPsUpdate.in.~in

Sample output JCD files include:

- JCD_Delete_output0.dat
- JCD_Insert_output0.dat
- JCD_PsSelect_output0.dat
- JCD_TableSelect_output0.dat
- JCD_Update_output0.dat

Sample output BPEL files include:

- BPEL_Delete_output0.dat
- BPEL_Insert_output0.dat
- BPEL_PsSelect_output0.dat
- BPEL_TableSelect_output0.dat
- BPEL_Update_output0.dat

6.1.1 Operations Used in the DB2 Connect Sample Projects

The following database operations are used in both BPEL and JCD sample Projects:

- Insert
- Update
- Delete

- Select (SelectAll as a BPEL Activity)

Assigning Operations in JCD

Database operations are listed as methods in the JCD. Perform the following steps to access these methods:

1. Create a Collaboration that contains a database OTD created from the DB2 Connect database.
2. Right-click the OTD listed in your Collaboration and then select **Select Method to Call** from the shortcut menu.
3. Browse to and select a method to call.

Assigning Operations in BPEL

You can associate an eInsight Business Process Activity with the eWay, both during the system design phase and during run time. To make this association:

1. Select the desired **receive** or **write** operation under the eWay in the Enterprise Explorer.
2. Drag the operation onto the eInsight Business Process canvas.

   The operation automatically changes to an Activity with an icon identifying the component that is the basis for the Activity.

At run time, the eInsight engine invokes each step in the order that you defined in the Business Process. Using the engine’s Web Services interface, the Activity in turn invokes the eWay. You can open a file specified in the eWay and view its contents before and after the Business Process is executed.

Note: *Inbound database eWays are only supported within BPEL Collaborations.*

6.1.2 About the eInsight Engine and eGate Components

You can deploy an eGate component as an Activity in an eInsight Business Process. Once you have associated the desired component with an Activity, the eInsight engine can invoke it using a Web Services interface.

Examples of eGate components that can interface with eInsight in this way are:

- Object Type Definitions (OTDs)
- An eWay
- Collaborations

Using the eGate Enterprise Designer and eInsight, you can add an Activity to a Business Process, then associate that Activity with an eGate component, for example, an eWay. When eInsight run the Business Process, it automatically invokes that component via its Web Services interface.

6.2 Steps Required to Run the Sample Projects

The following steps are required to run the sample projects that are contained in the **DB2ConnecteWayDocs.sar** file.

1. Run the SQL script.

   This creates the tables and records required by the sample Project.

2. Import the sample Projects.
3. Build, deploy, and run the sample Projects.

   You must do the following before you can run an imported sample Project:

   - Create an Environment
   - Configure the eWays
   - Create a Deployment Profile
   - Create and start a domain
   - Deploy the Project

4. Check the output.

6.3 Running the SQL Script

The data used for both the **JCD** and **BPEL** sample Projects are contained within a table called **db_employee**. You create this table by using the SQL statement **DB2Connect_sample_script.sql**, that is included in the sample Project.

The following SQL statement is designed for the sample Projects and can be run with any database tool.

```
drop table db_employee
go
create table db_employee (
  EMP_NO int,
  LAST_NAME varchar(30),
  FIRST_NAME varchar(30),
  RATE float,
  LAST_UPDATE datetime)
go
```

The sample Projects provided with the DB2 Connect eWay use input files to pass predefined data or conditions into the Collaboration or BPEL business process, which then transforms the database contents, and delivers the result set.

6.4 Importing a Sample Project

Sample eWay Projects are included as part of the installation CD-ROM package. To import a sample eWay Project to the Enterprise Designer do the following:

1. Extract the samples from the Enterprise Manager to a local file.

   Sample files are uploaded with the eWay's documentation SAR file, and then downloaded from the Enterprise Manager's Documentation tab. The **DB2Connect_eWay_Sample.zip** file contains the various sample Project ZIP files.

Note: *Make sure you save all unsaved work before importing a Project.*

2. From the Enterprise Designer's Project Explorer pane, right-click the Repository and select **Import Project** from the shortcut menu. The **Import Manager** appears.
3. Browse to the directory that contains the sample Project ZIP file. Select the sample file and click **Import**.
4. Click **Close** after successfully importing the sample Project.

6.5 Building and Deploying the prjDB2 Connect_BPEL Sample Project

The following provides step-by-step instructions for manually creating the **prjDB2 Connect_BPEL** sample Project.

Steps required to create the sample project include:

- **Creating a Project** on page 78
- **Creating the OTDs** on page 79
- **Creating the Business Process** on page 80
- **Creating the Connectivity Map** on page 94
- **Creating an Environment** on page 96
- **Configuring the eWays** on page 97
- **Creating the Deployment Profile** on page 100
- **Creating and Starting the Domain** on page 100
- **Building and Deploying the Project** on page 101

6.5.1 Creating a Project

The first step is to create a new Project in the Sun SeeBeyond Enterprise Designer.

1. Start the Enterprise Designer.

2. From the Project Explorer tree, right-click the Repository and select **New Project**. A new Project (**Project1**) appears on the Project Explorer tree.
3. Click twice on **Project1** and rename the Project (for this sample, **prjDB2 Connect_BPEL**).

6.5.2 Creating the OTDs

The sample Project requires three OTDs to interact with the DB2 Connect eWay. These OTDs include:

- A DB2 Connect Database OTD
- An Inbound DTD OTD
- An Outbound DTD OTD

Steps required to create a DB2 Connect Database OTD

These steps are explained in more detail beginning in the section **Steps to Create a New DB2 Connect OTD** on page 45.

1. Right-click your new Project in the Enterprise Designer's Project Explorer, and select **New** > **Object Type Definition**.

   The New Object Type Definition Wizard window appears.
2. Select the **DB2 Connect Database OTD Wizard** from the list of OTD Wizards and click **Next**.
3. Enter the connection information for the DB2 Connect database. For more information on the connection fields.
4. Click **Next**, and select the type of database object(s) you want to include in the sample Project. For this example, select the following:
   - Tables/Views/Aliases
   - Prepared Statements
5. Click **Add** to select tables from the DB2 Connect database. The **Add Tables** window appears.
6. Search for or Type in the name of the database. In this example we use the **DB_EMPLOYEE** table. Click **Select** when the database appears in the Results selection frame. Click **OK** to close the Add Tables window
7. Click **Next the Add Prepared Statements Wizard appears.**
8. Click **Add**, the Add Prepared Statement window appears. Enter the following:
   - Prepared Statement Name: Select_ps
   - SQL Statement:

```
select * from db_employee where emp_no > ? order by emp_no
```

Note: *In our example, the SQL statement includes the* **?** *placeholder for input. This placeholder represents the Where Clause.*

9 Click the **OK** button to close the Prepared Statement window, and then click **Next** on the Prepared Statements Wizard window.

10 Enter an OTD name. In this example, we use **otdDB2 Connect**.

11 Click **Next** and review your settings, then click **Finish** to create the OTD.

Steps required to create inbound and outbound DTD OTDs include:

1 Right-click your new Project in the Enterprise Designer's Project Explorer, and select **New** > **Object Type Definition**.

The New Object Type Definition Wizard window appears.

2 Select **DTD** from the list of OTD Wizards and click **Next**.

3 Browse to and then select a DTD file. For our example, select one of the following DTD files from the sample Project, and then click **Next**.

- otdInputDTD.dtd
- otdOutputDTD.dtd

4 The file you select appears in the Select Document Elements window. Click **Next**.

5 Click **Finish** to complete the DTD based OTD. Repeat this process again to create the second DTD file.

6.5.3 Creating the Business Process

Steps required to create the Business Process include:

- Creating the business process flow
- Configuring the modeling elements

Creating the Business Process Flow

The business process flow contains all the BPEL elements that make up a business process.

Steps to create a business process flow include:

1 Right-click your new Project in the Enterprise Designer's Project Explorer, and select **New** > **Business Process** from the shortcut menu. The eInsight Business Process Designer appears and **BusinessProcess1** is added to the Project Explorer tree. Rename **BusinessProcess1** to **bpInsert**.

2 Create four additional business processes and rename them as follows:

- bpUpdate
- bpDelete
- bpPsSelect
- bpTableSelect

3 Add the following activities to the Business Process Designer canvas.

Business Process	Activity
bpInsert	▪ FileClient.Receive ▪ FileClient.Write ▪ FileClient.Write ▪ otdDB2 Connect.DB_EMPLOYEEInsert (inside a Scope) ▪ otdInputDTD_DBemployees.unmarshal
bpUpdate	▪ FileClient.receive ▪ FileClient.write ▪ otdDB2 Connect.DB_EMPLOYEEUpdate ▪ FileClient.write
bpDelete	▪ FileClient.receive ▪ FileClient.write ▪ otdDB2 Connect.DB_EMPLOYEEDelete ▪ FileClient.write
bpPsSelect	▪ FileClient.receive ▪ FileClient.write ▪ otdDB2 Connect.Select_psPSSelectAll ▪ Decision ▪ FileClient.write (inside a Scope renamed “No record”) ▪ otdInputDTD_DBemployees.marshal (inside a Scope renamed “Records found”) ▪ FileClient.write (inside a Scope renamed “Records found”) ▪ FileClient.write
bpTableSelect	▪ FileClient.receive ▪ FileClient.write ▪ otdDB2 Connect.DB_EMPLOYEESelectAll ▪ otdInputDTD_DBemployees.marshal ▪ FileClient.write ▪ FileClient.write

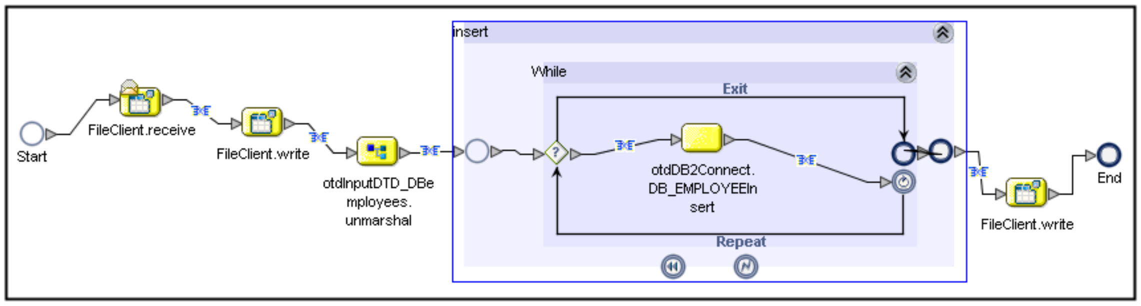
Configuring the bpInsert Modeling Elements

Business Rules, created between the Business Process Activities, allow you to configure the relationships between the input and output Attributes of the Activities using the Business Process Designer’s Business Rule Designer.

Once you have connected the modeling elements together, begin adding the business processes necessary to facilitate the Insert operation. See Figure 25 for an illustration of how all the modeling elements appear when connected.

Note: *Review the eInsight Business Process Manager User’s Guide for a more detailed description of the steps required to connect and add business rules to a modeling elements in a business process.*

Figure 25 bpInsert Business Process

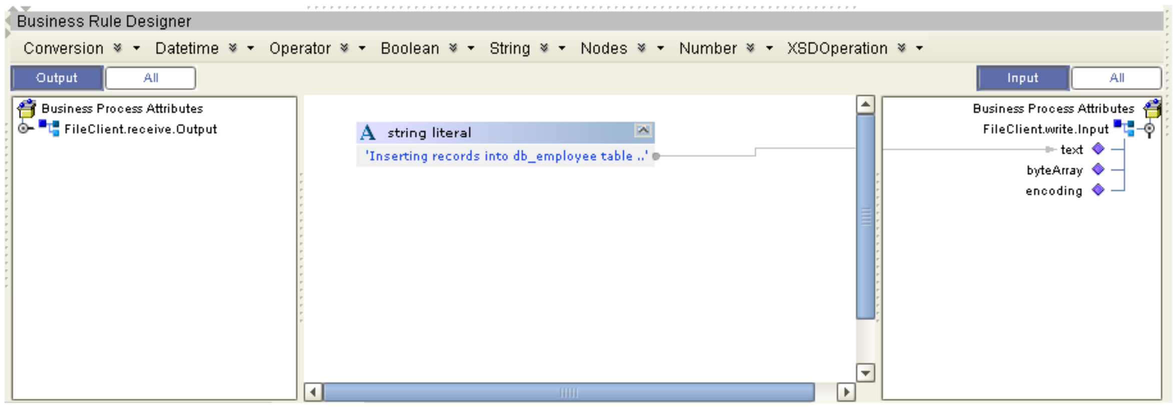


Steps required to configure the bpInsert business process:

Configure the following business rules in the bpInsert business process.

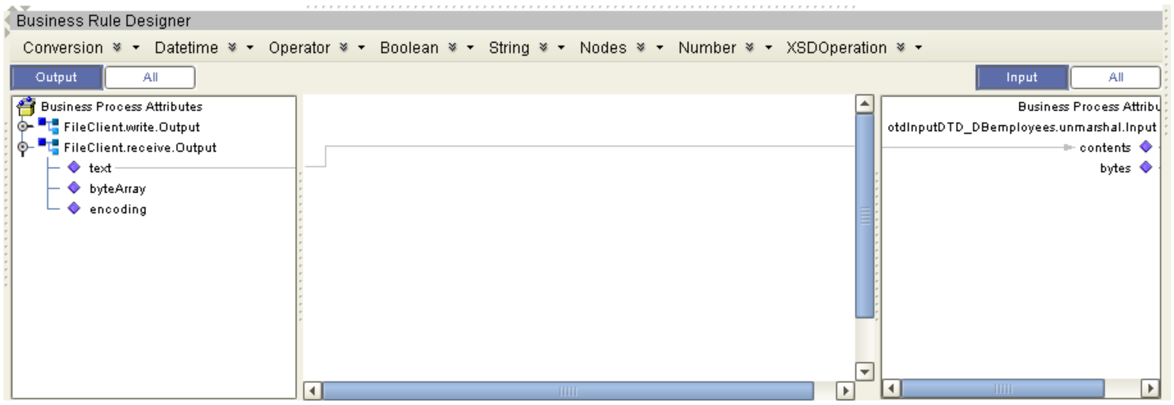
1. Configure the business rule between **FileClient.receive** and **FileClient.write** as seen in Figure 26.

Figure 26 bpInsert Business Rule # 1



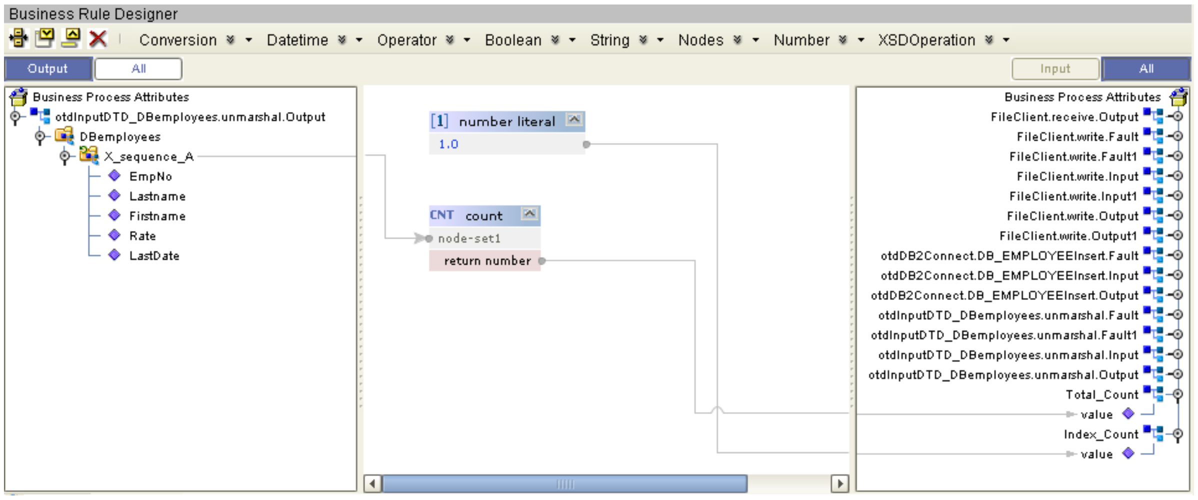
2. Configure the business rule between the **FileClient.write** Activity and **otdInputDTD_DBemployees.unmarshal** Activity as seen in Figure 27.

Figure 27 bpInsert Business Rule # 2



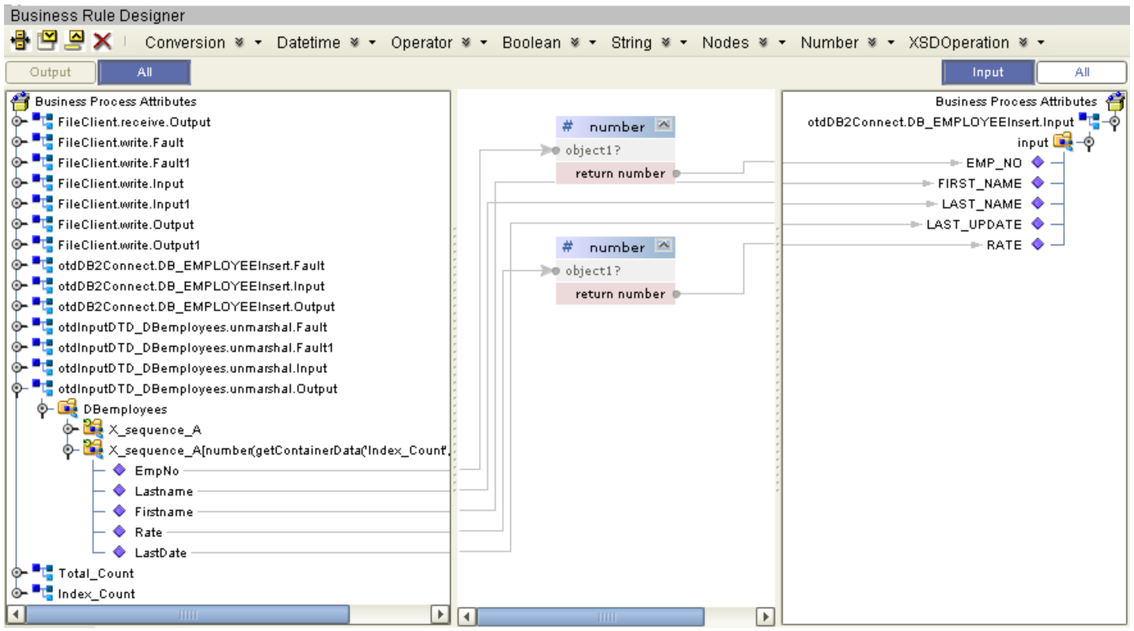
3 Configure the business rule between **otdInputDTD_DBemployees.unmarshal** and the **Insert** (Scope element). The **bpInsert** project uses integer variables **total_count** and **index_count** for multiple records insert as seen in Figure 28 and Figure 30.

Figure 28 bpInsert Business Rule # 3


Business Rule Designer
Conversion
Datetime
Operator
Boolean
String
Nodes
Number
XSDOperation
Output
All
Input
All
Business Process Attributes
otdInputDTD_DBemployees.unmarshal.Output
DBemployees
X_sequence_A
EmpNo
Lastname
Firstname
Rate
LastDate
[1] number literal
1.0
CNT count
node-set1
return number
Business Process Attributes
FileClient.receive.Output
FileClient.write.Fault
FileClient.write.Fault1
FileClient.write.Input
FileClient.write.Input1
FileClient.write.Output
FileClient.write.Output1
otdDB2Connect.DB_EMPLOYEEInsert.Fault
otdDB2Connect.DB_EMPLOYEEInsert.Input
otdDB2Connect.DB_EMPLOYEEInsert.Output
otdInputDTD_DBemployees.unmarshal.Fault
otdInputDTD_DBemployees.unmarshal.Fault1
otdInputDTD_DBemployees.unmarshal.Input
otdInputDTD_DBemployees.unmarshal.Output
Total_Count
value
Index_Count
value

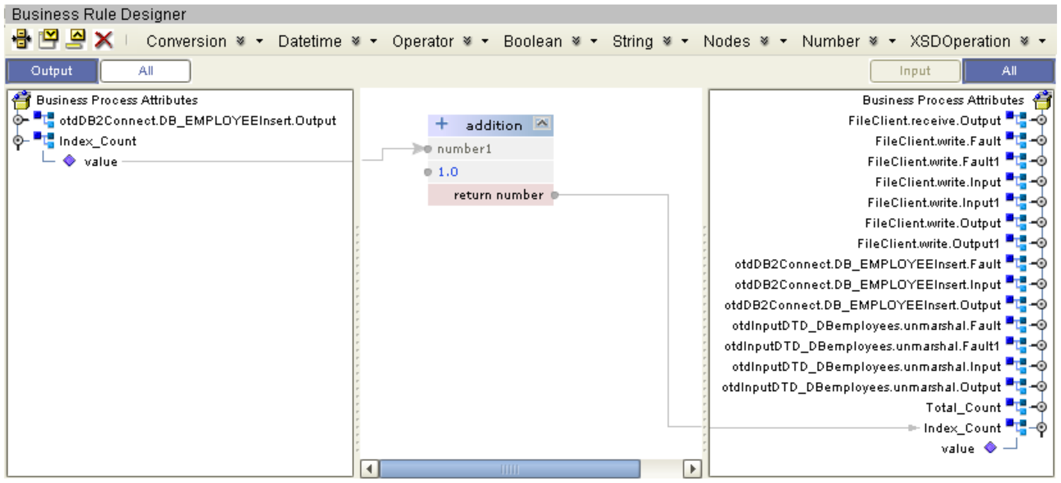
4 Configure the business rule in the **While** statement that connects to the **otdDB2 Connect.DB_EMPLOYEEInsert** Activity.

Figure 29 bpInsert Business Rule # 4


Business Rule Designer
Conversion
Datetime
Operator
Boolean
String
Nodes
Number
XSDOperation
Output
All
Input
All
Business Process Attributes
FileClient.receive.Output
FileClient.write.Fault
FileClient.write.Fault1
FileClient.write.Input
FileClient.write.Input1
FileClient.write.Output
FileClient.write.Output1
otdDB2Connect.DB_EMPLOYEEInsert.Fault
otdDB2Connect.DB_EMPLOYEEInsert.Input
otdDB2Connect.DB_EMPLOYEEInsert.Output
otdInputDTD_DBemployees.unmarshal.Fault
otdInputDTD_DBemployees.unmarshal.Fault1
otdInputDTD_DBemployees.unmarshal.Input
otdInputDTD_DBemployees.unmarshal.Output
DBemployees
X_sequence_A
X_sequence_A[number(getContainerData('Index_Count'
EmpNo
Lastname
Firstname
Rate
LastDate
Total_Count
Index_Count
# number
object1?
return number
# number
object1?
return number
Business Process Attributes
otdDB2Connect.DB_EMPLOYEEInsert.Input
input
EMP_NO
FIRST_NAME
LAST_NAME
LAST_UPDATE
RATE

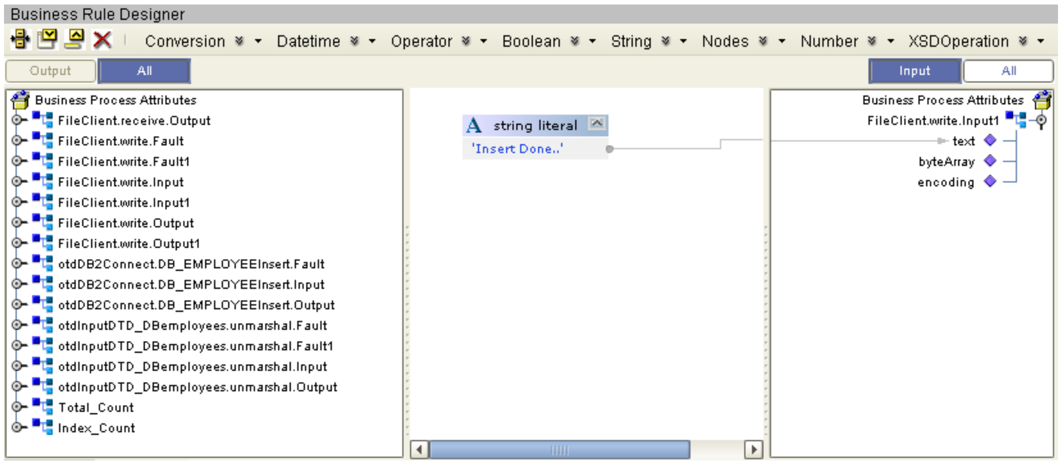
5 Configure the business rule in the **While** statement that connects from the **otdDB2 Connect.DB_EMPLOYEEInsert** Activity.

Figure 30 bpInsert Business Rule # 5


Business Rule Designer
Conversion
Datetime
Operator
Boolean
String
Nodes
Number
XSDOperation
Output
All
Input
All
Business Process Attributes
otdDB2Connect.DB_EMPLOYEEInsert.Output
Index_Count
value
addition
number1
1.0
return number
Business Process Attributes
FileClient.receive.Output
FileClient.write.Fault
FileClient.write.Fault1
FileClient.write.Input
FileClient.write.Input1
FileClient.write.Output
FileClient.write.Output1
otdDB2Connect.DB_EMPLOYEEInsert.Fault
otdDB2Connect.DB_EMPLOYEEInsert.Input
otdDB2Connect.DB_EMPLOYEEInsert.Output
otdInputDTD_DBemployees.unmarshal.Fault
otdInputDTD_DBemployees.unmarshal.Fault1
otdInputDTD_DBemployees.unmarshal.Input
otdInputDTD_DBemployees.unmarshal.Output
Total_Count
Index_Count
value

6 Configure the business rule from the **Insert** (Scope element) to the **FileClient.write** Activity.

Figure 31 bpInsert Business Rule # 6


Business Rule Designer
Conversion
Datetime
Operator
Boolean
String
Nodes
Number
XSDOperation
Output
All
Input
All
Business Process Attributes
FileClient.receive.Output
FileClient.write.Fault
FileClient.write.Fault1
FileClient.write.Input
FileClient.write.Input1
FileClient.write.Output
FileClient.write.Output1
otdDB2Connect.DB_EMPLOYEEInsert.Fault
otdDB2Connect.DB_EMPLOYEEInsert.Input
otdDB2Connect.DB_EMPLOYEEInsert.Output
otdInputDTD_DBemployees.unmarshal.Fault
otdInputDTD_DBemployees.unmarshal.Fault1
otdInputDTD_DBemployees.unmarshal.Input
otdInputDTD_DBemployees.unmarshal.Output
Total_Count
Index_Count
string literal
'Insert Done..'
Business Process Attributes
FileClient.write.Input1
text
byteArray
encoding

Configuring the bpUpdate Modeling Elements

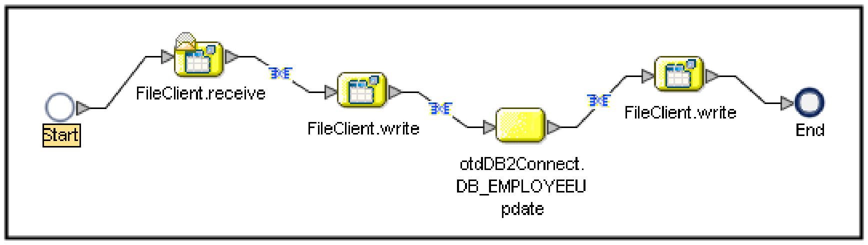
The bpUpdate business process describes how to update a record in the DB2 Connect database using the Business Process Designer.

Once you have connected the modeling elements together, begin adding the business processes necessary to facilitate the Update operation. Figure 32 illustrates how all the modeling elements appear when connected.

Note: *The where clause in the business rule reads the trigger value as a placeholder for input. This permits you to modify the query to select a specific record. Also note that all records are selected from the database when the TriggerUpdate.in file is empty.*

Note: *Review the eInsight Business Process Manager User's Guide for a more detailed description of the steps required to connect and add business rules to a modeling elements in a business process.*

Figure 32 bpUpdate Business Process

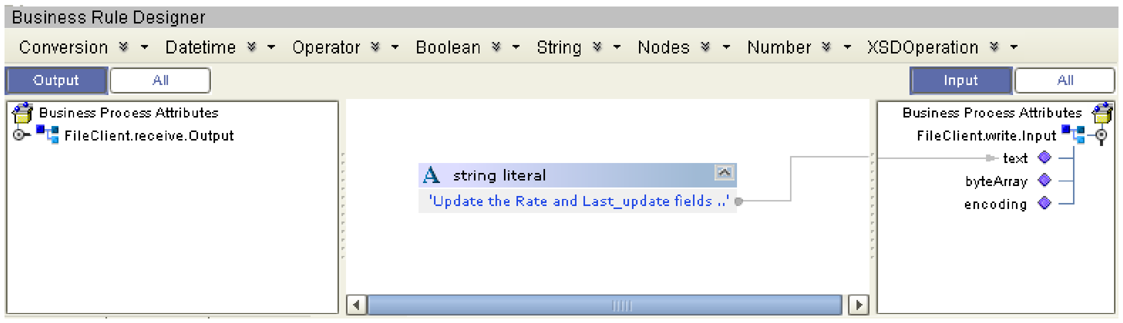


Steps required to configure the bpUpdate business process:

Configure the following business rules in the bpUpdate business process.

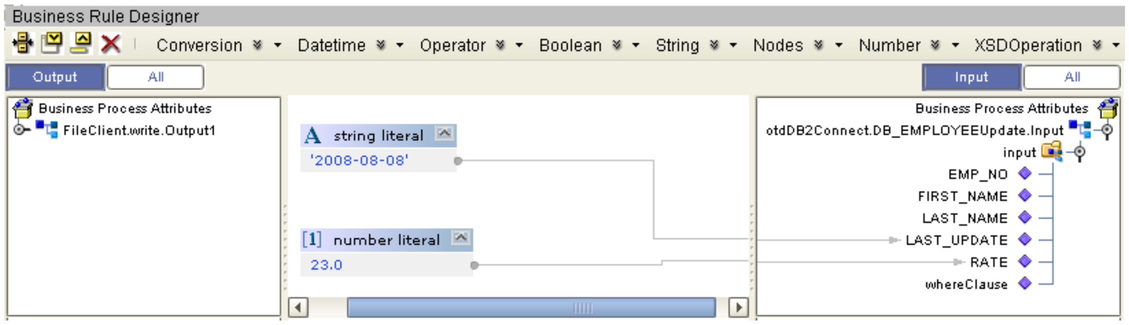
1 Configure the business rule between **FileClient.receive** and **FileCleint.write** as seen in Figure 33.

Figure 33 bpUpdate Business Rule # 1



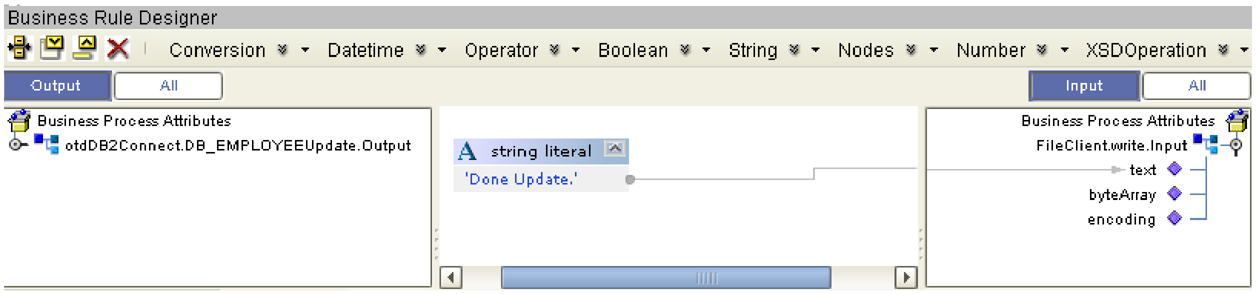
2 Configure the business rule between the **FileClient.write** Activity and **otdDB2 Connect.DB_EMPLOYEEUpdate** Activity as seen in Figure 34.

Figure 34 bpUpdate Business Rule # 2



3 Configure the business rule between **otdDB2 Connect.DB_EMPLOYEEUpdate** and the **FileClient.write** activity.

Figure 35 bpUpdate Business Rule # 3



4 Configure the business rule in the **While** statement that connects to the **otdDB2 Connect.DB_EMPLOYEEInsert** Activity.

5 Configure the business rule from the **Insert** (Scope element) to the **FileClient.write** Activity.

Configuring the bpDelete Modeling Elements

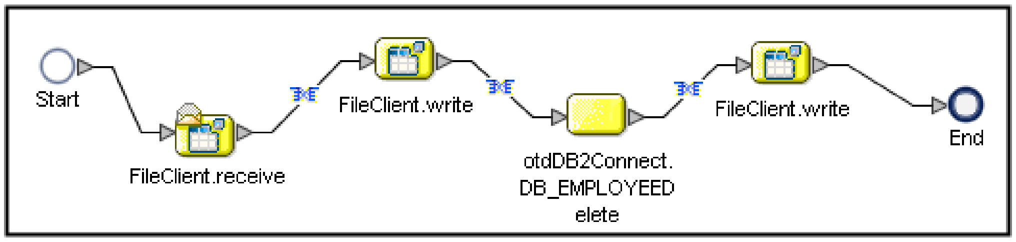
The bpDelete business process describes how to delete a record in the DB2 Connect database using the Business Process Designer.

Once you have connected the modeling elements together, begin adding the business processes necessary to facilitate the Delete operation. See Figure 36 for an illustration of how all the modeling elements appear when connected.

Note: *The where clause in the business rule reads the trigger value as a placeholder for input. This permits you to modify the query to select a specific record. Also note that all records are selected from the database when the TriggerDelete.in file is empty.*

Note: *Review the eInsight Business Process Manager User's Guide for a more detailed description of the steps required to connect and add business rules to a modeling elements in a business process.*

Figure 36 bpDelete Business Process

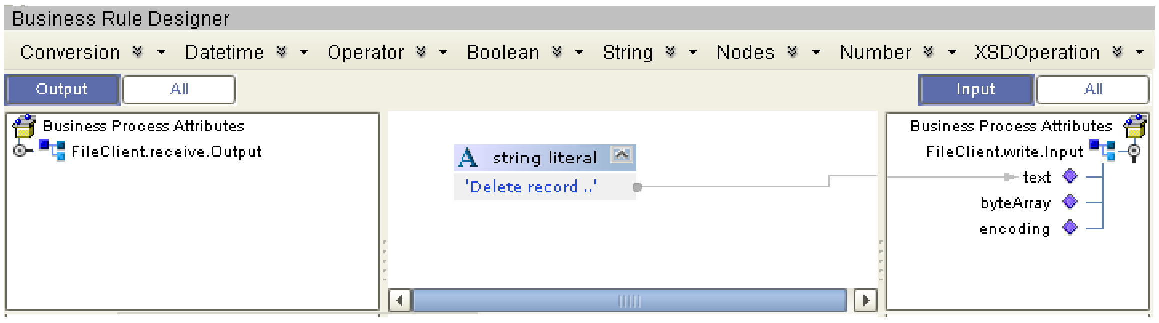


Steps required to configure the bpDelete business process:

Configure the following business rules to the bpDelete business process.

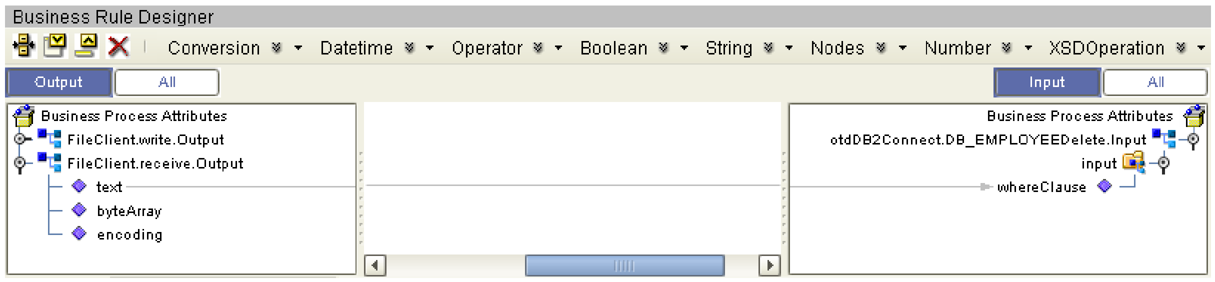
1 Configure the business rule between **FileClient.receive** and **FileCleint.write** as seen in Figure 37.

Figure 37 bpDelete Business Rule # 1



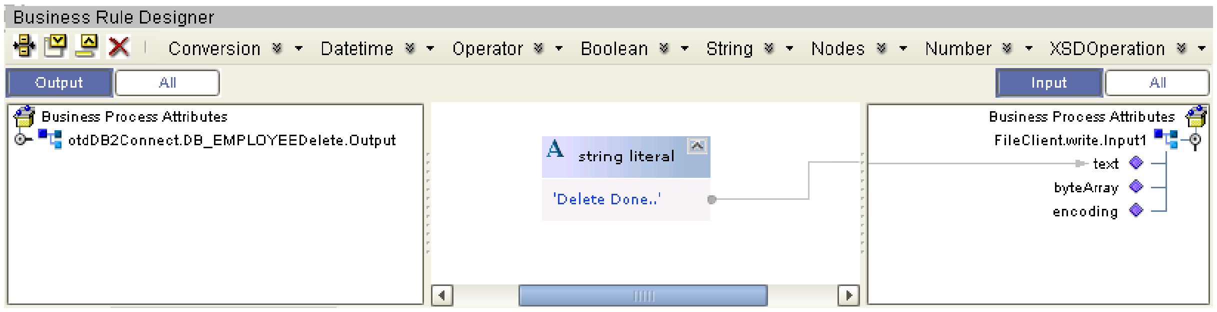
2 Configure the business rule between the **FileClient.write** Activity and **otdDB2 Connect.DB_EMPLOYEEDelete** Activity as seen in Figure 38.

Figure 38 bpDelete Business Rule # 2



3 Configure the business rule between the **otdDB2 Connect.DB_EMPLOYEEDelete** Activity and the **FileClient.write** Activity as seen in Figure 39.

Figure 39 bpDelete Business Rule # 3



Configuring the bpTableSelect Modeling Elements

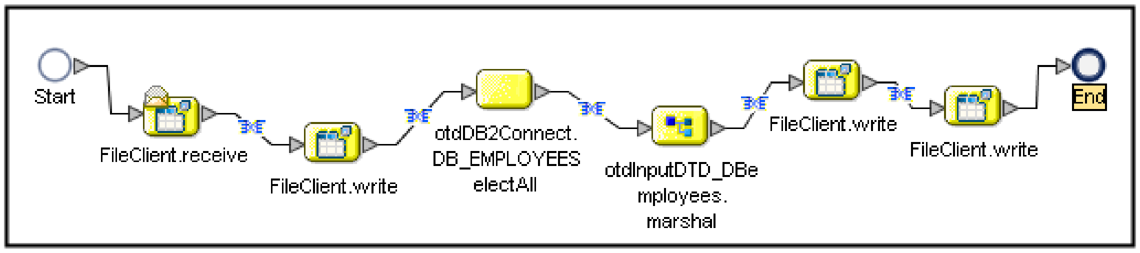
The bpTableSelect business process is describes how to select all records the DB2 Connect database using the Business Process Designer.

Once you have connected the modeling elements together, begin adding the business processes necessary to facilitate the SelectAll operation. See Figure 40 for an illustration of how all the modeling elements appear when connected.

Note: *The where clause in the business rule reads the trigger value as a placeholder for input. This permits you to modify the query to select a specific record. Also note that all records are selected from the database when the TriggerTableSelect.in file is empty.*

Note: *Review the eInsight Business Process Manager User's Guide for a more detailed description of the steps required to connect and add business rules to a modeling elements in a business process.*

Figure 40 bpTableSelect Business Process

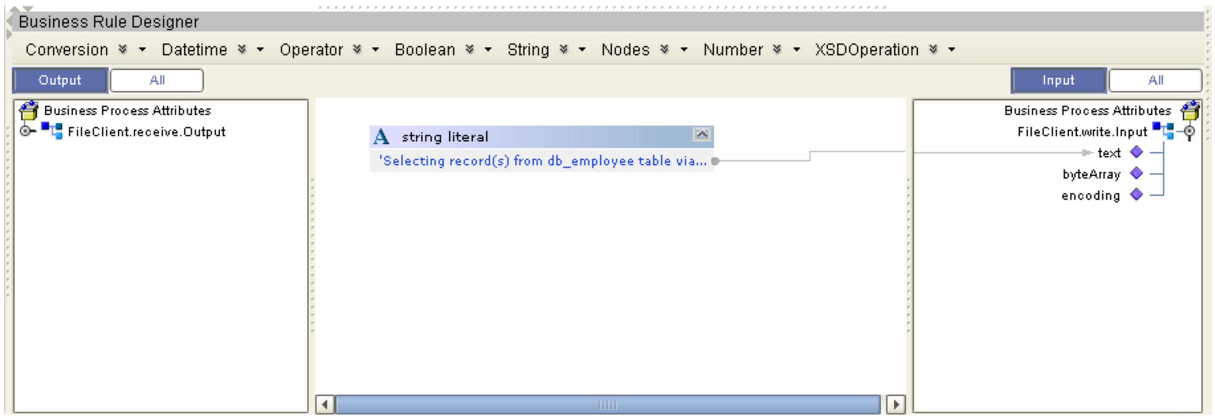


Steps required to configure the bpTableSelect business process:

Configure the following business rules in the bpTableSelect business process.

1 Configure the business rule between **FileClient.receive** and **FileCleint.write** as seen in Figure 41.

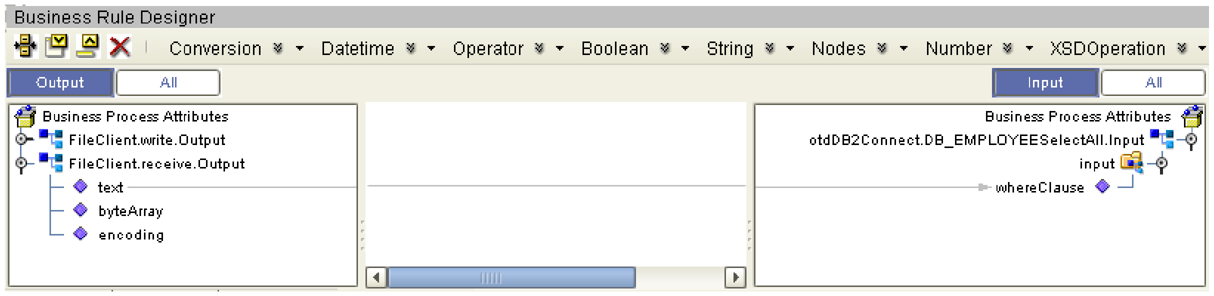
Figure 41 bpTableSelect Business Rule # 1



'Selecting record(s) from db_employee table via...

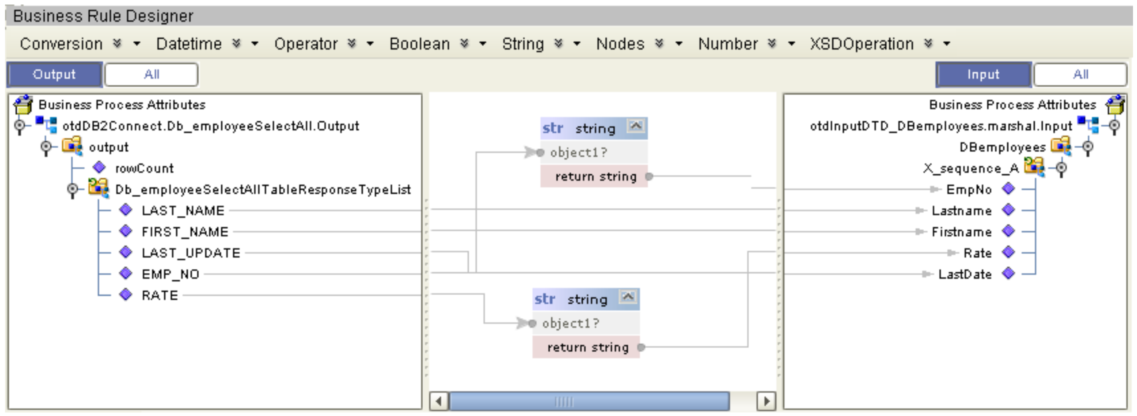
2 Configure the business rule between the **FileClient.write** Activity and **otdDB2 Connect.DB_EMPLOYEESelectAll** Activity as seen in Figure 42.

Figure 42 bpTableSelect Business Rule # 2



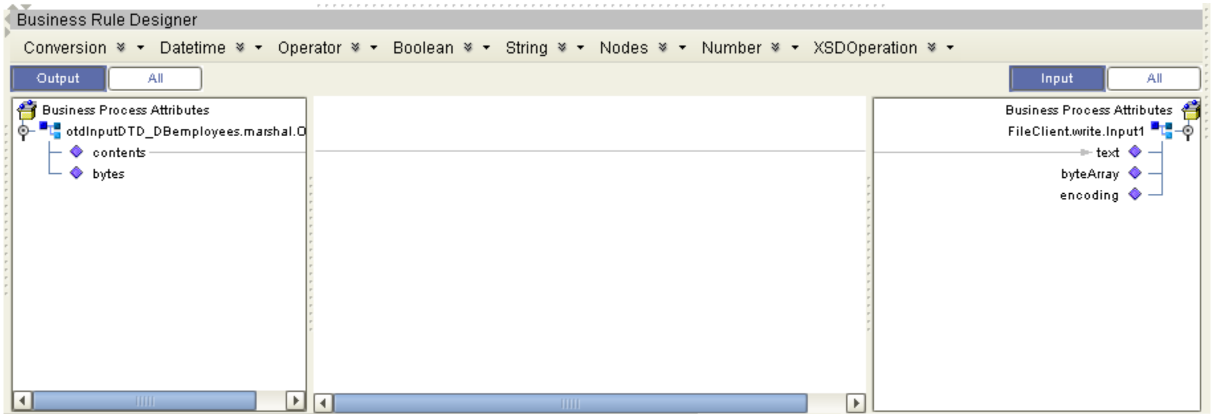
3 Configure the business rule between the **otdDB2 Connect.DB_EMPLOYEESelectAll** Activity and the **otdInputDTD_DBemployees.marshal** Activity as seen in Figure 43.

Figure 43 bpSelectTable Business Rule # 3



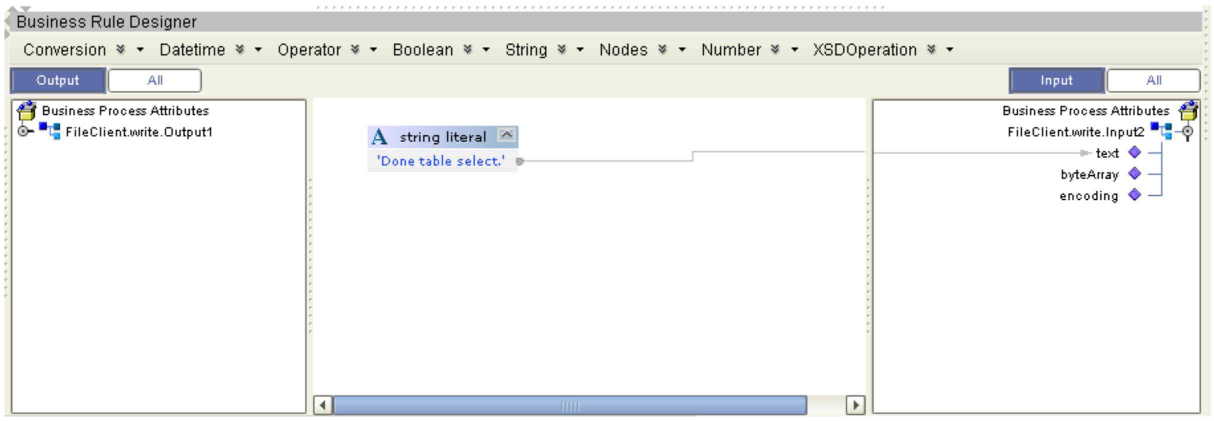
4 Configure the business rule between the **otdInputDTD_DBemployees.marshal** Activity and the **FileClient.write** Activity as seen in Figure 44.

Figure 44 bpTableSelect Business Rule # 4



5 Configure the business rule between the **FileClient.write** Activity and the **FileClient.write** Activity as seen in Figure 45.

Figure 45 bpTableSelect Business Rule # 5



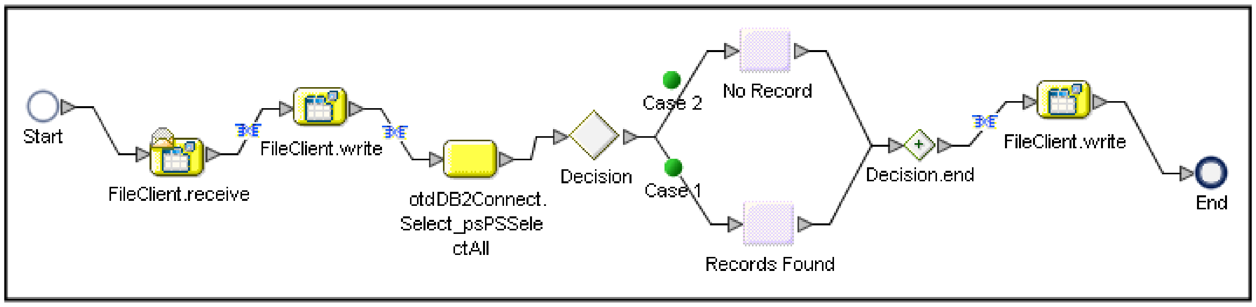
Configuring the bpPsSelect Modeling Elements

The bpPsSelect business process describes how to use a Prepared Statement query to select all records in the DB2 Connect database via the Business Process Designer.

Once you have connected the modeling elements together, begin adding the business processes necessary to facilitate the SelectAll operation. See Figure 46 for an illustration of how all the modeling elements appear when connected.

Note: *Review the eInsight Business Process Manager User's Guide for a more detailed description of the steps required to connect and add business rules to a modeling elements in a business process.*

Figure 46 bpPsSelect Business Process

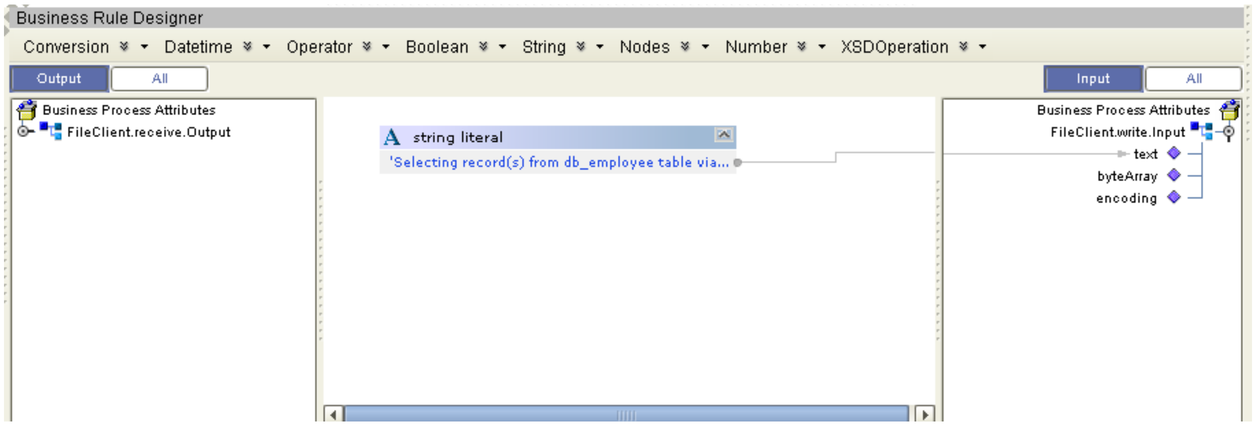


Steps required to configure the bpPsSelect business process:

Configure the following business rules in the bpPsSelect business process.

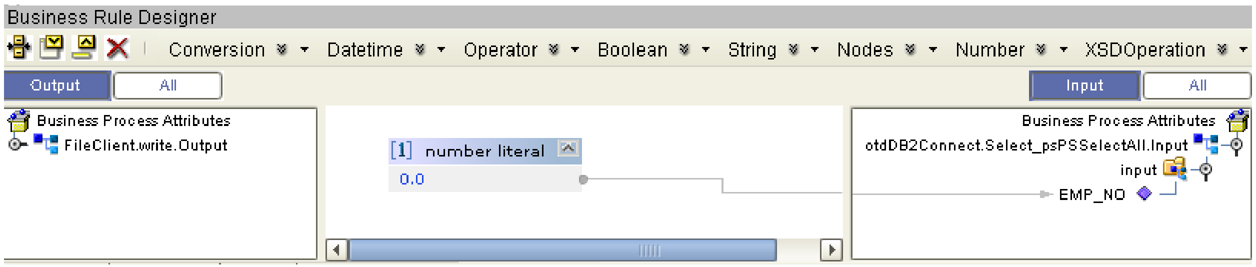
1 Configure the business rule between **FileClient.receive** and **FileCleint.write** as seen in Figure 41.

Figure 47 bpSelectTable Business Rule # 1



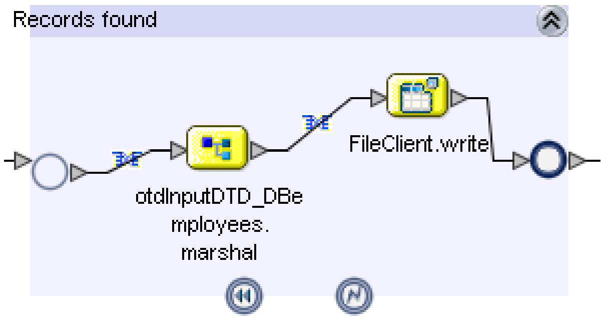
2 Configure the business rule between **FileClient.write** and **otdDB2 Connect.Select_psPSSelectAll** as seen in Figure 48.

Figure 48 bpSelectTable Business Rule # 2



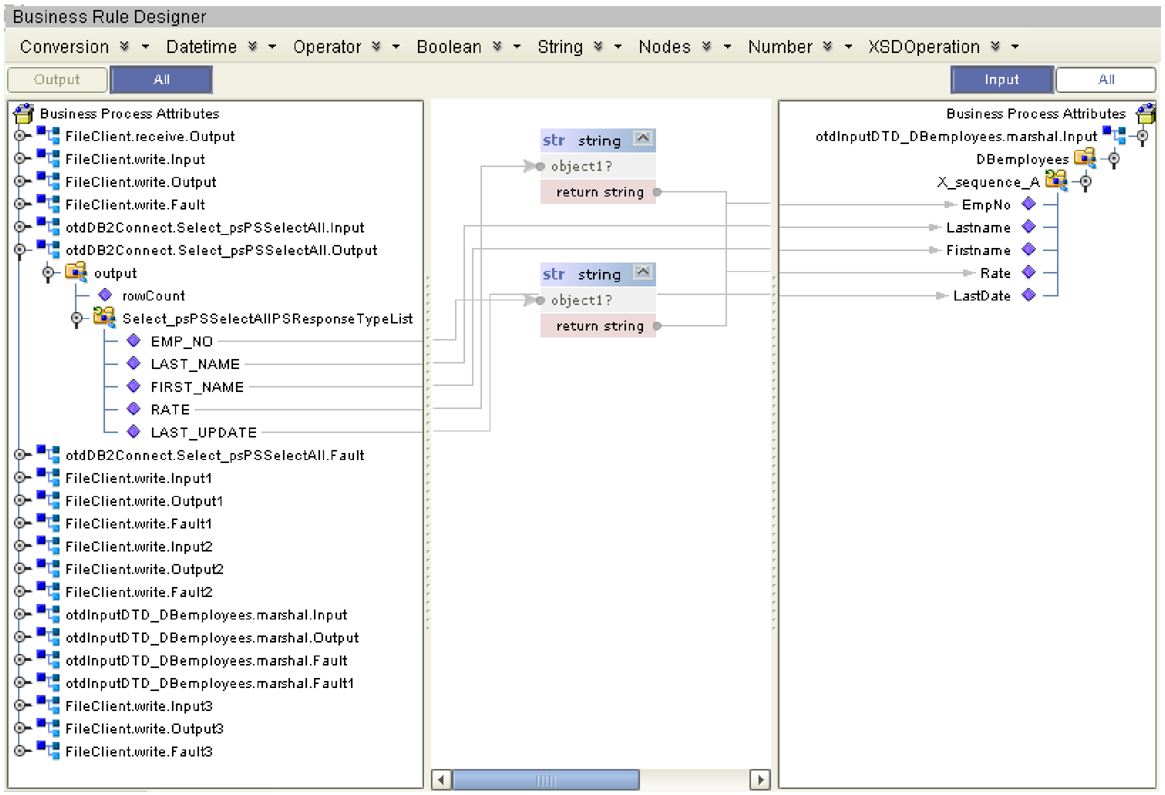
3 Configure **Case 1** of the Decision branching activity. This requires adding business rules between the **otdInputDTD_DBemployees.marshal** and the **FileClient.write** activities within the Scope element.

Figure 49 Activities within Case 1 Scope



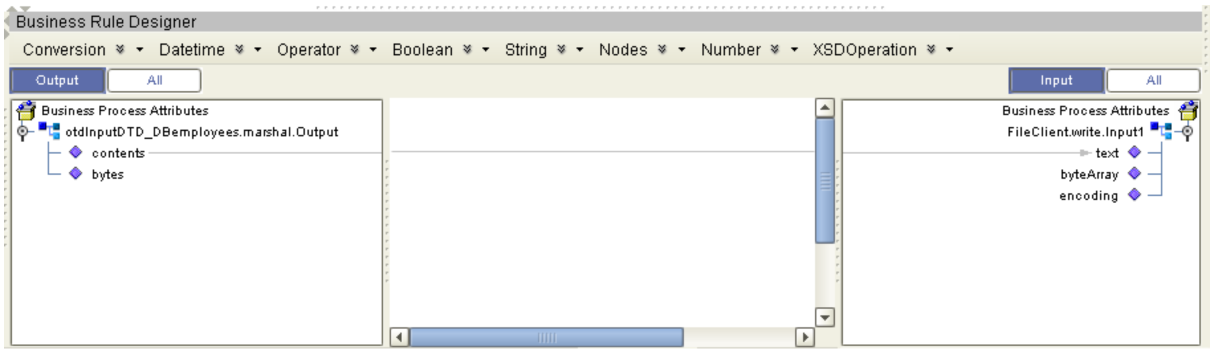
4 Configure the business rule between the start of the Scope element in **Case 1** and the **otdInputDTD_DBemployees.marshal** activity, as seen in Figure 50.

Figure 50 Case 1 Scope Business Rule # 1



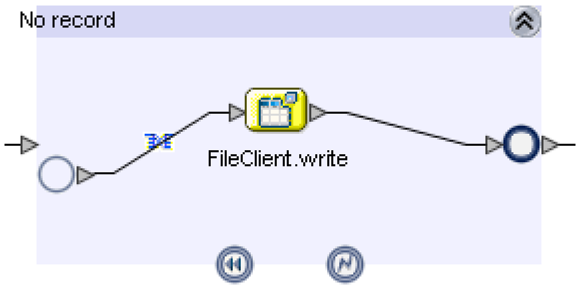
5 Configure the business rule between **otdInputDTD_DBemployees.marshal** and **FileCleint.write** in the Scope element, as seen in Figure 51.

Figure 51 Case 1 Scope Business Rule # 2



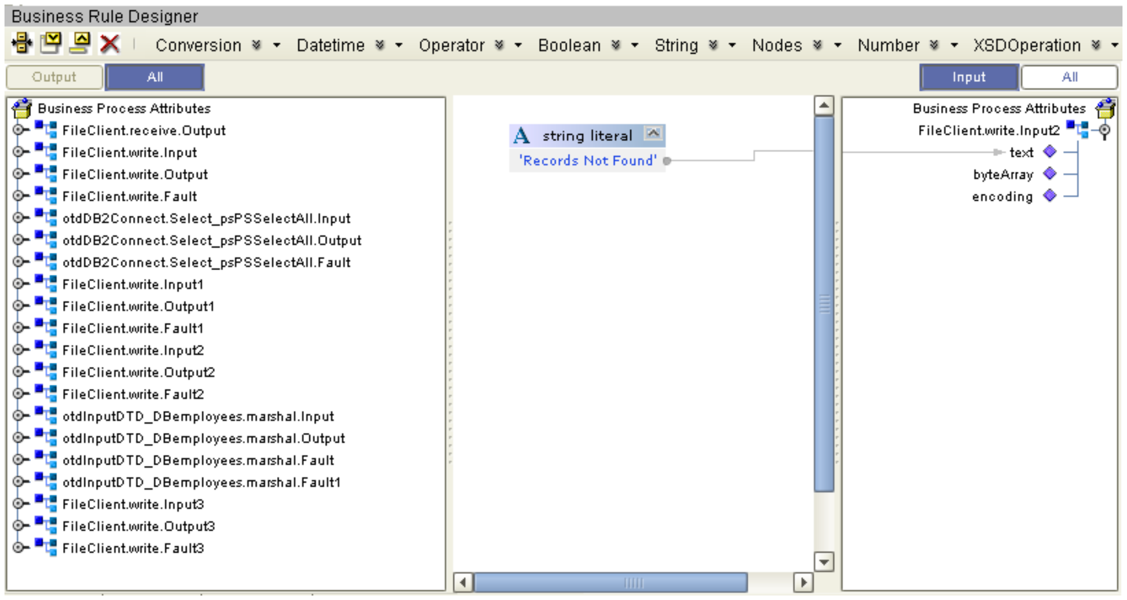
6 Configure Case 2 of the Decision branching activity. This requires adding business rules between the **otdInputDTD_DBemployees.marshal** and the **FileClient.write** activities within the Scope element.

Figure 52 Activities within Case 2 Scope



7 Configure the business rule between the start of the Scope element in **Case 2** and the **FileClient.Write** activity, as seen in Figure 53.

Figure 53 Case 2 Scope Business Rule # 1



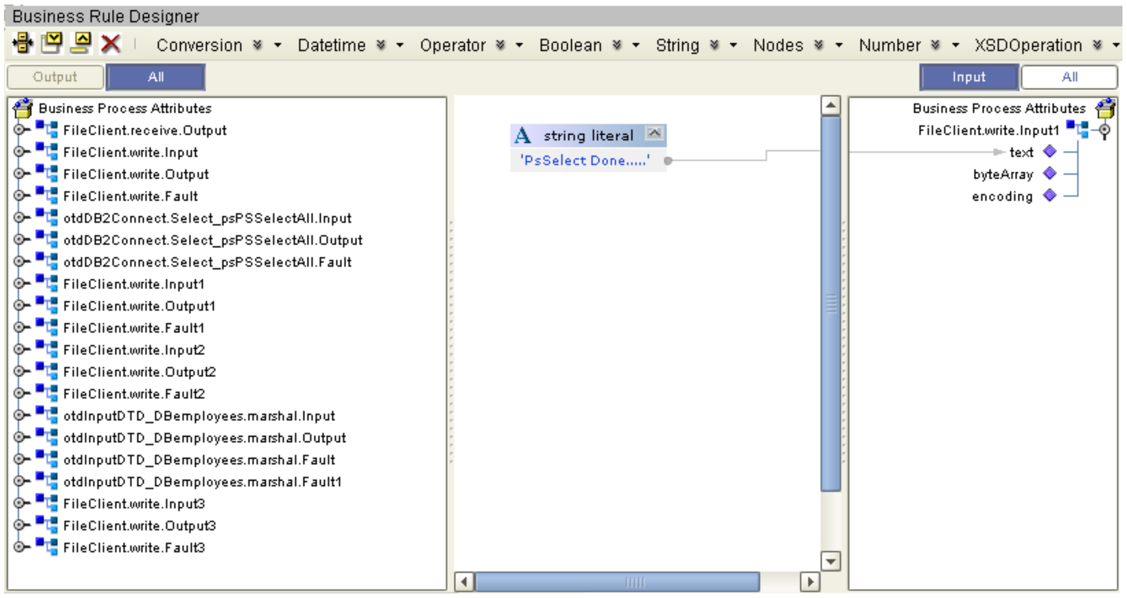
Business Process Attributes
- FileClient.receive.Output
- FileClient.write.Input
- FileClient.write.Output
- FileClient.write.Fault
- otdDB2Connect.Select_psPSSelectAll.Input
- otdDB2Connect.Select_psPSSelectAll.Output
- otdDB2Connect.Select_psPSSelectAll.Fault
- FileClient.write.Input1
- FileClient.write.Output1
- FileClient.write.Fault1
- FileClient.write.Input2
- FileClient.write.Output2
- FileClient.write.Fault2
- otdInputDTD_DBemployees.marshal.Input
- otdInputDTD_DBemployees.marshal.Output
- otdInputDTD_DBemployees.marshal.Fault
- otdInputDTD_DBemployees.marshal.Fault1
- FileClient.write.Input3
- FileClient.write.Output3
- FileClient.write.Fault3

'Records Not Found'

Business Process Attributes
- FileClient.write.Input2
  - text
  - byteArray
  - encoding

8 Configure the business rule between the **Decision.end** Element and the **FileClient.write** Activity, as seen in Figure 54.

Figure 54 bpSelectTable Business Rule # 3



Business Process Attributes
FileClient.receive.Output
FileClient.write.Input
FileClient.write.Output
FileClient.write.Fault
otdDB2Connect.Select_psPSSelectAll.Input
otdDB2Connect.Select_psPSSelectAll.Output
otdDB2Connect.Select_psPSSelectAll.Fault
FileClient.write.Input1
FileClient.write.Output1
FileClient.write.Fault1
FileClient.write.Input2
FileClient.write.Output2
FileClient.write.Fault2
otdInputDTD_DBemployees.marshal.Input
otdInputDTD_DBemployees.marshal.Output
otdInputDTD_DBemployees.marshal.Fault
otdInputDTD_DBemployees.marshal.Fault1
FileClient.write.Input3
FileClient.write.Output3
FileClient.write.Fault3

Business Process Attributes
FileClient.write.Input1

6.5.4 Creating the Connectivity Map

The Connectivity Map provides a canvas for assembling and configuring a Project’s components.

1. From the Project Explorer tree, right-click the new **prjDB2 Connect_BPEL** Project and select **New > Connectivity Map** from the shortcut menu.
2. The New Connectivity Map appears, and a node for the Connectivity Map is added under the Project on the Project Explorer tree labeled **CMap1**.

   Create four additional Connectivity Maps—**CMap2**, **CMap3**. **CMap4**, and **CMap5**—and rename them as follows:

   - cmDelete
   - cmInsert
   - cmPsSelect
   - cmTableSelect
   - cmUpdate

The icons in the toolbar represent the available components used to populate the Connectivity Map canvas.

Populating the Connectivity Map

Add the Project components to the Connectivity Map by dragging the icons from the toolbar to the canvas.

Each Connectivity Map in the **prjDB2 Connect_BPEL** sample Project requires the following components:

- File External Application (2)
- DB2 Connect External Application
- Business Process

Any eWay added to the Connectivity Map is associated with an External System. To establish a connection to DB2 Connect, first select DB2 Connect as an External System to use in your Connectivity Map.

To Select a DB2 Connect External System

1. Click the **External Application** icon on the Connectivity Map toolbar.
2. Select the external systems necessary to create your Project (for this sample, DB2 Connect and **File**). Icons representing the selected external systems are added to the Connectivity Map toolbar.
3. Rename the following components and then save changes to the Repository:
   - File1 to FileClientIN
   - File2 to FileClientOUT
   - DB2 Connect1 to eaDB2 ConnectOUT

To Select a DB2 Connect Business Process

1. Drag a business process from the Enterprise Explorer Project Explorer onto the corresponding Connectivity Map. For example, drag the **dbDelete** business process onto the **cmDelete** Connectivity Map.
2. Save your changes to the Repository

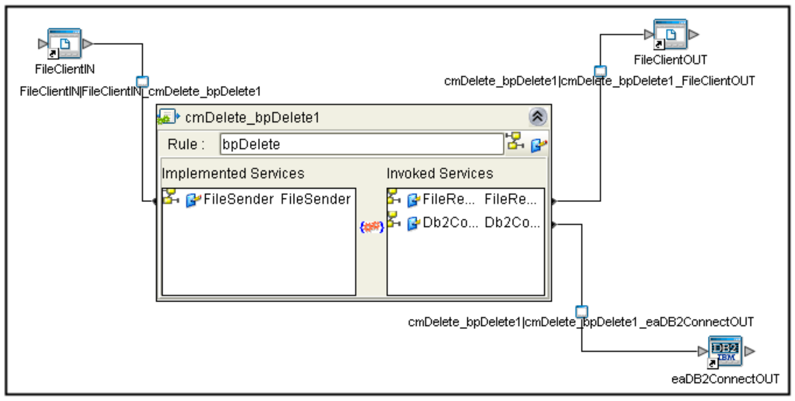
Binding the eWay Components

The final step in creating a Connectivity Map is binding the eWay components together.

To bind eWay Components Together

1. Open one of the Connectivity Maps and double-click a Business Process, for example the **bpDelete** Business Process in the **cmDelete** Connectivity Map. The **bpDelete** Binding dialog box appears.
2. From the **bpDelete** Binding dialog box, map **FileSender** (under Implemented Services) to the **FileClientIN** (File) External Application. To do this, click on **FileSender** in the **bpDelete** Binding dialog box, and drag the cursor to the **FileClientIN** External Application in the Connectivity Map. A link is now visible between **FileClientIN** and **bpDelete**.
3. From the **bpDelete** Binding dialog box, map **DB2 Connect_otdDB2 Connect** (under Invoked Services) to the **eaDB2 ConnectOUT** External Application.
4. From the **bpDelete** Binding dialog box, map **FileReceiver** to the **FileClientOUT** External Application, as seen in Figure 55.

Figure 55 Connectivity Map - Associating (Binding) the Project's Components



5 Minimize the **bpDelete** Binding dialog box by clicking the chevrons in the upper-right corner.

6 Save your current changes to the Repository, and then repeat this process for each of the other Connectivity Maps.

6.5.5 Creating an Environment

Environments include the external systems, Logical Hosts, integration servers and message servers used by a Project and contain the configuration information for these components. Environments are created using the Enterprise Designer's Environment Editor.

1 From the Enterprise Designer's Enterprise Explorer, click the **Environment Explorer** tab.

2 Right-click the Repository and select **New Environment**. A new Environment is added to the Environment Explorer tree.

3 Rename the new Environment to **envDB2 ConnectProj**.

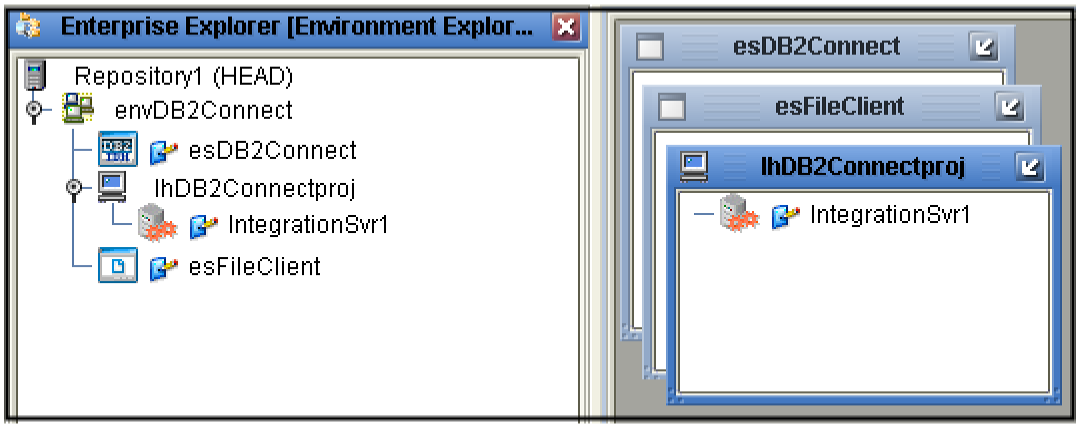
4 Right-click **envDB2 ConnectProj** and select **New DB2 Connect External System**. Name the External System **esDB2 Connect**. Click **OK**. **esDB2 Connect** is added to the Environment Editor.

5 Right-click **envDB2 ConnectProj** and select **New File External System**. Name the External System **esFileClient**. Click **OK**. **esFileClient** is added to the Environment Editor.

6 Right-click **envDB2 ConnectProj** and select **New Logical Host**. The **LogicalHost1** box is added to the Environment, and **LogicalHost1** is added to the Environment Editor tree.

7 Right-click **LogicalHost1** and select **New Sun SeeBeyond Integration Server**. A new Integration Server (**IntegrationSvr1**) is added to the Environment Explorer tree under **LogicalHost1** (see Figure 56).

Figure 56 Environment Editor - envDB2 ConnectProj



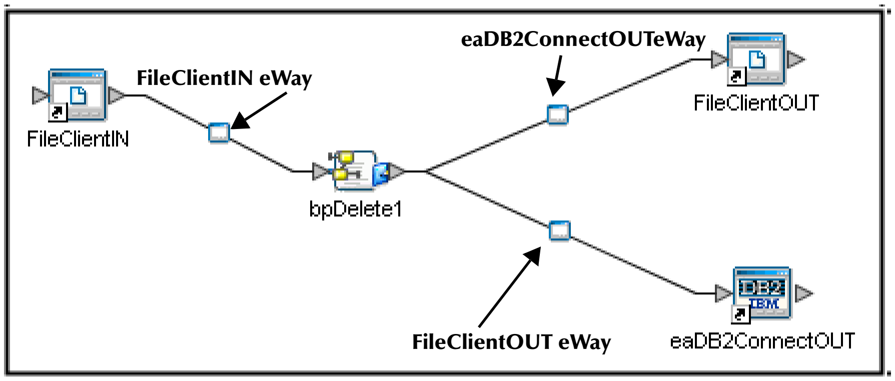
8 Save your current changes to the Repository.

6.5.6 Configuring the eWays

eWays facilitate communication and movement of data between the external applications and the eGate system. Each Connectivity Map in the The **prjDB2 Connect_BPEL** sample Project use three eWays that are represented as a nodes between the External Applications and the Business Process, as seen in Figure 57.

You must configure eWay properties in both the Connectivity Map and the Environment Explorer.

Figure 57 eWays in the cmDelete Connectivity Map



Steps to Configure the eWay Properties

Steps required to configure the eWay properties:

1 Double-click the **FileClientIN eWay on each of the Connectivity Maps and** modify the properties for your system, as seen in Table 26. Click **OK** to close the Properties Editor.

Table 26 FileClientIN eWay Property Settings

Connectivity Map	Property Name	Required Value
cmDelete	Input file name	TriggerDelete.in
cmInsert	Input file name	TriggerBpInsert.in
cmPsSelect	Input file name	TriggerPsSelect.in
cmTableSelect	Input file name	TriggerTableSelect.in
cmUpdate	Input file name	TriggerUpdate.in

2 Double-click the **FileClientIN eWay on each of the Connectivity Maps and** modify the properties for your system, as seen in Table 27. Click **OK** to close the Properties Editor.

Table 27 FileClientOUT eWay Property Settings

Connectivity Map	Property Name	Required Value
cmDelete	Input file name	BPEL_Delete_output%d.dat
cmInsert	Input file name	BPEL_Insert_output%d.dat
cmPsSelect	Input file name	BPEL_PsSelect_output%d.dat
cmTableSelect	Input file name	BPEL_TableSelect_output%d.dat.in
cmUpdate	Input file name	BPEL_Update_output%d.dat

Steps to Configure the Environment Explorer Properties

1 From the **Environment Explorer** tree, right-click the File External System (**esDB2 Connect** in this sample), and select **Properties**. The Properties Editor opens to the DB2 Connect eWay Environment configuration.

2 Modify the DB2 Connect eWay Environment configuration properties for your system, as seen in Table 28, and click **OK**.

Table 28 DB2 Connect eWay Environment Properties

Section	Property Name	Required Value
Configuration > Inbound DB2 Connect eWay > JDBC Connector settings	ServerName	Enter the name of the database server being used.
	DatabaseName	Enter the name of the particular database that is being used on the server.
	User	Enter the user account name for the database.
	Password	Enter the user account password for the database.

3 From the **Environment Explorer** tree, right-click the File External System (**esFileClient** in this sample), and select **Properties**. The Properties Editor opens to the DB2 Connect eWay Environment configuration.

4 Modify the File eWay Environment configuration properties for your system, as seen in Table 29, and click **OK**.

Table 29 File eWay Environment Properties

Section	Property Name	Required Value
Configuration > Inbound File eWay > Parameter Settings	Directory	Enter the directory that contains the input files (trigger files included in the sample Project). Trigger files include: ▪ TriggerBpInsert.in.~in ▪ TriggerDelete.in.~in ▪ TriggerPsSelect.in.~in ▪ TriggerTableSelect.in.~in ▪ TriggerUpdate.in.~in
Configuration > Outbound File eWay > Parameter Settings	Directory	Enter the directory where output files are written. In this sample Project, the output files include: ▪ BPEL_Delete_output0.dat ▪ BPEL_Insert_output0.dat ▪ BPEL_PsSelect_output0.dat ▪ BPEL_TableSelect_output0.dat ▪ BPEL_Update_output0.dat

6.5.7 Configuring the Integration Server

You must set your SeeBeyond Integration Server Password property before deploying your Project.

1 From the Environment Explorer, right-click **IntegrationSvr1** under your **Logical Host**, and select **Properties** from the shortcut menu. The Integration Server Properties Editor appears.

2 Click the **Password** property field under **SeeBeyond Integration Server Configuration**. An ellipsis appears in the property field.

3 Click the ellipsis. The **Password Settings** dialog box appears. Enter **STC** as the **Specific Value** and as the **Confirm Password**, and click **OK**.

4 Click **OK** to accept the new property and close the Properties Editor.

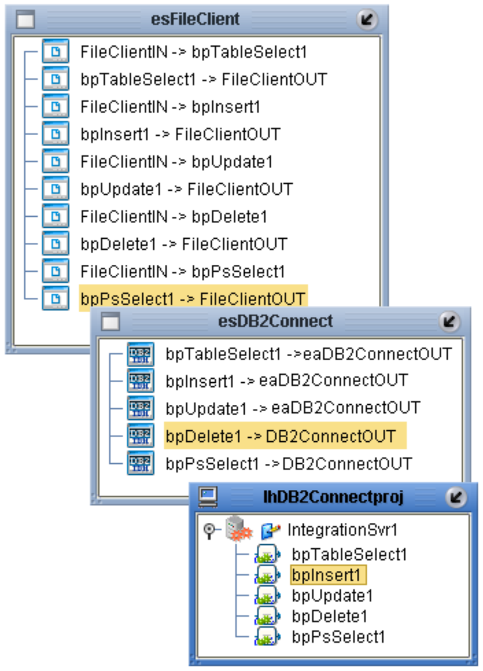
For more information on deploying a Project see the *Sun SeeBeyond Java™ Composite Application Platform Suite Deployment Guide*.

6.5.8 Creating the Deployment Profile

A Deployment Profile is used to assign services and message destinations to the integration server and message server. Deployment profiles are created using the Deployment Editor.

1. From the Enterprise Explorer's Project Explorer, right-click the **prjDB2 Connect_BPEL** Project and select **New** > **Deployment Profile**.
2. Enter a name for the Deployment Profile (for this sample **dpDB2 Connect_BPEL**). Select **envDB2 ConnectProj** as the Environment and click **OK**.
3. From the Deployment Editor toolbar, click the **Automap** icon. The Project's components are automatically mapped to their system windows, as seen in Figure 58.

Figure 58 Deployment Profile


esFileClient
FileClientIN -> bpTableSelect1
bpTableSelect1 -> FileClientOUT
FileClientIN -> bpInsert1
bpInsert1 -> FileClientOUT
FileClientIN -> bpUpdate1
bpUpdate1 -> FileClientOUT
FileClientIN -> bpDelete1
bpDelete1 -> FileClientOUT
FileClientIN -> bpPsSelect1
bpPsSelect1 -> FileClientOUT

esDB2Connect
bpTableSelect1 ->eaDB2ConnectOUT
bpInsert1 -> eaDB2ConnectOUT
bpUpdate1 -> eaDB2ConnectOUT
bpDelete1 -> DB2ConnectOUT
bpPsSelect1 ->DB2ConnectOUT

lhDB2Connectproj
IntegrationSvr1
bpTableSelect1
bpInsert1
bpUpdate1
bpDelete1
bpPsSelect1

6.5.9 Creating and Starting the Domain

To build and deploy your Project, you must first create a domain. A domain is an instance of a Logical Host. After the domain is created, the Project is built and then deployed.

Note: *You are only required to create a domain once when you install the Composite Application Platform Suite.*

Steps required to create and start the domain:

1. Navigate to your **<caps51>\logicalhost** directory (where <caps51> is the location of your Sun Java Composite Application Suite installation.

2 Double-click the **domainmgr.bat** file. The **Domain Manager** appears.

3 If you have already created a domain, select your domain in the Domain Manager and click the **Start an Existing Domain** button. Once your domain is started, a green check mark indicates that the domain is running.

4 If there are no existing domains, a dialog box indicates that you can create a domain now. Click **Yes**. The **Create Domain** dialog box appears.

5 Make any necessary changes to the **Create Domain** dialog box and click **Create**. The new domain is added to the Domain Manager. Select the domain and click the **Start an Existing Domain** button. Once your domain is started, a green check mark indicates that the domain is running.

Note: *For more information about creating and managing domains see the eGate Integrator System Administration Guide.*

6.5.10 Building and Deploying the Project

The Build process compiles and validates the Project’s Java files and creates the Project EAR file.

Build the Project

1 From the Deployment Editor toolbar, click the **Build** icon.

2 If there are any validation errors, a **Validation Errors** pane will appear at the bottom of the Deployment Editor and displays information regarding the errors. Make any necessary corrections and click **Build** again.

3 After the Build has succeeded you are ready to deploy your Project.

Deploy the Project

1 From the Deployment Editor toolbar, click the **Deploy** icon. Click **Yes** when the **Deploy** prompt appears.

2 A message appears when the project is successfully deployed. You can now test your sample.

6.5.11 Running the Sample Project

Perform the following steps to run your deployed sample Project:

1 Rename one of the trigger files included in the sample Project from **<filename>.in.~in** to **<filename>.in** to run the corresponding operation.

The File eWay polls the directory every five seconds for the input file name (as defined in the Inbound File eWay Properties window). The Business Process then transforms the data, and the File eWay sends the output to an Output file name (as defined in the outbound File eWay Properties window).

The Where Clause defined in the business rule recognizes the trigger as a placeholder for input, allowing a set condition, such as emp_no = 100, to determine the type of output data.

You can modify the following input files to view different output.

- TriggerTableSelect.in
- TriggerDelete.in
- TriggerUpdate.in

Having no content in these files causes the operation to read all records.

2 Verify the output data by viewing the sample output files. See **About the DB2 Connect eWay Sample Projects** on page 74 for more details on the types of output files used in this sample Project. The output files may change depending on the number of times you execute the sample Project, the input file, and also the content of your database table.

6.6 Creating the prjDB2 Connect_JCD Sample Project

The following provides step-by-step instructions for manually creating the prjDB2 Connect_JCD sample Project. The Type 2 Legacy driver is used for this project.

Steps required to create the sample project include:

- **Creating a Project** on page 102
- **Creating the OTDs** on page 103
- **Creating a Connectivity Map** on page 104
- **Creating the Collaboration Definitions (Java)** on page 105
- **Binding the eWay Components** on page 115
- **Creating an Environment** on page 116
- **Configuring the eWays** on page 117
- **Creating and Starting the Domain** on page 121
- **Building and Deploying the Project** on page 122
- **Running the Sample** on page 122

6.6.1 Creating a Project

The first step is to create a new Project in the Enterprise Designer.

1 Start the Enterprise Designer.

2 From the Project Explorer tree, right-click the Repository and select **New Project**. A new Project (**Project1**) appears on the Project Explorer tree.

3 Click twice on **Project1** and rename the Project (for this sample, **prjDB2Connect_JCD**).

6.6.2 Creating the OTDs

The sample Project requires three OTDs to interact with the DB2 Connect eWay. These OTDs include:

- DB2 Connect Database OTD
- Inbound DTD OTD
- Outbound DTD OTD

Steps required to create a DB2 Connect Database OTD:

1. Right-click your new Project in the Enterprise Designer's Project Explorer, and select **New** > **Object Type Definition**.

   The New Object Type Definition Wizard window appears.

2. Select the **DB2 Connect Database OTD Wizard** from the list of OTD Wizards and click **Next**.

3. Enter the connection information for the DB2 Connect database. Connection fields include:

   - Database:
   - Username
   - Password:

4. Click **Next**, and select the types of database object you want to include in the sample Project. For our example, select the following:

   - Tables/Views/Aliases
   - Prepared Statements

5. Click **Add** to select tables from the DB2 Connect database. The **Add Tables** window appears.

6. Search for or Type in the name of the database. In this example we use the **DB_EMPLOYEE** table. Click **Select** when the database appears in the Results selection frame. Click **OK** to close the Add Tables window

7. Click **Next the Add Prepared Statements Wizard appears.**

8. Click **Add**, the Add Prepared Statement window appears. Enter the following:

   - Prepared Statement Name: Select_ps
     - SQL Statement:

       ```
       select * from db_employee where emp_no > ? order by
       emp_no
       ```

   - Prepared Statement Name: Insert_ps
     - SQL Statement:

       ```
       INSERT intoDB_EMPLOYEE(EMP_NO,LAST_NAME,FIRST_NAME,RATE,
       LAST_UPDATE) values(?, ? , ?, ?, ?)
       ```

- Prepared Statement Name: Update_ps
  - SQL Statement:

```
UPDATE DB_EMPLOYEE SET RATE = 19 , LAST_UPDATE = '2008-
08-08'  where EMP_NO = ?
```

Note: *In this example, the SQL statement includes the* **?** *placeholder for input. This placeholder represents the Where Clause.*

9 Click the **OK** button to close the Prepared Statement window, and then click **Next** on the Prepared Statements Wizard window.

10 Enter an OTD name. In this example, use **otdDB2Connect**.

11 Click **Next** and review your settings, then click **Finish** to create the OTD.

Steps required to create inbound and outbound DTD OTDs include:

1 Right-click your new Project in the Enterprise Designer's Project Explorer, and select **New** > **Object Type Definition**.

The New Object Type Definition Wizard window appears.

2 Select **DTD** from the list of OTD Wizards and click **Next**.

3 Browse to and then select a DTD file. For this example, select one of the following DTD files from the sample Project, and then click **Next**.

- otdInputDTD.dtd
- otdOutputDTD.dtd

4 The file you select appears in the Select Document Elements window. Click **Next**.

5 Click **Finish** to complete the DTD based OTD. Repeat this process again to create the second DTD file.

6.6.3 Creating a Connectivity Map

The Connectivity Map provides a canvas for assembling and configuring a Project's components.

Steps required to create a new Connectivity Map:

1 From the Project Explorer tree, right-click the new **prjDB2Connect_JCD** Project and select **New > Connectivity Map** from the shortcut menu.

2 The New Connectivity Map appears and a node for the Connectivity Map is added under the Project, on the Project Explorer tree labeled **CMap1**.

Create four additional Connectivity Maps—**CMap2**, **CMap3**. **CMap4**, and **CMap5**— and rename them as follows:

- cmDelete
- cmPsInsert
- cmPsSelect

- cmTableSelect
- cmPsUpdate

The icons in the toolbar represent the available components used to populate the Connectivity Map canvas.

Populating the Connectivity Map

Add the Project components to the Connectivity Map by dragging the icons from the toolbar to the canvas.

Each Connectivity Map in the **prjDB2Connect_JCD** sample Project requires the following components:

- File External Application (2)
- DB2 Connect External Application
- Service

Any eWay added to the Connectivity Map is associated with an External System. To establish a connection to DB2 Connect, first select DB2 Connect as an External System to use in your Connectivity Map.

Steps required to select a DB2 Connect External System:

1. Click the **External Application** icon on the Connectivity Map toolbar.
2. Select the external systems necessary to create your Project (for this sample, **DB2 Connect** and **File**). Icons representing the selected external systems are added to the Connectivity Map toolbar.
3. Rename the following components and then save changes to the Repository:
   - File1 to FileClientIN
   - File2 to FileClientOUT
   - DB2 Connect1 to eaDB2 ConnectOUT
4. Rename each Connectivity Map Service to match the intended operation, as for example:
   - jcdDelete
   - jcdPsInsert
   - jcdPsSelect
   - jcdTableSelect
   - jcdPsUpdate

6.6.4 Creating the Collaboration Definitions (Java)

The next step is to create Collaborations using the **Collaboration Definition Wizard (Java)**. Since the sample Project includes five database operations, you must create five separate Collaboration Definitions (Java), or JCDs. Once you create the Collaboration

Definitions, you can write the Business Rules of the Collaborations using the Collaboration Editor.

JCDs required for the **prjDB2Connect_JCD** sample include:

- jcdDelete
- jcdPsInsert
- jcdPsSelect
- jcdTableSelect
- jcdPsUpdate

jcdDelete Collaboration

Steps required to create the jcdDelete Collaboration:

1. From the Project Explorer, right-click the sample Project and select **New** > **Collaboration Definition (Java)** from the shortcut menu. The **Collaboration Definition Wizard (Java)** appears.
2. Enter a Collaboration Definition name (for this sample **jcdDelete**) and click **Next**.
3. For Step 2 of the wizard, from the Web Services Interfaces selection window, double-click **Sun SeeBeyond** > **eWays** > **File** > **FileClient** > **receive**. The File Name field now displays **receive.** Click **Next**.
4. For Step 3 of the wizard, from the Select OTDs selection window, double-click **prjDB2Connect_JCD** > **otdALL** > **otdDB2Connect**. The **otdDB2Connect** OTD is added to the Selected OTDs field.
5. Click the **Up One Level** button to return to the Repository. Double-click **Sun SeeBeyond** > **eWays** > **File** > **FileClient**. The **Selected OTDs** field now lists the **FileClient** OTD.
6. Click **Finish**. The Collaboration Editor with the new **jcdDelete** Collaboration appears in the right pane of the Enterprise Designer.

jcdPsInsert Collaboration

Steps required to create the jcdInsert Collaboration:

1. From the Project Explorer, right-click the sample Project and select **New** > **Collaboration Definition (Java)** from the shortcut menu. The **Collaboration Definition Wizard (Java)** appears.
2. Enter a Collaboration Definition name (for this sample **jcdPsInsert**) and click **Next**.
3. For Step 2 of the wizard, from the Web Services Interfaces selection window, double-click **Sun SeeBeyond** > **eWays** > **File** > **FileClient** > **receive**. The File Name field now displays **receive.** Click **Next**.
4. For Step 3 of the wizard, from the Select OTDs selection window, double-click **prjDB2 Connect_JCD** > **otdALL** > **otdDB2 Connect**. The **otdDB2Connect** OTD is added to the Selected OTDs field.

5 In the same window, double-click **otdInputDTD_DBemployees**. The **otdInputDTD_DBemployees** OTD is added to the Selected OTDs field.

Note: *The otdOutputDTD_DBemployees OTD is created from the otdInputDTD.dtd that is included in the Sample Project.*

6 Click the **Up One Level** button to return to the Repository. Double-click **Sun SeeBeyond** > **eWays** > **File** > **FileClient**. The **Selected OTDs** field now lists the **FileClient** OTD.

7 Click **Finish**. The Collaboration Editor with the new **jcdPsInsert** Collaboration appears in the right pane of the Enterprise Designer.

jcdPsSelect Collaboration

Steps required to create the jcdPsSelect Collaboration:

1 From the Project Explorer, right-click the sample Project and select **New** > **Collaboration Definition (Java)** from the shortcut menu. The **Collaboration Definition Wizard (Java)** appears.

2 Enter a Collaboration Definition name (for this sample **jcdPsSelect**) and click **Next**.

3 For Step 2 of the wizard, from the Web Services Interfaces selection window, double-click **Sun SeeBeyond** > **eWays** > **File** > **FileClient** > **receive**. The File Name field now displays **receive.** Click **Next**.

4 For Step 3 of the wizard, from the Select OTDs selection window, double-click **prjDB2 Connect_JCD** > **otdALL** > **otdDB2 Connect**. The **otdDB2Connect** OTD is added to the Selected OTDs field.

5 In the same window, double-click **otdOutputDTD_DBemployee**. The **otdOutputDTD_DBemployee** OTD is added to the Selected OTDs field.

Note that the otdOutputDTD_DBemployee OTD is created from the otdOutputDTD.dtd that is included in the Sample Project.

6 Click the **Up One Level** button to return to the Repository. Double-click **Sun SeeBeyond** > **eWays** > **File** > **FileClient**. The **Selected OTDs** field now lists the **FileClient** OTD.

7 Click **Finish**. The Collaboration Editor with the new **jcdPsSelect** Collaboration appears in the right pane of the Enterprise Designer.

jcdTableSelect Collaboration

Steps required to create the jcdTableSelect Collaboration:

1 From the Project Explorer, right-click the sample Project and select **New** > **Collaboration Definition (Java)** from the shortcut menu. The **Collaboration Definition Wizard (Java)** appears.

2 Enter a Collaboration Definition name (for this sample **jcdTableSelect**) and click **Next**.

3 For Step 2 or the wizard, from the Web Services Interfaces selection window, double-click **Sun SeeBeyond** > **eWays** > **File** > **FileClient** > **receive**. The File Name field now displays **receive.** Click **Next**.

4 For Step 3 of the wizard, from the Select OTDs selection window, double-click **prjDB2Connect_JCD** > **otdALL** > **otdDB2Connect**. The **otdDB2Connect** OTD is added to the Selected OTDs field.

5 In the same window, double-click **otdOutputDTD_DBemployee**. The **otdOutputDTD_DBemployee** OTD is added to the Selected OTDs field.

Note: *The otdOutputDTD_DBemployee OTD is created from the otdOutputDTD.dtd that is included in the Sample Project.*

6 Click the **Up One Level** button to return to the Repository. Double-click **Sun SeeBeyond** > **eWays** > **File** > **FileClient**. The **Selected OTDs** field now lists the **FileClient** OTD.

7 Click **Finish**. The Collaboration Editor with the new **jcdTableSelect** Collaboration appears in the right pane of the Enterprise Designer.

jcdPsUpdate Collaboration

Steps required to create the jcdUpdate Collaboration:

1 From the Project Explorer, right-click the sample Project and select **New** > **Collaboration Definition (Java)** from the shortcut menu. The **Collaboration Definition Wizard (Java)** appears.

2 Enter a Collaboration Definition name (for this sample **jcdPsUpdate**) and click **Next**.

3 For Step 2 of the wizard, from the Web Services Interfaces selection window, double-click **Sun SeeBeyond** > **eWays** > **File** > **FileClient** > **receive**. The File Name field now displays **receive.** Click **Next**.

4 For Step 3 of the wizard, from the Select OTDs selection window, double-click **prjDB2Connect_JCD > otdALL > otdDB2Connect**. The **otdDB2Connect** OTD is added to the Selected OTDs field.

5 Click the **Up One Level** button to return to the Repository. Double-click **Sun SeeBeyond** > **eWays** > **File** > **FileClient**. The **Selected OTDs** field now lists the **FileClient** OTD.

6 Click **Finish**. The Collaboration Editor with the new **jcdPsUpdate** Collaboration appears in the right pane of the Enterprise Designer.

6.6.5 Create the Collaboration Business Rules

The next step in the sample is to create the Business Rules of the Collaboration using the Collaboration Editor.

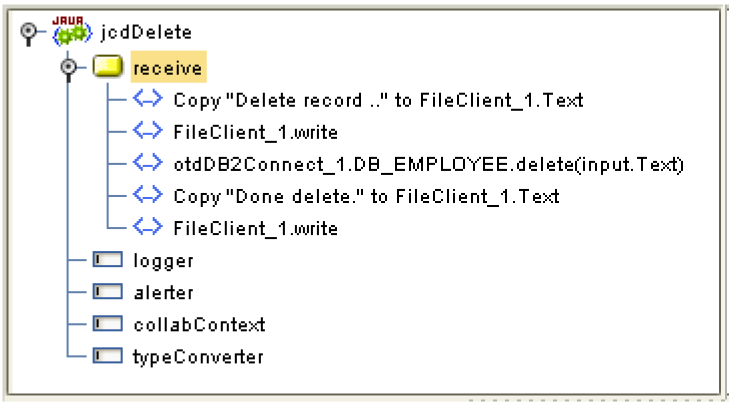
Creating the jcdDelete Business Rules

The **jcdDelete** Collaboration implements the Input Web Service Operation to read the **TriggerDelete.in** file and then delete the record **emp_no = 500**. The Collaboration also writes a message to **JCD_Delete_output0.dat** to confirm a deleted record.

Note: *The where clause in the business rule reads the trigger value as a placeholder for input. This permits you to modify the query to select a specific record. Also note that all records are selected from the database when the TriggerDelete.in file is empty.*

The **jcdDelete** Collaboration contains the Business Rules displayed in Figure 59.

Figure 59 jcdDelete Business Rules



```
jcdDelete
  receive
    Copy "Delete record .." to FileClient_1.Text
    FileClient_1.write
    otdDB2Connect_1.DB_EMPLOYEE.delete(input.Text)
    Copy "Done delete." to FileClient_1.Text
    FileClient_1.write
  logger
  alerter
  collabContext
  typeConverter
```

Sample code from the jcdPsInsert Includes:

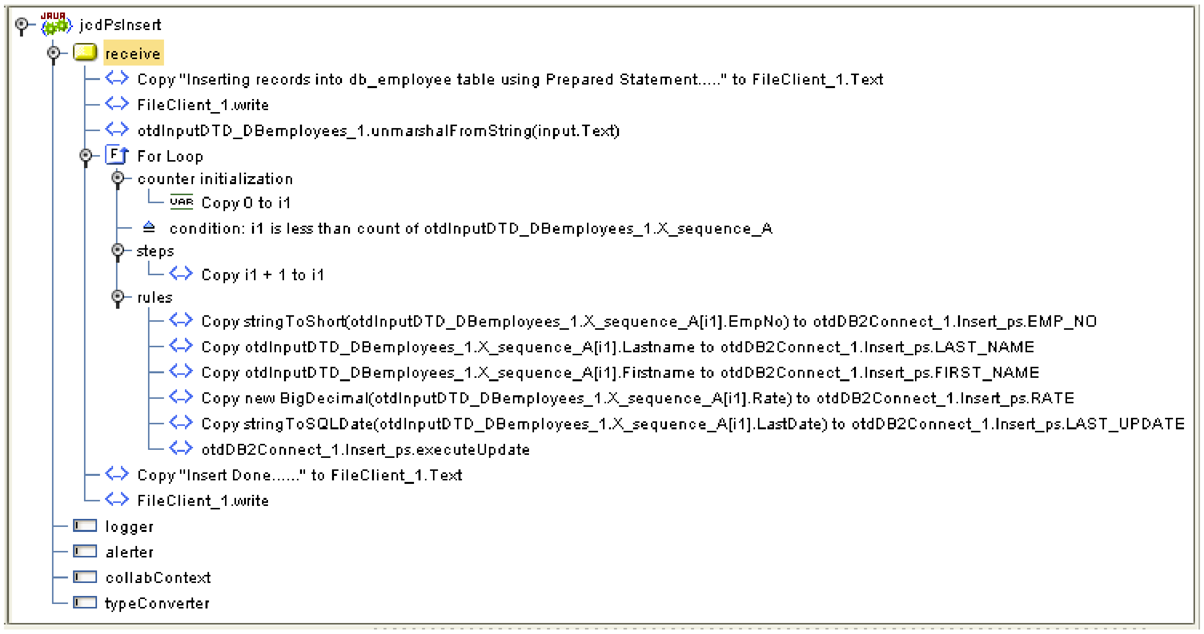
```
package prjDB2Connect_JCDjcdALL;


public class jcdDelete
{

    public com.stc.codegen.logger.Logger logger;

    public com.stc.codegen.alerter.Alerter alerter;

    public com.stc.codegen.util.CollaborationContext collabContext;

    public com.stc.codegen.util.TypeConverter typeConverter;

    public void receive(
com.stc.connector.appconn.file.FileTextMessage input,
otdDB2Connect.OtdDB2ConnectOTD otdDB2Connect_1,
com.stc.connector.appconn.file.FileApplication FileClient_1 )
        throws Throwable
    {
        FileClient_1.setText( "Delete record .." );
        FileClient_1.write();
        otdDB2Connect_1.getDB_EMPLOYEE().delete( input.getText() );
        FileClient_1.setText( "Done delete." );
        FileClient_1.write();
    }

}
```

Creating the jcdPsInsert Business Rules

The **jcdPsInsert** Collaboration implements the Input Web Service Operation to read the **TriggerJCEPsInsert.in**. file. It then unmarshals data from the input data into the **otdInputDTD_DBEmployees** OTD, calls the **otdDB2 Connect** OTD, and inserts records into the database via a For Loop. The Collaboration also writes a message to **JCD_Insert_output0.dat** to confirm an inserted record.

The **jcdInsert** Collaboration contains the Business Rules displayed in Figure 60.

Figure 60 jcdInsert Business Rules



```
jcdPsInsert
  receive
    Copy "Inserting records into db_employee table using Prepared Statement....." to FileClient_1.Text
    FileClient_1.write
    otdInputDTD_DBemployees_1.unmarshalFromString(input.Text)
    For Loop
      counter initialization
        Copy 0 to i1
      condition: i1 is less than count of otdInputDTD_DBemployees_1.X_sequence_A
      steps
        Copy i1 + 1 to i1
      rules
        Copy stringToShort(otdInputDTD_DBemployees_1.X_sequence_A[i1].EmpNo) to otdDB2Connect_1.Insert_ps.EMP_NO
        Copy otdInputDTD_DBemployees_1.X_sequence_A[i1].Lastname to otdDB2Connect_1.Insert_ps.LAST_NAME
        Copy otdInputDTD_DBemployees_1.X_sequence_A[i1].Firstname to otdDB2Connect_1.Insert_ps.FIRST_NAME
        Copy new BigDecimal(otdInputDTD_DBemployees_1.X_sequence_A[i1].Rate) to otdDB2Connect_1.Insert_ps.RATE
        Copy stringToSQLDate(otdInputDTD_DBemployees_1.X_sequence_A[i1].LastDate) to otdDB2Connect_1.Insert_ps.LAST_UPDATE
        otdDB2Connect_1.Insert_ps.executeUpdate
    Copy "Insert Done......" to FileClient_1.Text
    FileClient_1.write
  logger
  alerter
  collabContext
  typeConverter
```

Sample code from the jcdPsInsert Includes:

```
package prjDB2Connect_JCDjcdALL;


public class jcdPsInsert
{

    public com.stc.codegen.logger.Logger logger;

    public com.stc.codegen.alerter.Alerter alerter;

    public com.stc.codegen.util.CollaborationContext collabContext;

    public com.stc.codegen.util.TypeConverter typeConverter;

    public void receive(
com.stc.connector.appconn.file.FileTextMessage input,
otdDB2Connect.OtdDB2ConnectOTD otdDB2Connect_1,
com.stc.connector.appconn.file.FileApplication FileClient_1,
dtd.otdInputDTD_654315252.DBemployees otdInputDTD_DBemployees_1,
dtd.otdOutputDTD1750519912.DBemployee otdOutputDTD_DBemployee_1 )
        throws Throwable
    {
        FileClient_1.setText( "Inserting records into db_employee
table using Prepared Statement....." );
```

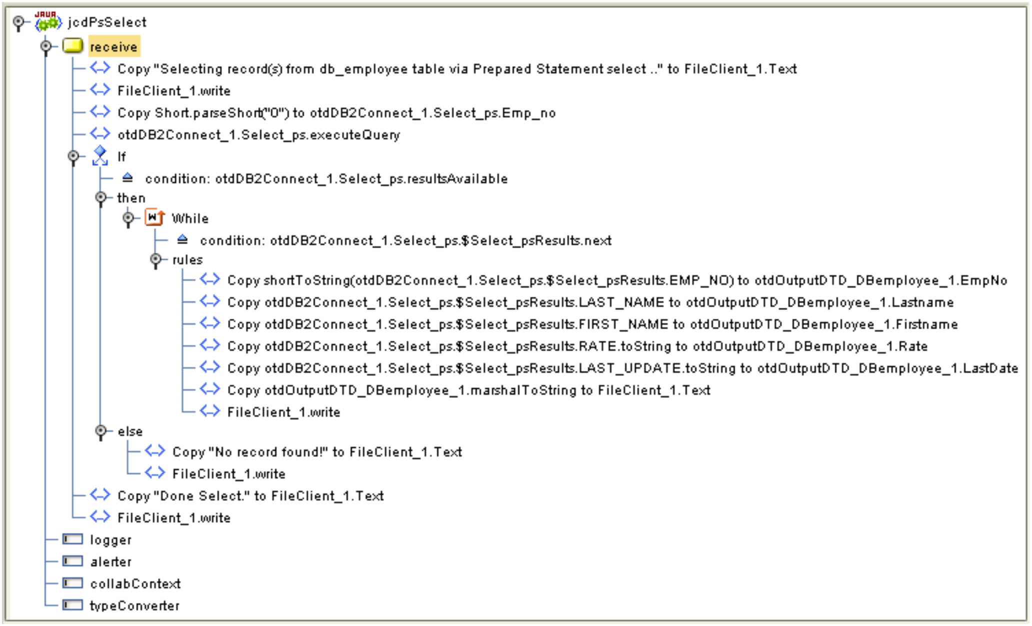
```
        FileClient_1.write();
        otdInputDTD_DBemployees_1.unmarshalFromString(
input.getText() );
        for (int i1 = 0; i1 <
otdInputDTD_DBemployees_1.countX_sequence_A(); i1 += 1) {
            otdDB2Connect_1.getInsert_ps().setEMP_NO(
typeConverter.stringToShort(
otdInputDTD_DBemployees_1.getX_sequence_A( i1 ).getEmpNo(), "#",
false, 0 ) );
            otdDB2Connect_1.getInsert_ps().setLAST_NAME(
otdInputDTD_DBemployees_1.getX_sequence_A( i1 ).getLastname() );
            otdDB2Connect_1.getInsert_ps().setFIRST_NAME(
otdInputDTD_DBemployees_1.getX_sequence_A( i1 ).getFirstname() );
            otdDB2Connect_1.getInsert_ps().setRATE( new
java.math.BigDecimal( otdInputDTD_DBemployees_1.getX_sequence_A( i1
).getRate() ) );
            otdDB2Connect_1.getInsert_ps().setLAST_UPDATE(
typeConverter.stringToSQLDate(
otdInputDTD_DBemployees_1.getX_sequence_A( i1 ).getLastDate(), "yyyy-
MM-dd hh:mm:ss", false, "" ) );
            otdDB2Connect_1.getInsert_ps().executeUpdate();
        }
        FileClient_1.setText( "Insert Done......" );
        FileClient_1.write();
    }

}
```

Creating the jcdPsSelect Business Rules

The **jcdPsSelect** Collaboration implements the Input Web Service Operation to read the **TriggerPsSelect.in** file. It then copies the database resultset (as noted in the prepared statement query) into the **otdInputDTD_DBEmployee** OTD and selects all available records from the database. The Collaboration also writes a message to **JCD_PsSelect_output0.dat** to confirm when records are selected, or when no records are available.

The **jcdPsSelect** Collaboration contains the Business Rules displayed in Figure 61.

Figure 61 jcdPsSelect



```
jcdPsSelect
  receive
    Copy "Selecting record(s) from db_employee table via Prepared Statement select .." to FileClient_1.Text
    FileClient_1.write
    Copy Short.parseShort("0") to otdDB2Connect_1.Select_ps.Emp_no
    otdDB2Connect_1.Select_ps.executeQuery
    If
      condition: otdDB2Connect_1.Select_ps.resultsAvailable
      then
        While
          condition: otdDB2Connect_1.Select_ps.$Select_psResults.next
          rules
            Copy shortToString(otdDB2Connect_1.Select_ps.$Select_psResults.EMP_NO) to otdOutputDTD_DBemployee_1.EmpNo
            Copy otdDB2Connect_1.Select_ps.$Select_psResults.LAST_NAME to otdOutputDTD_DBemployee_1.Lastname
            Copy otdDB2Connect_1.Select_ps.$Select_psResults.FIRST_NAME to otdOutputDTD_DBemployee_1.Firstname
            Copy otdDB2Connect_1.Select_ps.$Select_psResults.RATE.toString to otdOutputDTD_DBemployee_1.Rate
            Copy otdDB2Connect_1.Select_ps.$Select_psResults.LAST_UPDATE.toString to otdOutputDTD_DBemployee_1.LastDate
            Copy otdOutputDTD_DBemployee_1.marshalToString to FileClient_1.Text
            FileClient_1.write
      else
        Copy "No record found!" to FileClient_1.Text
        FileClient_1.write
    Copy "Done Select." to FileClient_1.Text
    FileClient_1.write
  logger
  alerter
  collabContext
  typeConverter
```

Sample code from the jcdPsSelect Includes:

```
package prjDB2Connect_JCDjcdALL;


public class jcdPsSelect
{

    public com.stc.codegen.logger.Logger logger;

    public com.stc.codegen.alerter.Alerter alerter;

    public com.stc.codegen.util.CollaborationContext collabContext;

    public com.stc.codegen.util.TypeConverter typeConverter;

    public void receive(
com.stc.connector.appconn.file.FileTextMessage input,
com.stc.connector.appconn.file.FileApplication FileClient_1,
otdDB2Connect.OtdDB2ConnectOTD otdDB2Connect_1,
dtd.otdInputDTD_654315252.DBemployees otdInputDTD_DBemployees_1,
dtd.otdOutputDTD1750519912.DBemployee otdOutputDTD_DBemployee_1 )
        throws Throwable
    {
        FileClient_1.setText( "Selecting record(s) from db_employee
table via Prepared Statement select .." );
        FileClient_1.write();
        otdDB2Connect_1.getSelect_ps().setEmp_no( Short.parseShort(
"0" ) );
        otdDB2Connect_1.getSelect_ps().executeQuery();
        if (otdDB2Connect_1.getSelect_ps().resultsAvailable()) {
            while
(otdDB2Connect_1.getSelect_ps().get$Select_psResults().next()) {
                otdOutputDTD_DBemployee_1.setEmpNo(
typeConverter.shortToString(
otdDB2Connect_1.getSelect_ps().get$Select_psResults().getEMP_NO(),
"#", false, "" ) );
```

```
                otdOutputDTD_DBemployee_1.setLastname(
otdDB2Connect_1.getSelect_ps().get$Select_psResults().getLAST_NAME()
);
                otdOutputDTD_DBemployee_1.setFirstname(
otdDB2Connect_1.getSelect_ps().get$Select_psResults().getFIRST_NAME()
);
                otdOutputDTD_DBemployee_1.setRate(
otdDB2Connect_1.getSelect_ps().get$Select_psResults().getRATE().toStr
ing() );
                otdOutputDTD_DBemployee_1.setLastDate(
otdDB2Connect_1.getSelect_ps().get$Select_psResults().getLAST_UPDATE(
).toString() );
                FileClient_1.setText(
otdOutputDTD_DBemployee_1.marshalToString() );
                FileClient_1.write();
            }
        } else {
            FileClient_1.setText( "No record found!" );
            FileClient_1.write();
        }
        FileClient_1.setText( "Done Select." );
        FileClient_1.write();
    }

}
```

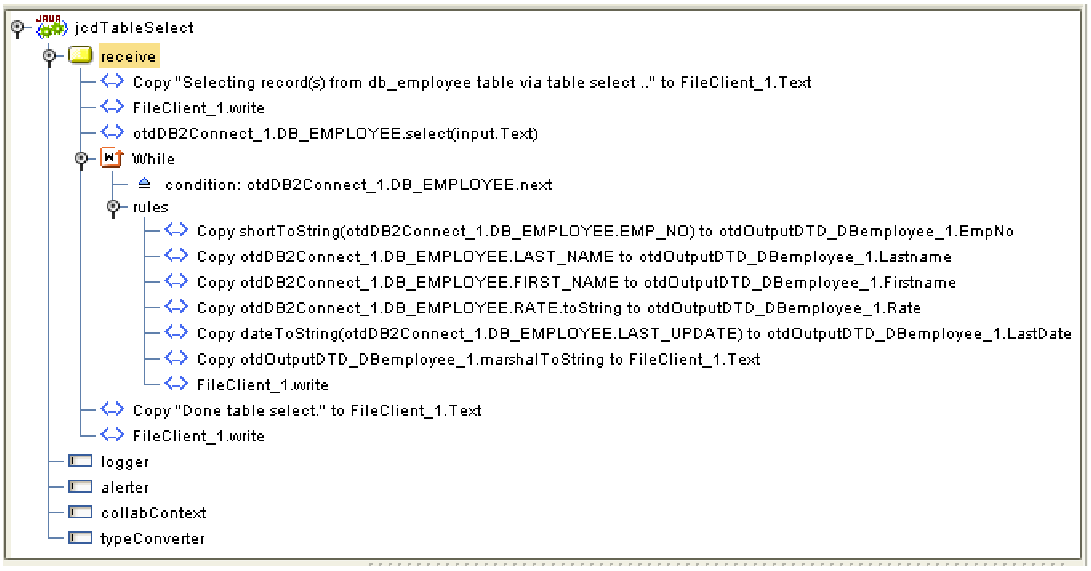
Creating the jcdTableSelect Business Rules

The **jcdTableSelect** Collaboration implements the Input Web Service Operation to read the **TriggerTableSelect.in** file. It then copies the database resultset into the **otdInputDTD_DBEmployee** OTD and selects all available records from the database that meet the criteria **emp_no = 100**. The Collaboration also writes a message to **JCD_TableSelect_output0.dat** to confirm when records are selected, or when no records are available.

Note: *The where clause in the business rule reads the trigger value as a placeholder for input. This permits you to modify the query to select a specific record. Also note that all records are selected from the database when the TriggerTableSelect.in file is empty.*

The **jcdTableSelect** Collaboration contains the Business Rules displayed in Figure 62.

Figure 62 jcdTableSelect



```
jcdTableSelect
  receive
    Copy "Selecting record(s) from db_employee table via table select .." to FileClient_1.Text
    FileClient_1.write
    otdDB2Connect_1.DB_EMPLOYEE.select(input.Text)
    While
      condition: otdDB2Connect_1.DB_EMPLOYEE.next
      rules
        Copy shortToString(otdDB2Connect_1.DB_EMPLOYEE.EMP_NO) to otdOutputDTD_DBemployee_1.EmpNo
        Copy otdDB2Connect_1.DB_EMPLOYEE.LAST_NAME to otdOutputDTD_DBemployee_1.Lastname
        Copy otdDB2Connect_1.DB_EMPLOYEE.FIRST_NAME to otdOutputDTD_DBemployee_1.Firstname
        Copy otdDB2Connect_1.DB_EMPLOYEE.RATE.toString to otdOutputDTD_DBemployee_1.Rate
        Copy dateToString(otdDB2Connect_1.DB_EMPLOYEE.LAST_UPDATE) to otdOutputDTD_DBemployee_1.LastDate
        Copy otdOutputDTD_DBemployee_1.marshalToString to FileClient_1.Text
        FileClient_1.write
    Copy "Done table select." to FileClient_1.Text
    FileClient_1.write
  logger
  alerter
  collabContext
  typeConverter
```

Sample code from the jcdTableSelect Includes:

```
package prjDB2Connect_JCDjcdALL;


public class jcdTableSelect
{

    public com.stc.codegen.logger.Logger logger;

    public com.stc.codegen.alerter.Alerter alerter;

    public com.stc.codegen.util.CollaborationContext collabContext;

    public com.stc.codegen.util.TypeConverter typeConverter;

    public void receive(
com.stc.connector.appconn.file.FileTextMessage input,
otdDB2Connect.OtdDB2ConnectOTD otdDB2Connect_1,
dtd.otdOutputDTD1750519912.DBemployee otdOutputDTD_DBemployee_1,
com.stc.connector.appconn.file.FileApplication FileClient_1 )
        throws Throwable
    {
        FileClient_1.setText( "Selecting record(s) from db_employee
table via table select .." );
        FileClient_1.write();
        otdDB2Connect_1.getDB_EMPLOYEE().select( input.getText() );
        while (otdDB2Connect_1.getDB_EMPLOYEE().next()) {
            otdOutputDTD_DBemployee_1.setEmpNo(
typeConverter.shortToString(
otdDB2Connect_1.getDB_EMPLOYEE().getEMP_NO(), "#", false, "" ) );
            otdOutputDTD_DBemployee_1.setLastname(
otdDB2Connect_1.getDB_EMPLOYEE().getLAST_NAME() );
            otdOutputDTD_DBemployee_1.setFirstname(
otdDB2Connect_1.getDB_EMPLOYEE().getFIRST_NAME() );
            otdOutputDTD_DBemployee_1.setRate(
otdDB2Connect_1.getDB_EMPLOYEE().getRATE().toString() );
            otdOutputDTD_DBemployee_1.setLastDate(
typeConverter.dateToString(
otdDB2Connect_1.getDB_EMPLOYEE().getLAST_UPDATE(), "yyyy-MM-dd
hh:mm:ss", false, "" ) );
```

```
                FileClient_1.setText(
otdOutputDTD_DBemployee_1.marshalToString() );
                FileClient_1.write();
           }
           FileClient_1.setText( "Done table select." );
           FileClient_1.write();
      }

}
```

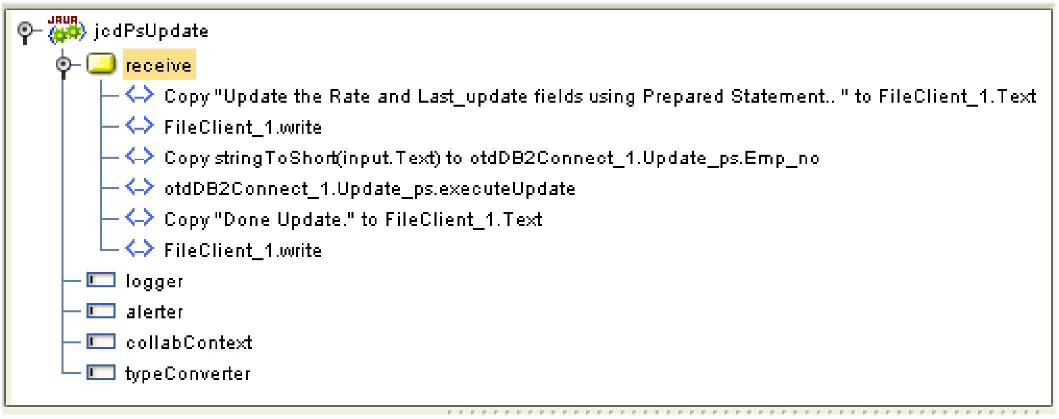
Creating the jcdPsUpdate Business Rules

The **jcdPsUpdate** Collaboration implements the Input Web Service Operation to read the **TriggerPsUpdate.in**. file and then update the record **emp_no = 300**. The Collaboration also writes a message to **JCE_PsUpdate_output0.dat** to confirm an updated record.

Note: *The where clause in the business rule reads the trigger value as a placeholder for input. This permits you to modify the query to select a specific record. Also note that all records are selected from the database when the TriggerUpdate.in file is empty.*

The **jcdUpdate** Collaboration contains the Business Rules displayed in Figure 63.

Figure 63 jcdPsUpdate


```
jcdPsUpdate
  receive
    Copy "Update the Rate and Last_update fields using Prepared Statement.. " to FileClient_1.Text
    FileClient_1.write
    Copy stringToShort(input.Text) to otdDB2Connect_1.Update_ps.Emp_no
    otdDB2Connect_1.Update_ps.executeUpdate
    Copy "Done Update." to FileClient_1.Text
    FileClient_1.write
  logger
  alerter
  collabContext
  typeConverter
```

Sample code from the jcdPsUpdate Includes:

```
package prjDB2Connect_JCDjcdALL;


public class jcdPsUpdate
{

    public com.stc.codegen.logger.Logger logger;

    public com.stc.codegen.alerter.Alerter alerter;

    public com.stc.codegen.util.CollaborationContext
collabContext;

    public com.stc.codegen.util.TypeConverter typeConverter;

    public void receive(
com.stc.connector.appconn.file.FileTextMessage input,
otdDB2Connect.OtdDB2ConnectOTD otdDB2Connect_1,
com.stc.connector.appconn.file.FileApplication FileClient_1 )
        throws Throwable
    {
        FileClient_1.setText( "Update the Rate and Last_update
fields using Prepared Statement.. " );
        FileClient_1.write();
        otdDB2Connect_1.getUpdate_ps().setEmp_no(
typeConverter.stringToShort( input.getText(), "#", false, 0 ) );
        otdDB2Connect_1.getUpdate_ps().executeUpdate();
        FileClient_1.setText( "Done Update." );
        FileClient_1.write();
    }

}
```

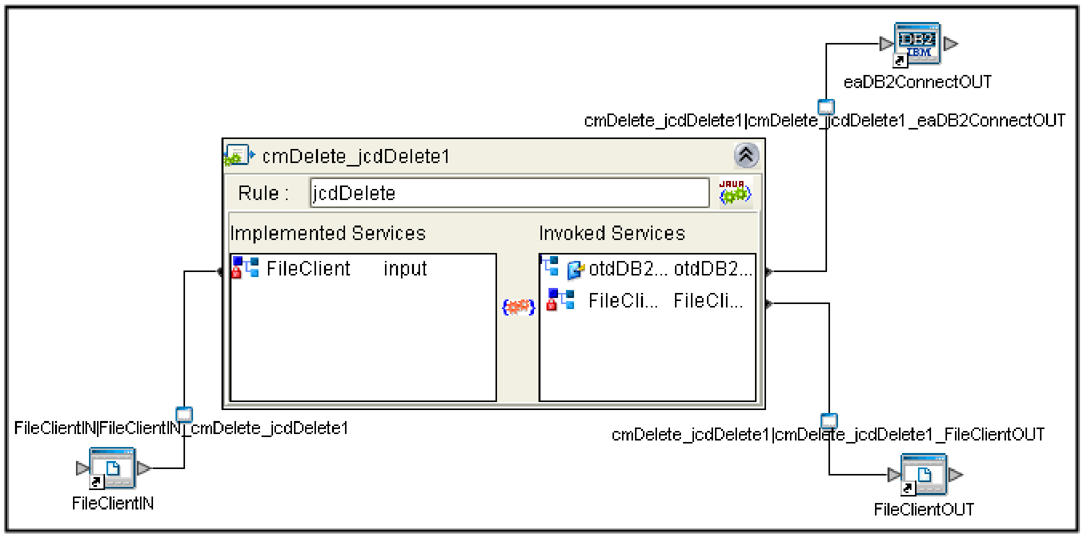
6.6.6 Binding the eWay Components

The final step in creating a Connectivity Map is binding the eWay components together.

Steps required to bind eWay components together:

1. Double-click a Connectivity Map—in this example **cmDelete**—in the Project Explorer tree. The **cmDelete** Connectivity Map appears in the Enterprise Designers canvas.
2. Drag and drop the **jcdDelete** Collaboration from the Project Explorer to the **jcdDelete** Service. The Service icon “gears” change from red to green.
3. Double-click the **jcdDelete** Service. The **jcdDelete** Binding dialog box appears.
4. Map the input **FileClient** (under Implemented Services) to the **FileClientIN** (File) External Application. To do this, click on **FileSender** in the **bpDelete** Binding dialog box, and drag the cursor to the **FileClientIN** External Application in the Connectivity Map. A link is now visible between **FileClientIN** and **bpDelete**.
5. From the **bpDelete** Binding dialog box, map **otdDB2Connect_1** (under Invoked Services) to the **eaDB2ConnectOUT** External Application.
6. From the **bpDelete** Binding dialog box, map **FileClient_1** to the **FileClientOUT** External Application, as seen in Figure 55.

Figure 64 Connectivity Map - Associating (Binding) the Project’s Components



7 Minimize the **jcdDelete** Binding dialog box by clicking the chevrons in the upper-right corner.

8 Save your current changes to the Repository, and then repeat this process for each of the other Connectivity Maps.

6.6.7 Creating an Environment

Environments include the external systems, Logical Hosts, integration servers and message servers used by a Project and contain the configuration information for these components. Environments are created using the Enterprise Designer’s Environment Editor.

Steps required to create an Environment:

1 From the Enterprise Designer’s Enterprise Explorer, click the **Environment Explorer** tab.

2 Right-click the Repository and select **New Environment**. A new Environment is added to the Environment Explorer tree.

3 Rename the new Environment to **envDB2Connect**.

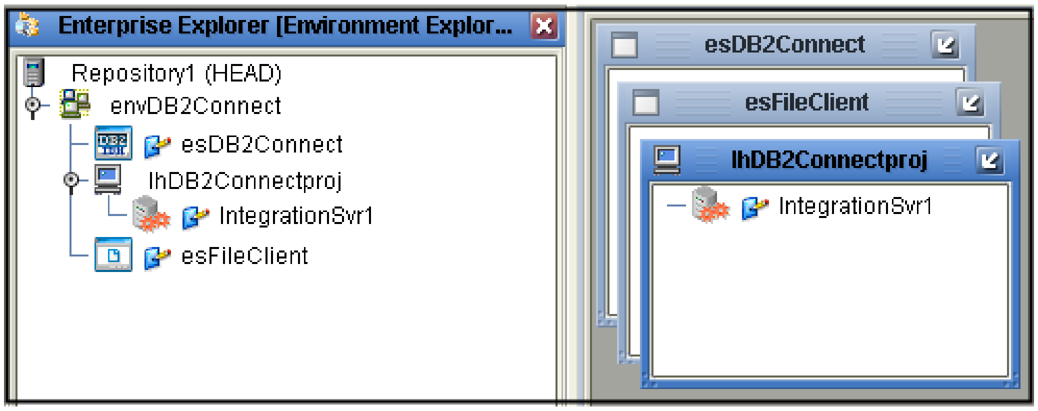
4 Right-click **envDB2ConnectProj** and select **New DB2Connect External System**. Name the External System **esDB2Connect.** Click **OK**. **esDB2Connect** is added to the Environment Editor.

5 Right-click **envDB2ConnectProj** and select **New File External System**. Name the External System **esFileClient**. Click **OK**. **esFileClient** is added to the Environment Editor.

6 Right-click **envDB2ConnectProj** and select **New Logical Host**. Name the Logical Host **lhDB2Connectproj**.The Logical Host box is added to both the Environment and the Environment Editor tree.

7 Right-click **lhDB2Connectproj** and select **New Integration Server**. A new Integration Server (**IntegrationSvr1**) is added to the Environment Explorer tree under **lhDB2Connectproj** (see Figure 56).

Figure 65 Environment Editor - envDB2 ConnectProj



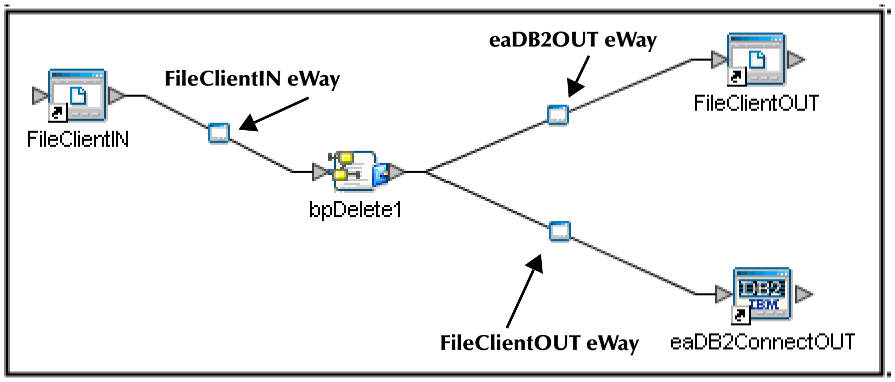
8 Save your current changes to the Repository.

6.6.8 Configuring the eWays

eWays facilitate communication and movement of data between the external applications and the eGate system. Each Connectivity Map in the **prjDB2Connect_JCD** sample Project uses three eWays that are represented as nodes between the External Applications and the Business Process, as seen in Figure 66.

You must configure eWay properties in both the Connectivity Map and the Environment Explorer.

Figure 66 eWays in the cmDelete Connectivity Map



Configuring the eWay Properties

Steps required to configure the eWay properties:

1. Double-click the **FileClientIN eWay on each of the Connectivity Maps and** modify the properties for your system, as seen in Table 30. Click **OK** to close the Properties Editor.

Table 30 FileClientIN eWay Property Settings

Connectivity Map	Property Name	Required Value
cmDelete	Input file name	TriggerDelete.in
cmPsInsert	Input file name	TriggerJCEPsInsert.in
cmPsSelect	Input file name	TriggerPsSelect.in
cmTableSelect	Input file name	TriggerTableSelect.in
cmPsUpdate	Input file name	TriggerPsUpdate.in

2. Double-click the **FileClientIN eWay on each of the Connectivity Maps and** modify the properties for your system, as seen in Table 31. Click **OK** to close the Properties Editor.

Table 31 FileClientOUT eWay Property Settings

Connectivity Map	Property Name	Required Value
cmDelete	Input file name	JCE_Delete_output%d.dat
cmPsInsert	Input file name	JCE_PsInsert_output%d.dat
cmPsSelect	Input file name	JCE_PsSelect_output%d.dat
cmTableSelect	Input file name	JCE_TableSelect_output%d.dat.in
cmPsUpdate	Input file name	JCE_PsUpdate_output%d.dat

Configuring the Environment Explorer Properties

Steps required to configure the Environment Explorer properties:

1. From the **Environment Explorer** tree, right-click the File External System (**esDB2 Connect** in this sample), and select **Properties**. The Properties Editor opens to the DB2 Connect eWay Environment configuration.

2. Modify the DB2 Connect eWay Environment configuration properties for your system, as seen in Table 32, and click **OK**.

Table 32 DB2 Connect eWay Environment Properties

Section	Property Name	Required Value
Configuration > Inbound DB2 Connect eWay > JDBC Connector settings	ServerName	Enter the name of the database server being used.
	DatabaseName	Enter the name of the particular database that is being used on the server.
	User	Enter the user account name for the database.
	Password	Enter the user account password for the database.

3 From the **Environment Explorer** tree, right-click the File External System (**esFileClient** in this sample), and select **Properties**. The Properties Editor opens to the DB2 Connect eWay Environment configuration.

4 Modify the File eWay Environment configuration properties for your system, as seen in Table 33, and click **OK**.

Table 33 File eWay Environment Properties

Section	Property Name	Required Value
Configuration > Inbound File eWay > Parameter Settings	Directory	Enter the directory that contains the input files (trigger files included in the sample Project). Trigger files include: ▪ TriggerDelete.in.~in ▪ TriggerJCEPsInsert.in.~in ▪ TriggerPsSelect.in.~in ▪ TriggerTableSelect.in.~in ▪ TriggerPsUpdate.in.~in
Configuration > Outbound File eWay > Parameter Settings	Directory	Enter the directory where output files are written. In this sample Project, the output files include: ▪ JCE_Delete_output0.dat ▪ JCE_PsInsert_output0.dat ▪ JCE_PsSelect_output0.dat ▪ JCE_TableSelect_output0.dat ▪ JCE_PsUpdate_output0.dat

6.6.9 Creating the Deployment Profile

A Deployment Profile is used to assign services and message destinations to the integration server and message server. Deployment profiles are created using the Deployment Editor.

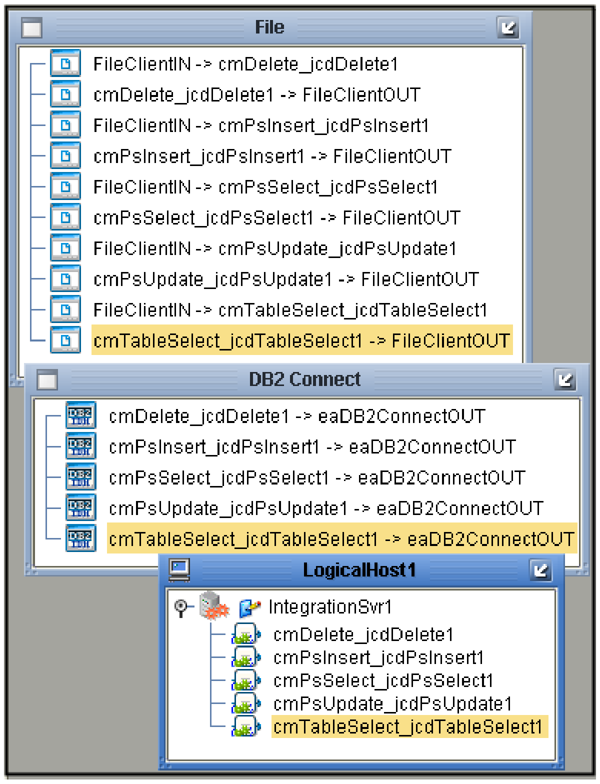
Steps required to create the Deployment Profile:

1 From the Enterprise Explorer's Project Explorer, right-click the **prjDB2Connect_JCD** Project and select **New** > **Deployment Profile**.

2 Enter a name for the Deployment Profile (for this sample **dpDB2Connect_JCD**). Select **envDB2ConnectProj** as the Environment and click **OK**.

3 From the Deployment Editor toolbar, click the **Automap** icon. The Project's components are automatically mapped to their system windows, as seen in Figure 58.

Figure 67 Deployment Profile



File

FileClientIN -> cmDelete_jcdDelete1
cmDelete_jcdDelete1 -> FileClientOUT
FileClientIN -> cmPsInsert_jcdPsInsert1
cmPsInsert_jcdPsInsert1 -> FileClientOUT
FileClientIN -> cmPsSelect_jcdPsSelect1
cmPsSelect_jcdPsSelect1 -> FileClientOUT
FileClientIN -> cmPsUpdate_jcdPsUpdate1
cmPsUpdate_jcdPsUpdate1 -> FileClientOUT
FileClientIN -> cmTableSelect_jcdTableSelect1
cmTableSelect_jcdTableSelect1 -> FileClientOUT

DB2 Connect

cmDelete_jcdDelete1 -> eaDB2ConnectOUT
cmPsInsert_jcdPsInsert1 -> eaDB2ConnectOUT
cmPsSelect_jcdPsSelect1 -> eaDB2ConnectOUT
cmPsUpdate_jcdPsUpdate1 -> eaDB2ConnectOUT
cmTableSelect_jcdTableSelect1 -> eaDB2ConnectOUT

LogicalHost1

IntegrationSvr1
cmDelete_jcdDelete1
cmPsInsert_jcdPsInsert1
cmPsSelect_jcdPsSelect1
cmPsUpdate_jcdPsUpdate1
cmTableSelect_jcdTableSelect1

6.6.10 Creating and Starting the Domain

To build and deploy your Project, you must first create a domain. A domain is an instance of a Logical Host. After the domain is created, the Project is built and then deployed.

Steps required to create and start the domain:

1 Navigate to your **<caps51>\logicalhost** directory (where <caps51> is the location of your Sun Java Composite Application Suite installation.

2 Double-click the **domainmgr.bat** file. The **Domain Manager** appears.

3 If you have already created a domain, select your domain in the Domain Manager and click the **Start an Existing Domain** button. Once your domain is started, a green check mark indicates that the domain is running.

4 If there are no existing domains, a dialog box indicates that you can create a domain now. Click **Yes**. The **Create Domain** dialog box appears.

5 Make any necessary changes to the **Create Domain** dialog box and click **Create**. The new domain is added to the Domain Manager. Select the domain and click the **Start an Existing Domain** button. Once your domain is started, a green check mark indicates that the domain is running.

For more information about creating and managing domains see the *eGate Integrator System Administration Guide*.

6.6.11 Building and Deploying the Project

The Build process compiles and validates the Project's Java files and creates the Project EAR file.

Build the Project

1 From the Deployment Editor toolbar, click the **Build** icon.

2 If there are any validation errors, a **Validation Errors** pane will appear at the bottom of the Deployment Editor and displays information regarding the errors. Make any necessary corrections and click **Build** again.

3 After the Build has succeeded you are ready to deploy your Project.

Deploy the Project

1 From the Deployment Editor toolbar, click the **Deploy** icon. Click **Yes** when the **Deploy** prompt appears.

2 A message appears when the project is successfully deployed. You can now test your sample.

6.6.12 Running the Sample

Additional steps are required to run the deployed sample Project.

Steps required to run the sample Project:

1 Rename one of the trigger files included in the sample Project from **<filename>.in.~in** to **<filename>.in** to run the corresponding operation.

The File eWay polls the directory every five seconds for the input file name (as defined in the Inbound File eWay Properties window). The JCD then transforms the data, and the File eWay sends the output to an Output file name (as defined in the outbound File eWay Properties window).

The Where Clause defined in the business rule recognizes the trigger as a placeholder for input, allowing a set condition, such as emp_no = 100, to determine the type of output data.

You can modify the following input files to view different output.

- TriggerTableSelect.in
- TriggerDelete.in

- TriggerPsUpdate.in

Having no content in these files causes the operation to read all records.

Verify the output data by viewing the sample output files. See **About the DB2 Connect eWay Sample Projects** on page 74 for more details on the types of output files used in this sample Project. The output files may change depending on the number of times you execute the sample Project, the input file, and also the content of your database table.

Appendix A

Common DataType Conversions

This section lists conversions between the DB2 Connect and OTD/Java datatypes, the types of methods to use for each datatype, and the value or size of the data element that can be used.

What's in this Appendix:

A.1 Common DataType Conversions

The DB2 Connect eWay supports the following data types:.

Table 1 Standard Data Types Supported by the DB2 Connect eWay

DB2 Data Type	OTD/Java Data Type	Java Method or New Constructor to Use (Default: Java Method)	Sample Data
Int	Int	**Integer** java.lang.Integer.parseInt(String s)	123
Smallint	BigDecimal	**Call a New Constructor BigDecimal**: java.math.BigDecimal(String)	123
Number	BigDecimal	**Call a New Constructor BigDecimal**: java.math.BigDecimal(String)	123
Decimal	BigDecimal	**Call a New Constructor BigDecimal**: java.math.BigDecimal(String)	123
BigInteger	Long	**Long:** java.lang.Long.parseLong(String)	123
Short	Short	**Short**: java.lang.Short.parseShort(String)	123

DB2 Data Type	OTD/Java Data Type	Java Method or New Constructor to Use (Default: Java Method)	Sample Data
Real	Float	**Float**: java.lang.Float.parseFloat(String)	2454.56
Float	Double	**Double**: java.sql.Double.parseDouble(String)	2454.56
Double	Double	**Double**: java.sql.Double.parseDouble(String)	2454.56
Timestamp	Timestamp	**TimeStamp**: java.sql.TimeStamp.valueOf(String)	2003-09-04 23:55:59
Time	Time	**Time**: java.sql.Time.valueOf(String)	11:15:33
Date	Date	**Date**: java.sql.Date.valueOf(String)	2003-09-04
Varchar2	String	Direct Assign	Any character
Char	String	Direct Assign	Any character

A.2 Converting Data Types in the DB2 Connect eWay

When working with data in the DB2 Connect eWay OTD's, you may need to do a data conversion. The following tables show input/output data for data conversion:

Table 2 Insert and Update Operations Datatype Conversions (Text/String input data)

DB2 Data Type	OTD/Java Data Type	Java Method or New Constructor to Use (Default: Java Method)	Sample Data
Int	Integer	**Integer** java.lang.Integer.toString(Int)	123
Smallint	BigDecimal	**BigDecimal**: java.math.BigDecimal.toString()	123
Number	BigDecimal	**BigDecimal**: java.math.BigDecimal.toString()	123
Decimal	BigDecimal	**BigDecimal**: java.math.BigDecimal.toString()	123

DB2 Data Type	OTD/Java Data Type	Java Method or New Constructor to Use (Default: Java Method)	Sample Data
Short	Short	**Short**: java.lang.Short.toString(short)	123
Real	Float	**Float**: java.lang.Float.toString(Float)	2454.56
Float	Double	**Double**: java.sql.Double.parseDouble(String)	2454.56
Double	Double	**Double**: java.sql.Double.parseDouble(String)	2454.56
Timestamp	Timestamp	**TimeStamp**: java.sql.TimeStamp.toString()	2003-09-04 23:55:59
Time	Time	**Time**: java.sql.Time.toString()	11:15:33
Date	TimeStamp	**Date**: java.sql.Date.toString()	2003-09-04
Varchar2	String	Direct Assign	Any character
Char	String	Direct Assign	Any character

Index