SeeBeyond ICAN Suite

JDBC/ODBC eWay Intelligent Adapter User's Guide

Release 5.0.1

S E E B E Y O N D ®

Contents

Chapter 5

Using the JDBC eWay Database Wizard 33

Chapter 6

Working with the Sample Projects 45

Introduction

This document describes how to install and configure the JDBC/ODBC eWay Intelligent Adapter.

This introduction includes the following:

- **"eWay Overview" on page 7**

1.1 eWay Overview

The JDBC/ODBC eWay enables the eGate system to exchange data with external databases. This document describes how to install and configure the JDBC/ODBC eWay.

1.1.1 Operational Overview

The JDBC/ODBC eWay uses Java Collaborations to interact with one or more external databases. By using a Java Collaboration Service it is possible for eGate components such as eWay Intelligent Adapters to connect to external databases and execute business rules.

1.1.2 Supported Operating Systems

The JDBC/ODBC eWay is available on the following operating systems:

- Windows 2003, Windows XP and Windows 2000
- Solaris 8 and 9
- HP Tru64 5.1A
- HP-UX 11.0 and 11i
- HPUX 11i V2 (11.23)
- IBM AIX 5.1L and 5.2
- Red Hat Enterprise Linux AS 2.1 (Intel)
- Red Hat Linux 8 (Intel)

Although the JDBC eWay, Repository, and Logical Hosts run on the platforms listed above, the Enterprise Designer requires the Windows operating system. Enterprise Manager can run on any platform that supports Internet Explorer.

1.1.3 System Requirements

The performance and functionality of the JDBC/ODBC eWay depends on the driver(s) selected. Certain drivers may not support all JDBC features. Consult the documentation for your respective driver(s) for more information.

To use the JDBC/ODBC eWay, you need the following:

- An eGate Participating Host
- A TCP/IP network connection.

Host Requirements

The appropriate driver type for your database.

A list of drivers from third party vendors is available at:

http://industry.java.sun.com/products/jdbc/drivers

Installation

This chapter describes how to install the JDBC/ODBC eWay. It includes the following information:

- **Installing the JDBC eWay** on page 9
- **After Installation** on page 10

3.1 Installing the JDBC eWay

During the eGate Integrator installation process, the Enterprise Manager, a web-based application, is used to select and upload eWays (eWay.sar files) from the eGate installation CD-ROM to the Repository.

The installation process includes installing the following components:

- Installing the Repository
- Uploading products to the Repository
- Downloading components (such as Enterprise Designer and Logical Host)
- Viewing product information home pages

Follow the instructions for installing the eGate Integrator in the SeeBeyond *ICAN Suite Installation Guide*, and include the following steps:

- On the Enterprise Manager, select the **JDBCeWay.sar** (to install the eWay) file to upload.
- On the Enterprise Manager, select the **FileeWay.sar** (to install the File eWay, used in the sample Project) file to upload.
- On the Enterprise Manager, install the **JDBCeWayDocs.sar** (to install the documentation, and the sample) file to upload.
- On the Enterprise Manager click on the Documentation tab. Click the document link, or the sample file link. For the sample project, it is recommended that you extract the file to another file location prior to importing it using the Enterprise Explorer's Import Project tool.

For additional information on how to use eGate, please see the *eGate Integrator Tutorial*.

Continue installing the eGate Enterprise Designer as instructed.

3.2   After Installation

Once the eWay is installed and configured it must then be incorporated into a Project before it can perform its intended functions. See the *eGate Integrator User's Guide* for more information on incorporating the eWay into an eGate Project.

Properties of the JDBC/ODBC eWay

This chapter describes how to set the properties of the JDBC/ODBC eWay.

4.1 Working with eWay Property Sheets

On the Properties sheet window and using the descriptions below, enter the information necessary for the eWay to establish a connection to the external application.

4.2 JDBC/ODBC eWay Properties

The JDBC/ODBC's Properties are organized as follows:

- **Outbound JDBC eWay Properties** on page 12
- **Outbound JDBC Non-Transactional eWay Properties** on page 15
- **Outbound JDBC XA eWay Properties** on page 18
- **Inbound JDBC eWay Properties** on page 21
- **Outbound JDBC eWay Environment Properties** on page 22
- **Outbound JDBC Non-Transactional eWay Environment** on page 25
- **Outbound JDBC XA eWay Environment Properties** on page 28
- **Inbound eWay Environment Properties** on page 31

4.2.1. Outbound JDBC eWay Properties

The DataSource settings define the properties used to interact with the external database.

Figure 1 The Outbound JDBC eWay Properties

Description

Description

Specifies the Java class in the JDBC driver that is used to implement the ConnectionPoolDataSource interface.

Required Values

The applicable Java class name. The configured default is **JDBC Connection Pool Datasource**.

InitialPoolSize

Description

Specifies the number of physical connections the pool contains when it is created.

Required Value

A number indicating the initial number of physical connections. The configured default is **2**.

LoginTimeOut

Description

Specifies the number of seconds the driver attempts to log in to the database before timing out.

Required Value

A number indicating the maximum number of seconds allotted for logging in before the process is timed out. A value of **0** (zero) indicates that there is no set limit. The configured default is **0**.

MaxIdleTime

Description

Specifies the maximum number of seconds that a physical connection may remain unused before it is closed.

Required Value

A number indicating the maximum number of seconds that the connection may remain idle before it is closed. valid numeric value. A value of **0** (zero) indicates that there is no set limit. The default is **0**.

MaxPoolSize

Description

Specifies the maximum number of physical connections the pool contains.

Required Value

A number indicating the maximum number of physical connections. A value of **0** (zero) indicates that there is no set limit. The configured default is **10**.

MaxStatements

Description

Specifies the maximum total number of statements that the pool keeps open.

Required Value

A number indicating the number of statements. A value of **0** (zero) indicates that the caching of statements is disabled. The configured default is **1000**.

MinPoolSize

The minimum number of physical connections the pool should keep available at all times.

Required Value

A valid numeric value. A value of **0** (zero) indicates that there are no physical connections in the pool and that new connections will be created as needed. The configured default is **2**.

NetworkProtocol

Description

Specifies the network protocol used to communicate with the server.

Required Values

The applicable network protocol.

PropertyCycle

Description

Specifies the interval in seconds that the pool waits before enforcing the current policy. This policy is defined by the values of the other connection pool properties in this deployment descriptor.

Required Values

A number indicating the maximum number of seconds allotted for the pool to wait before enforcing the current policy.

RoleName

Description

Specifies the initial SQL role name.

Required Values

The applicable initial SQL role name (String).

4.2.2 Outbound JDBC Non-Transactional eWay Properties

Figure 2 Properties of the Outbound JDBC Non-Transactional eWay
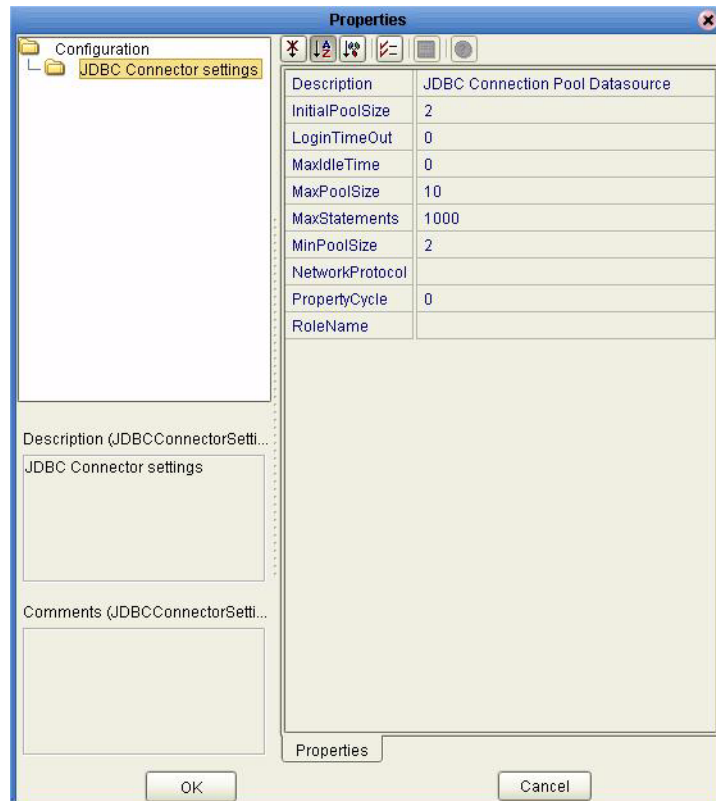
Description

Description

Specifies the Java class in the JDBC driver that is used to implement the ConnectionPoolDataSource interface.

Required Values

The applicable Java class name. The configured default is **JDBC Connection Pool Datasource**.

InitialPoolSize

Description

Specifies the number of physical connections the pool contains when it is created.

Required Value

A number indicating the initial number of physical connections. The configured default is **2**.

LoginTimeOut

Description

The number of seconds driver will wait before attempting to log in to the database before timing out.

Required Value

A number indicating the maximum number of seconds allotted for logging in before the process is timed out. A value of **0** (zero) indicates that there is no set limit. The configured default is **0**.

MaxIdleTime

Description

The maximum number of seconds that a physical connection may remain unused before it is closed.

Required Value

A number indicating the maximum number of seconds that the connection may remain idle before it is closed. valid numeric value. A value of **0** (zero) indicates that there is no set limit. The default is **0**.

MaxPoolSize

Description

The maximum number of physical connections the pool contains.

Required Value

A number indicating the maximum number of physical connections. A value of **0** (zero) indicates that there is no set limit. The configured default is **10**.

MaxStatements

Description

Specifies the maximum total number of statements that the pool keeps open.

Required Value

A number indicating the number of statements. A value of **0** (zero) indicates that the caching of statements is disabled. The configured default is **1000**.

MinPoolSize

The minimum number of physical connections the pool should keep available at all times.

Required Value

A number indicating the minimum number of physical connections. A value of **0** (zero) indicates that there are no physical connections in the pool and that new connections will be created as needed. The configured default is **2**.

NetworkProtocol

Description

Specifies the network protocol used to communicate with the server.

Required Values

The applicable network protocol.

PropertyCycle

Description

Specifies the interval in seconds that the pool waits before enforcing the current policy. This policy is defined by the values of the other connection pool properties in this deployment descriptor.

Required Values

A number indicating the maximum number of seconds allotted for the pool to wait before enforcing the current policy.

RoleName

Description

Specifies the initial SQL role name.

Required Values

The applicable initial SQL role name (String).

4.2.3. Outbound JDBC XA eWay Properties

Figure 3 The Outbound JDBC XA eWay Properties
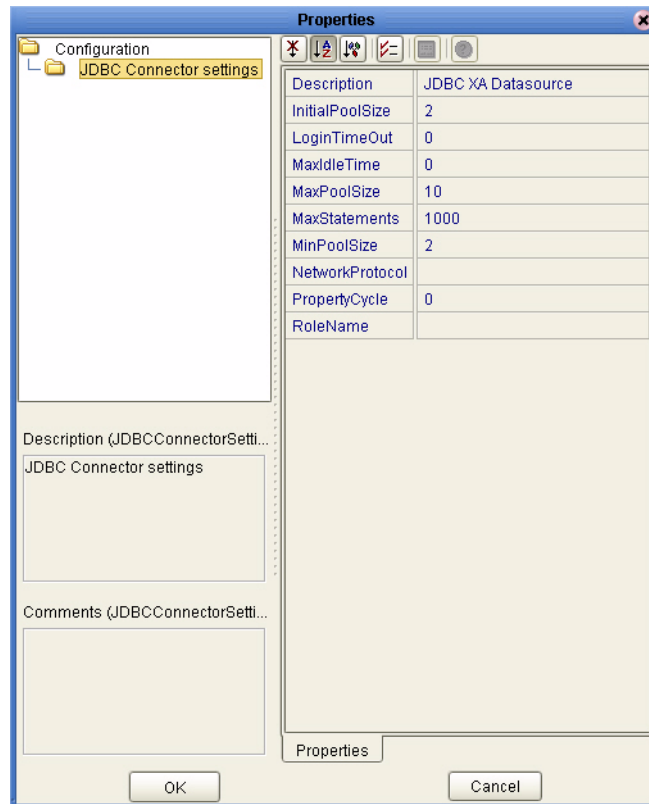
Description

Description

Specifies the Java class in the JDBC driver that is used to implement the ConnectionPoolDataSource interface.

Required Values

The applicable Java class name. The configured default is **JDBC XA Datasource**.

InitialPoolSize

Description

Specifies the number of physical connections for the pool to contain when it is created.

Required Value

A number indicating the initial number of physical connections. The configured default is **2**.

LoginTimeOut

Description

Specifies the number of seconds the driver attempts to log in to the database before timing out.

Required Value

A number indicating the maximum number of seconds allotted for logging in before the process is timed out. A value of **0** (zero) indicates that there is no set limit. The configured default is **0**.

MaxIdleTime

Description

Specifies the maximum number of seconds that a physical connection may remain unused before it is closed.

Required Value

A number indicating the maximum number of seconds that the connection may remain idle before it is closed. valid numeric value. A value of **0** (zero) indicates that there is no set limit. The default is **0**.

MaxPoolSize

Description

Specifies the maximum number of physical connections the pool contains.

Required Value

A number indicating the maximum number of physical connections. A value of **0** (zero) indicates that there is no set limit. The configured default is **10**.

MaxStatements

Description

Specifies the maximum total number of statements the pool keeps open.

Required Value

A number indicating the number of statements. A value of **0** (zero) indicates that the caching of statements is disabled. The configured default is **1000**.

MinPoolSize

Specifies the minimum number of physical connections the pool retains as available at all times.

Required Value

A number indicating the minimum number of physical connections. A value of **0** (zero) indicates that there should be no physical connections in the pool and new connections should be created as needed.

NetworkProtocol

Description

Specifies the network protocol used to communicate with the server.

Required Values

The applicable network protocol.

PropertyCycle

Description

Specifies the interval in seconds that the pool waits before enforcing the current policy. This policy is defined by the values of the other connection pool properties in this deployment descriptor.

Required Values

A number indicating the maximum number of seconds allotted for the pool to wait before enforcing the current policy.

RoleName

Description

Specifies the initial SQL role name.

Required Values

The applicable initial SQL role name (String).

4.2.4. Inbound JDBC eWay Properties

Figure 4 Properties of the Inbound JDBC eWay

Pollmilliseconds

Description

Specifies the polling interval in milliseconds.

Required Value

A number indicating the rate in milliseconds at which the database directory is queried. The configured default is **5000** (5 seconds).

PreparedStatement

Description

Prepared Statement used to query against the database.

Required Value

The Prepared Statement must be the same Prepared Statement you created using the Database OTD Wizard. Only SELECT Statement is allowed. Additionally, no place holders should be specified. There should not be any "?" in the Prepared Query.

4.2.5. Outbound JDBC eWay Environment Properties

Before deploying your eWay, you will need to set the properties of the eWay environment using the following descriptions.

Figure 5 Outbound JDBC eWay Environment Configuration

ClassName

Description

Specifies the Java class in the JDBC driver that is used to implement the **ConnectionPoolDataSource/javax.sql.XADataSource** interface. Change as needed for your driver.

Required Values

An applicable class name. The default is **com.SeeBeyond.jdbcx.jdbc.JDBCDataSource**.

DatabaseName

Description

Specifies the name of the particular database instance being used by the server.

Required Values

An applicable database name.

DataSourceName

Description

Specifies the name of the **XADataSource** or **ConnectionPoolDataSource** implementation, to which the DataSource object delegates behind the scenes when there is connection pooling or distributed transaction management being done.

Required Value

The name of the **XADataSource** or **ConnectionPoolDataSource** implementation. This property is Optional. In most cases, leave this box empty.

Delimiter

Description

Specifies the delimiter character to be used in the DriverProperties prompt.

Required Value

An applicable character indicating the selected delimiter. The default is **#**.

Description

Description

Provides a description of the database.

Required Value

A appropriate description of the database. The configured default is **JDBC Connection Pool Datasource**.

DriverProperties

Description

Specifies the driver properties. If you choose to not use the JDBC driver that is shipped with this eWay, you will need to add the drivers properties to the eWay. Often times the DataSource implementation will need to execute additional properties to assure a connection. The additional methods will need to be identified in the Driver Properties.

Required Value

Set the driver properties according to driver vendor's instruction. For example:

```
setDefTdpName#DBSQL##setWorkspace#Navigator##
```

Password

Description

Specifies the password used to access the database. This is used in conjunction with the **User** property.

Required Values

The correct password assigned to the User name.

PortNumber

Description

Specifies the I/O port number to which the server is listening for connection requests.

Required Values

The appropriate port number. For example, 4100.

ServerName

Description

Specifies the host name of the external database server.

Required Values

An applicable server name.

User

Description

Specifies an available user name with authority to access the database. This is used in conjunction with the **Password** property.

Required Values

An applicable user name.

4.2.6 Outbound JDBC Non-Transactional eWay Environment

Figure 6 Outbound JDBC Non-Transactional eWay Environment Properties

ClassName

Description

Specifies the Java class in the JDBC driver that is used to implement the **ConnectionPoolDataSource/javax.sql.XADataSource** interface. Change as needed for your driver.

Required Values

An applicable class name. The default is **com.SeeBeyond.jdbcx.jdbc.JDBCDataSource**.

DatabaseName

Description

Specifies the name of the particular database instance being used by the server.

Required Values

An applicable database name.

DataSourceName

Description

Specifies the name of the **XADataSource** or **ConnectionPoolDataSource** implementation, to which the DataSource object delegates behind the scenes when there is connection pooling or distributed transaction management being done.

Required Value

The name of the **XADataSource** or **ConnectionPoolDataSource** implementation. This property is Optional. In most cases, leave this box empty.

Delimiter

Description

Specifies the delimiter character to be used in the DriverProperties prompt.

Required Value

An applicable character indicating the selected delimiter. The default is **#**.

Description

Description

Provides a description of the database.

Required Value

A appropriate description of the database. The configured default is **JDBC non-Transactional Connection Pool Datasource**.

DriverProperties

Description

Specifies the driver properties. If you choose to not use the JDBC driver that is shipped with this eWay, you will need to add the drivers properties to the eWay. Often times the DataSource implementation will need to execute additional properties to assure a connection. The additional methods will need to be identified in the Driver Properties.

Required Value

Set the driver properties according to driver vendor's instruction. For example:

```
setDefTdpName#DBSQL##setWorkspace#Navigator##
```

Password

Description

Specifies the password used to access the database. This is used in conjunction with the **User** property.

Required Values

The correct password assigned to the User name.

PortNumber

Description

Specifies the I/O port number to which the server is listening for connection requests.

Required Values

The appropriate port number. For example, 4100.

ServerName

Description

Specifies the host name of the external database server.

Required Values
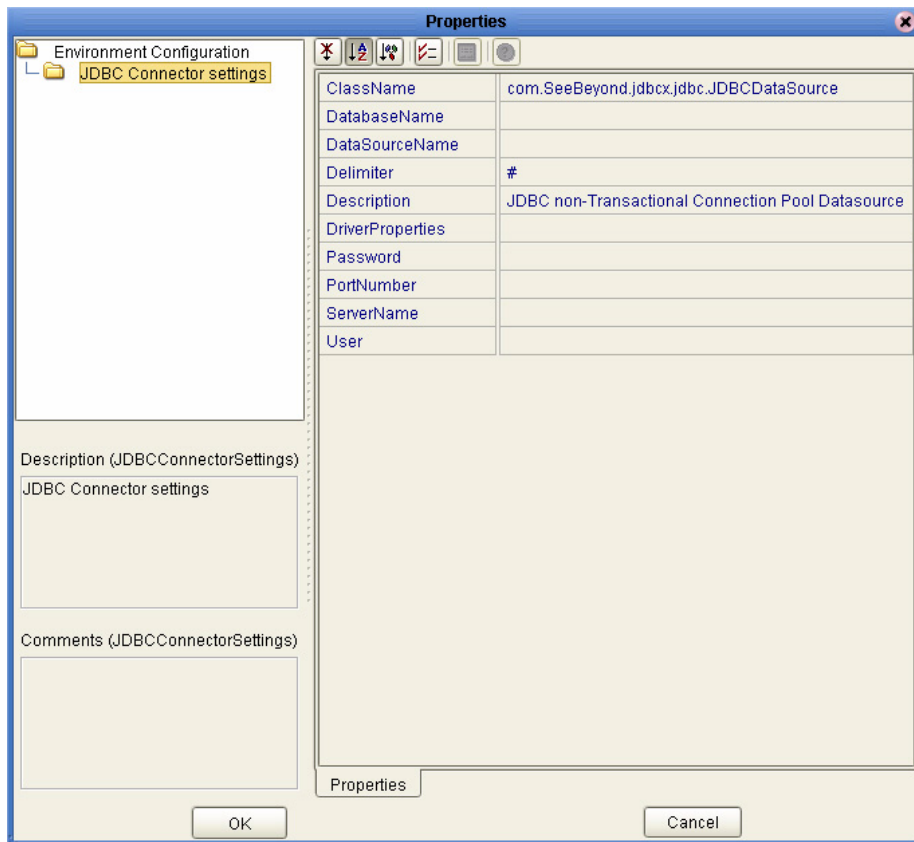
An applicable server name.

User

Description

Specifies an available user name with authority to access the database. This is used in conjunction with the **Password** property.

Required Values

An applicable user name.

4.2.7 Outbound JDBC XA eWay Environment Properties

Figure 7 Outbound JDBC XA eWay Environment Properties

ClassName

Description

Specifies the Java class in the JDBC driver that is used to implement the
ConnectionPoolDataSource/javax.sql.XADataSource interface. Change as needed for
your driver.

Required Values

An applicable class name. The default is **com.SeeBeyond.jdbcx.jdbc.JDBCDataSource**.

DatabaseName

Description

Specifies the name of the particular database instance being used by the server.

Required Values

An applicable database name.

DataSourceName

Description

Specifies the name of the **XADataSource** or **ConnectionPoolDataSource** implementation, to which the DataSource object delegates behind the scenes when there is connection pooling or distributed transaction management being done.

Required Value

The name of the **XADataSource** or **ConnectionPoolDataSource** implementation. This property is Optional. In most cases, leave this box empty.

Delimiter

Description

Specifies the delimiter character to be used in the DriverProperties prompt.

Required Value

An applicable character indicating the selected delimiter. The default is **#**.

Description

Description

Provides a description of the database.

Required Value

A appropriate description of the database. The configured default is **JDBC XA Datasource**.

DriverProperties

Description

Specifies the driver properties. If you choose to not use the JDBC driver that is shipped with this eWay, you will need to add the drivers properties to the eWay. Often times the DataSource implementation will need to execute additional properties to assure a connection. The additional methods will need to be identified in the Driver Properties.

Required Value

Set the driver properties according to driver vendor's instruction. For example:

```
setDefTdpName#DBSQL##setWorkspace#Navigator##
```

Password

Description

Specifies the password used to access the database. This is used in conjunction with the **User** property.

Required Values

The correct password assigned to the User name.

PortNumber

Description

Specifies the I/O port number to which the server is listening for connection requests.

Required Values

The appropriate port number. For example, 4100.

ServerName

Description

Specifies the host name of the external database server.

Required Values
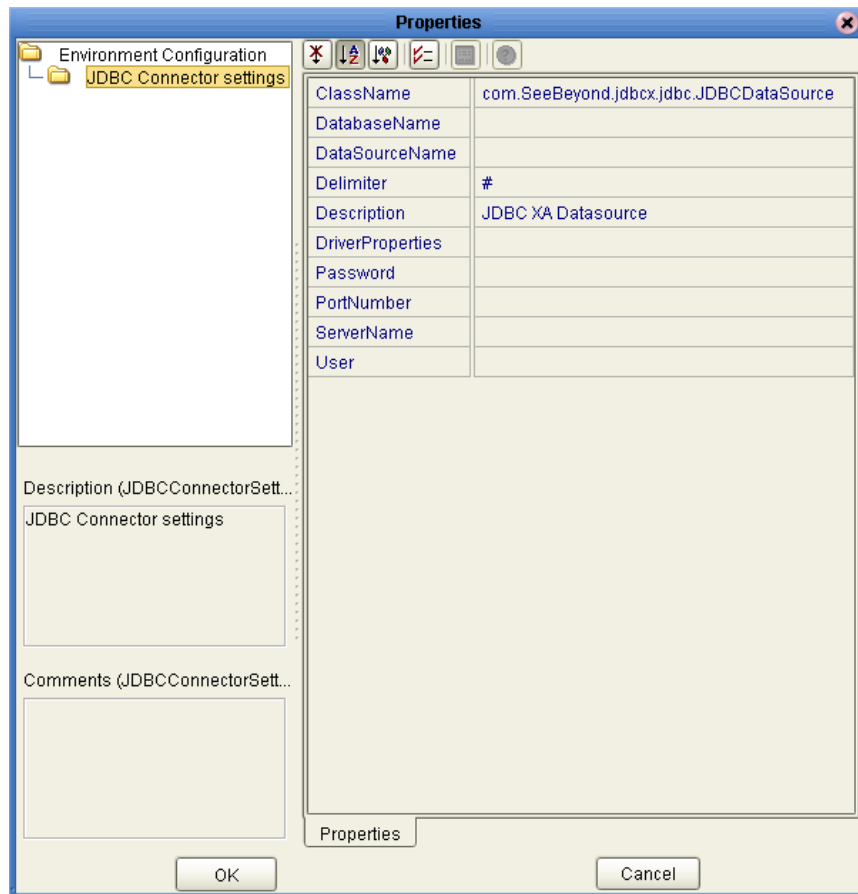
An applicable server name.

User

Description

Specifies an available user name with authority to access the database. This is used in conjunction with the **Password** property.

Required Values

An applicable user name.

4.2.8. **Inbound eWay Environment Properties**

Figure 8 Inbound eWay Environment

ClassName

Description

Specifies the Java class in the JDBC driver that is used to implement the ConnectionPoolDataSource/javax.sql.XADataSource interface. Change this as needed for your driver.

Required Values

An applicable class name.

Password

Description

Specifies the password used to access the database. This is used in conjunction with the User property.

Required Values

The correct password assigned to the User name.

URL

Description

This is the JDBC URL required to gain access to the database. The URL usually starts with **jdbc**; followed by <subprotocol> and ends with information that identifies the data source. For example:

```
jdbc:<driver>:<data-source-name>[;<attribute-name>=<attribute-value>]
```

If you do not select URL in the **connection method** this parameter is ignored. For more information on the JDBC URL, please consult the documentation of your specific driver.

Required Values

The applicable JDBC URL.

User

Description

Specifies an available user name with authority to access the database. This is used in conjunction with the password property.

Required Values

An applicable user name.

Using the JDBC eWay Database Wizard

This chapter describes how to use the JDBC eWay Database Wizard to build OTDs.

5.1 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.

The OTD Wizard allows the addition and removal of columns/nodes in an OTD. Nodes with the same name and type as existing nodes are allowed by the wizard, but should not be created, and will result in generic code generation errors upon OTD activation of the OTD.

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

Steps Required to Create an OTD Include:

- **Select Wizard Type** on page 34
- **Connect To Database** on page 35
- **Select Database Objects** on page 35
- **Select Table/Views** on page 36
- **Select Procedures** on page 40
- **Add Prepared Statements** on page 41
- **Specify the OTD Name** on page 43

Select Wizard Type

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

2. The **Select Wizard Type** window appears, displaying the available **OTD** wizards (see Figure 9).

Figure 9 OTD Wizard Selection
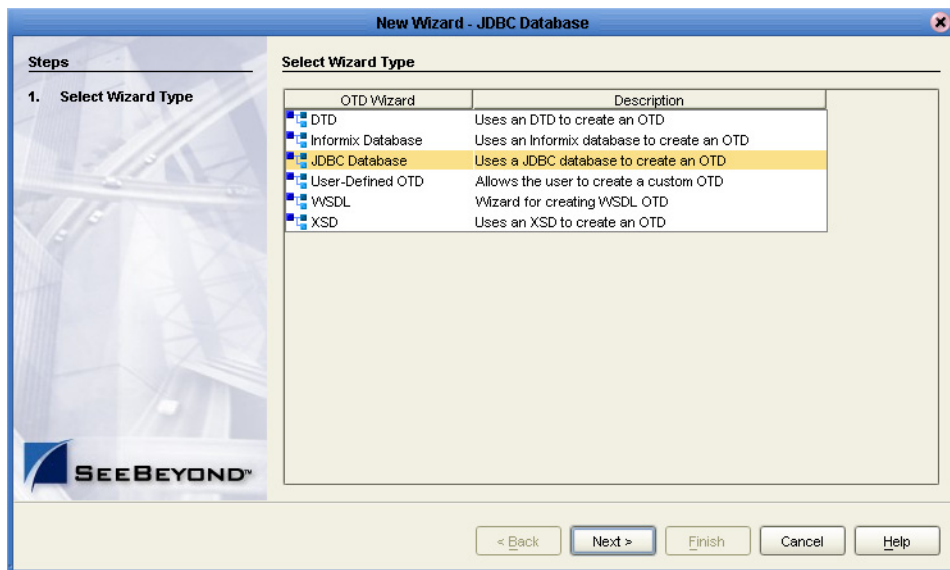
3. From the list, select the JDBC Database OTD and click **Next**. The **Specify Database Connection Information** window appears (see Figure 10).

Figure 10 Database Connection Information
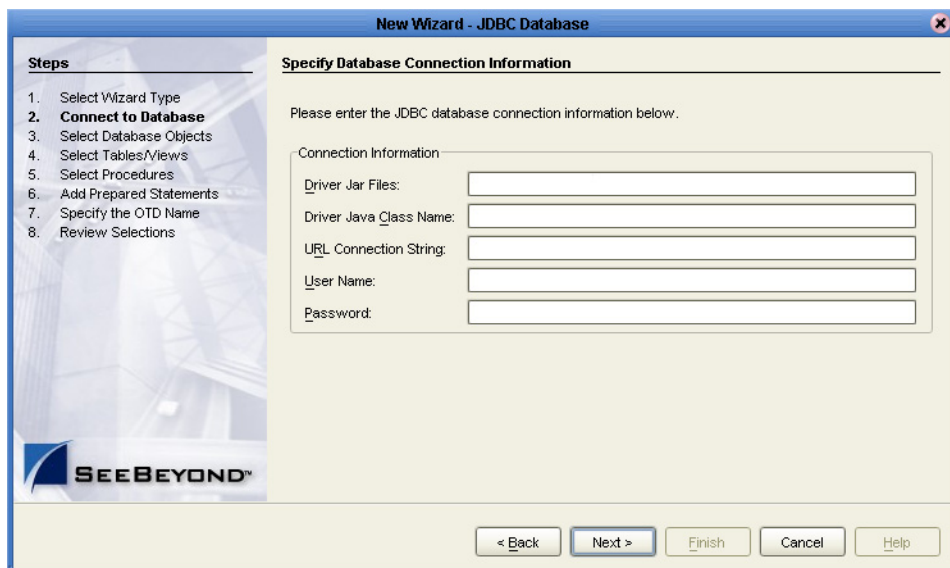
Connect To Database

1   On the Specify Database Connection Information window, enter the following:

- **Driver Jar Files** – the complete path to driver Jar files, separated by semicolons.
- **Driver Java Class Name** – the driver class name.
- **URL Connection String** – the URL connection string.
- **User Name** – the user name required to log into the database.
- **Password** – the password required to log into the database.

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

Select Database Objects

1   On the Select Database Objects window, select **Tables/Views**, **Procedures**, and **Prepared Statements** check boxes (see Figure 11).

Figure 11   Select Database Objects
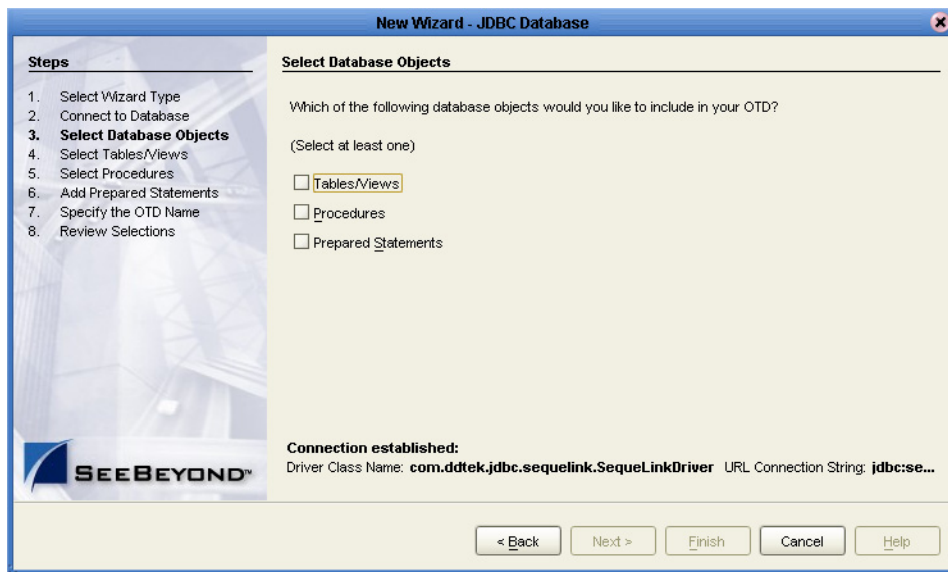
2   Click **Next**. The **Select Tables/Views** window appears. See **Figure 12 on page 36**.

Figure 12   Select Tables/Views
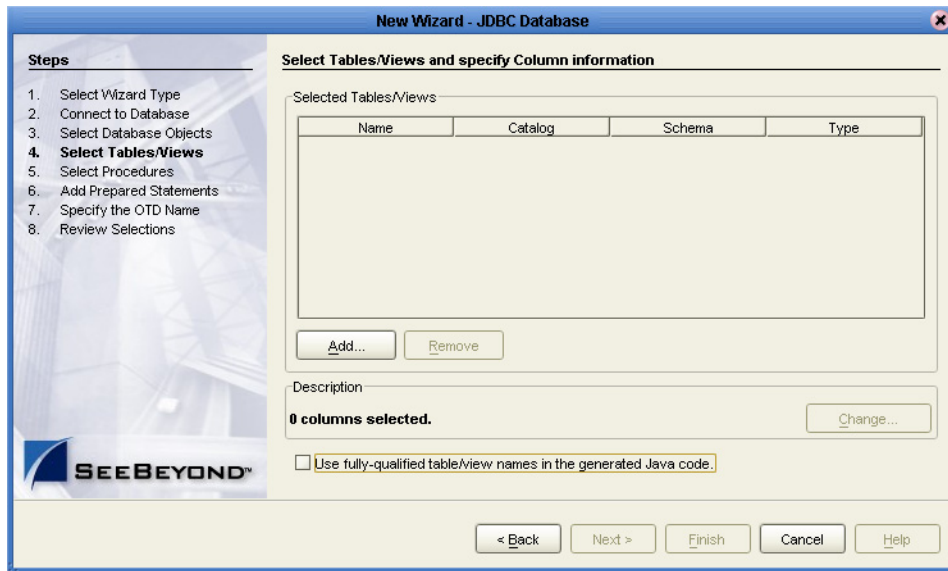
Note: *Views are read-only and are for informational purposes only.*

Note: *Not all drivers provide metadata information such as column names and data types. If your table does not have column names and data types, add them before saving the OTD.*

The OTD Wizard allows the addition and removal of columns/nodes in an OTD. Nodes with the same name and type as existing nodes are allowed by the wizard, but should not be created, and will result in generic code generation errors upon OTD activation of the OTD.

Select Table/Views

1 On the Select Tables/Views window, click the **Add** button. The **Add Tables** window appears. See **Figure 13 on page 37**.

Figure 13 Add Tables Window
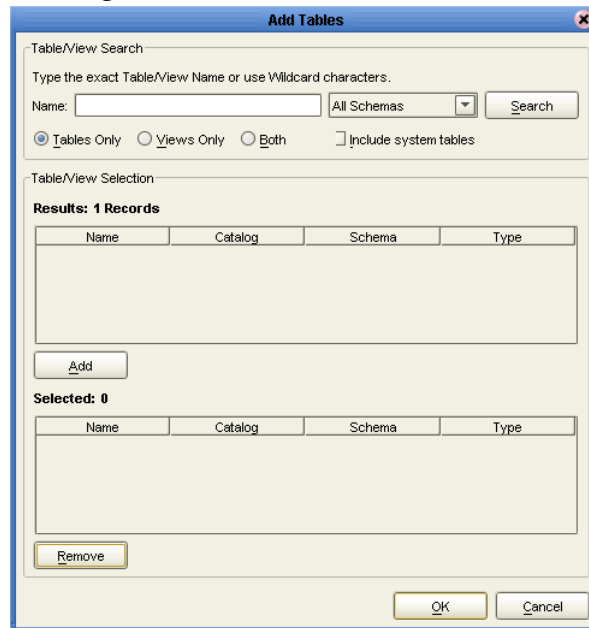
2   In the **Add Tables** window, select whether your selection criteria will include table data, view only data, both, and/or system tables.

3   From the **Table/View Name** drop down list, select the location of your database table and click **Search** (see Figure 14).

Note: *Click Search to find the desired table or tables. You can also use wildcard characters to search for a table or view. Available wildcard characters include the "?", "_", and "*". For example, you can use "AB?CD", "AB_CD", or "AB*CD". However, do not use "%". Using this character results in nothing being returned.*

Figure 14 Database Wizard - All Schemes
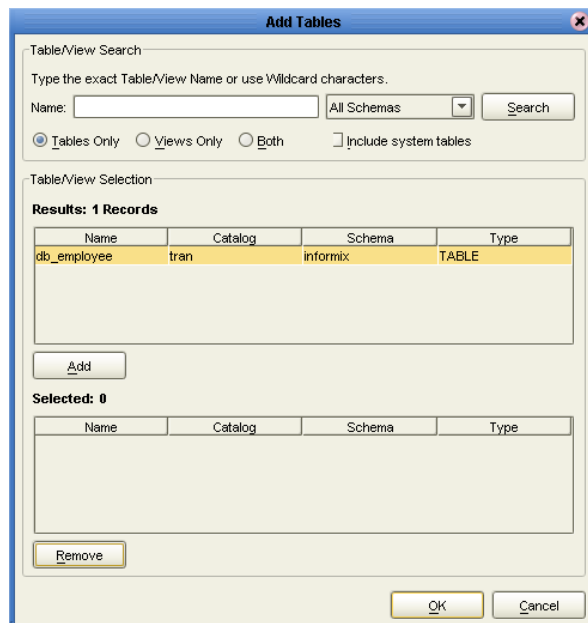
4   Select a table and click **OK**. The selected table is added to the **Selected Tables/
    Views** window. See Figure 15.

Figure 15   Selected Tables/Views window with a table selected
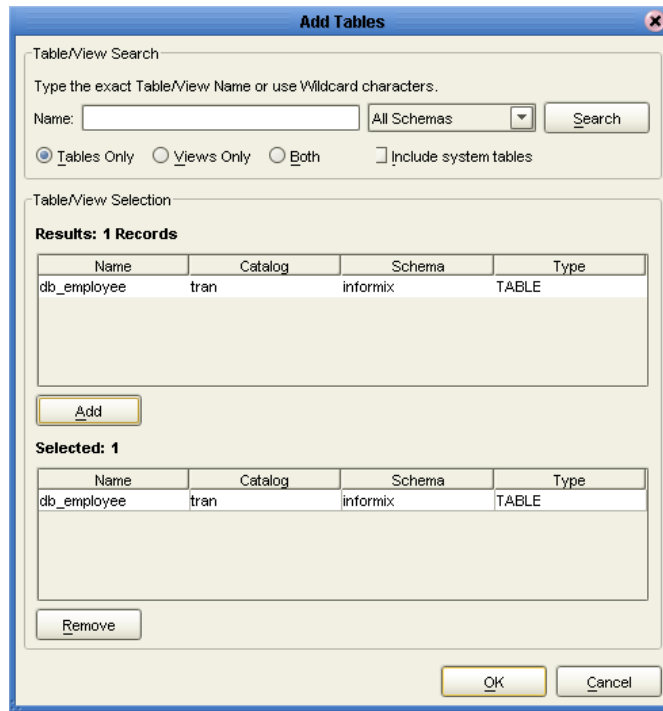
5   On the Selected Tables/Views window, review your selected tables. To make
    changes to the selected Table or view, click **Change**, or click **Next** if no additional
    changes are required (see Figure 16).

Figure 16   Selected table and column window
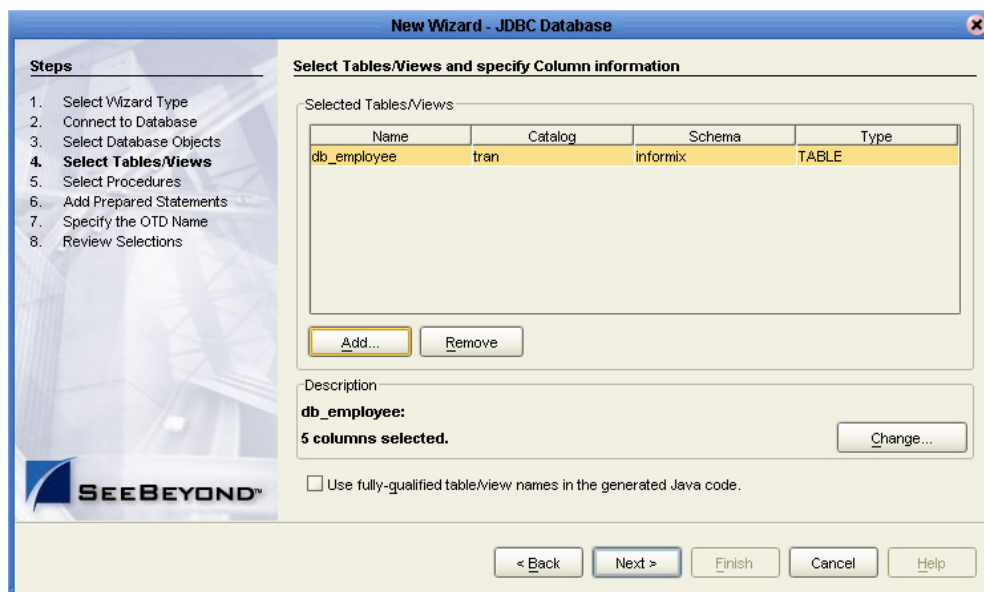
6   If you click **Change**, the Table/View Columns window appears, allowing you to select or deselect any table columns. You can also change the data type for each table by highlighting the data type and selecting a different data type from the drop-down list. Double check the datatypes the driver is returning. If they are not correct, change them to the appropriate types (see Figure 17).

The buttons in this window include:

- **Select All** – allows you to select all columns.

- **Invert Selection** – allows you to invert the order of the selected columns.

- **Clear All** – allows you to deselect all columns.

- **Advanced** – allows you to perform advanced operations with the columns. See the *eGate Integrator User's Guide* for details.

- **Restore Metadata** – allow you to restore the data to its original state before you made any changes via the wizard; returns you to the **Specify Database Connection** window.

Figure 17   Table/View Columns window
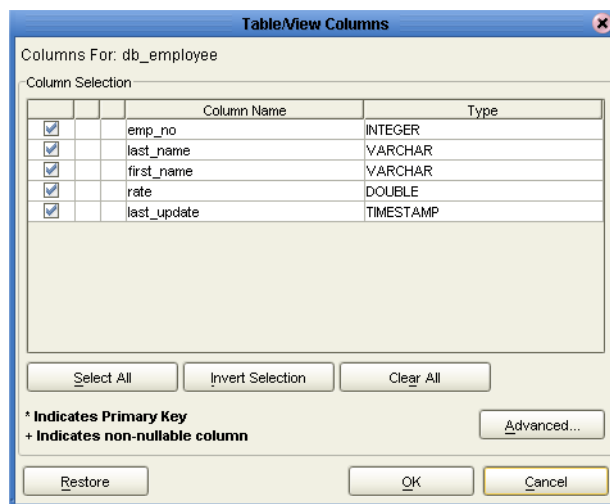
7   When you are finished using this window, click **OK** to save your changes and return to the **Select Tables/Views** window.

*Note: When using Prepared Statement packages, select **Use fully qualified table/view names in the generated code**. See **Figure 18 on page 40**.*

Select Procedures

1 On the **Select Procedures and specify Resultset and Parameter Information** window, click **Add** (see Figure 18).

Figure 18   Select Procedures window
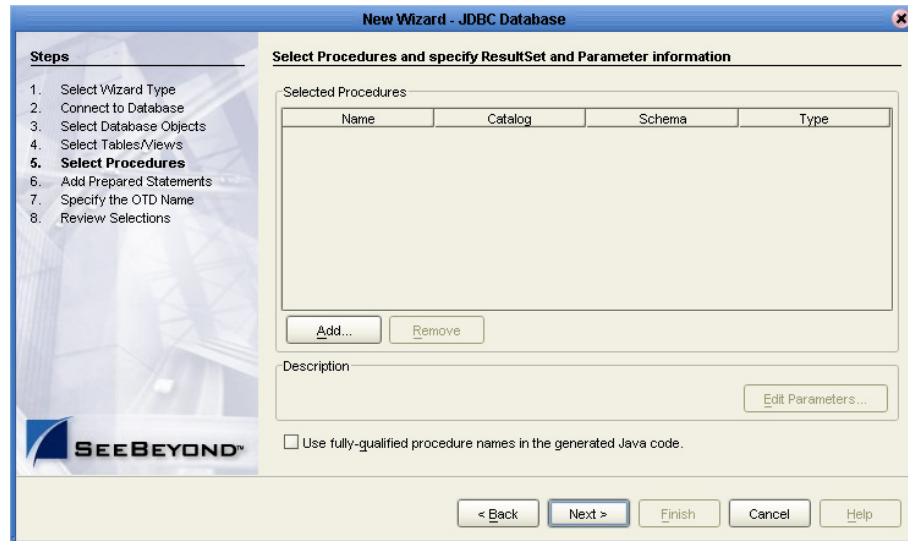
2 On the **Select Procedures** window, enter the name of a Procedure or select a table from the drop down list. Click **Search**. Wildcard characters can also be used.

Note: *On some connected databases, you must use lower case schema names when calling stored procedures.*

3 In the resulting **Procedure Selection** list box, select a Procedure (see Figure 19). Click **OK**.

Figure 19   Add Procedures
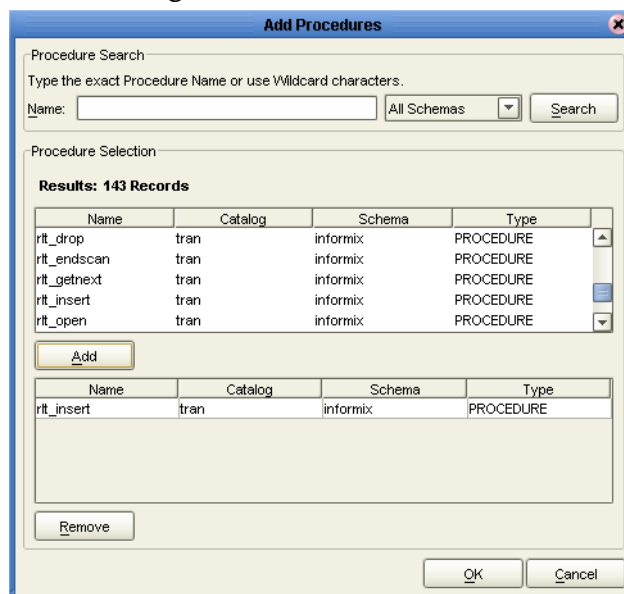
4   On the **Select Procedures and specify Resultset and Parameter Information** window click **Edit Parameters** to make any changes to the selected Procedure (see Figure 20).

Figure 20   Procedure Parameters
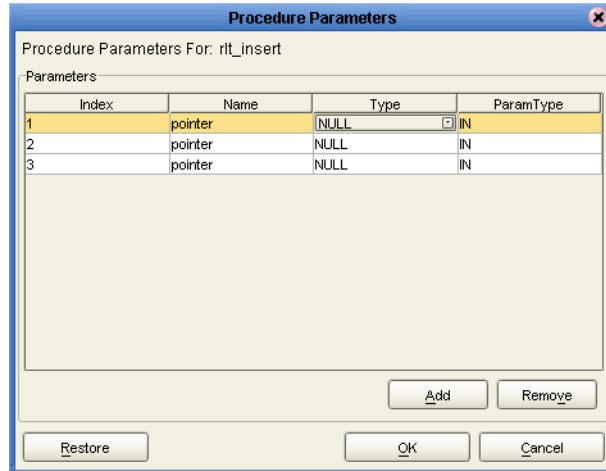
5   To restore the data type, click **Restore**. When finished, click **OK**.

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

Note: *At this time, Resultset is not supported.*

Add Prepared Statements

1   On the **Add Prepared Statements** window, click **Add**. The **Add Prepared Statement** window appears (see Figure 21).

2   Enter the name of a Prepared Statement and create an SQL statement using the SQL Statement window.

Figure 21   Prepared SQL Statement
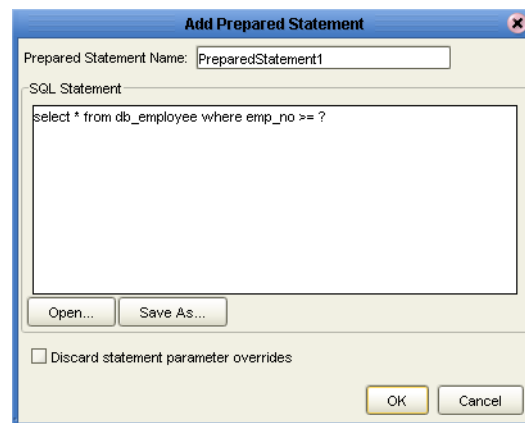
3 Click **Save As** to save the statement with a name, or click the **OK** button to exit the window.

On the **Add Prepared Statement** window, the name you assigned to the Prepared Statement appears (seeFigure 22).

Figure 22 Add Prepared Statement window
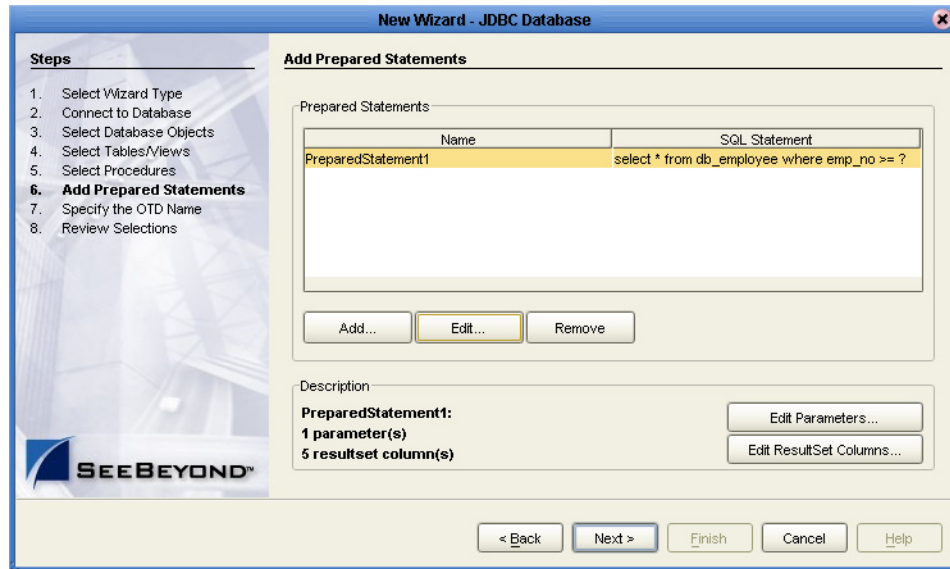
4 To edit the parameters, click **Edit Parameters**. You can change the datatype by clicking in the **Type** field and selecting a different type from the list (see Figure 23).

Figure 23 Edit the Prepared Statement Parameters.
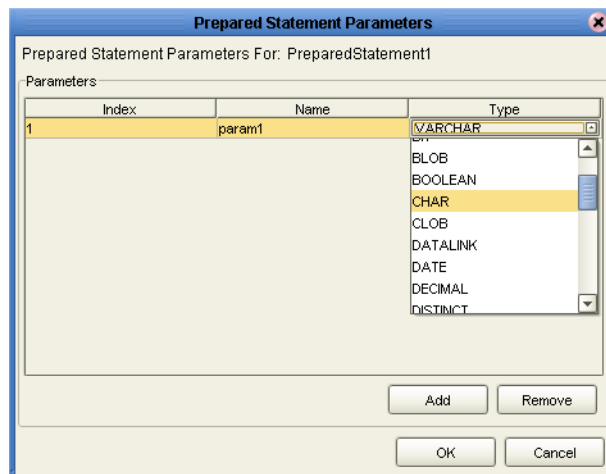
5 Click **Add** if you want to add additional parameters to the Statement or highlight a row and click **Remove** to remove it. Click **OK** to save your changes and exit the window. Double check the datatypes the driver is returning. If they are not correct, change them to the appropriate types.

6 To edit the Resultset Columns, click **Edit Resultset Columns**. Both the Name and Type are editable. Click **OK** to save your changes and exit the window (see **Figure 24 on page 43**.

Figure 24 ResultSet Columns
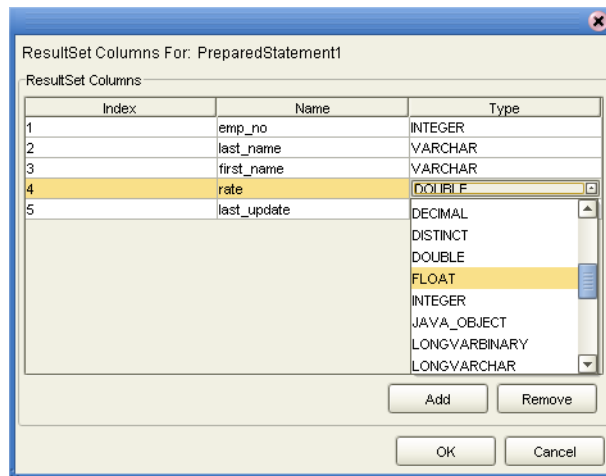
Specify the OTD Name

1 On the **Specify the OTD Name** window, enter a name for the OTD. The OTD contains the selected tables and the package name of the generated classes. See **Figure 25**.

Figure 25 Naming an OTD
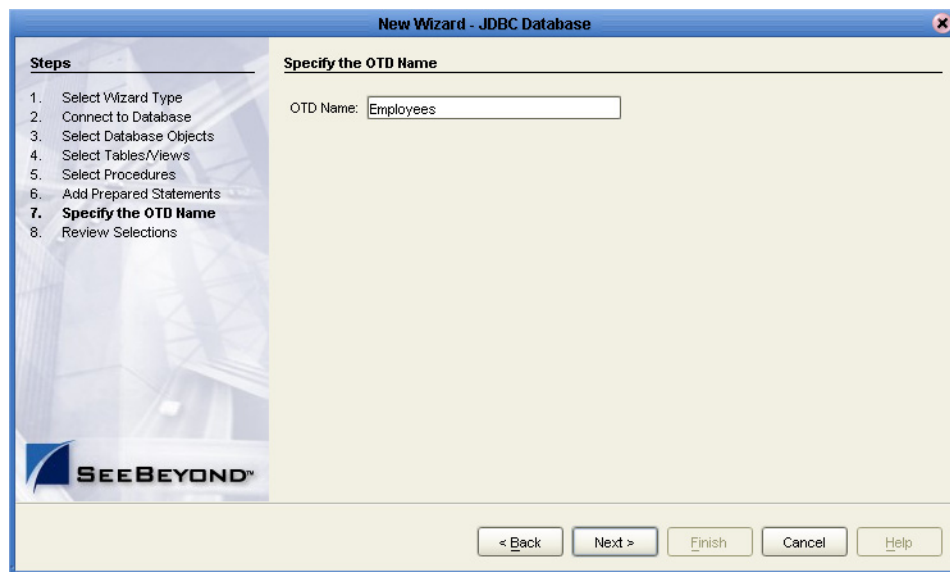
2 Review the summary of the OTD (see **Figure 26 on page 44**). Click **Back** if necessary, to return to a previous screen to make any corrections **.**

Figure 26   Database Wizard - Summary
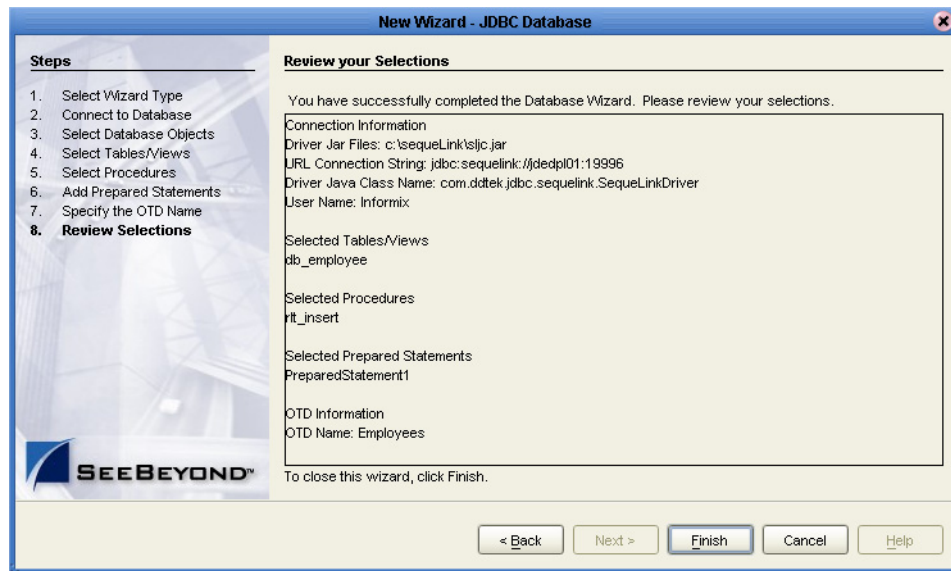
If you are satisfied with the OTD information, click **Finish** to begin generating the OTD. When complete, the resulting **OTD** appears in the OTD Editor.

Working with the Sample Projects

This chapter describes how to build an JDBC eWay project in a production environment.

This Chapter Includes:

Note: *While several steps are required to create, activate, and deploy a Project, only the steps containing information relevant to the JDBC eWay are included in this chapter. For more detailed information on how to complete a sample Project, see the eGate Integrator Tutorial.*

6.1 Sample Projects Overview

Sample Projects are designed to provide an overview of the basic functionality of the JDBC/ODBC eWay by identifying how information is passed between eGate and supported external databases.

Sample Projects Include:

JDBC_JCE_Sample – demonstrates how to use the insert, update, delete, and select employee records from the db_employee table using eGate Integrator.

JDBC_BPEL_Sample – demonstrates how to insert, update, delete, and select employee records from the db_employee table using eInsight's BPEL business process.

Sample Data Used:

Data used for the sample Projects are contained within a table called **db_employee**. The table has the following columns:

Table 1   Sample Project data – db_employee table

Column Name	Data Type	Data Length
emp_no	INTEGER	10
last_name	STRING	30
first_name	STRING	30
rate	FLOAT	15
last_update	DATE	19

6.2 Locating and Importing the Sample Projects

The eWay sample Projects are included in the **JDBCeWayDocs.sar**. This file is uploaded separately from the JDBC eWay SAR file during installation.

Upload the JDBCeWayDocs.sar file to the Repository and begin downloading the sample Projects using the **DOCUMENTATION** tab in the Enterprise Manager to a folder of your choosing.

To use the sample Project, first import it into the SeeBeyond Enterprise Designer using the Enterprise Designer Project Import utility.

To Import the Sample Project:

1   From the Enterprise Designer's Project Explorer pane, right-click the Repository and select **Import**.

2   In the **Import Manager** window, browse to the directory that contains the sample Project zip file.

3   Select the sample file and then click **Open**.

4   Click the **Import** button. If the import was successful, then click the **OK** button on the **Import Status** window.

6.3   Running the Sample Projects

Steps required to run a sample Project include:

- Setting the Properties
- Creating the external Environment
- Deploying the Project
- Running the Sample

6.3.1   Setting the Properties

The sample uses both inbound and outbound File eWays, as well as an outbound JDBC eWay. Use the following information to configure the sample Project eWays. For additional information on the eWay properties, see **JDBC/ODBC eWay Properties** on page 11.

To Configure File eWays:

1   On the Connectivity Map canvas double-click the **Inbound File eWay**. The **Properties** window for the Inbound File eWay opens.

2   Modify the parameter settings for your system. Change the Directory and Input file name to match the location and name of the sample data file.

3   Click **OK** to close the **Properties** window.

4   On the Connectivity Map, double-click the Outbound File eWay. The **Properties** window for the Outbound File eWay opens.

5   Modify the required parameter settings for your system, including the target Directory and Output file name. For the included samples, change the Directory field to **<valid path to the directory where the output file will be stored>**. The Output File Name to **Output1.dat**. For the remaining parameters, use the default settings.

6   Click **OK** to close the Properties window.

To Configure the Outbound JDBC eWay:

1   On the Connectivity Map, double-click the JDBC eWay.

2   The **Properties** window for the JDBC eWay opens. Modify the parameter settings for your system. Click **OK** to close the Properties window.

6.3.2.   Creating the External Environment

To review the components of the Sample project, there is an Inbound and an Outbound File eWay, an eWay, and a Collaboration.

To create the external environment for the Sample project:

1   On the Environment Explorer, highlight and right-click the JDBC profile.

2   Select **Properties** and enter the configuration information required for your Outbound JDBCeWay. See **Working with eWay Property Sheets** on page 11.

6.3.3 Deploying a Project

For instruction on deploying a sample Project, see the *eGate Integrators User's Guide*.

6.3.4. Running the Sample

For instruction on how to run a Sample project, see the *eGate Integrator Tutorial*.

Once the process has completed, the Output file in the target directory configured in the Outbound File eWay will contain all records retrieved from the database in a text format.

6.4   A Word about Drivers

Drivers are uniquely different in what they do and the type of functions they support. The JDBC/ODBC eWay allows you to pick and choose which driver is best suited for your application environment.

There are, or can be, significant differences and limitations between drivers. The performance and functionality of the JDBC/ODBC eWay depends on the selected driver(s). Certain drivers may not support all JDBC features. Consult the documentation for your respective drivers) for more information.

6.4.1 Drivers Used to Create Sample Projects

Sample Projects included with this eWay use the DataDirect Technology SequeLink **sljc.jar** driver. When using any driver to build OTDs, be sure to include the absolute path in the Database Connection Information window. See **Figure 10 on page 34**. Alternately, you can also copy the jar file to:

```
ican50\logicalhost\stcis\lib.
```

To set up a spylog, insert the following string in the JDBC eWay Environment Properties window.

```
setSpyAttributes#log=(file)c:/temp/spy.log;logTName=yes##
```

You must also copy the **spy.jar** driver to:

```
ican50\logicalhost\stcis\lib.
```

6.4.2 Troubleshooting Your Drivers

Refer to the following when troubleshooting Driver issues.

▪ The ReceiveOne operation in BPEL is not supported when using inbound functions with some drivers.

- Some drivers do not support Updatable Resultsets. If you find this to be the case, use a Prepared Statement to Update, Insert, and Delete data.

- Not all drivers provide metadata information such as column names and data types. If your table does not have column names and data types, add them before saving the OTD.

6.5 Using the Sample Project in eInsight

To begin using the sample eInsight Business Process project, you must first import the project and view it from within the Enterprise Designer using the Enterprise Designer Project Import utility. Import the **JDBC_BPEL_Sample** file contained in the eWay sample folder on the installation CD-ROM.

Note: *eInsight is a Business Process modeling tool. If you have not purchased eInsight, contact your sales representative for information on how to do so.*

Before recreating the sample Business Process, review the *eInsight Business Process Manager User's Guide* and the *eGate Integrator Tutorial*.

6.5.1 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:

- Java Messaging Service (JMS)

- 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 runs the Business Process, it automatically invokes that component via its Web Services interface.

6.6 Working with Business Process Activities

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, select the desired operation, found under the Project OTD in Enterprise Explorer and drag it onto the eInsight Business Process canvas.

The following operations are available:

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

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: For information on how to create a database OTD, refer to **Using the Database OTD Wizard** *on page 33.*

The table below shows the inputs and outputs to each of these eInsight operations:

eInsight Operation	Input	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
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, and definition of new item to be inserted	Returns status.
Delete	where() clause	Returns status.

6.7 Working with the JDBC_BPEL_Sample Sample Project

The **JDBC_BPEL_Sample** sample Project uses four business processes to demonstrate database operations.

Operations used in the sample Project include:

- Select One
- Update
- Insert
- Delete

Using the where() Clause

A BPEL where() clause statement is used throughout the samples. It can be joined by AND/OR with conditions of "=", "!=", "<>", "<", ">", "<=", ">=".

For example: where() Clause such as where column2=2 AND column1=1 OR column3=3 is valid

Note: Refer to the eInsight Business Process Manager User's Guide for specific information on creating and using a Business Process in eInsight

Using Triggers

Sample Projects in this chapter use triggers to initiate the business process. You can find triggers for each of the operations in the sample Project folder.

6.7.1 SelectOne

The Select One database operation describes how to retrieve the last name of an employee in the database. In this sample, SelectOne operation uses a where() clause to define the criteria for selecting, and returning the first row in the database that meets required conditions.

This business process describes the account retrieval process seen in **Figure 27**.

Figure 27   Employee Name Retrieval

Business Process:

1 The File eWay subscribes to an external directory and activates a trigger (emp_no > 0) to query the database.

2 A where() clause loops through the database and queries the first record matching the criteria.

Figure 28 BP_GetEmployee Business Process - Find Name
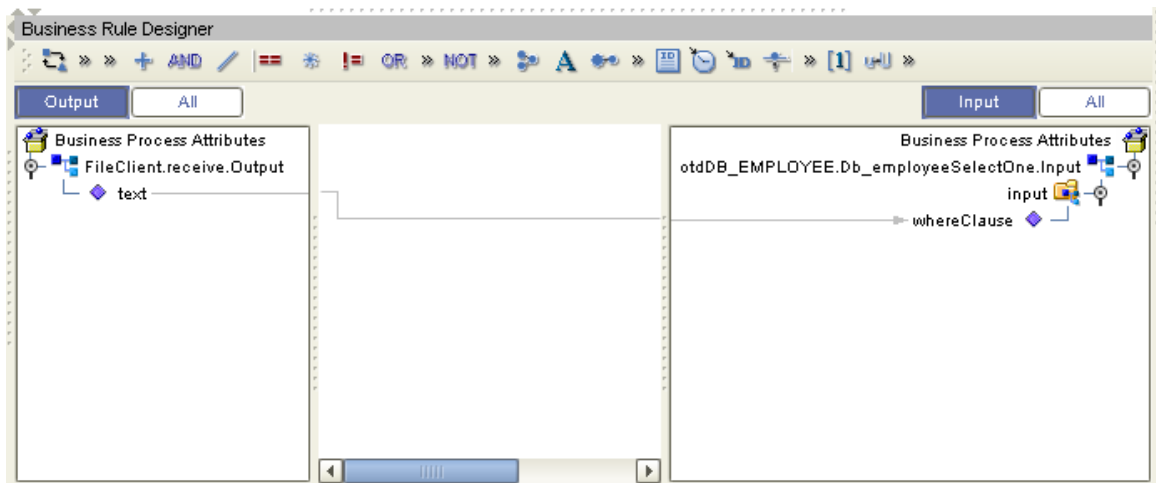
3 The employee's last name (last_name) is written into an input file, before passing onto the Outbound File eWay.

Figure 29 BP_GetEmployee Business Process - Send Name
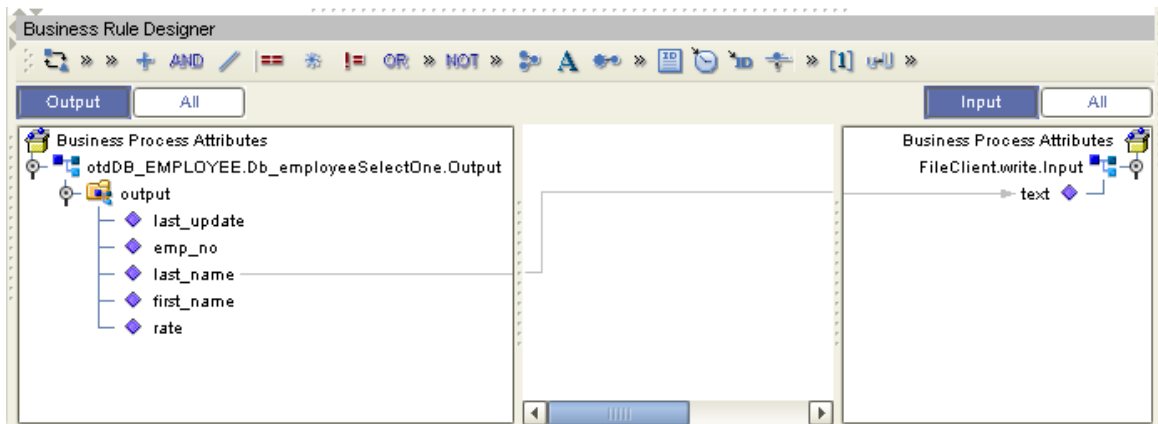
6.7.2 Insert

The Insert operation inserts a new row in the db_employee table. An empty trigger is used to initiate the operation. Data for the Insert is provided by using String Literal found in the business process.

Figure 30 shows a sample eInsight Business Process using the Insert operation.

Figure 30 Insert Sample Business Process

Figure 31 shows the input data used for the Insert operation.

Figure 31 Insert Input
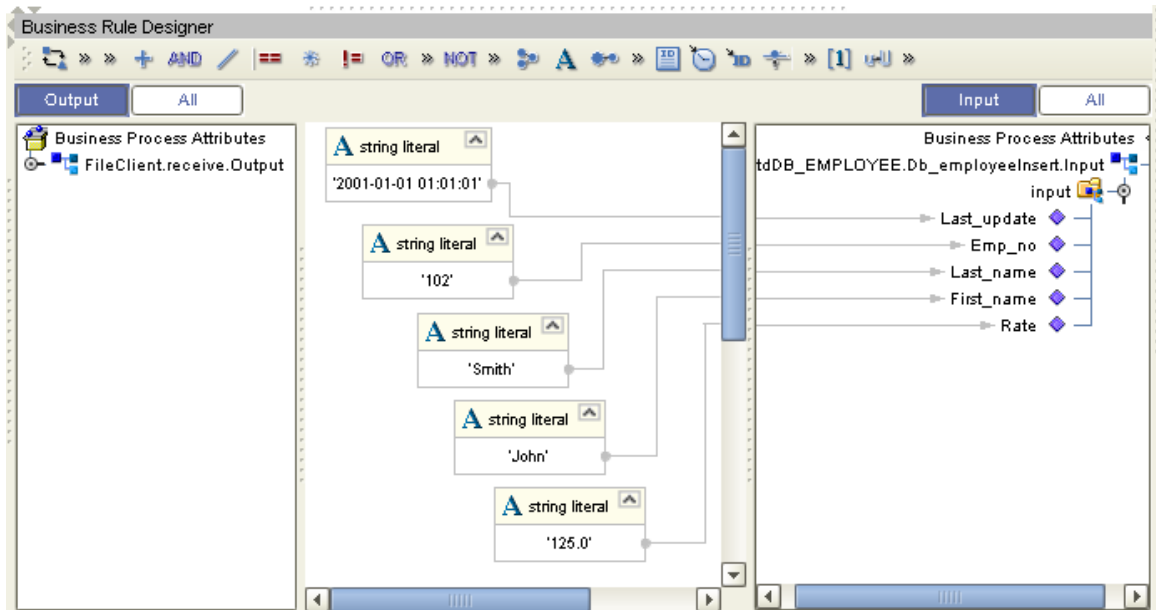
Figure 32 shows the output of the Insert operation, which is a status indicating the number of rows created.

Figure 32 Insert Output
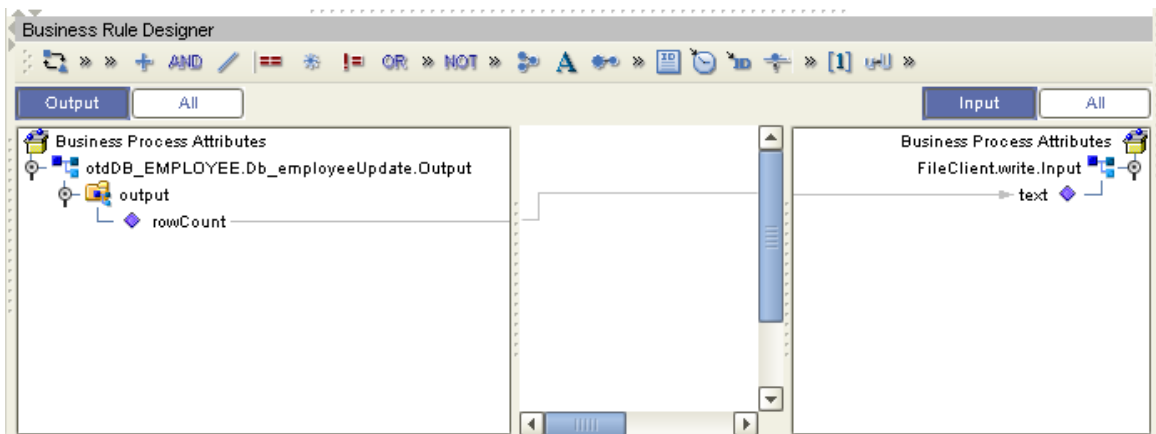
6.7.3 Update

The Update operation updates rows that fit certain criteria defined in a where() clause. An empty trigger is used to initiate the operation.

Figure 33 shows a sample eInsight Business Process using the Update operation. In this process, the operation updates a rate for a certain employee.

Figure 33 Update Sample Business Process

Figure 34 shows the definition of the where() clause for the Update operation. In this example, employee number 115 is updated to have a new rate of 168.75.

Figure 34 Update Input

Figure 35 shows the output of the Update operation, which is a status indicating the number of rows updated.

Figure 35 Update Output

6.7.4 Delete

The Delete operation deletes rows that match the criteria defined in a where() clause. An empty trigger is used to initiate the operation. The output is a status of how many rows where deleted.

Figure 36 shows a sample eInsight Business Process using the Delete operation. In this process, the operation deletes a row that matches a certain employee number.

Figure 36 Delete Sample Business Process

Figure 37 shows the definition of the where() clause for the Delete operation. In this example, employee 115 is deleted.

Figure 37 Delete Input

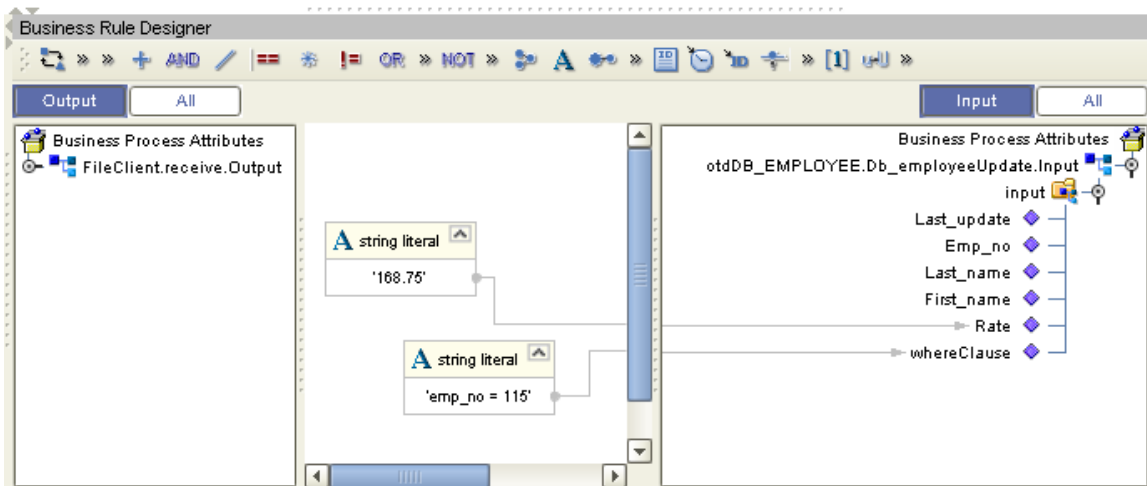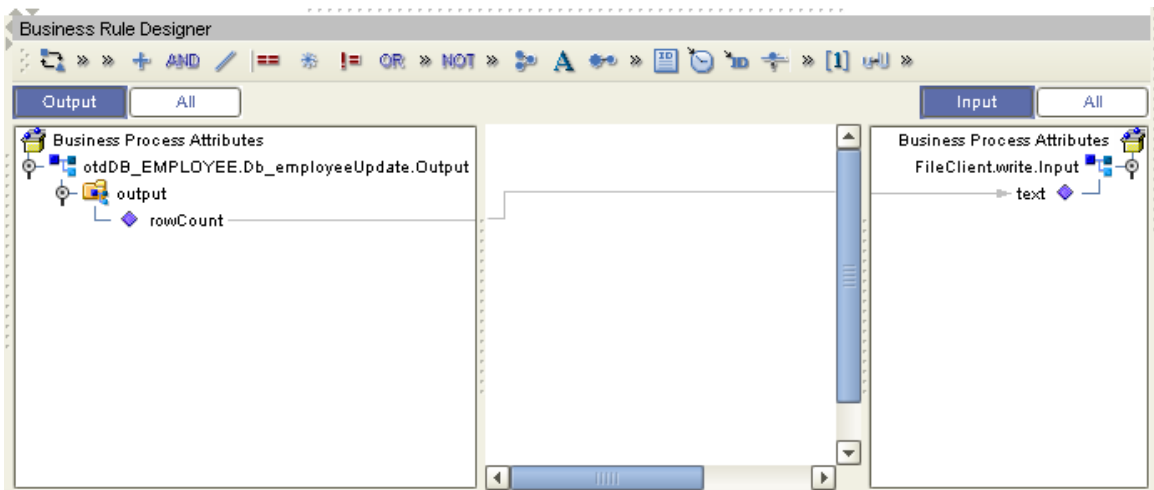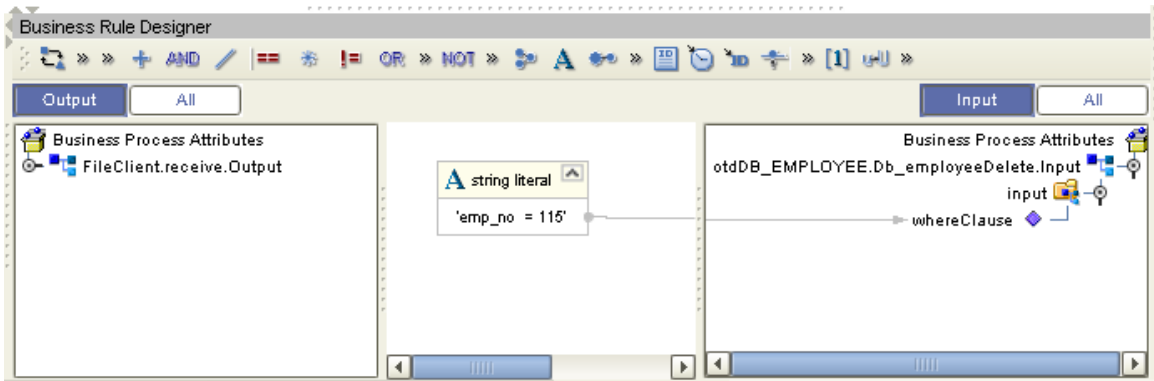**Figure 38** shows the output of the Delete operation, which is a status indicating the number of rows deleted.

Figure 38 Delete Output
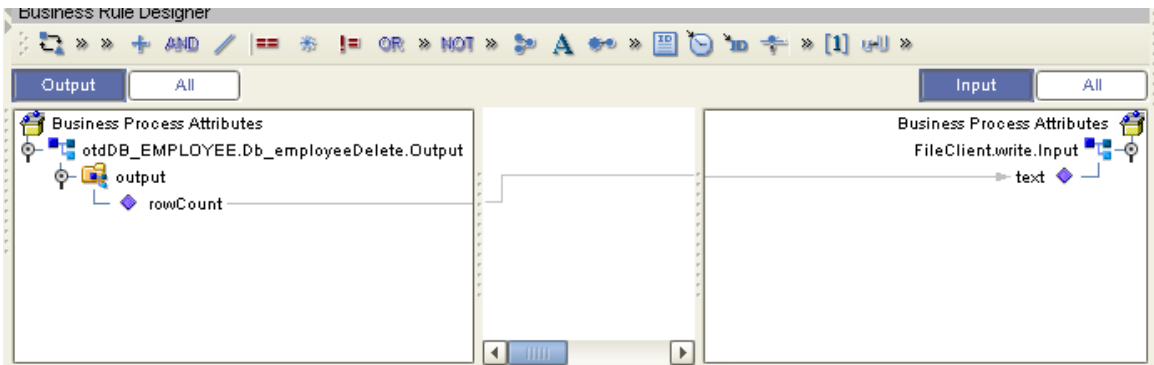
6.8   Additional BPEL Operation Examples

Listed below are examples of the two additional operations—SelectAll and SelectMultiple—that are not included in the sample Project.

6.8.1   SelectAll

The input to a SelectAll operation uses a where() clause to define the criteria for selecting, and returning all rows from the database.

Figure 39 shows a sample eInsight Business Process using the SelectAll operation. In this process, the SelectAll operation returns all rows where the emp_no is >=0.

Figure 39   SelectAll Sample Business Process

Figure 40 shows the definition of the where() clause for the SelectAll operation.

Figure 40   SelectAll Input

Figure 41 shows the output for the SelectAll operation. For each row selected during the operation, the employee number and last name are concatenated and passed out as text using the FileClient.write.

Figure 41 SelectAll Output
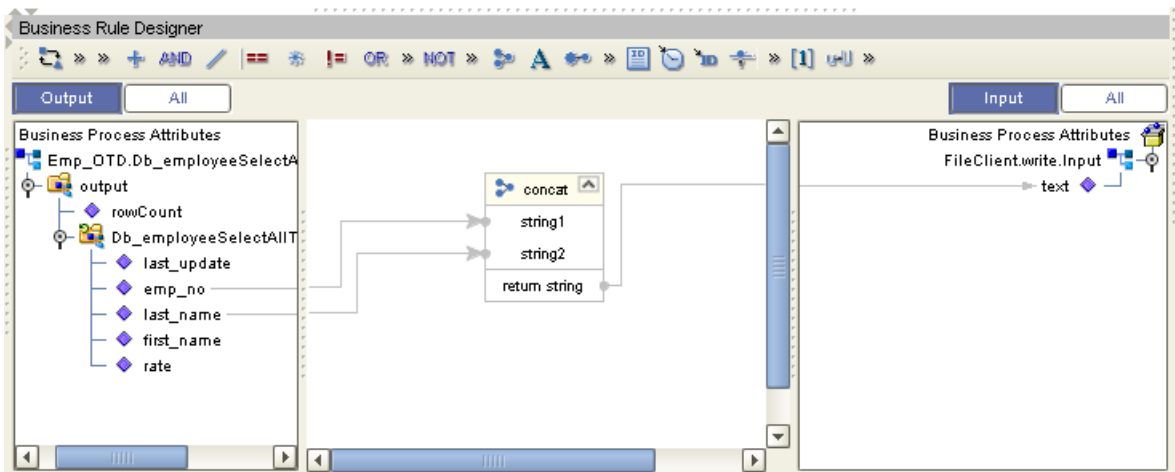
6.8.2 SelectMultiple

The input to a SelectMultiple operation uses a where() clause to define the criteria for selecting, and returning a specific number of rows from the database.

Note: *In order to return multiple rows using the SelectMultiple function, the results of the function must be marshalled into an OTD using a repeat element.*

Figure 42 shows a sample eInsight Business Process using the SelectMultiple operation. In this process, the SelectMultiple operation returns three rows specified in the where() clause.

Figure 42 SelectMultiple Sample Business Process

Figure 43 shows three rows requested using the where() clause. In this example, we are requesting everything from the table where emp_no is equal to 103, 109, and 117.
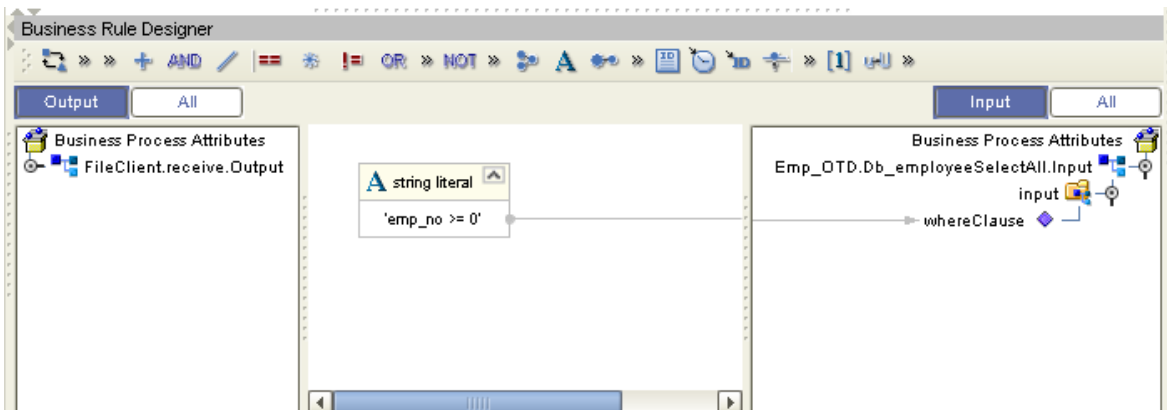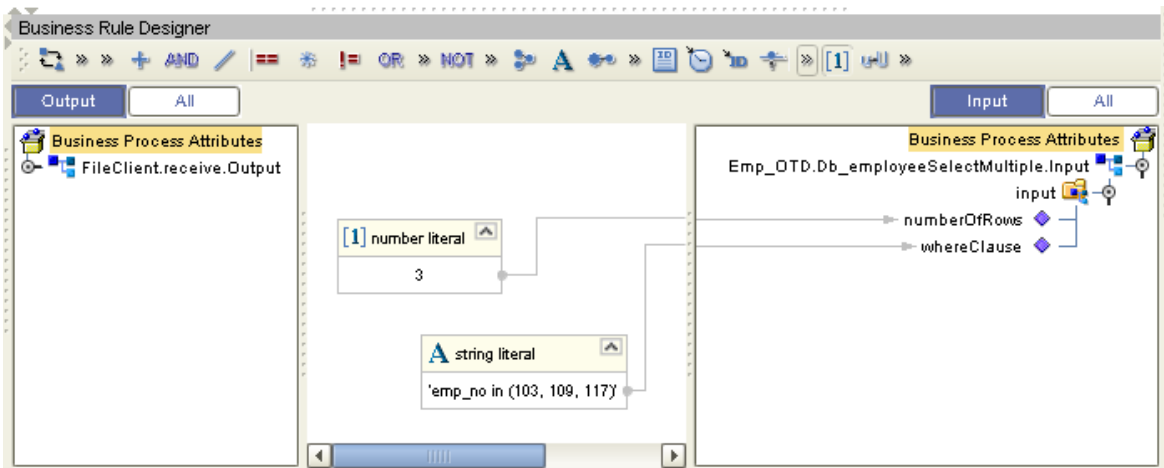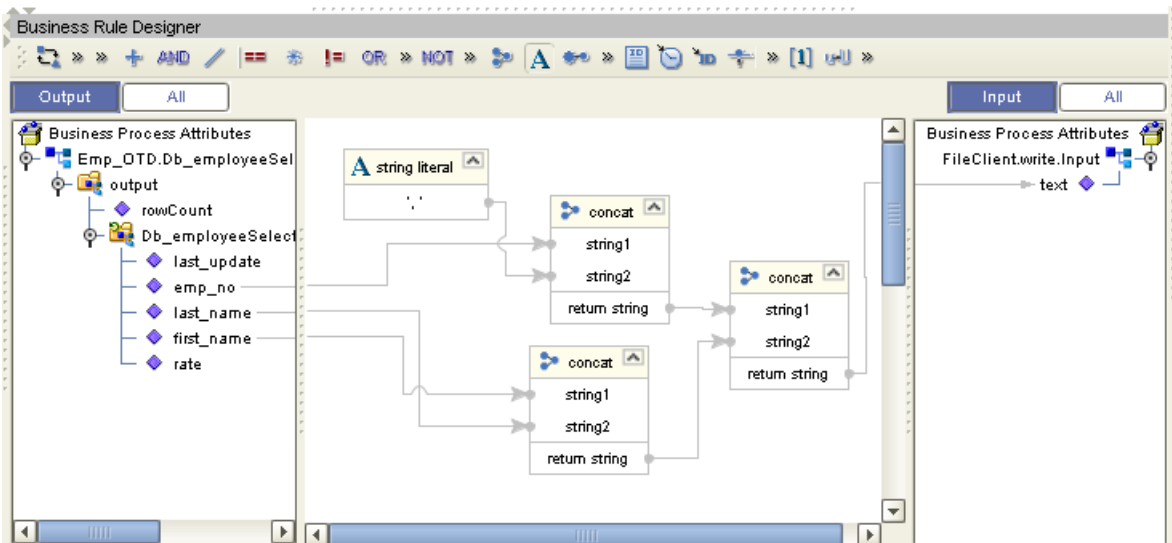
Figure 43 SelectMultiple Input

Figure 44 shows the output for the SelectMultiple operation. For each row selected during the operation, the employee number, first and last name are returned as text using the FileClient.write.

Figure 44 SelectMultiple Output

6.9 Using the Sample Projects in eGate

To import the **JDBC_JCE_Sample** eGate sample Project, follow the instructions given in **Locating and Importing the Sample Projects** on page 46.

6.9.1 Working with the Sample Projects in eGate

Input data for the sample Project first passes into a Collaboration, which then interacts with the external database to insert, update, delete, or select information in the db_employee table. Resulting data then passes to an output file for verification.

Figure 45 Database project flow
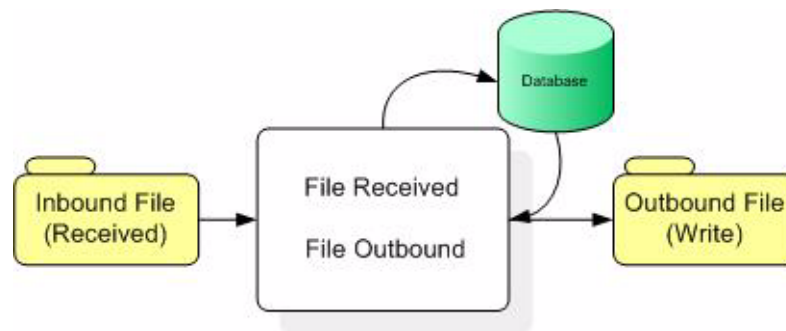
Additional instructions on working with sample Projects are provided in the *eGate Integrator Tutorial*.

Note: Outbound database eWays are available when using a JCE Collaboration. To poll the database, you must use the Scheduler or the File eWay.

JDBC_JCE_Sample

The **JDBC_JCE_Sample** sample Project uses input files to pass data into Collaborations. There are four Collaborations that demonstrate the Insert, Update, Delete, and Select operations, and one Collaboration to demonstrate a Prepared Statement. Results are written out to an output file and the input files are contained in the sample Project folder.

Collaborations used in the sample Project include:

jceTableInsert – inserts a new row for a "John Smith" that contains a rate of "125" and time stamp of "2004-02-06 11:50:00".

jceTableUpdate – updates any record with a last name like "Smith" to have a last name of "Doe" and a new rate of "33.88".

jceTableDelete – deletes the table db_employee.

jceTableSelect – selects all employees with an employee number greater than "100" and writes the results of the employee number and last name to a text file.

jcePreparedStatement – uses a prepared statement to select all records from the db_employee table, and write the last names to a text file.

6.10 Sample of Supported Data Types

The following is an example of some possible data types. Check your driver documentation for more information:

Table 2 Sample of Standard Data Types Supported by the eWay

Data Type	Description	Column Length
Number	Variable-length numeric data. Maximum precision p and/or scale s is 38.	Variable for each row. The maximum space required for a given column is 21 bytes per row.
VarChar2	Variable-length character data, with maximum length size bytes or characters.	Variable for each row, up to 4000 bytes per row. Consider the character set (single-byte or multibyte) before setting size. A maximum size must be specified.
Char	Fixed-length character data of length size bytes or characters.	Fixed for every row in the table (with trailing blanks); maximum size is 2000 bytes per row, default size is 1 byte per row. Consider the character set (single-byte or multibyte) before setting size.
Raw	Variable-length raw binary data.	Variable for each row in the table, up to 2000 bytes per row. A maximum size must be specified. Provided for backward compatibility.
Long	Variable-length character data.	Variable for each row in the table, up to 2^{32} - 1 bytes, or 2 gigabytes, per row. Provided for backward compatibility.
Clob		Up to 2^{32} - 1 bytes, or 64k.

Data Type	Description	Column Length
Date	Fixed-length date and time data, ranging from Jan. 1, 4712 B.C.E. to Dec. 31, 4712 C.E.	Fixed at 7 bytes for each row in the table. Default format is a string (such as DD-MON-RR) specified by the NLS_DATE_FORMAT parameter. Oracle expects a format of: "YYYY-MM-DD; hh:mm:ss.x".

6.11 Converting Sample Data Types in the eWay

When working with data in the JDBC eWay OTD's, you may need to do a data conversion. The following tables is an example of sample conversions.:

Table 3   Insert and Update Operations Sample Datatype Conversions (Text/String input data)

Data Type	OTD/Java Data Type	Java Method or New Constructor to Use (Default: Java Method)	Sample Data
Int	BigDecimal	**Call a New Constructor BigDecimal**: java.math.BigDecimal(String)	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)	147.78
Real	Double	**Double**: java.lang.Double.parseDouble(String)	147.78
Float	Double	**Double**: java.lang.Double.parseDouble(String)	147.78
Double	Double	**Double**: java.lang.Double.parseDouble(String)	147.78
Date	TimeStamp	**TimeStamp**: java.sql.TimeStamp.valueOf(String)	2003-08-11 11:47:39.0
Varchar2	String	Direct Assign	Any character
Char	String	Direct Assign	Any character

Data Type	OTD/Java Data Type	Java Method or New Constructor to Use (Default: Java Method)	Sample Data
Long Char	String	Direct Assign	Any character
Raw	Byte[]	**String**: java.lang.String.getBytes()	Any character
Long Raw	Byte[]	**String**: java.lang.String.getBytes()	Any character
CLOB	Clob	See Appendix	Any character

Table 4 Sample Select Operation Datatype Conversion (Text/String output data)

Data Type	OTD/Java Data Type	Methods To Use	Sample Data
Int	BigDecimal	**BigDecimal**: java.math.toString()	123
SmallInt	BigDecimal	**BigDecimal**: java.math.toString()	123
Number	BigDecimal	**BigDecimal**: java.math.toString()	123
Decimal	BigDecimal	**BigDecimal**: java.math.toString()	123.67
Real	Double	**Double**: java.lang.Double.toString(double)	123
Float	Double	**Double**: java.lang.Double.toString(double)	123
Double	Double	**Double**: java.lang.Double.parseDouble(String)	123
Date	TimeStamp	**TimeStamp**: java.sql.TimeStamp.valueof()	2003-08-11 11:47:39.0
Varchar2	String	Direct Assign	Any characters
Char	String	Direct Assign	Any Characters
Raw	Byte[]	N/A	N/A
Long Raw	Byte[]	N/A	N/A
CLOB	Clob	N/A	N/A

6.12 Using OTDs with Tables, Views, and Stored Procedures, and Prepared Statements

Tables, Views, and Stored Procedures, and Prepared Statements are manipulated through OTDs. Common operations include insert, delete, update, and query.

Note: *Some drivers do not support updatable ResultSets. Use a Prepared Statement instead.*

6.12.1 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.

By default, the Table OTD has UpdatableConcurrency and ScrollTypeForwardOnly. The type of result returned by the select() method can be specified using:

- SetConcurrencytoUpdatable
- SetConcurrencytoReadOnly
- SetScrollTypetoForwardOnly
- SetScrollTypetoScrollSensitive
- SetScrollTypetoInsensitive

The methods should be called before executing the select() method. For example,

```
getDBEmp().setConcurToUpdatable();
getDBEmp().setScroll_TypeToScrollInsensitive();
getDBEmp().getDB_EMPLOYEE().select("");
```

Note: *The above select() method is driver dependent and may not work with all drivers.*

The Query Operation

To perform a query operation on a table

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

2 Loop through the ResultSet using the **next()** method.

3 Process the return record within a **while()** loop.

For example:

```
package SelectSales;
public class Select
{

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,db_employee.Db_employeeOTD
      db_employee_1,employeedb.Db_employee employeedb_db_employee_1 )
      throws Throwable
        {
      //@map:Db_employee.select(Text)
        db_employee_1.getDb_employee().select( input.getText() );
      //while
        while (db_employee_1.getDb_employee().next()) {
      //@map:Copy EMP_NO to Employee_no
        employeedb_db_employee_1.setEmployee_no(
      java.lang.Integer.toString(
      db_employee_1.getDb_employee().getEMP_NO() ) );
      //@map:Copy LAST_NAME to Employee_lname
        employeedb_db_employee_1.setEmployee_lname(
      db_employee_1.getDb_employee().getLAST_NAME() );
      //@map:Copy FIRST_NAME to Employee_fname
        employeedb_db_employee_1.setEmployee_fname(
      db_employee_1.getDb_employee().getFIRST_NAME() );
      //@map:Copy RATE to Rate
        employeedb_db_employee_1.setRate( java.lang.Double.toString(
      db_employee_1.getDb_employee().getRATE() ) );
      //@map:Copy LAST_UPDATE to Update_date
        employeedb_db_employee_1.setUpdate_date(
      db_employee_1.getDb_employee().getLAST_UPDATE().toString() );
      }
      //@map:Copy employeedb_db_employee_1.marshalToString to Text
        FileClient_1.setText(
        employeedb_db_employee_1.marshalToString() );
      //@map:FileClient_1.write
        FileClient_1.write();
        }

        }
```

The Insert Operation

To perform an insert operation on a table

  1 Execute the **insert()** method. Assign a field.

  2 Insert the row by calling **insertRow()**

  This example inserts an employee record.

```
 //DB_EMPLOYEE.insert
     Table_OTD_1.getDB_EMPLOYEE().insert();
 //Copy EMP_NO to EMP_NO
     insert_DB_1.getInsert_new_employee().setEmployee_no(
     java.lang.Integer.parseInt(
       employeedb_with_top_db_employee_1.getEmployee_no() ) );

 //@map:Copy Employee_lname to Employee_Lname
     insert_DB_1.getInsert_new_employee().setEmployee_Lname(
     employeedb_with_top_db_employee_1.getEmployee_lname() );

 //@map:Copy Employee_fname to Employee_Fname
     insert_DB_1.getInsert_new_employee().setEmployee_Fname(
     employeedb_with_top_db_employee_1.getEmployee_fname() );

 //@map:Copy java.lang.Float.parseFloat(Rate) to Rate
     insert_DB_1.getInsert_new_employee().setRate(
     java.lang.Float.parseFloat(
     employeedb_with_top_db_employee_1.getRate() ) );
```

```
//@map:Copy java.sql.Timestamp.valueOf(Update_date) to Update_date
   insert_DB_1.getInsert_new_employee().setUpdate_date(
   java.sql.Timestamp.valueOf(
   employeedb_with_top_db_employee_1.getUpdate_date() ) );
   Table_OTD_1.getDB_EMPLOYEE().insertRow();

}
```

The Update Operation

To perform an update operation on a table

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

  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()**.

```
//SalesOrders_with_top_SalesOrders_1.unmarshalFromString(Text)
   SalesOrders_with_top_SalesOrders_1.unmarshalFromString(
   input.getText() );

//SALES_ORDERS.update("SO_num =99")
   DB_sales_orders_1.getSALES_ORDERS().update( "SO_num ='01'" );

//while
   while (DB_sales_orders_1.getSALES_ORDERS().next()) {

//Copy SalesOrderNum to SO_num
   DB_sales_orders_1.getSALES_ORDERS().setSO_num(
   SalesOrders_with_top_SalesOrders_1.getSalesOrderNum() );

//Copy CustomerName to Cust_name
   DB_sales_orders_1.getSALES_ORDERS().setCust_name(
   SalesOrders_with_top_SalesOrders_1.getCustomerName() );

//Copy CustomerPhone to Cust_phone
   DB_sales_orders_1.getSALES_ORDERS().setCust_phone(
   SalesOrders_with_top_SalesOrders_1.getCustomerPhone() );

//SALES_ORDERS.updateRow
   DB_sales_orders_1.getSALES_ORDERS().updateRow();
}

//Copy "Update completed" to Text
   FileClient_1.setText( "Update completed" );

//FileClient_1.write
   FileClient_1.write();
}
```

The Delete Operation

To perform a delete operation on a table

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

  2 Move to the row that you want to delete.

  3 Delete the row by calling **deleteRow()**.

In this example DELETE an employee.

```
//DB_EMPLOYEE.delete("EMP_NO = '".concat(EMP_NO).concat("'"))
  Table_OTD_1.getDB_EMPLOYEE().delete( "EMP_NO = '".concat(
  employeedb_with_top_db_employee_1.getEMP_NO() ).concat( "'" ) );
}
```

6.12.2 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. These allow 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
    {
        //
@map:employeedb_with_top_db_employee_1.unmarshalFromString(Text)
        employeedb_with_top_db_employee_1.unmarshalFromString(
input.getText() );

        //@map:Copy java.lang.Integer.parseInt(Employee_no) to
Employee_no
```

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

            //@map:Copy Employee_lname to Employee_Lname
            insert_DB_1.getInsert_new_employee().setEmployee_Lname(
    employeedb_with_top_db_employee_1.getEmployee_lname() );

            //@map:Copy Employee_fname to Employee_Fname
            insert_DB_1.getInsert_new_employee().setEmployee_Fname(
    employeedb_with_top_db_employee_1.getEmployee_fname() );

            //@map:Copy java.lang.Float.parseFloat(Rate) to Rate
            insert_DB_1.getInsert_new_employee().setRate(
    java.lang.Float.parseFloat(
    employeedb_with_top_db_employee_1.getRate() ) );

            //@map:Copy java.sql.Timestamp.valueOf(Update_date) to
    Update_date
            insert_DB_1.getInsert_new_employee().setUpdate_date(
    java.sql.Timestamp.valueOf(
    employeedb_with_top_db_employee_1.getUpdate_date() ) );

            //@map:Insert_new_employee.execute
            insert_DB_1.getInsert_new_employee().execute();

            //@map:insert_DB_1.commit
            insert_DB_1.commit();

            //@map:Copy "procedure executed" to Text
            FileClient_1.setText( "procedure executed" );

            //@map:FileClient_1.write
            FileClient_1.write();
        }

    }
```

6.12.3 Prepared Statement

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

You can use prepared statements 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.

To Insert

```
public void receive( com.stc.connector.appconn.file.FileTextMessage
input,com.stc.connector.appconn.file.FileApplication
FileClient_1,pS_Insert.PS_InsertOTD PS_Insert_1 )
    throws Throwable
    {
        //@map:Copy Text to Param1
        PS_Insert_1.getPSInsert().setParam1( input.getText() );
```

```
//@map:Copy "James" to Param2
PS_Insert_1.getPSInsert().setParam2( "James" );


//@map:Copy "M" to Param3
PS_Insert_1.getPSInsert().setParam3( "M" );


//@map:Copy "Smith" to Param4
PS_Insert_1.getPSInsert().setParam4( "Smith" );


//@map:Copy "HR" to Param5
PS_Insert_1.getPSInsert().setParam5( "HR" );


//@map:Copy "9995551212" to Param6
PS_Insert_1.getPSInsert().setParam6( "9995551212" );


//@map:PSInsert.executeUpdate
PS_Insert_1.getPSInsert().executeUpdate();


//@map:Copy "Record Inserted" to Text
FileClient_1.setText( "Record Inserted" );


//@map:FileClient_1.write
FileClient_1.write();
}

}
```

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

To Update

```
public void receive( com.stc.connector.appconn.file.FileTextMessage
input,com.stc.connector.appconn.file.FileApplication
FileClient_1,pS_Update.PS_UpdateOTD PS_Update_1 )
    throws Throwable
{
    //@map:Copy Text to Param1
    PS_Update_1.getPSUpdate().setParam1( input.getText() );


    //@map:Copy "6265551212" to Param2
    PS_Update_1.getPSUpdate().setParam2( "6265551212" );


    //@map:PSUpdate.executeUpdate
    PS_Update_1.getPSUpdate().executeUpdate();


    //@map:Copy "Record Updated" to Text
    FileClient_1.setText( "Record Updated" );


    //@map:FileClient_1.write
    FileClient_1.write();
}
```

```
    }
```

6.12.4 Batch Operations

While the Java API used by SeeBeyond does not support traditional bulk insert or update operations, there is an equivalent feature that can achieve comparable results, with better performance. 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();
```

6.13 Alerting and Logging

eGate provides an alerting and logging feature. This allows monitoring of messages and captures any adverse messages in order of severity based on configured severity level and higher. To enable Logging, please see the *eGate Integrator User's Guide*.

Index