SeeBeyond ICAN Suite

DB2 Connect eWay Intelligent Adapter User's Guide

Release 5.0.2

Version 20041119140626.

Contents

Chapter 1

Introduction

This document describes how to install and configure the DB2 Connect eWay.

This Chapter Includes:

1.1 Overview

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.*

1.2 Supported Operating Systems

The DB2 Connect eWay is available on the following operating systems:

- Windows 2000, Windows XP, and Windows Server 2003
- HP-UX 11.0, 11i (PA-RISC)
- IBM AIX 5.1L and 5.2
- Sun Solaris 8 and 9
- Red Hat Enterprise Linux AS 2.1 (Intel x86)
- Connecting to z/OS running DB2 Connect 7.1

Although the DB2 Connect eWay, the Repository, and Logical Hosts run on the operating systems listed above, the Enterprise Designer requires the Windows operating system. Enterprise Manager can run on any operating system that supports Internet Explorer.

1.3 System Requirements

The system requirements for the DB2 Connect eWay are the same as for eGate Integrator. For information, refer to the SeeBeyond *ICAN Suite Installation Guide*. It is also helpful to review the **Readme.txt** for any additional requirements prior to installation. The **Readme.txt** is located on the installation CD-ROM.

DB2 Connect is available from IBM. You must install the software prior to installing and configuring the DB2 Connect eWay.

Using IBM Drivers on Windows or Unix Operating Systems

- DB2 Connect available from IBM, installed on the same machine as the Enterprise Designer and Logical Host. This is required to utilize the build tool.

Note: *To enable Web Services, you must install and configure the SeeBeyond ICAN Suite eInsight Business Process Manager.*

Using IBM Drivers for Connecting to a z/OS Operating System

To use the DB2 Connect eWay connecting to an z/OS operating system, you must meet the System Requirements.

Note: *If you have chosen to use the IBM drivers for DB2, please be aware that these drivers do not support updatable ResultSets. It is recommended that you use a prepared statement to update and insert data.*

1.4 External System Requirements

The DB2 Connect eWay supports the following software on external systems:

- DB2 Connect version 8.1.
- DB2 Universal Database (UDB) version 7.1 when connecting to DB2 running on an z/OS operating system.
- DB2 Universal Database (UDB) version V5R1 when connecting to DB2 running on an AS/400 operating system.

Chapter 2

Installation

This chapter describes how to install the DB2 Connect eWay.

This Chapter Includes:

2.1 Installing the DB2 Connect 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 **DB2ConnecteWay.sar** (to install the DB2 Connect 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 **DB2ConnecteWayDocs.sar** (to install the documentation and the sample) file to upload.
- On the Enterprise Manager under the Documentation tab, click on 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.

Continue installing the eGate Enterprise Designer as instructed.

Note: *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.*

2.2 After Installation

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

To Create the .jar file

1. Locate the **db2java.zip** file in your DB2 Connect installation directory.
2. Copy the zip file to:
   `<egate_logicalhost>:\\ICAN50\edesigner\userdir\modules\ext\db2connectadapter`
3. Rename the zip file from **db2java.zip** to **db2java.jar**.

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

Once the .jar file is created, you can load it in a number of ways, including:

- Loading the file using Enterprise Designer
- Copying the file directly into the Logical Host

To Load the .jar file Using Enterprise Designer

When using ICAN, do the following:

1. On the Environment tab, create a new Logical Host in your DB2 Environment.
2. Right-click the new Logical Host and select **Upload File** on the menu.
3. On the Upload Third Party Files window, click **Add**. Navigate to the **db2java.jar** you copied to:

   ```
   <egate_logicalhost>:\\ICAN50\edesigner\userdir\modules\ext\db2connectadapter
   ```
4. Click **OK**.
5. Exit and then restart Enterprise Designer.

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.

Note: *Failing to follow these instructions results in the eWay not recognizing your DB2 database.*

To Copy the .jar file Directly into the Logical Host

1. Copy the file to the following location and rename it from **db2java.zip** to **db2java.jar**:

   ```
   <egate_logicalhost>:\\stcis\lib.
   ```

Note: *The <egate_logicalhost>:\\stcis\lib location only appears after running a project.*

Chapter 3

Properties of the DB2 Connect eWay

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

This Chapter Includes:

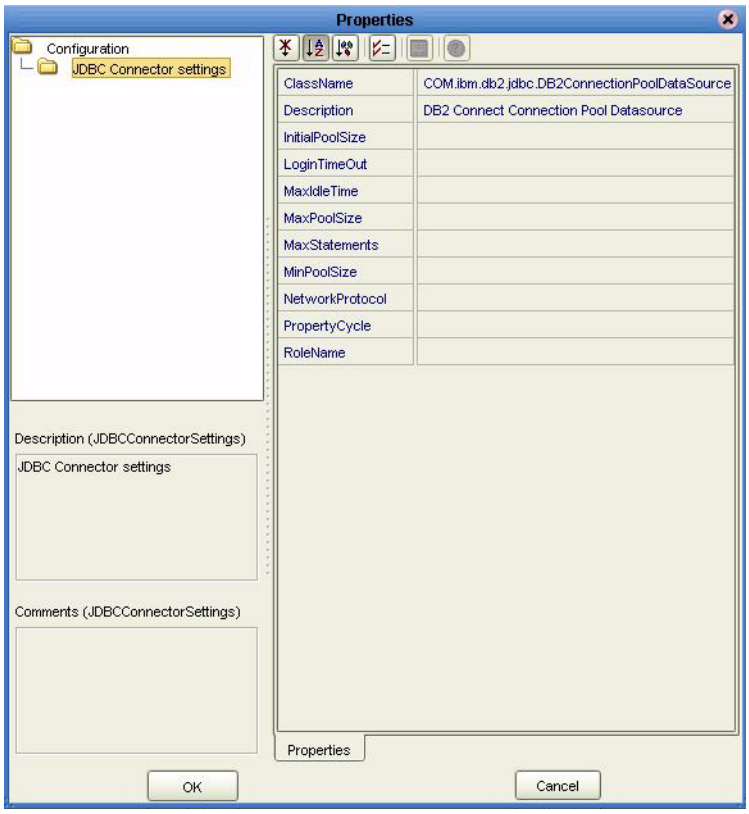
3.1 Working with eWay Property Sheets

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.

3.1.1. Setting the Properties in the Outbound eWay

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

Figure 1 The eWay Properties



ClassName

Description

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

Required Values

A valid class name.

The default is **COM.ibm.db2jdbc.DB2ConnectionPoolDataSource**.

Description

Description

Enter a description for the database.

Required Value

A valid string.

InitialPoolSize

Description

Enter a number for the physical connections the pool should contain when it is created.

Required Value

A valid numeric value.

LoginTimeOut

Description

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

Required Value

A valid numeric value.

MaxIdleTime

Description

The maximum number of seconds that a physical connection may remain unused before it is closed. 0 (zero) indicates that there is no limit.

Required Value

A valid numeric value.

MaxPoolSize

Description

The maximum number of physical connections the pool should keep available at all times. 0 (zero) indicates that there is no maximum.

Required Value

A valid numeric value.

MaxStatements

Description

The maximum total number of statements that the pool should keep open. 0 (zero) indicates that the caching of statements is disabled.

Required Value

A valid numeric value.

MinPoolSize

The minimum number of physical connections the pool should keep available at all times. 0 (zero) indicates that there should be no physical connections in the pool and the new connections should be created as needed.

Required Value

A valid numeric value.

NetworkProtocol

Description

The network protocol used to communicate with the server.

Required Values

Any valid string.

PropertyCycle

Description

The interval, in seconds, that the pool should wait before enforcing the current policy defined by the values of the other connection pool properties in this deployment descriptor.

Required Values

A valid numeric value.

RoleName

Description

An initial SQL role name.

Required Values

Any valid string.

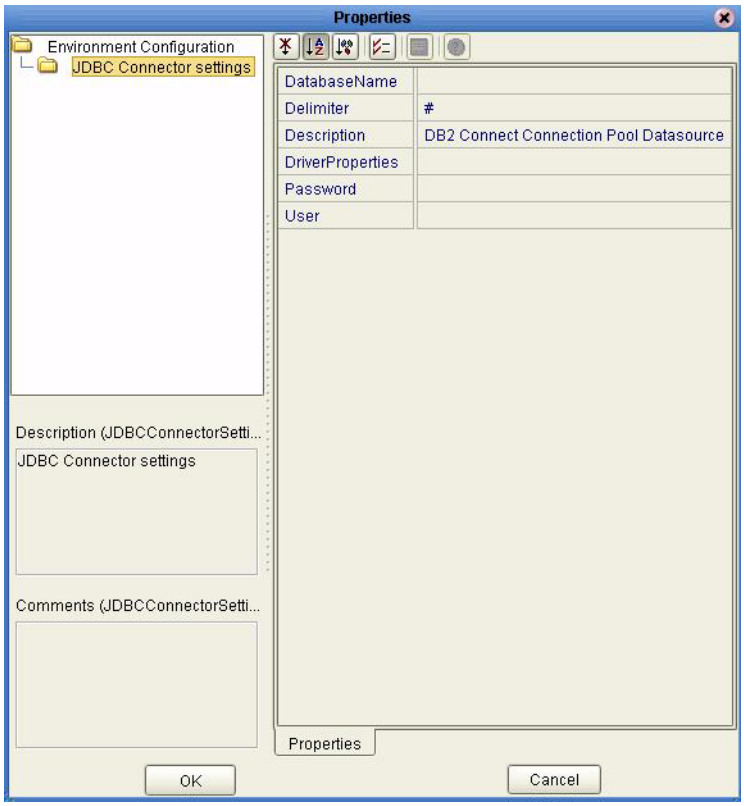
Before deploying your eWay, you will need to set the properties of the eWay environment using the following descriptions.

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

3.1.2 Setting the Environment Properties in the Outbound eWay

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

Figure 2 Environment Settings of the Outbound DB2 Connect eWay on Windows and Unix



DatabaseName

Description

Specifies the name of the database instance as configured on the local client.

Required Values

Any valid string.

Delimiter

Description

This is the delimiter character to be used in the DriverProperties prompt.

Required Value

The default is #

Description

Description

Enter a description for the database.

Required Value

A valid string.

DriverProperties

Description

If you choose to not to 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 methods to assure a connection. The additional methods will need to be identified in the Driver Properties.

Required Value

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 execute the method setURL, give the method a String for the URL "setURL#<url>##".

Password

Description

Specifies the password used to access the database.

Required Values

Any valid string.

User

Description

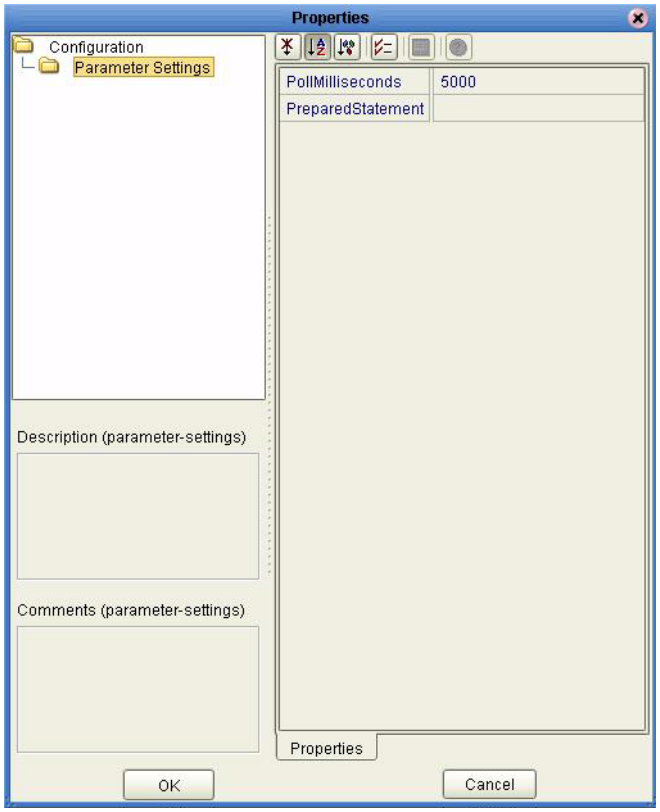
Specifies the user name the eWay uses to connect to the database.

Required Values

Any valid string.

3.1.3. Setting the Properties in the Inbound eWay

Figure 3 Properties of the Inbound eWay



Pollmilliseconds

Description

Polling interval in milliseconds.

Required Value

A valid numeric value. The default is 5000.

PreparedStatement

Description

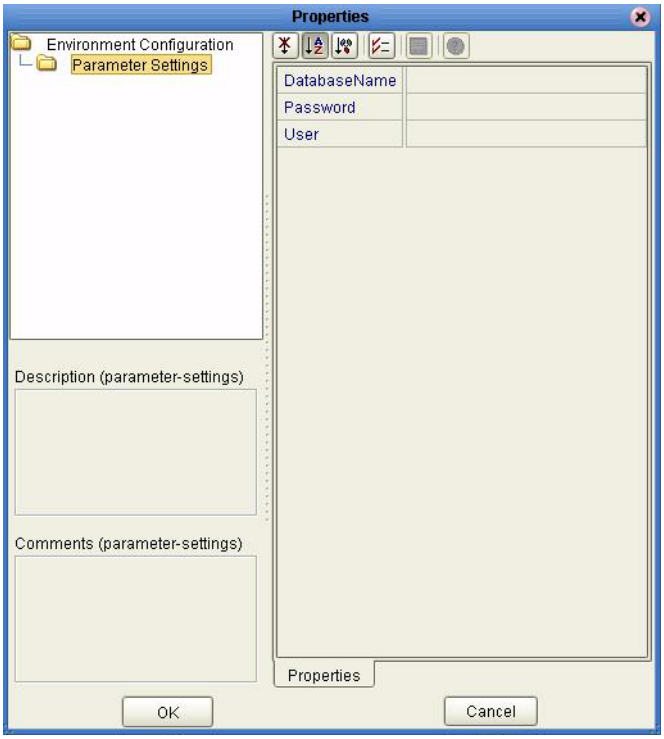
Prepared Statement used for polling against the database.

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.

3.1.4. Setting the Environment Properties in the Inbound eWay

Figure 4 Inbound eWay Environment



DatabaseName

Description

Specifies the name of the database instance.

Required Values

Any valid string.

Password

Description

Specifies the password used to access the database.

Required Values

Any valid string.

User

Description

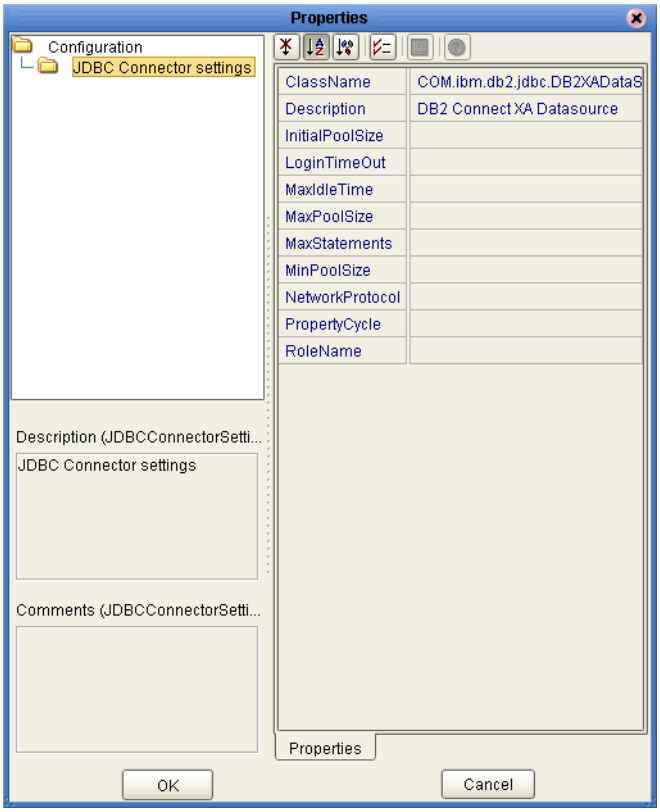
Specifies the user name the eWay uses to connect to the database.

Required Values

Any valid string.

3.1.5. Setting the Properties in the Outbound DB2 XA eWay

Figure 5 Properties of the Outbound XA eWay



ClassName

Description

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

Required Values

A valid class name.

The default is **COM.ibm.db2.jdbc.DB2XADataSource**.

Description

Description

Enter a description for the database. The default is **DB2 Connect XA Datasource**.

Required Value

A valid string.

InitialPoolSize

Description

Enter a number for the physical connections the pool should contain when it is created.

Required Value

A valid numeric value.

LoginTimeOut

Description

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

Required Value

A valid numeric value.

MaxIdleTime

Description

The maximum number of seconds that a physical connection may remain unused before it is closed. 0 (zero) indicates that there is no limit.

Required Value

A valid numeric value.

MaxPoolSize

Description

The maximum number of physical connections the pool should keep available at all times. 0 (zero) indicates that there is no maximum.

Required Value

A valid numeric value.

MaxStatements

Description

The maximum total number of statements that the pool should keep open. 0 (zero) indicates that the caching of statements is disabled.

Required Value

A valid numeric value.

MinPoolSize

Description

The minimum number of physical connections the pool should keep available at all times. 0 (zero) indicates that there should be no physical connections in the pool and the new connections should be created as needed.

Required Value

A valid numeric value.

NetworkProtocol

Description

The network protocol used to communicate with the server.

Required Values

Any valid string.

PropertyCycle

Description

The interval, in seconds, that the pool should wait before enforcing the current policy defined by the values of the other connection pool properties in this deployment descriptor.

Required Values

A valid numeric value.

RoleName

Description

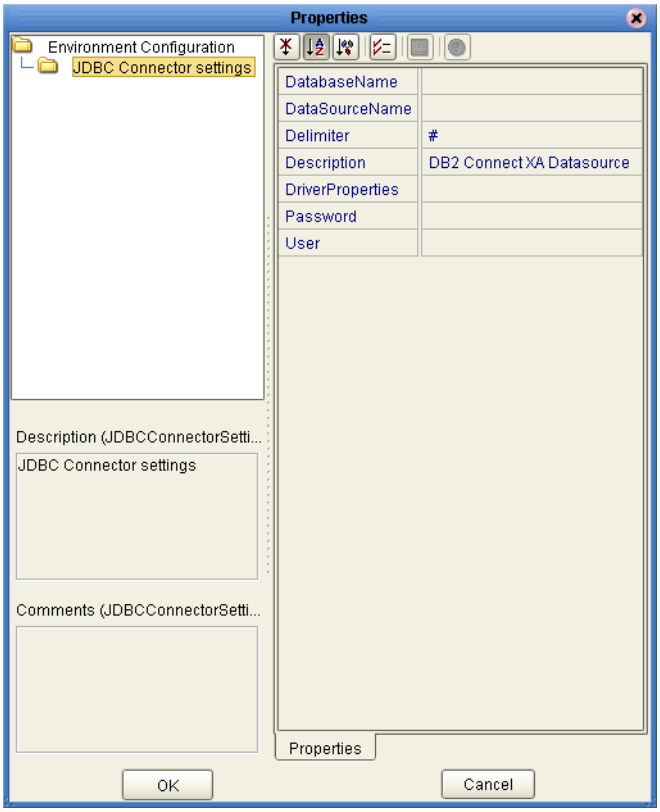
An initial SQL role name.

Required Values

Any valid string.

3.1.6. Setting the Environment Properties in the Outbound DB2 XA eWay

Figure 6 Inbound eWay Environment



DatabaseName

Description

Specifies the name of the database instance.

Required Values

Any valid string.

DataSourceName

Description

Provide the name of the ConnectionPoolDataSource object that the DataSource object delegates behind the scenes when connection pooling or distributed transaction management is being done.

Required Value

Optional. In most cases, leave this box empty.

Delimiter

Description

This is the delimiter character to be used in the DriverProperties prompt.

Required Value

The default is **#.**

Description

Description

Enter a description for the database. The default is **DB2 Connect XA Datasource**.

Required Value

A valid string.

DriverProperties

Description

If you choose to not to 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

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 execute the method setURL, give the method a String for the URL “setURL#jdbc:db2:<database>##”.

Password

Description

Specifies the password used to access the database.

Required Values

Any valid string.

User

Description

Specifies the user name the eWay uses to connect to the database.

Required Values

Any valid string.

Chapter 4

Using the DB2 Connect eWay Database Wizard

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

This Chapter Includes:

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.

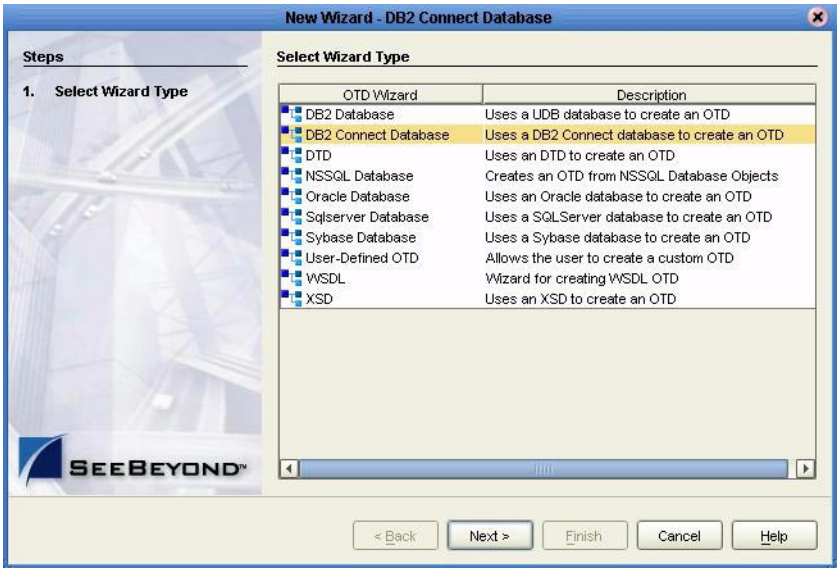
Note: *Database OTDs are not messagable. For more information on messagable OTDs, see the eGate Integrator User's Guide.*

Select Wizard Type

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

2 From the OTD Wizard Selection window, select the **DB2 Connect Database** and click **Next**. See **Figure 7**.

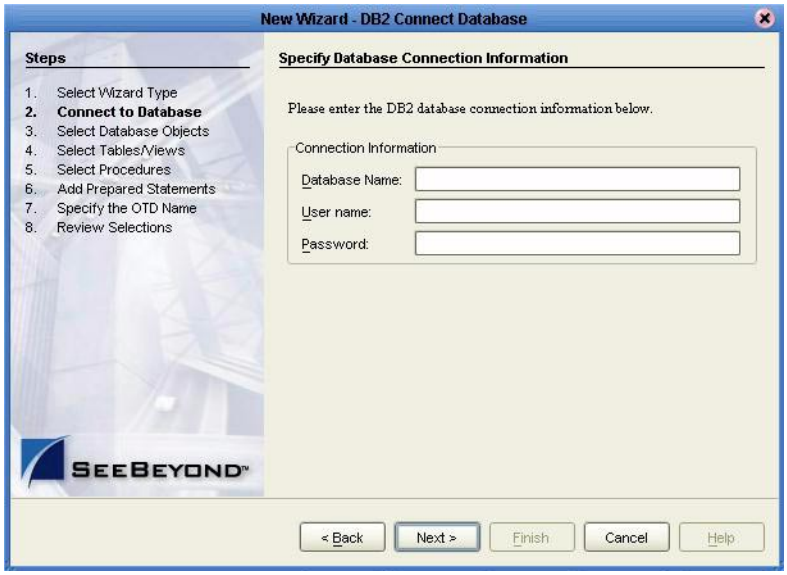
Figure 7 OTD Wizard Selection



Connect to Database

3 Specify the **Database Name** that you configured in DB2 Connect**.** Enter a **UserName** and **Password** and click **Next**. See **Figure 8**.

Figure 8 Database Connection Information

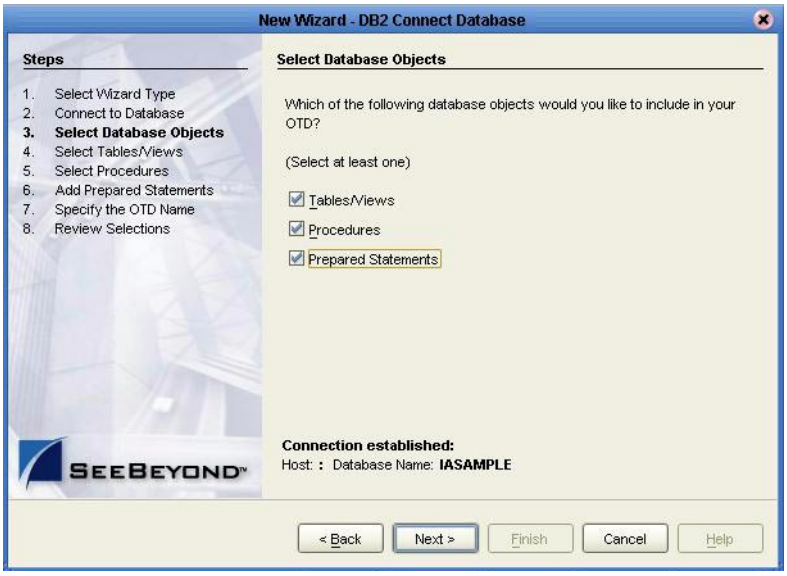


Select Database Objects

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 9**.

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

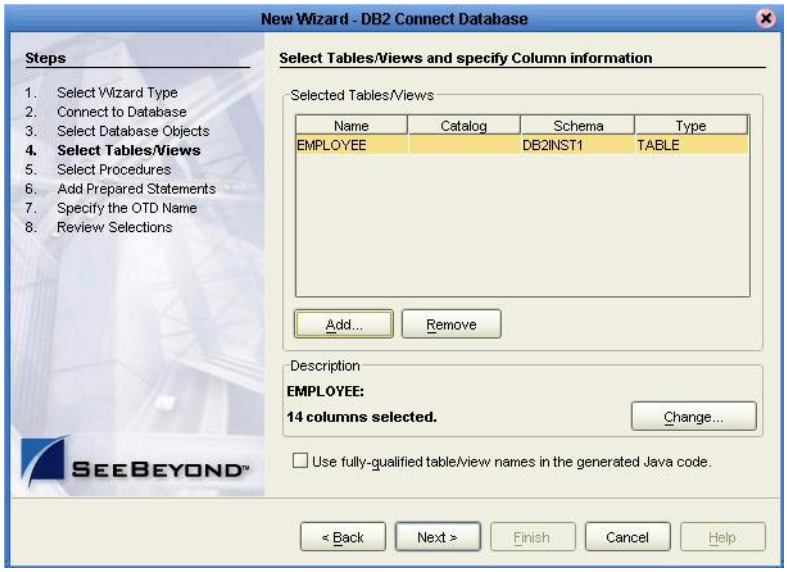
Figure 9 Select Database Objects



Select Table/Views

1 In the **Select Tables/Views** window, click **Add**. See **Figure 10**.

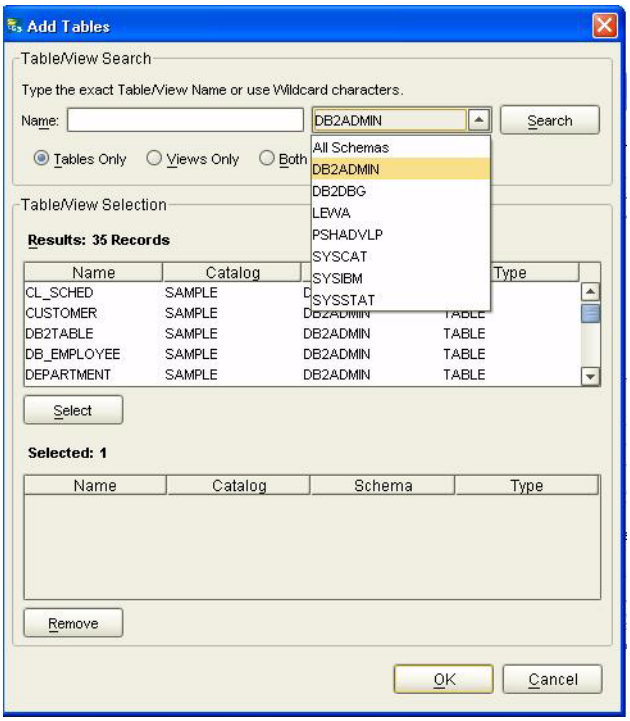
Figure 10 Select Tables/Views



2 In the **Add Tables** window, select if 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 11**. 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".

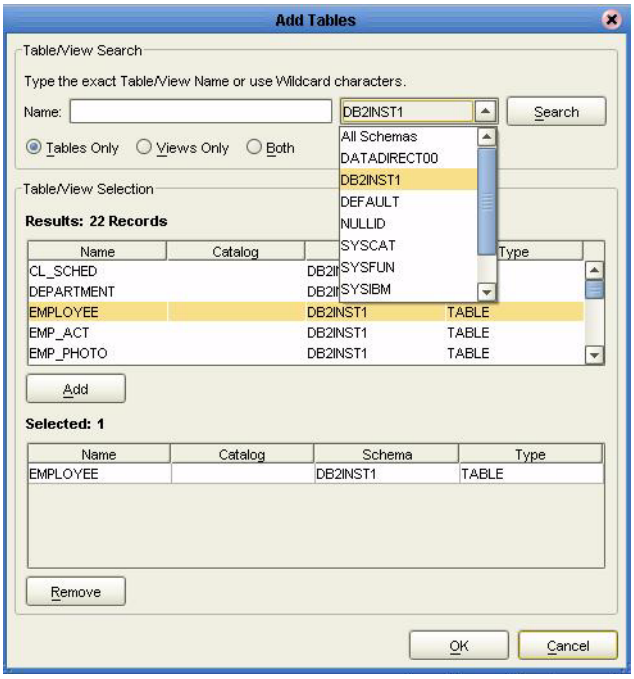
Figure 11 Database Wizard - All Schemes



4 Select the table of choice and click **OK**.

The table selected is added to the **Selected** window. See **Figure 12**.

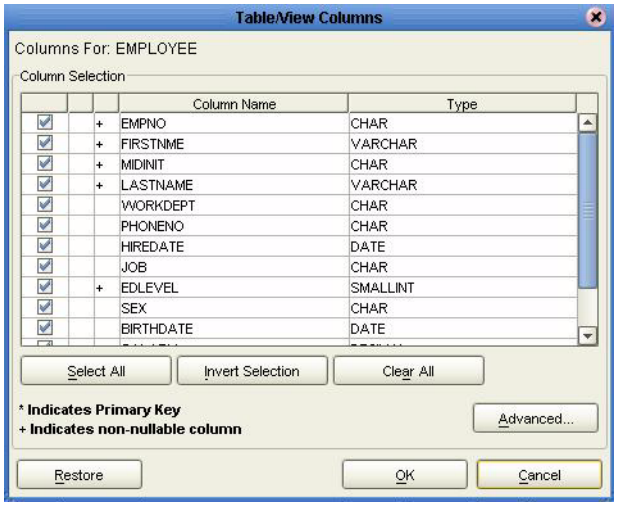
Figure 12 Selected Tables/Views window with a table selected



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.

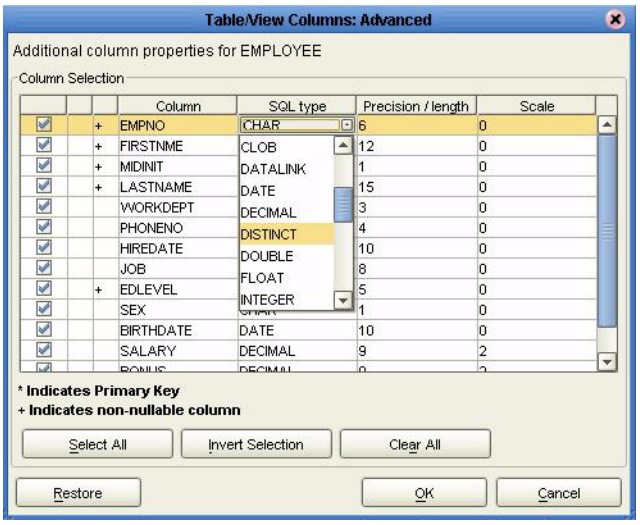
6 If you clicked **Change** on the Selected Tables/Views window, you can select or deselect your table columns on the **Table/View Columns** window. You can also change the data type for each table by highlighting the data type and selecting a different one from the drop down. See **Figure 13**.

Figure 13 Table/View Columns



7 Click **Advanced** to change the data type, percision/length, or scale. In general, do not change the percision/length or the scale. Once you have finished your table choices, click **OK**. See **Figure 14**.

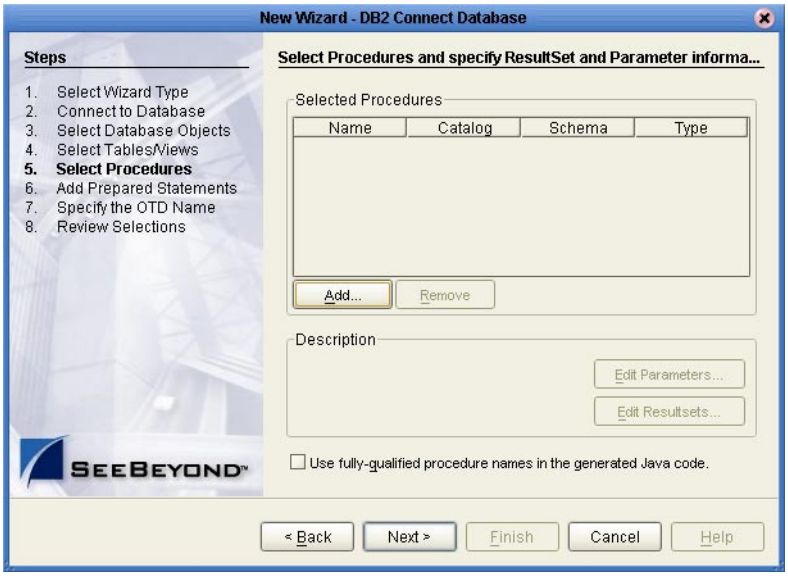
Figure 14 Table/View Columns — Advanced



Select Procedures

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

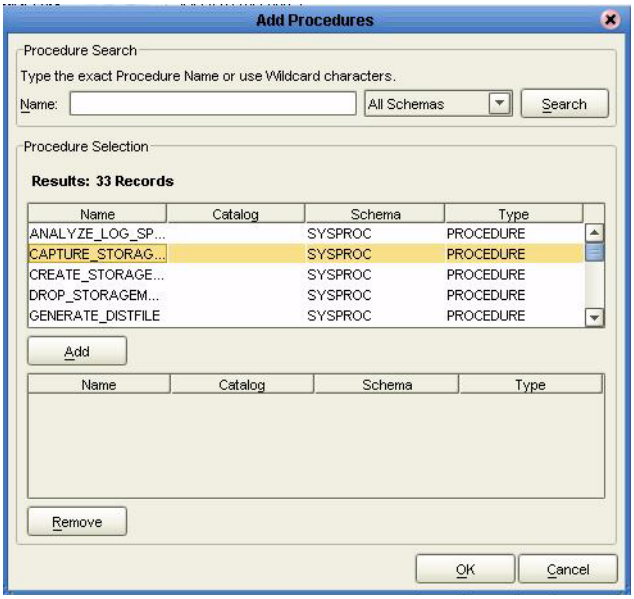
Figure 15 Select Procedures and specify Resultset and Parameter Information



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.

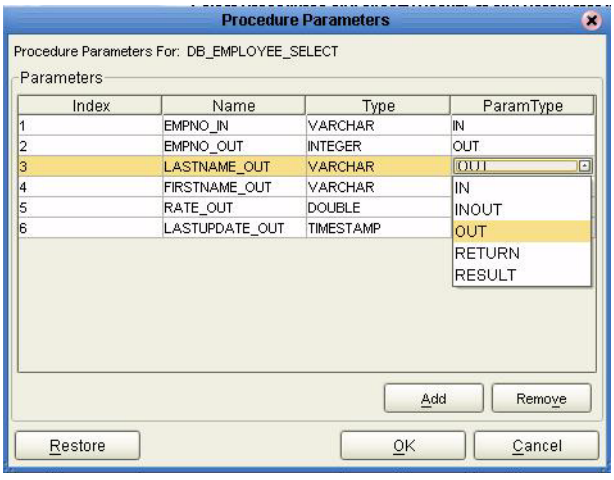
3 In the resulting **Procedure Selection** list box, select a Procedure. Click **OK**.

Figure 16 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 17**.

Figure 17 Procedure Parameters

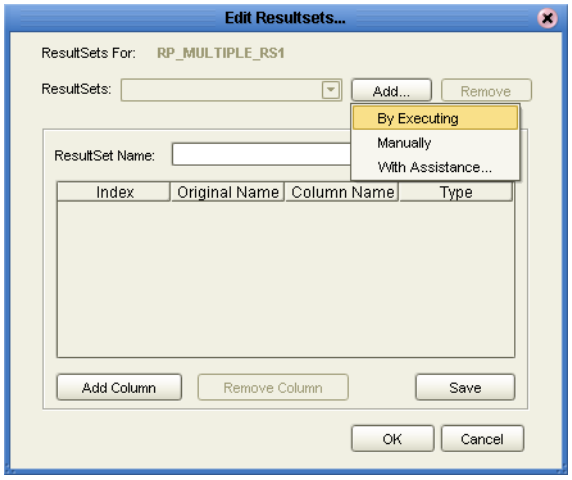


5 To restore the data type, click **Restore**. When finished, click **OK**.

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.

Figure 18 Edit Resultset



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

"**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 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 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.

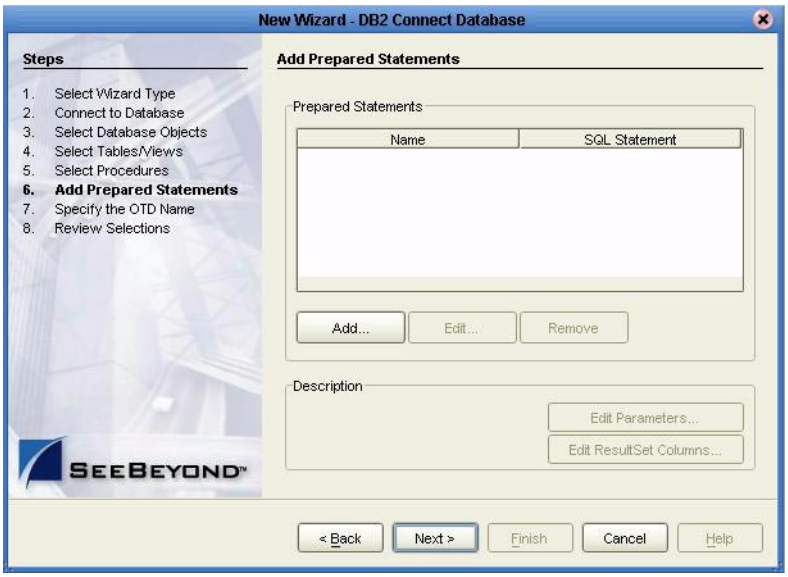
If you modify the ResultSet generated by the "**Execute**" mode of the Database Wizard you need to make sure the indexes match the Stored Procedure. This assures your ResultSet indexes are preserved.

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

Add Prepared Statements

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

Figure 19 Prepared Statement



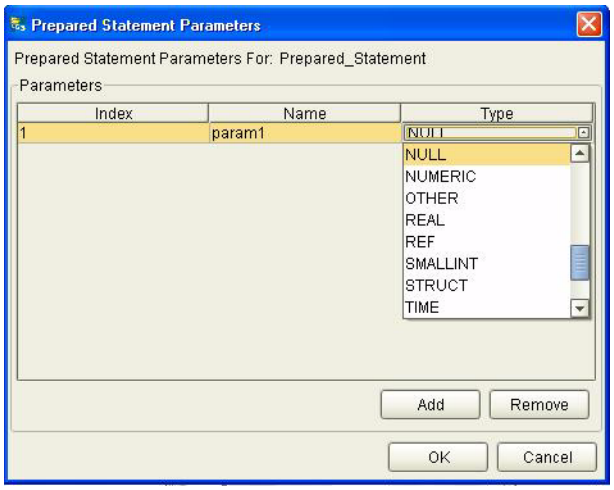
2 Enter the name of a Prepared Statement or create a SQL statement by clicking in the SQL Statement window. When finished creating the statement, click **Save As** giving the statement a name. This name will appear as a node in the OTD. Click **OK**. See **Figure 20**.

Figure 20 Prepared SQL Statement

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 datatype by clicking in the **Type** field and selecting a different type from the list.

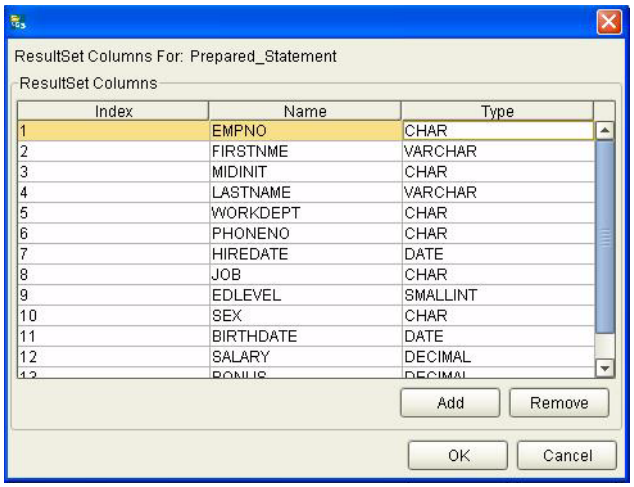
4 Click **Add** if you want to add additional parameters to the Statement or highlight a row and click **Remove** to remove it. Click **OK**. See **Figure 21**.

Figure 21 Edit the Prepared Statement Parameters



1 To edit the Resultset Columns, click **Edit Resultset Columns**. Both the Name and Type are editable. Click **OK**. See **Figure 22**.

Figure 22 ResultSet Columns

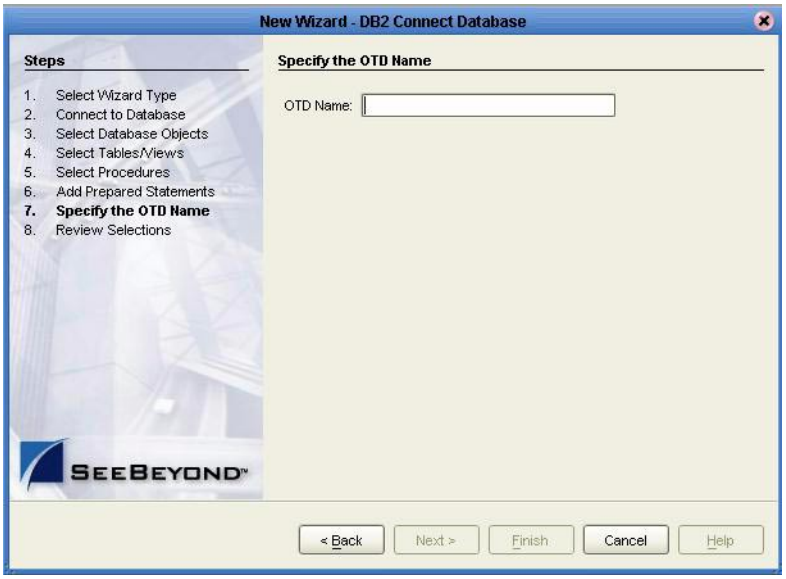


2 On the Add Prepared Statements window, click **OK**.

Specify the OTD Name

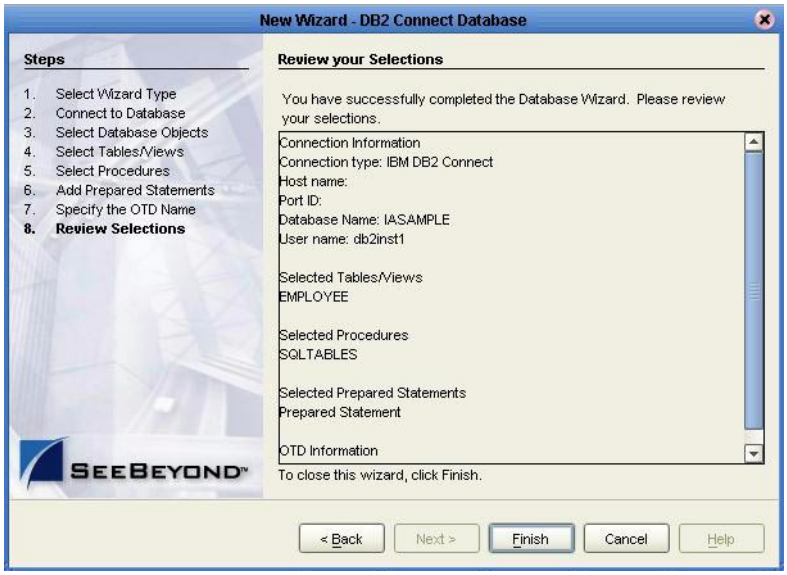
1 Enter a name for the OTD. The OTD contains the selected tables and the package name of the generated classes. See **Figure 23**.

Figure 23 Naming an OTD



2 View the summary of the OTD. If you find you have made a mistake, click **Back** and correct the information. If you are satisfied with the OTD information, click **Finish** to begin generating the OTD. See **Figure 24**.

Figure 24 Database Wizard - Summary



The resulting **OTD** will appear on the Enterprise Designer's canvas.

Chapter 5

Building an eWay Project

This chapter discusses how to build a DB2 Connect eWay project in a production environment.

This Chapter Includes:

5.1 eInsight 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.

5.2 Using the Sample Project in eInsight

To begin using the sample eInsight Business Process project, you will need to import the project and view it from within the Enterprise Designer using the Enterprise

Designer Project Import utility. Import the **DB2Connect_BPEL_Sample.zip** file contained in the eWay sample folder on the installation CD-ROM.

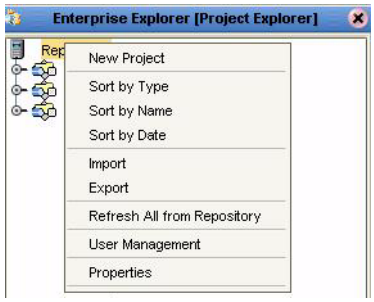
Note: *eInsight is a Business Process modeling tool. If you have not purchased eInsight, contact your sales representative for information on how to do so.*

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

Importing the Sample Project

1 On the Enterprise Explorer highlight the repository and right click. Select **Import Project**. See **Figure 25**.

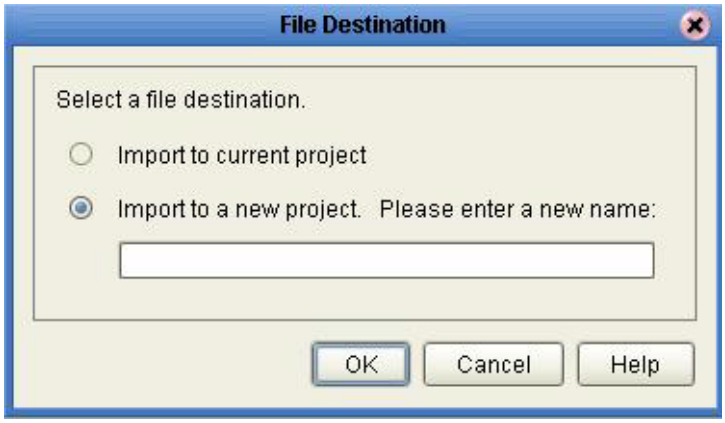
Figure 25 Importing the sample project



1 In the **Select File to Import** window, browse to the location of the sample folder and select the following .zip file **DB2Connect_sampleBPEL.zip** and click **Open**.

2 On the **File Destination** window, select **Import a new project. Please enter a new name**. Enter a name for the sample project and click OK. See **Figure 26**.

Figure 26 Select the project file destination



3 Click the **Refresh All From Repository** icon located on the **Enterprise Explorer** toolbar.

The Business Process

The data used for this sample project is contained within a table called DBEmployee. The table has the following columns:

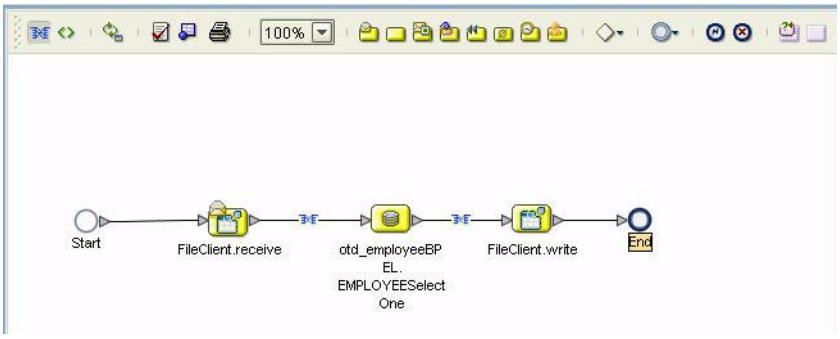
Table 1 Sample Project Data

Column Name	Mapping	Data Type	Data Length
EMPNO	EMPNO	char	6
FIRSTNME	FIRSTNME	varchar	12
MIDINIT	MIDINIT	char	1
LASTNAME	LASTNAME	varchar	15
WORKDEPT	WORKDEPT	char	3
PHONENO	PHONENO	char	4
HIREDATE	HIREDATE	date	10
JOB	JOB	char	8
EDLEVEL	EDLEVEL	smallint	5
SEX	SEX	char	1
BIRTHDATE	BIRTHDATE	date	10
SALARY	SALARY	decimal	9
BONUS	BONUS	decimal	9
COMM	COMM	decimal	9

The sample project consists of an input file containing data that is passed into a database collaboration, and then written out to an output file

4 Refer to the *eInsight Business Process Manager User's Guide* for specific information on how to create and use a Business Process.

Figure 27 Sample Project Business Process



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 **receive** or **write** operation under the eWay in the 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 DB2 Connect eWay. You can open a file specified in the eWay and view its contents before and after the Business Process is executed.

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

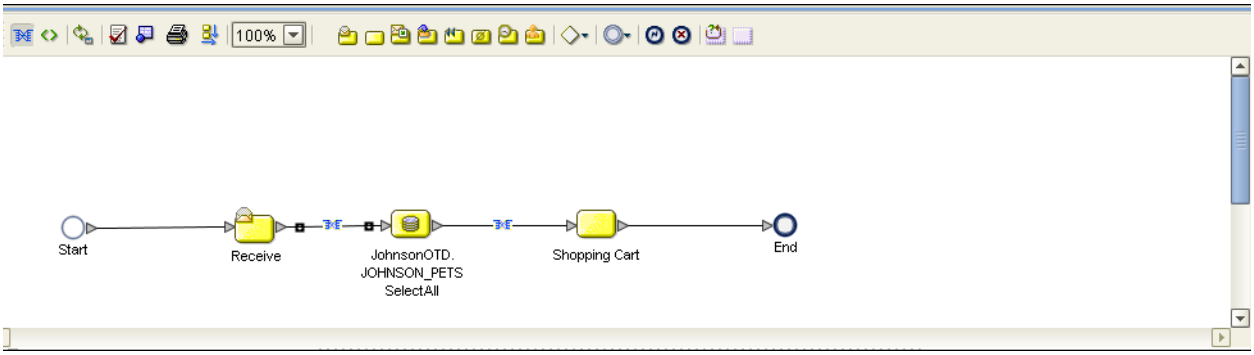
eInsight Operation	Input	Output
SelectAll	where() clause (optional)	Returns all rows that fit the condition of the where() clause
SelectMultiple	number of rows where() clause. Optional	Returns the number of rows specified that fit the condition of the where() clause when using a repeating node to output all rows.
SelectOne	number of rows 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.

5.2.1 SelectAll

The input to a SelectAll operation is an optional where() clause. The where() clause defines to which criteria rows must adhere to be returned. In the SelectAll operation, all items that fit the criteria are returned. If the where() clause is not specified, all rows are returned.

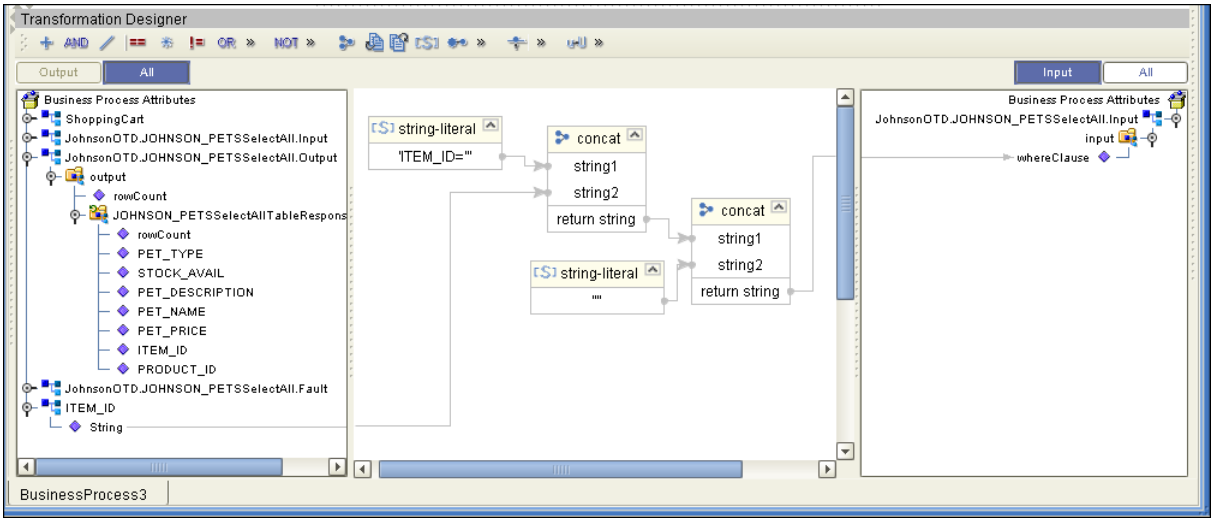
The figure below shows a sample eInsight Business Process using the SelectAll operation. In this process, the SelectAll operation returns all rows where the ITEM_ID matches the selected ITEM_ID to the shopping cart.

Figure 28 SelectAll Sample Business Process



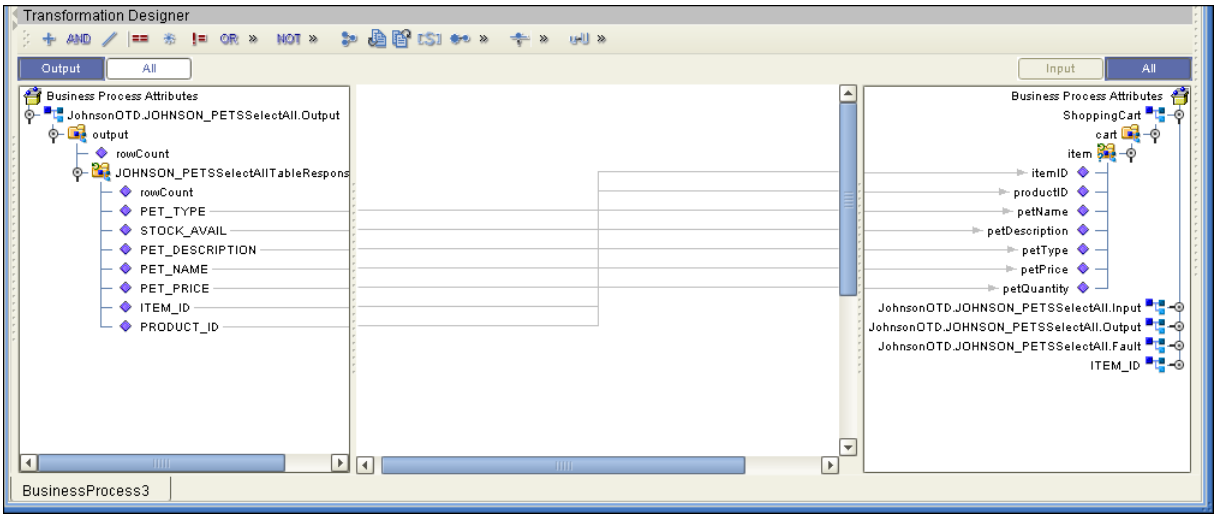
The figure below shows the definition of the where() clause for the SelectAll operation.

Figure 29 SelectAll Input



The figure below shows the definition of the output for the SelectAll operation. For each row selected during the operation, the shopping cart shows the columns of those rows as defined here.

Figure 30 SelectAll Output

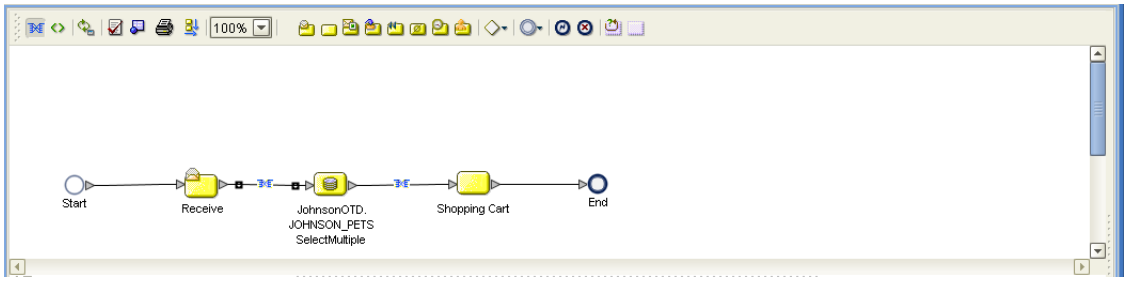


5.2.2 SelectMultiple

The input to a SelectMultiple operation is the number of rows to be selected and a where() clause. The number of rows indicates how many rows the SelectMultiple operation returns. The where() clause defines to which criteria rows must adhere to be returned.

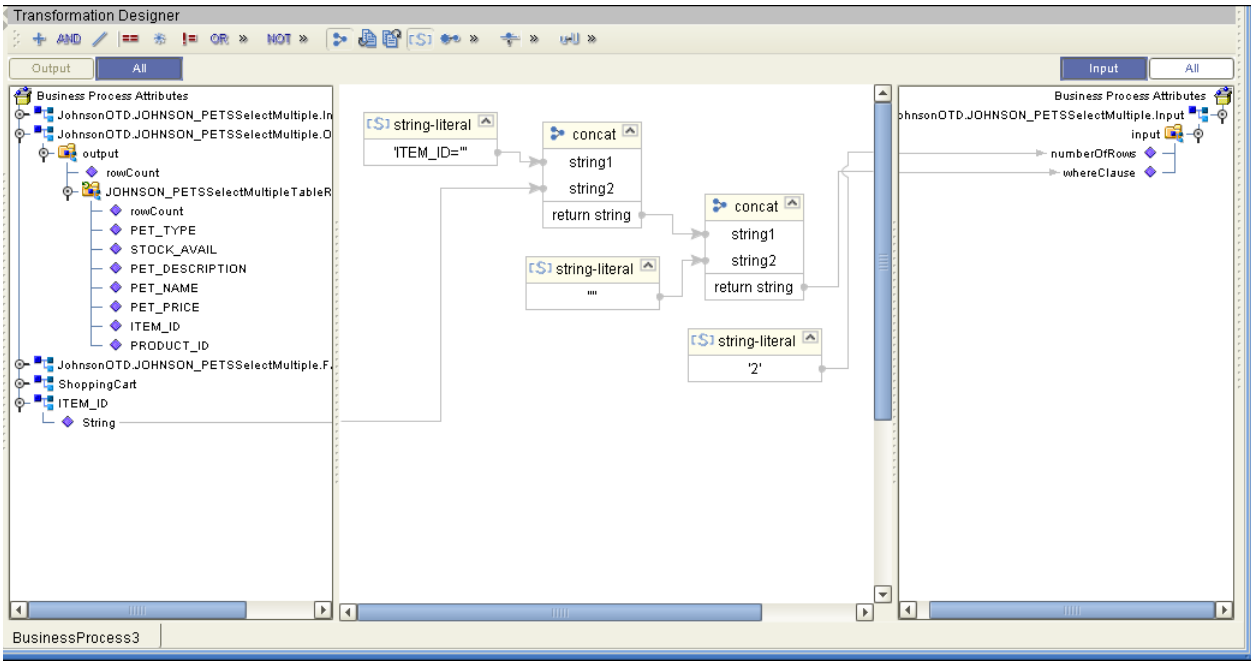
The figure below shows a sample eInsight Business Process using the SelectMultiple operation. In this process, the SelectMultiple operation returns the first two rows where the ITEM_ID matches the selected ITEM_ID to the shopping cart.

Figure 31 SelectMultiple Sample Business Process



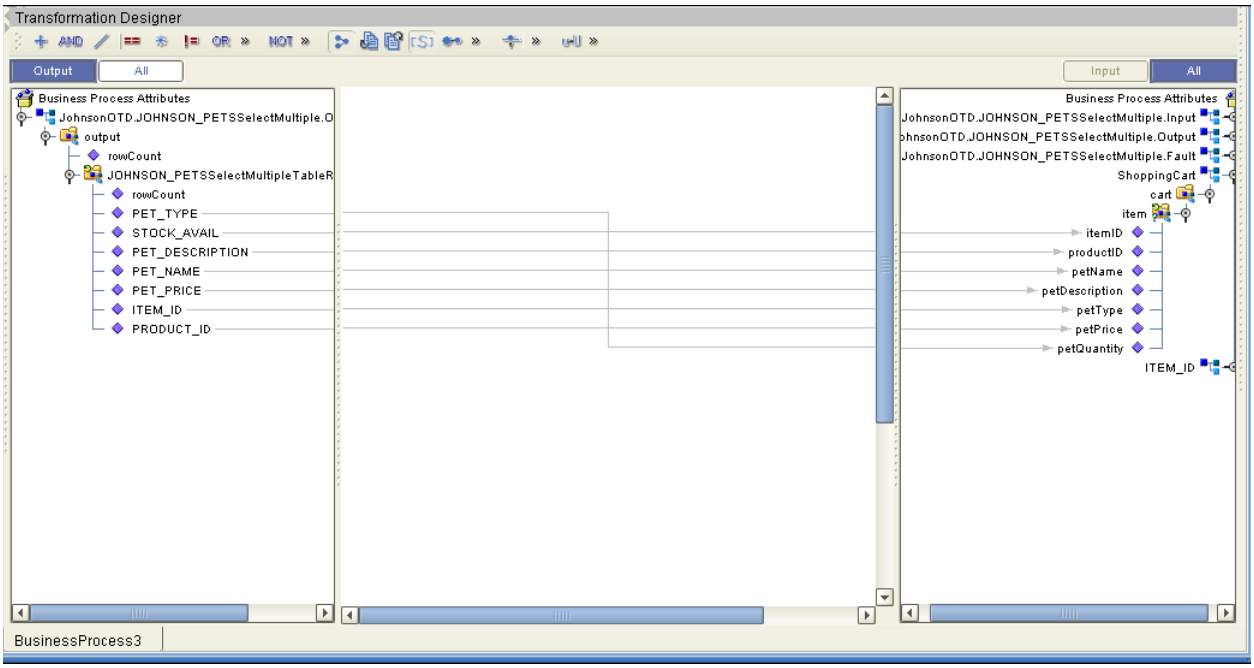
The figure below shows the definition of the number of rows and where() clause into the input for the SelectMultiple operation. You could also use an empty string or Item_ID='123'.

Figure 32 SelectMultiple Input



The figure below shows the definition of the output for the SelectMultiple operation. For each row selected during the operation, the shopping cart shows the columns of those rows as defined here.

Figure 33 SelectMultiple Output

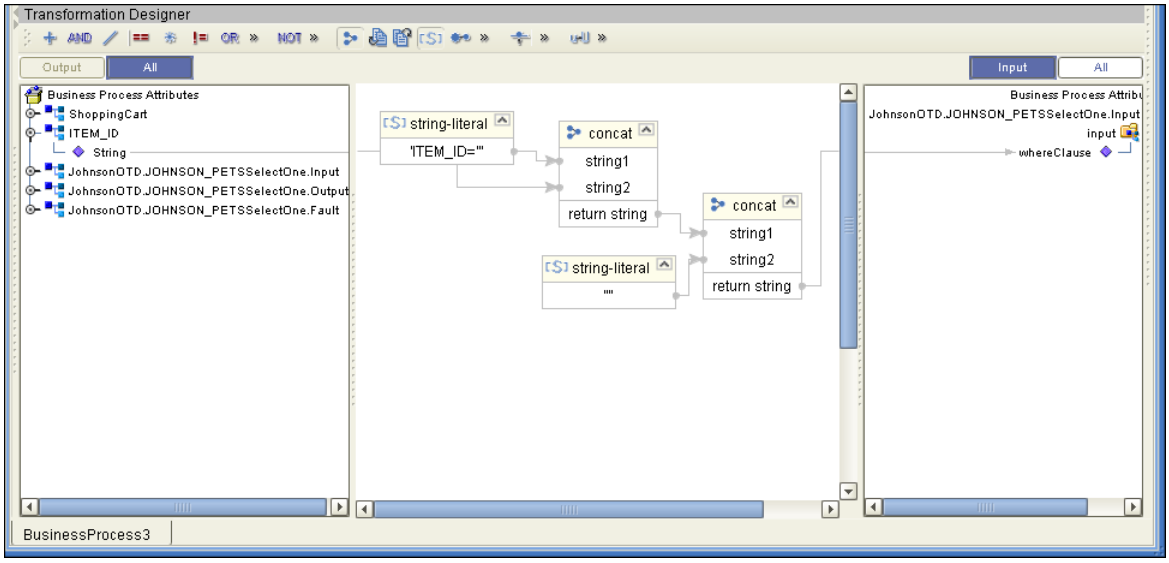


5.2.3 SelectOne

The input to a SelectOne operation is a where() clause. The where() clause defines to which criteria rows must adhere to be selected for the operation. In the SelectOne operation, the first row that fits the criteria is returned.

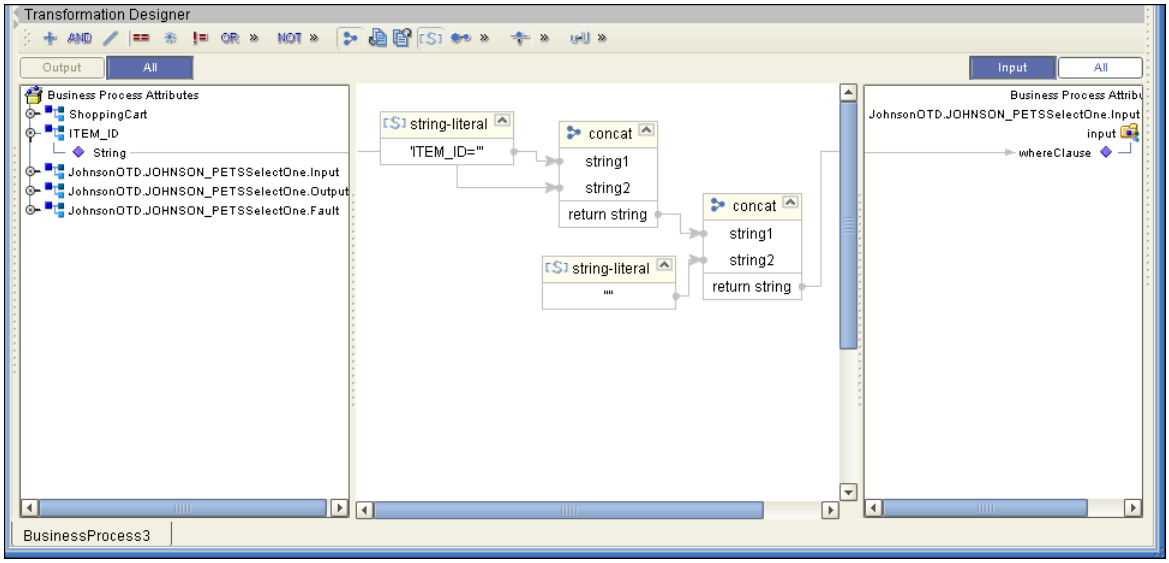
The figure below shows a sample eInsight Business Process using the SelectOne operation. In this process, the SelectOne operation returns the first row where the ITEM_ID matches the specified ITEM_ID to the shopping cart.

Figure 34 SelectOne Sample Business Process



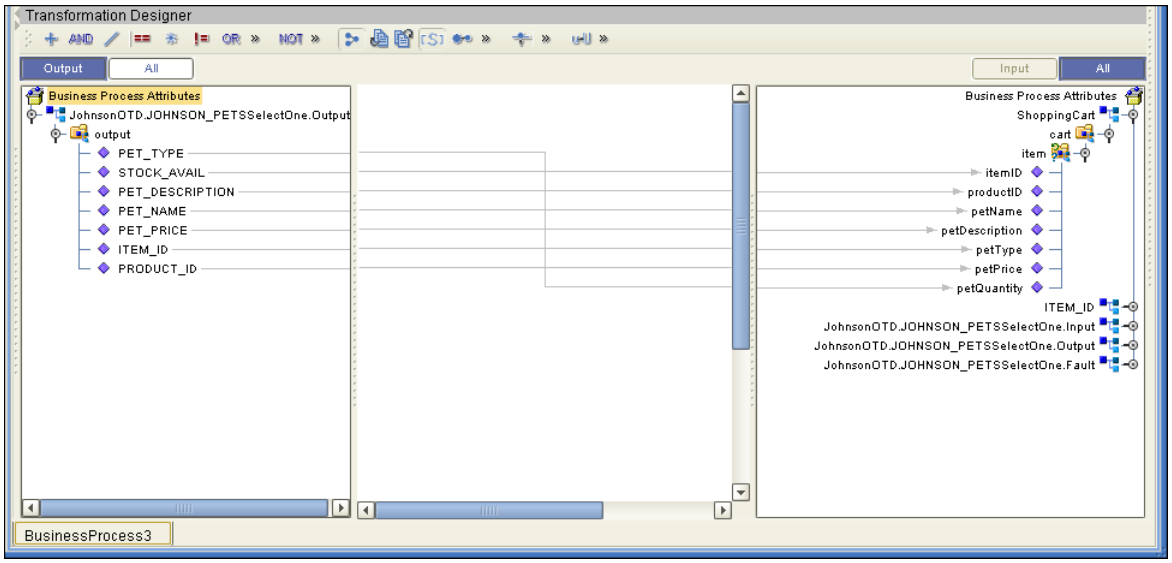
The figure below shows the definition of the where() clause for the SelectOne operation.

Figure 35 SelectOne Input



The figure below shows the definition of the output for the SelectOne operation. For the first row selected during the operation, the shopping cart shows the columns of that row as defined here.

Figure 36 SelectOne Output

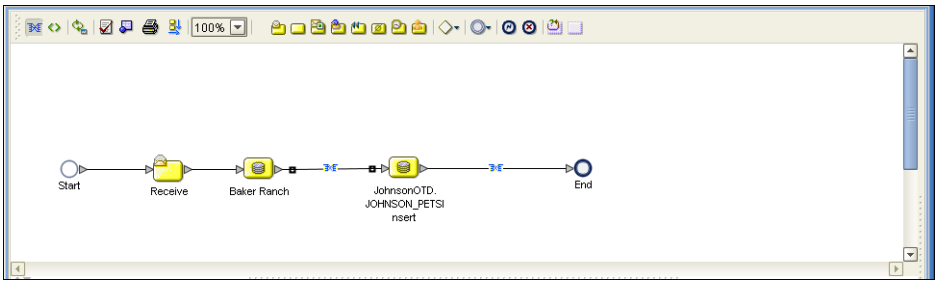


5.2.4 Insert

The Insert operation inserts a row. The input to an Insert operation is a where() clause. The where() clause defines to which criteria rows must adhere to be selected for the operation. In the Insert operation, the first row that fits the criteria is returned.

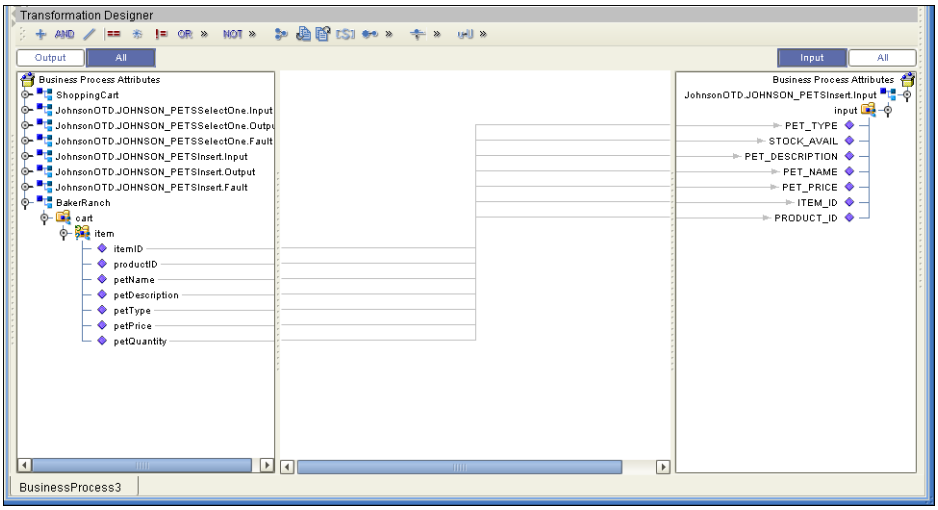
The figure below shows a sample eInsight Business Process using the Insert operation. In this process, the operation inserts a new row into the database to accommodate a new item provided by a vendor.

Figure 37 Insert Sample Business Process



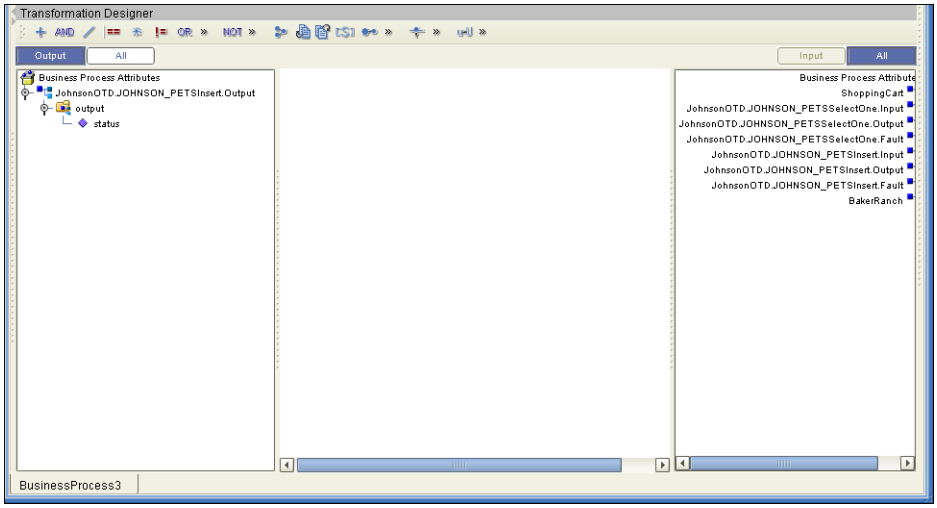
The figure below shows the definition of the input for the Insert operation.

Figure 38 Insert Input



The figure below shows the output of the Insert operation, which is a status indicating the number of rows created.

Figure 39 Insert Output

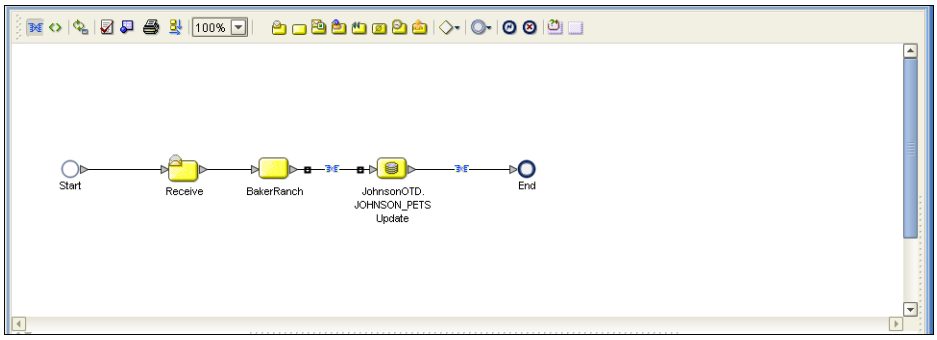


5.2.5 Update

The Update operation updates rows that fit certain criteria defined in a where() clause.

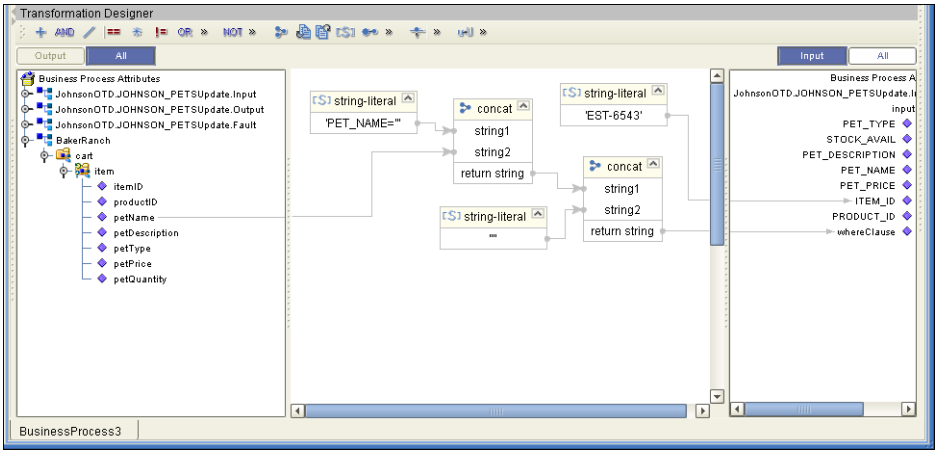
The figure below shows a sample eInsight Business Process using the Update operation. In this process, the operation updates the ITEM_ID for all items with a certain name to ESR_6543.

Figure 40 Update Sample Business Process



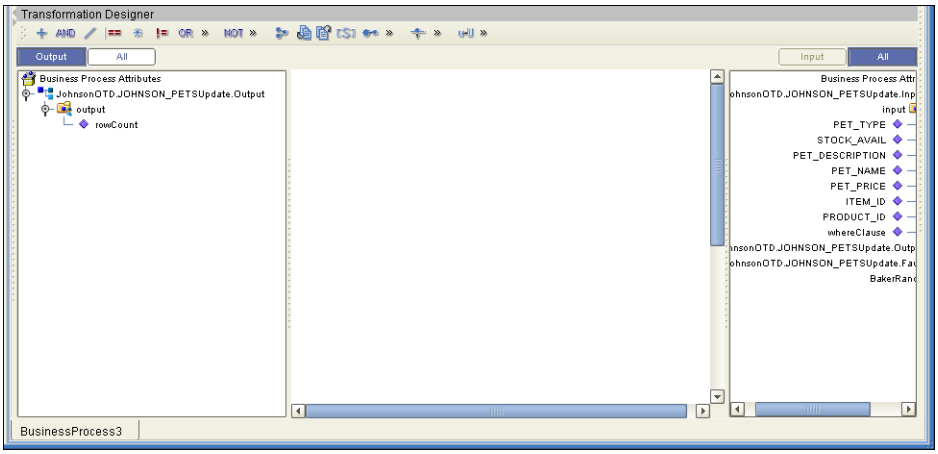
The figure below shows the definition of the where() clause for the Update operation.

Figure 41 Update Input



The figure below shows the output of the Update operation, which is a status indicating the number of rows updated.

Figure 42 Update Output



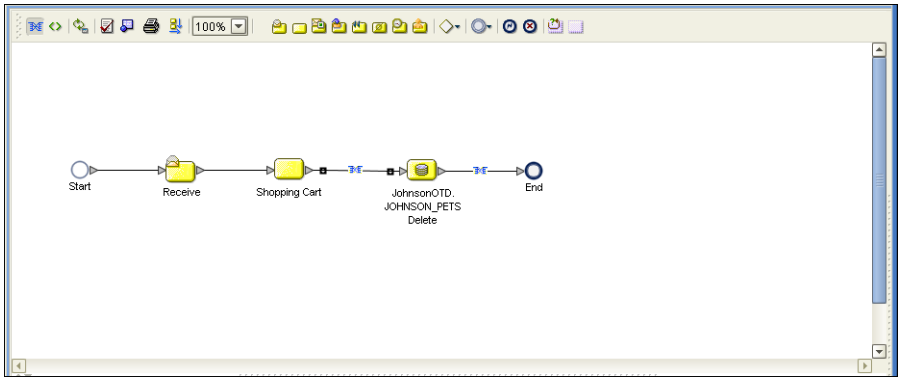
5.2.6 Delete

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

The figure below shows a sample eInsight Business Process using the Delete operation. In this process, the operation deletes rows with a certain product ID from the shopping cart.

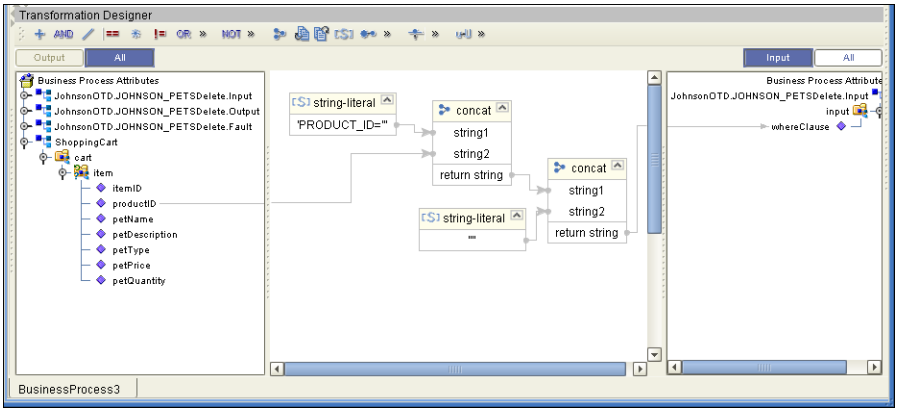
Note: *If a where() clause is not defined, all rows will be deleted.*

Figure 43 Delete Sample Business Process



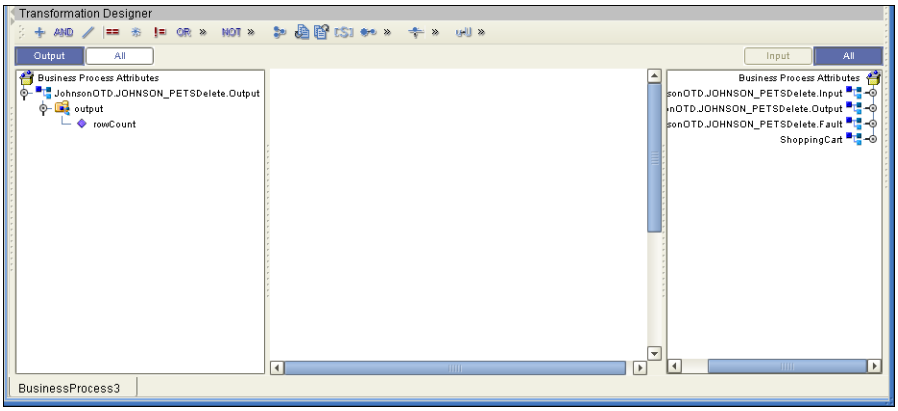
The figure below shows the definition of the where() clause for the Delete operation.

Figure 44 Delete Input



The figure below shows the output of the Delete operation, which is a status indicating the number of rows deleted.

Figure 45 Delete Output



5.3 Using the Sample Project in eGate

To import the sample project **DB2Connect_JCE_Sample.zip** follow the instructions given in **Importing the Sample Project** on page 37.

5.3.1. Working with the Sample Project in eGate

This sample project selects columns from the table DBEmployee and publishes the record to an output file.

The data used for this projects is within a table called DBEmployee. The table contains the following columns:

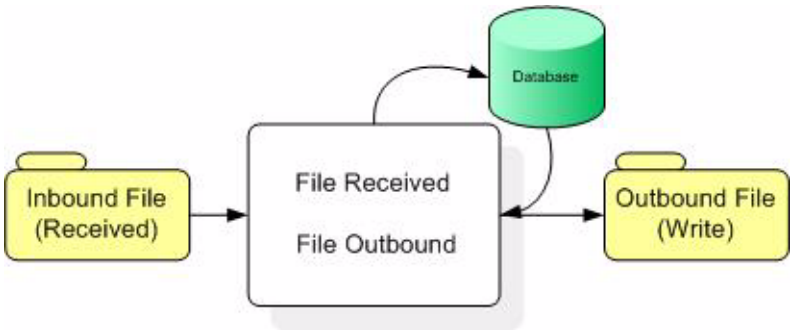
Table 2 Sample Project Data

Column Name	Mapping	Data Type	Data Length
EMPNO	EMPNO	char	6
FIRSTNME	FIRSTNME	varchar	12
MIDINIT	MIDINIT	char	1
LASTNAME	LASTNAME	varchar	15
WORKDEPT	WORKDEPT	char	3
PHONENO	PHONENO	char	4
HIREDATE	HIREDATE	date	10
JOB	JOB	char	8
EDLEVEL	EDLEVEL	smallint	5
SEX	SEX	char	1
BIRTHDATE	BIRTHDATE	date	10
SALARY	SALARY	decimal	9

Column Name	Mapping	Data Type	Data Length
BONUS	BONUS	decimal	9
COMM	COMM	decimal	9

The sample project consists of an input file containing data that is passed into a collaboration and out to the database from which data is retrieved and passed back into the collaboration and then to an output file.

Figure 46 Database project flow



To work with the sample project, follow the instructions given in the *eGate Tutorial*.

5.3.2. Configuring the eWays

The sample uses an inbound and an outbound File eWay as well as an outbound DB2 Connect eWay. To configure the sample projects eWays, use the following information. For additional information on the DB2 Connect properties, see **Working with eWay Property Sheets** on page 10.

To configure the Inbound File eWay:

1. On the Connectivity Map canvas, double click the eWay icon located between the **FileIn** and **jcdEmployeeSelect1**.
2. On the resulting **Templates** window, select **Inbound File eWay** and click **OK**.
3. On the **Properties** window, enter the appropriate configurations for the Inbound File eWay. See the ***File eWay User's Guide*** for information on how to specifically configure the File eWay. For this sample, the default settings are used.
4. When you have completed your selections, click **OK**.

To configure the Outbound DB2 Connect eWay:

1. On the Connectivity Map canvas, double click the eWay icon located between the **jcdEmployeeSelect1** and **DB2 Connect1** database.
2. On the resulting **Templates** window, select **Outbound DB2** and click **OK**.
3. On the Properties window, enter the appropriate configurations for the Outbound DB2 eWay and click **OK**. See **Working with eWay Property Sheets** on page 10. For this sample, the default settings are used.

4 When you have completed your selections, click **OK**.

To configure the Outbound File eWay:

1 On the Connectivity Map canvas, double click the eWay icon located between **jcdEmployeeSelect1** and **FileOut** eWay.

2 On the resulting **Templates** window, select **Outbound File eWay** and click **OK**.

3 On the **Properties** window, enter the appropriate configurations for the Outbound File eWay. See the ***File eWay User's Guide*** for information on how to specifically configure the File eWay. For this sample, 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, the default settings are used.

4 When you have completed your selections, click **OK**.

5.3.3. Creating an External Environment

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

To create the external environment for the Sample project:

5 On the Environment Explorer, highlight and right-click the DB2 profile. Select **Properties**. Enter the configuration information required for your Outbound DB2 Connect eWay. See **Working with eWay Property Sheets** on page 10.

5.3.4 Deploying a Project

To deploy a project, please see the "*eGate Integrators User's Guide*".

5.3.5. Running the Sample

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

5.4 Common DataType Conversions

Table 3 The DB2 Connect eWay Insert or Update Operations for Text/String Input Data

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

DB2 Data Type	OTD/Java Data Type	Java Method or New Constructor to Use (Default: Java Method)	Sample Data
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
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

Table 4 The DB2 Connect eWay Select Operations for Text/String Output 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

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

5.5 Using OTDs with Tables, Views, and Stored Procedures

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

5.5.1 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 a query on a table.

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();
    }
```

```
}
```

5.5.2 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. 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
    {
        //
@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:Copy "procedure executed" to Text
        FileClient_1.setText( "procedure executed" );

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

5.5.3 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.

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();
        }

    }
```

5.5.4 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();
```

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 OTD, 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 OTD 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 OTD nodes to the Business Rules. Below is a code snippet that can be generated by the Collaboration Editor:

```
// resultsAvailable() will be true if there's an update count and/or a
result set available.
// note, it should not be called indiscriminantly because each time
the results pointer is
// advanced via getMoreResults() call.
while (getSPIn().getSpS_multi().resultsAvailable())
{
    // check if there's an update count
    if (getSPIn().getSpS_multi().getUpdateCount() > 0)
    {
        logger.info("Updated
"+getSPIn().getSpS_multi().getUpdateCount()+" rows");
    }
// each result set node has an available() method (similar to OTD's)
that tells the user
// whether this particular result set is available. note, JDBC does
support access to
// more than one result set at a time, i.e., cannot drag from 2
distinct result sets
// simultaneously
    if (getSPIn().getSpS_multi().getNormRS().available())
    {
    while (getSPIn().getSpS_multi().getNormRS().next())
    {
    logger.info("Customer Id =
"+getSPIn().getSpS_multi().getNormRS().getCustomerId());
    logger.info("Customer Name =
"+getSPIn().getSpS_multi().getNormRS().getCustomerName());
    }
    if (getSPIn().getSpS_multi().getDbEmployee().available())
    {
    while (getSPIn().getSpS_multi().getDbEmployee().next())
    {
    logger.info("EMPNO =
"+getSPIn().getSpS_multi().getDbEmployee().getEMPNO());
    logger.info("ENAME =
"+getSPIn().getSpS_multi().getDbEmployee().getENAME());
    logger.info("JOB =
"+getSPIn().getSpS_multi().getDbEmployee().getJOB());
    logger.info("MGR =
"+getSPIn().getSpS_multi().getDbEmployee().getMGR());
    logger.info("HIREDATE =
"+getSPIn().getSpS_multi().getDbEmployee().getHIREDATE());
    logger.info("SAL =
"+getSPIn().getSpS_multi().getDbEmployee().getSAL());
    logger.info("COMM =
"+getSPIn().getSpS_multi().getDbEmployee().getCOMM());
    logger.info("DEPTNO =
"+getSPIn().getSpS_multi().getDbEmployee().getDEPTNO());
    }
}
```

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.

5.6 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***.

5.7 Additional Sample

An additional sample **DB2_Connect_Adittional_Sample.zip** is provided to illustrate an Update, Insert, and Delete within a Prepared Statement. To use this sample do the following:

1. On the Enterprise Manager, click the **Documentation** tab.

2 Click on **DB2 Connect eWay Intelligent Adapter**.

3 Click on the **Download Sample** link.

4 Click on the **DB2_Connect_Adittional_Sample.zip** file.

5 Unzip this file to a local file.

Index

R

S

U