SeeBeyond ICAN Suite

DB2 Universal Database eWay Intelligent Adapter User's Guide

Release 5.0.6

SeeBeyond®

Contents

Chapter 4

DB2 Wizard Operation   32

Chapter 5

Implementing the DB2 eWay   46

Introducing the DB2 Universal Database eWay

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

What's in This Chapter

1.1 About DB2 Universal Database

A database consists of a collection of information that is organized so that it can be easily accessed, managed, and updated. DB2 Universal Database is a database that handles the development and deployment of critical solutions such as:

- On demand business
- Business intelligence
- Content management
- Enterprise Resource Planning
- Customer Relationship Management

DB2 reduces the complexity of data management by eliminating, simplifying, and automating tasks associated with maintaining an enterprise-class database. It provides a foundation of information integration technologies, including federation, replication, Web services, and XML.

1.2   About the DB2 Universal Database eWay

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

Note: The DB2 Universal Database eWay connects to DB2 via the DataDirect driver which is packaged with the eWay.

1.3   What's New in This Version

The DB2 eWay includes driver support for DataBase Drivers JDBC 3.4.

1.4   About This Document

This guide explains how to install, configure, and operate the SeeBeyond® Integrated Composite Application Network Suite™ (ICAN) DB2 eWay Intelligent Adapter, referred to as the DB2 eWay throughout this guide.

1.4.1   What's in This Document

This document includes the following chapters:

- **Chapter 1 "Introducing the DB2 Universal Database eWay":** Provides an overview description of the product as well as high-level information about this document.

- Chapter 2 **"Installing the eWay"**: Describes the system requirements and provides instructions for installing the DB2 Universal Database eWay.

- Chapter 3 **"Configuring the DB2 eWay"**: Provides instructions for configuring the eWay to communicate with your legacy systems.

- Chapter 4 **"DB2 Wizard Operation"**: Provides information about .sag files and using the DB2 Universal Database wizard.

- Chapter 5 **"Implementing the DB2 eWay"**: Provides instructions for installing and running the sample Projects.

1.4.2   Scope

This document describes the process of installing, configuring, and running the DB2 Universal Database eWay.

This document does not cover the Java methods exposed by this eWay. For information on the Java methods, download and view the DB2 Universal Database eWay Javadoc files from the Enterprise Manager.

1.4.3 Intended Audience

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

1.4.4 Document Conventions

The following writing conventions are observed throughout this document.

Table 1 Writing Conventions

Text	Convention	Example
Button, file, icon, parameter, variable, method, menu, and object names.	**Bold** text	▪ Click **OK** to save and close. ▪ From the **File** menu, select **Exit**. ▪ Select the **logicalhost.exe** file. ▪ Enter the **timeout** value. ▪ Use the **getClassName()** method. ▪ Configure the **Inbound** File eWay.
Command line arguments and code samples	`Fixed` font. Variables are shown in ***bold italic***.	`bootstrap -p` ***`password`***
Hypertext links	**Blue** text	**http://www.seebeyond.com**

1.4.5 Screenshots

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

1.5 Related Documents

The following SeeBeyond documents provide additional information about the ICAN product suite:

- *eGate Integrator User's Guide*
- *SeeBeyond ICAN Suite Installation Guide*

1.6 SeeBeyond Web Site

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

http://www.seebeyond.com

1.7 Feedback

If you have any feedback on any SeeBeyond documentation, please send an E-mail to:

docfeedback@seebeyond.com

Installing the eWay

This chapter describes how to install the DB2 Universal Database eWay.

What's in This Chapter

- **"Supported Operating Systems" on page 11**
- **"System Requirements" on page 12**
- **"Supported External Applications" on page 12**
- **"Installing the eWay Product Files" on page 12**
- **"After You Install" on page 13**

2.1 Supported Operating Systems

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

- Windows Server 2003, Windows XP, and Windows 2000
- HP Tru64 5.1A
- HP-UX 11.0, 11i (PA-RISC), and 11i v2.0 (11.23)
- IBM AIX 5.1L and 5.2
- Sun Solaris 8 and 9
- SuSE Linux Enterprise Server 8 (Intel x86)
- z/OS 1.3 and 1.4

Although the DB2 Universal Database eWay, the Repository, and Logical Hosts run on the platforms listed above, the Enterprise Designer requires the Windows operating system. For more information, see the *eGate Integrator User's Guide.*

2.1.1 WebLogic and WebSphere Application Server Support

In addition to the operating systems listed above, this eWay is also supported on the following application servers:

- WebSphere Application Server, version 5.0
- WebLogic Application Server, version 8.1

These are limited to outbound mode using Java Collaborations. For additional information see the *eGate Integrator User's Guide*.

2.2 System Requirements

The system requirements for the DB2 eWay are the same as for eGate Integrator. For information, refer to the *ICAN 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.

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

2.3 Supported External Applications

The DB2 eWay supports the following software on external systems:

- DB2 Universal Database (UDB) version 8.2 for Windows/UNIX.
- DB2 Universal Database (UDB) version 7.1 when connecting to DB2 running on an OS/390 operating system when using DataDirect drivers.
- DB2 Universal Database (UDB) version V5R1 when connecting to DB2 running on an AS/400 operating system.
- Driver support for DataDirect Drivers JDBC 3.4

2.4 Installing the eWay Product Files

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 *ICAN Installation Guide*, and include the following steps:

- On the Enterprise Manager, select the **DB2eWay.sar** (to install the DB2 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 **DB2eWayDocs.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. It is recommended that you extract the sample project file to another 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.

2.5 After You Install

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.

Configuring the DB2 eWay

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

What's in This Chapter

- **"Setting the Properties of the Inbound DB2 eWay on Windows or Unix Operating Systems" on page 15**

- **"Setting the Environment Properties of the Inbound DB2 eWay on Windows or Unix Operating Systems" on page 16**

- **"Setting the Properties of the Outbound DB2 eWay on Windows or Unix Operating Systems" on page 17**

- **"Setting the Environment Properties of the Outbound DB2 eWay on Windows or Unix Operating Systems" on page 20**

- **"Setting the Properties of the Inbound DB2 eWay Connecting to z/OS or and AS/400 Operating System" on page 23**

- **"Setting the Environment Properties of the Inbound DB2 eWay Connecting to z/OS or an AS/400 Operating Systems" on page 24**

- **"Setting the Properties of the Outbound DB2 eWay Connecting to z/OS or an AS/400 Operating System" on page 26**

- **"Setting the Environment Properties of the Outbound DB2 eWay Connecting to z/OS or an AS/400 Operating Systems" on page 28**

3.1 Properties of DB2 eWay on Windows or Unix Operating Systems

The following parameter descriptions are used for you to enter the necessary information, on the Properties window, for the eWay to establish a connection to the external application.

3.1.1 Setting the Properties of the Inbound DB2 eWay on Windows or Unix Operating Systems
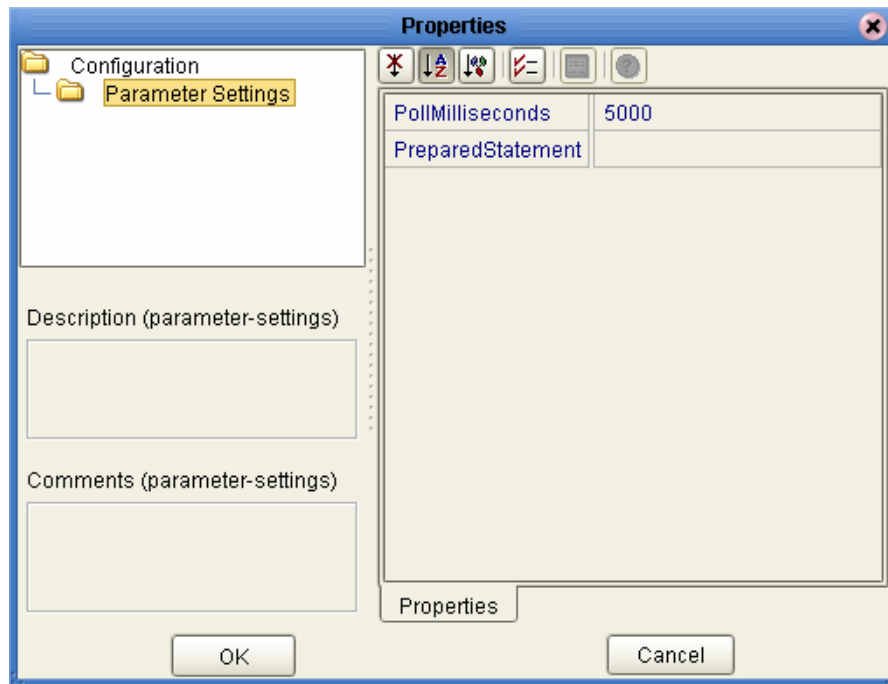
PollMilliseconds

Description

Specify the polling interval between database queries in milliseconds.

Required Values

A valid numeric value. The default is 5000.

PreparedStatement

Description

Specify the Prepared Statement used to query the database.

Required Values

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

3.1.2 Setting the Environment Properties of the Inbound DB2 eWay on Windows or Unix Operating Systems

DatabaseName

Description

Specify the name of the database instance.

Required Values

Any valid string.

Password

Description

Specify the password used to access the database.

Required Values

Any valid string.

PortNumber

Description

Specify the I/O port number on which the server is listening for connection requests.

Required Values

A valid port number. The default is 50000.

ServerName

Description

Specify the host name of the external database server.

Required Values

Any valid string.

User

Description

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

Required Values

Any valid string.

3.1.3 Setting the Properties of the Outbound DB2 eWay on Windows or Unix Operating Systems

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

Note: Not all parameters are supported in the current release, please contact SeeBeyond for more information.

Figure 1 The eWay Properties
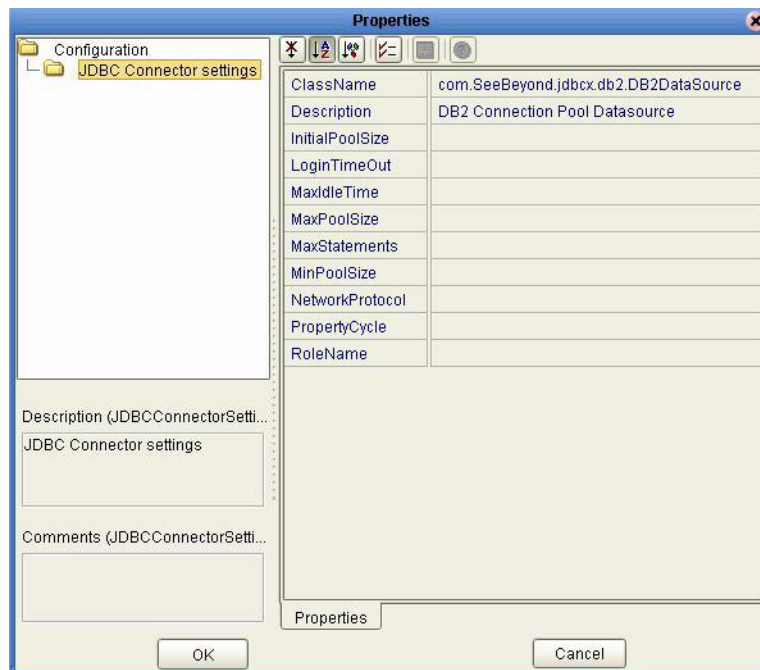
ClassName

Description

Specify 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.SeeBeyond.jdbcx.db2.DB2DataSource**.

Description

Description

Enter a description for the database.

Required Values

A valid string.

InitialPoolSize

Description

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

Required Values

A valid numeric value.

LoginTimeOut

Description

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

Required Values

A valid numeric value.

MaxIdleTime

Description

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

Required Values

A valid numeric value.

MaxPoolSize

Description

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

Required Values

A valid numeric value.

MaxStatements

Description

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

Required Values

A valid numeric value.

MinPoolSize

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

Required Values

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 must 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.

3.1.4 Setting the Environment Properties of the Outbound DB2 eWay on Windows or Unix Operating Systems

Before deploying your eWay, you must set the properties of the eWay environment using the following descriptions.

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

DatabaseName

Description

Specify the name of the database instance.

Required Values

Any valid string.

Delimiter

Description

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

Required Values

The default is #.

Description

Description

Enter a description for the database.

Required Values

A valid string.

DriverProperties

Description

Use the JDBC driver that is shipped with this eWay. If you need to set any additional properties to assure a connection, you can set them in the driver properties.

Required Values

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>##".

Note: *The setSpyAttributes, contained in the following examples (between the last set of double octothorps [##] within each example), are used for debugging purposes and need not be used on every occasion.*

Windows & Unix example:

setURL#jdbc:Seebeyond:db2://<server>:50000;DatabaseName=<database>**##setSpyAttributes#log=(file)c:/temp/spy.log;logTName=yes##**

zOS & AS/400 example:

setURL#jdbc:SeeBeyond:db2://<server>:446;locationName=<location>;collectionId=<collection>**##setLocationName#<location>##setCollectionID#<collection>##setSpyAttributes#log=(file)c:/temp/spy.log;logTName=yes##**

Password

Description

Specify the password used to access the database.

Required Values

Any valid string.

PortNumber

Description

Specify the I/O port number on which the server is listening for connection requests.

Required Values

A valid port number. The default is 50000.

ServerName

Description

Specify the host name of the external database server.

Required Values

Any valid string.

User

Description

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

Required Values

Any valid string.

3.2 Properties of the DB2 eWay Connecting to z/OS or an AS/400 Operating System

The following parameter descriptions are used for you to enter the necessary information, on the Properties window, for the eWay to establish a connection to the external application.

3.2.1 Setting the Properties of the Inbound DB2 eWay Connecting to z/OS or and AS/400 Operating System

PollMilliseconds

Description

Specify the polling interval between database queries in milliseconds.

Required Values

A valid numeric value. The default is 5000.

PreparedStatement

Description

Specify the Prepared Statement used to query the database.

Required Values

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

3.2.2 Setting the Environment Properties of the Inbound DB2 eWay Connecting to z/OS or an AS/400 Operating Systems
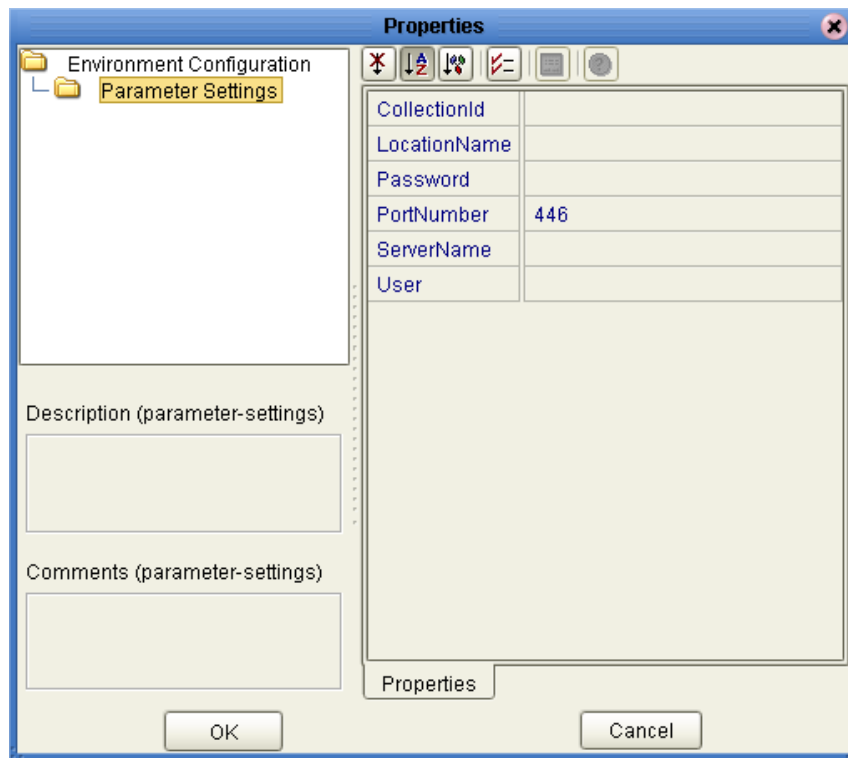
CollectionId

Description

Specify the CollectionID for the DB2 database that is being used on AS400/zOS.

Required Values

Any valid String.

LocationName

Description

Specify the Location Name for the DB2 database that is being used on AS400/zOS.

Required Values

Any valid String.

Password

Description

Specify the password used to access the database.

Required Values

Any valid String.

PortNumber

Description

The TCP port number. PortNumber is used for DataSource connections only.

Required Values

Any valid String. The default port number is 446.

ServerName

Description

Specify the name of the database server being used.

Required Values

Any valid String.

User

Description

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

Required Values

Any valid String.

3.2.3 Setting the Properties of the Outbound DB2 eWay Connecting to z/OS or an AS/400 Operating System

Figure 3   Outbound DB2 eWay Connecting to an AS400
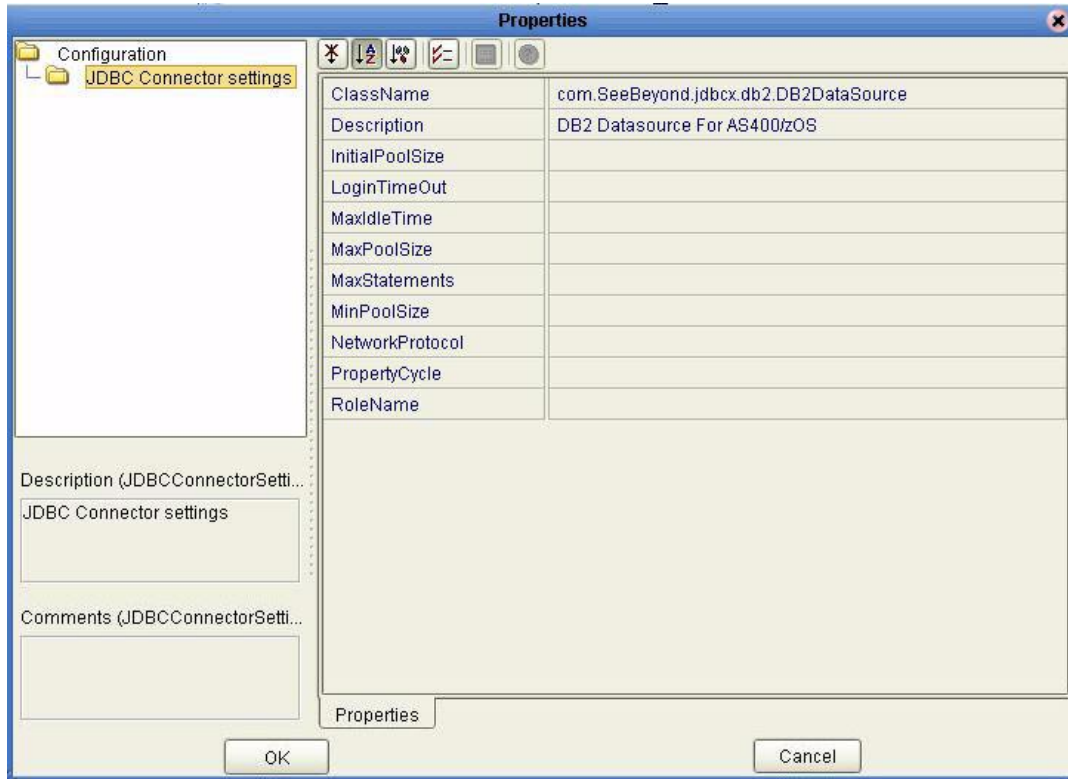
ClassName

Description

Specify 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.SeeBeyond.jdbcx.db2.DB2DataSource**.

Description

Description

Enter a description for the database.

Required Values

A valid string.

InitialPoolSize

Description

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

Required Values

A valid numeric value.

LoginTimeOut

Description

The number of seconds the driver waits before attempting to log into the database before timing out.

Required Values

A valid numeric value.

MaxIdleTime

Description

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

Required Values

A valid numeric value.

MaxPoolSize

Description

The maximum number of physical connections the pool keeps 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 keeps open. 0 (zero) indicates that the caching of statements is disabled.

Required Values

A valid numeric value.

MinPoolSize

The minimum number of physical connections the pool keeps 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 Values

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 waits 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.

3.2.4 Setting the Environment Properties of the Outbound DB2 eWay Connecting to z/OS or an AS/400 Operating Systems

Before deploying your eWay, you must set the properties of the eWay environment using the following descriptions.

Figure 4   Properties of the DB2 eWay Connecting to z/OS or an AS/400 Operating System
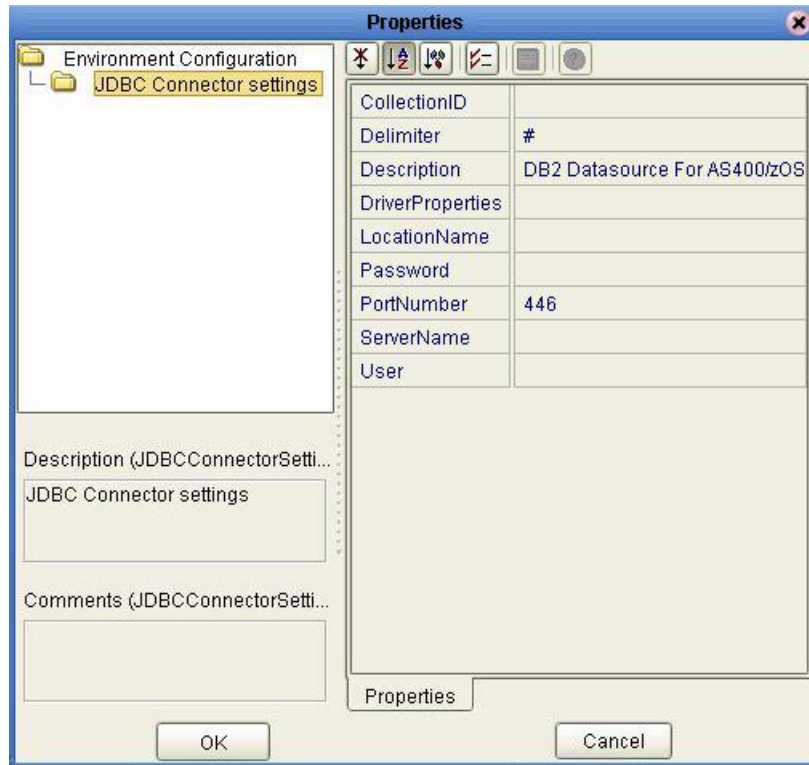
CollectionID

Description

The collection or group of packages to which a package is bound.

Required Values

Any valid string.

Delimiter

Description

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

Required Values

The default is #.

Description

Description

Enter a description for the database.

Required Values

A valid string.

DriverProperties

Description

Use the JDBC driver that is shipped with this eWay. If you need to set any additional properties to assure a connection, you can set them in the driver properties.

Required Values

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>##".

If you are using Spy Log. Optional:

"setURL#jdbc:SeeBeyond:db2://<server>:446;locationName=<location>;collectionId=<collection>##setLocationName#<location>##setCollectionID#<collection>##setSpyAttributes#log=(file)c:/temp/spy.log;logTName=yes##"

LocationName

Description

Specify the name of the DB2 location that you want to access.

Required Values

Any valid string.

Password

Description

Specify the password used to access the database.

Required Values

Any valid string.

PortNumber

Description

Specify the I/O port number on which the server is listening for connection requests.

Required Values

A valid port number. The default is 446.

ServerName

Description

Specify the host name of the external database server.

Required Values

Any valid string.

User

Description

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

Required Values

Any valid string.

DB2 Wizard Operation

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

What's in This Chapter

- **"Select Wizard Type" on page 32**
- **"Connect to Database" on page 33**
- **"Select Database Objects" on page 35**
- **"Select Table/Views" on page 35**
- **"Select Procedures" on page 40**
- **"Add Prepared Statements" on page 42**
- **"Specify the OTD Name" on page 44**

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 OTD's are not messagable. For more information on messagable OTD's, 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 Database** and click **Next**. See **Figure 5**.

Figure 5 OTD Wizard Selection
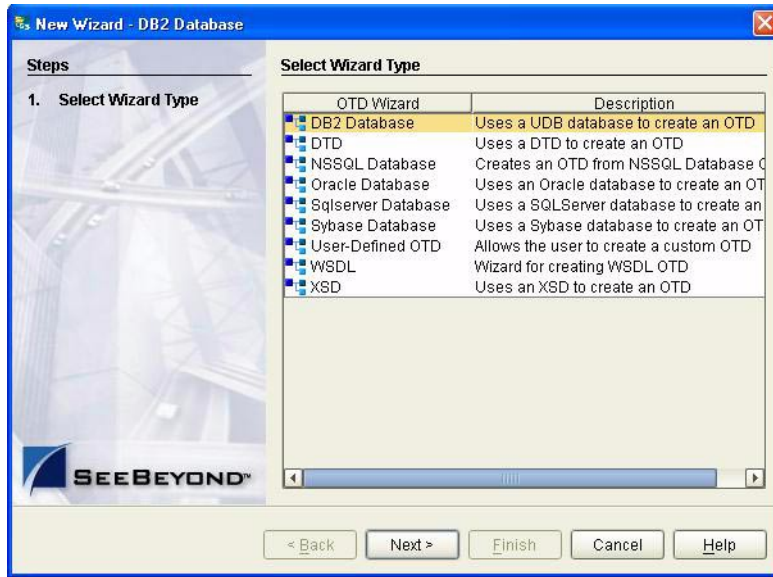
Connect to Database

1 Select the **Connection Type** using the drop-down list (see Figure 6). The rest of the Connection Information fields displayed will depend on your selection.

Figure 6 Database Connection Information
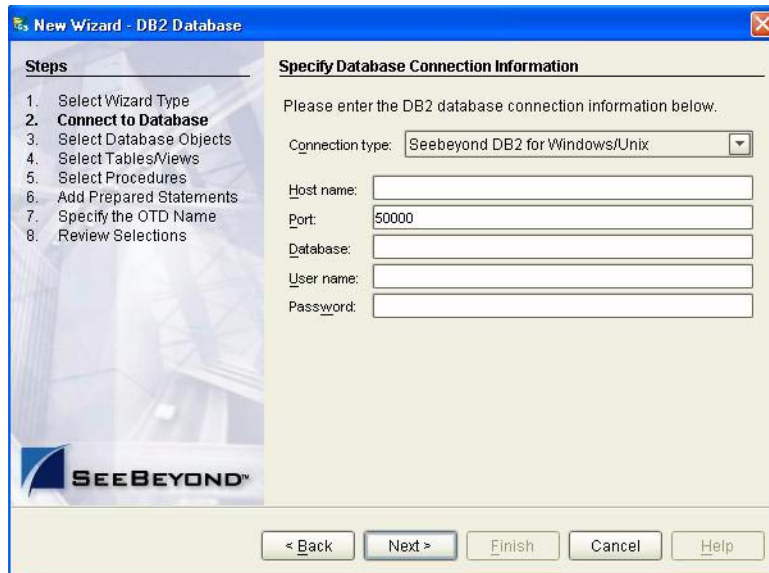
2   Specify the applicable connection information for your database including:

- **Host Name** - The server where DB2 resides.

- **Port** - The port number of DB2.

- **Location** (AS/400 and z/OS specific) - The name of the DB2 subsystem. To find the location of the DB2 subsystem, use the Database Query Tool to issue the following query: select current server from sysibm.sysdummy1.

- **Database** (Windows/Unix specific) - The name of the database instance.

- **Collection** (AS/400 and z/OS specific) - The name that identifies a group of packages.

- **User Name** - The user name that the eWay uses to connect to the database.

- **Password** - The password used to access the database.

Note: *Additional connection parameters for the DB2 OTD wizard are available. An* ***Optional Parameters*** *field will display in the OTD Wizard if the DB2_ConnectionInfo.txt file is present in the edesigner\usrdir\modules\ext\db2adapter directory. Please contact SeeBeyond for more information on Optional Parameters.*

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

Packages

This eWay uses a DataDirect driver (previously known as Merant) to execute SQL calls in DB2. The DataDirect driver requires packages to be created in the DB2 System. Packages do not contain specific SQL statements like static SQL packages but rather dynamic sections, used like cursors to help facilitate the driver's executing of dynamic SQL queries and returning results.

Creating packages on the server, also known as binding packages, needs only be done once. The first user of the OTD Wizard must have bind permission to create the packages. Without bind authority the user receives an error message when the driver attempts to bind the packages and they will be unable to issue any SQL call. Packages are created automatically, under the Collection ID, when the user fills in the Wizard entries. If the Collection ID is left as blank, it will generate the packages under NULLID.

The driver creates SQL packages on the database including: DDJC330A, DDJC330B, DDJC330C, etc. When connecting, the driver queries a system table to determine whether the default packages exist on the system. If none exist, the driver creates them.

Note: *SQL applications that execute dynamic SQL against DB2 need to have packages bound on the server. In the case of some IBM native tools this may not be obvious because the packages are already installed on the database by default.*

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 (see Figure 7).

Figure 7 Select Database Objects
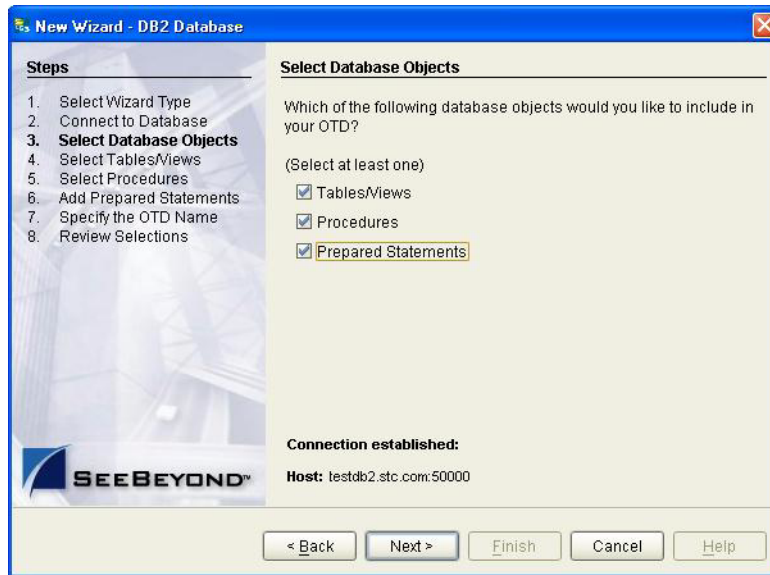
2 Click **Next** to continue. The **Select Tables/Views** window appears.

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

Select Table/Views

1 In the **Select Tables/Views** window, click **Add** (see Figure 8).

Figure 8 Select Tables/Views
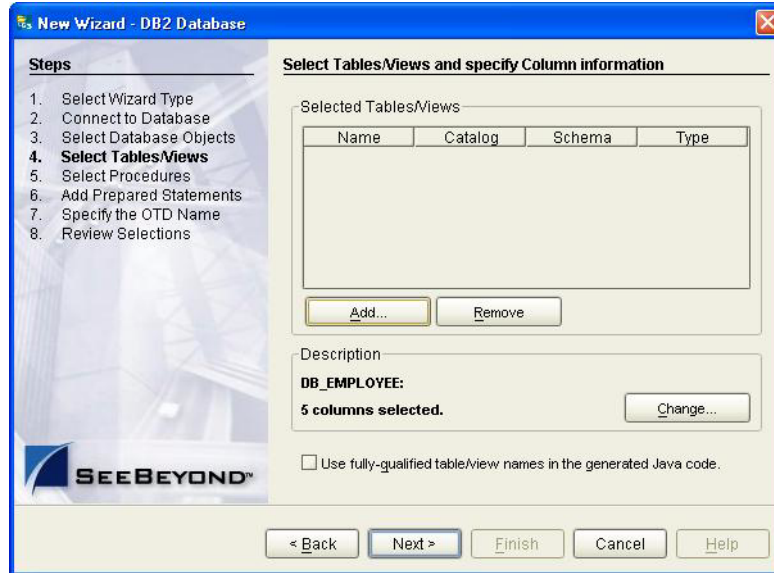
2   In the **Add Tables** window, select the type of criteria to be used for your search, consisting of table data, view only data, or both. You can include system tables in your search by selecting the checkbox.

3   From the **Table/View Name** drop down list, select the location of your database table and click **Search** (see Figure 9). 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 9 Database Wizard - All Schemes
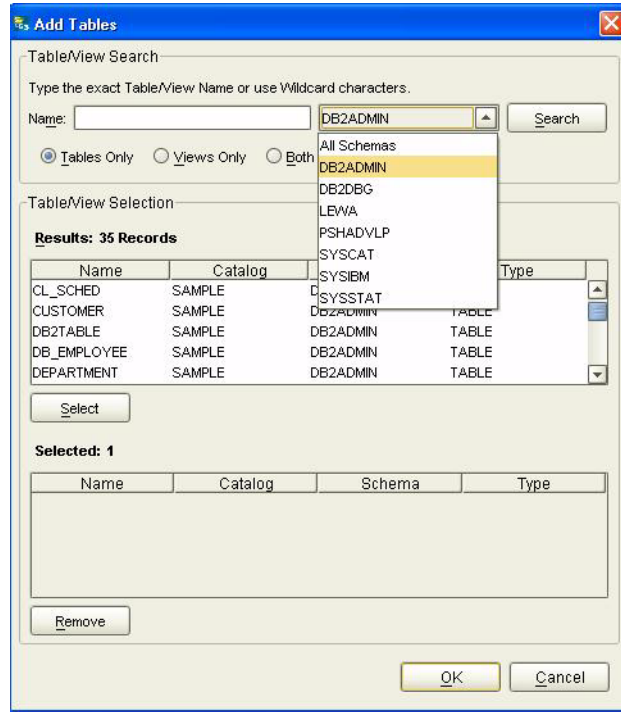
4 Select the table of choice and click **OK**.

The table selected is added to the **Selected** window (see Figure 10).

Figure 10 Selected Tables/Views window with a table selected
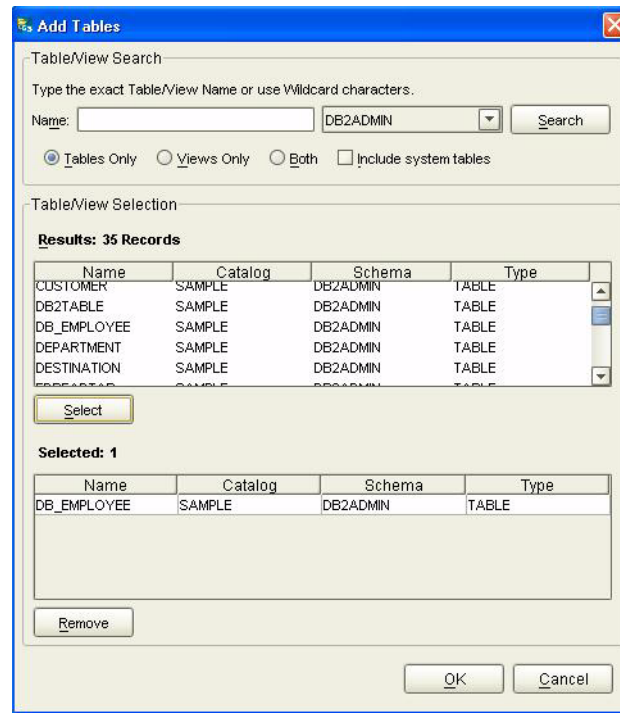
5   On the **Selected Tables/Views** window, review the table(s) you have selected. To make changes to the selected Table or View, click **Change**. If you do not wish to make any additional changes, click **Next** to continue.

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

Figure 11 Table/View Columns
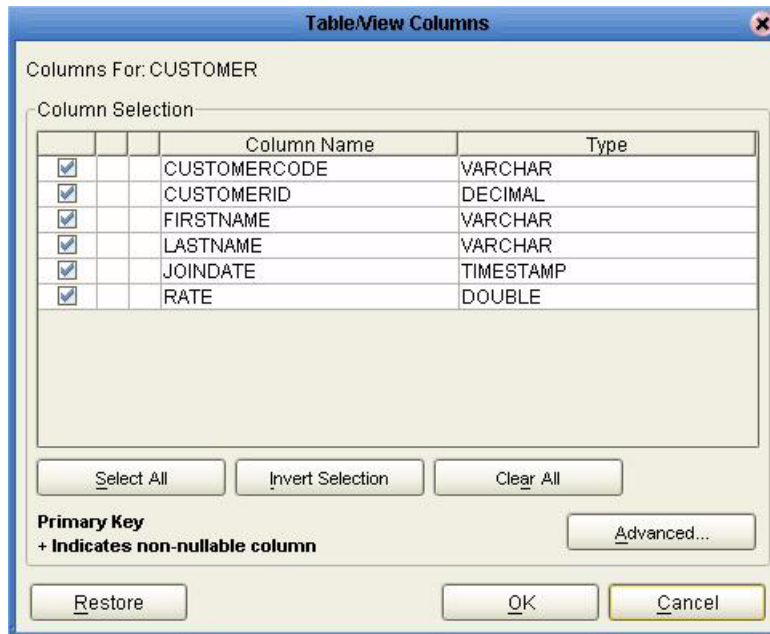
7 Click **Advanced** to change the data type, precision/length, or scale. In general, do not change the precision/length or the scale. Once you have finished your table choices, click **OK** (see Figure 12).

Figure 12 Table/View Columns — Advanced
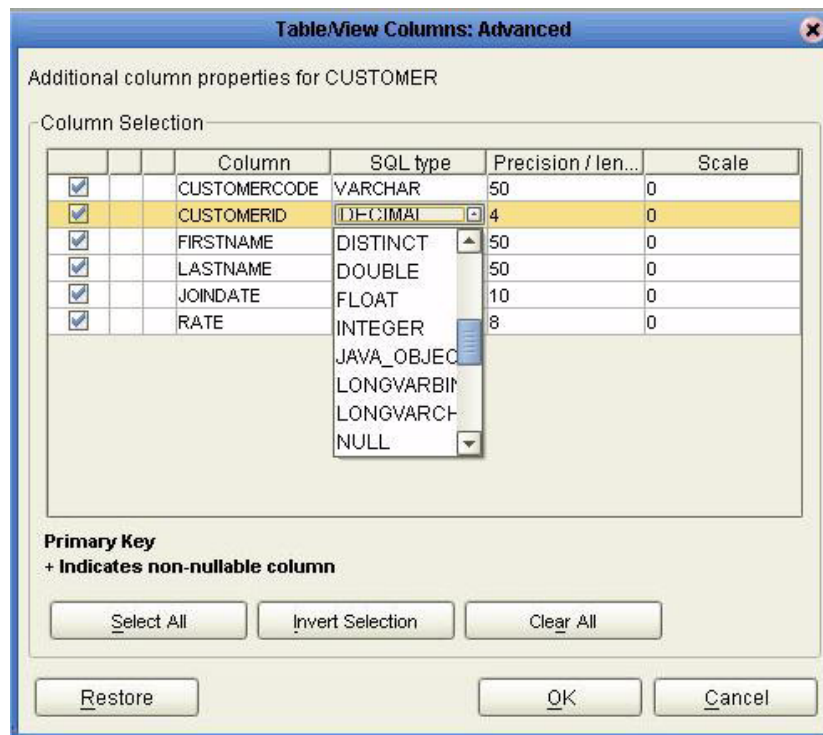
Select Procedures

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

Figure 13 Select Procedures and specify Resultset and Parameter Information
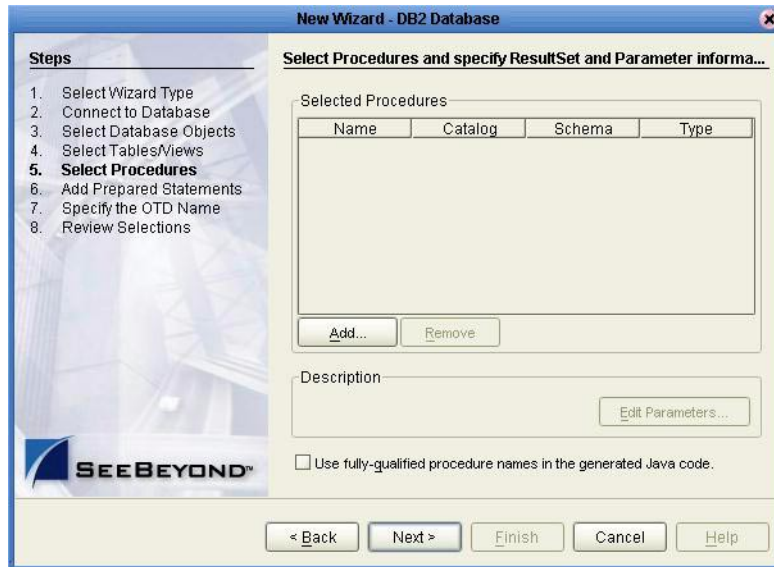
2 On the **Select Procedures** window, enter the name of a Procedure or select a table from the drop down list. Click **Search**. Wildcard characters can also be used.

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

Figure 14 Add Procedures
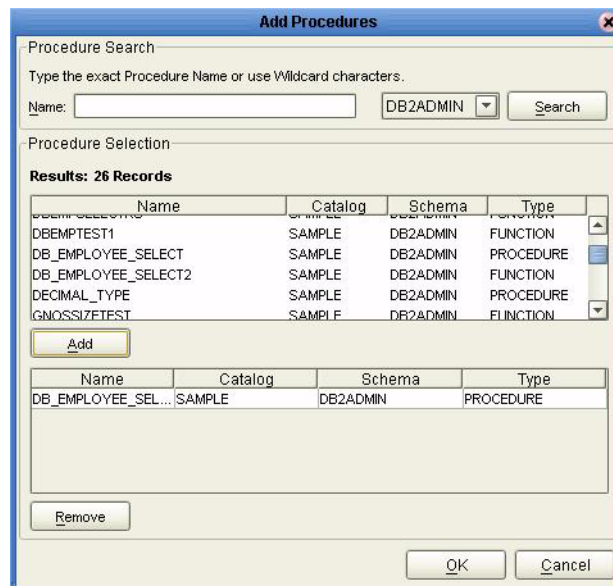
4 On the **Select Procedures and specify Resultset and Parameter Information** window click **Edit Parameters** to make any changes to the selected Procedure (see Figure 15).

Figure 15 Procedure Parameters
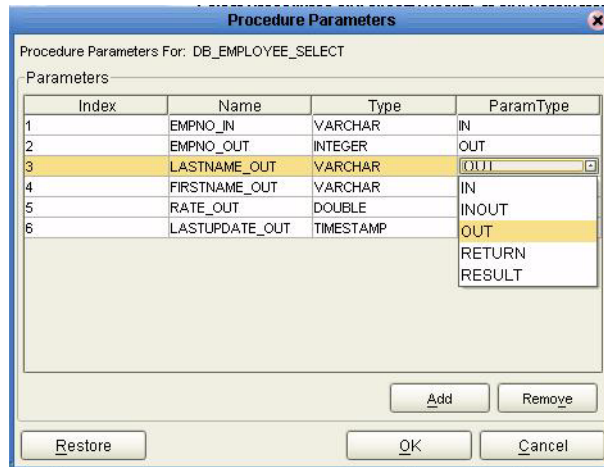
5 To restore the data type, click **Restore**. When finished, click **OK**.

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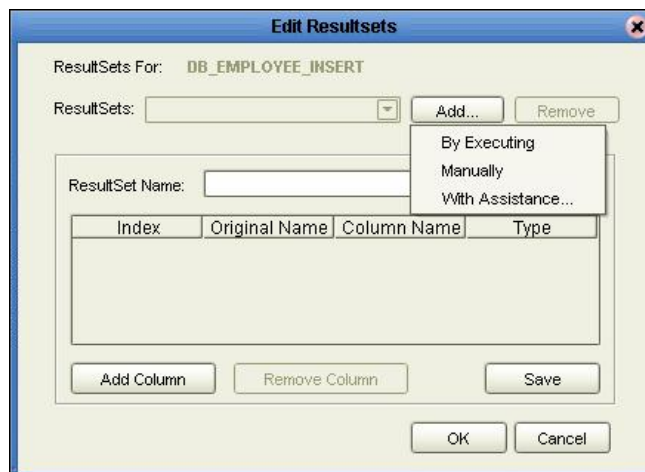
The DBWizard provides three different ways to generate the ResultSet nodes of a Stored Procedure. They are the "**By Executing**", "**Manually**", and "**With Assistance**" modes.

The "**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. If there are multiple ResultSets and the "**By Executing**" mode does not return all ResultSets, one will use the other modes to generate the ResultSet nodes.

The "**With Assistance**" mode enables 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 the "**Assist**" mode, highlight the execute statement up to and including the table name(s) before executing the query.

The "**Manually**" mode is the most flexible way to generate the result set nodes. It enables 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, the "**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

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

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

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

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

Figure 17 Prepared Statement
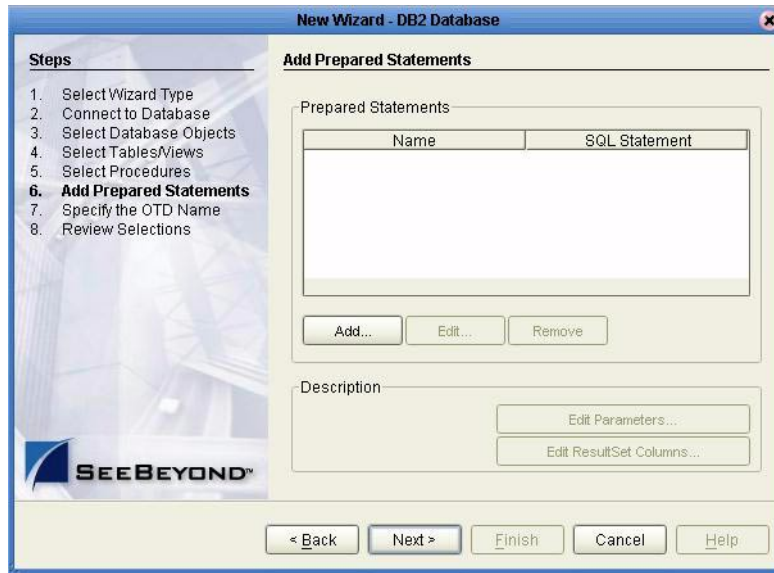
2 Enter the name of a Prepared Statement or create a SQL statement by clicking in the SQL Statement window. When finished creating the statement, click **Save As** giving the statement a name. This name appears as a node in the OTD. Click **OK** (see Figure 18).

Figure 18 Prepared SQL Statement
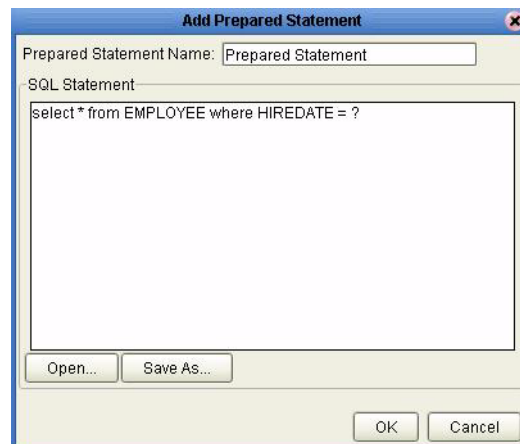
3 On the **Add Prepared Statement** window, the name you assigned to the Prepared Statement appears. To edit the parameters, click **Edit Parameters**. You can change the datatype by clicking in the **Type** field and selecting a different type from the list.

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

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

Figure 19 Edit the Prepared Statement Parameters
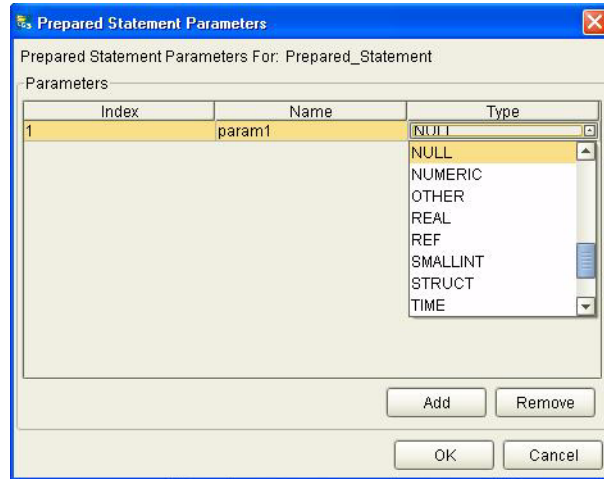
5   To edit the Resultset Columns, click **Edit Resultset Columns**. Both the Name and Type are editable. Click **OK** (see Figure 20).

Figure 20 ResultSet Columns
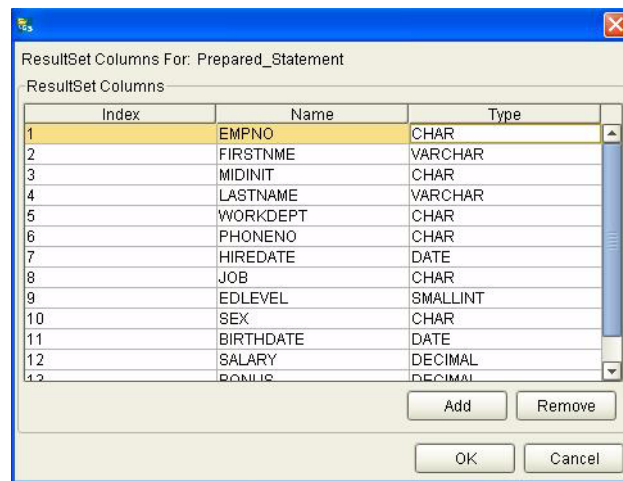
6   On the **Add Prepared Statements** window, click **OK**.

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 21).

Figure 21 Naming an OTD

2 View the summary of the OTD. Click **Back** to review previous screens, or click **Finish** to begin generating the OTD (see Figure 22).

Figure 22 Database Wizard - Summary

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

Implementing the DB2 eWay

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

What's in This Chapter

- **"eInsight Engine and eGate Components" on page 46**
- **"Using the Sample Project in eInsight" on page 46**
- **"Using the Sample Project in eGate" on page 57**
- **"Common DataType Conversions" on page 60**
- **"Using OTDs with Tables, Views, Stored Procedures, and Prepared Statements" on page 61**
- **"Alerting and Logging" on page 78**

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

Note: *For inbound BPEL Collaborations, use an eGate Scheduler. For more information on Schedulers see "eGate Integrator for eInsight Enterprise Service Bus Users Guide."*

5.2 Using the Sample Project in eInsight

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

Project Import utility. Import the **DB2_sampleBPEL.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 Integrator Tutorial*.

Importing the Sample Project

1 From 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 files.

   Sample Projects are contained within the zip file **DB2_eWay_Sample**, which is downloaded from the Repository to a folder of your choosing, (see **Installing the eWay** on page 11). Once downloaded, unzip the file and extract the following sample Projects:

   ◆ DB2_sampleBPEL.zip

   ◆ DB2_sampleJCE.zip

3 Select a sample Project zip file and click **Import**.

4 After importing the file, click **OK** on the Import Status window. You can now import another zip file, or click **Close** to exit the Import Manager window.

The Business Process

The data used for this sample project is contained within a table called DBEmployee. The table has 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	4
JOB	JOB	char	8
EDLEVEL	EDLEVEL	smallint	2
SEX	SEX	char	1
BIRTHDATE	BIRTHDATE	date	4

Column Name	Mapping	Data Type	Data Length
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

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

Figure 23   Sample Project Business Process
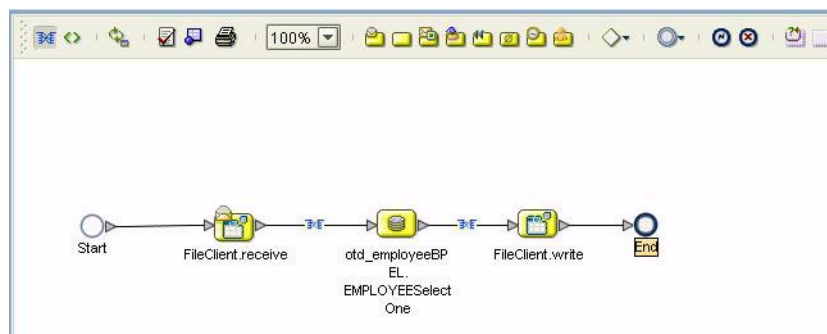
You can associate an eInsight Business Process Activity with the eWay, both during the system design phase and during run time. To make this association, select the desired **receive** or **write** operation under the eWay in the Enterprise Explorer and drag it onto the eInsight Business Process canvas. 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 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
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.
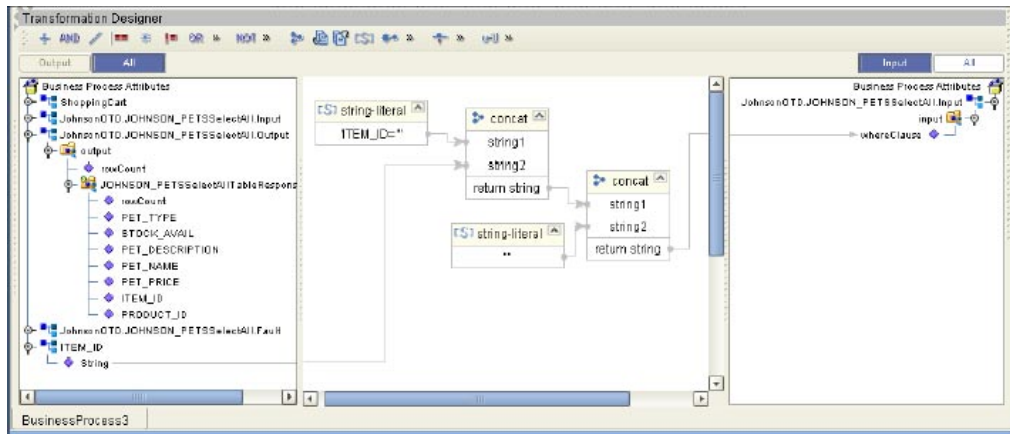
The figure below shows a sample eInsight Business Process using the SelectAll operation. In this process, the SelectAll operation returns all rows where the ITEM_ID matches the selected ITEM_ID to the shopping cart.
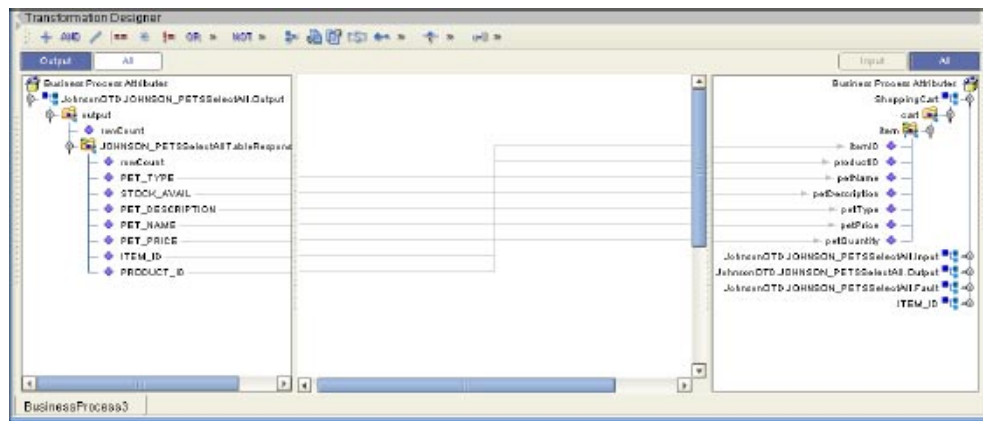
Figure 24 SelectAll Sample Business Process
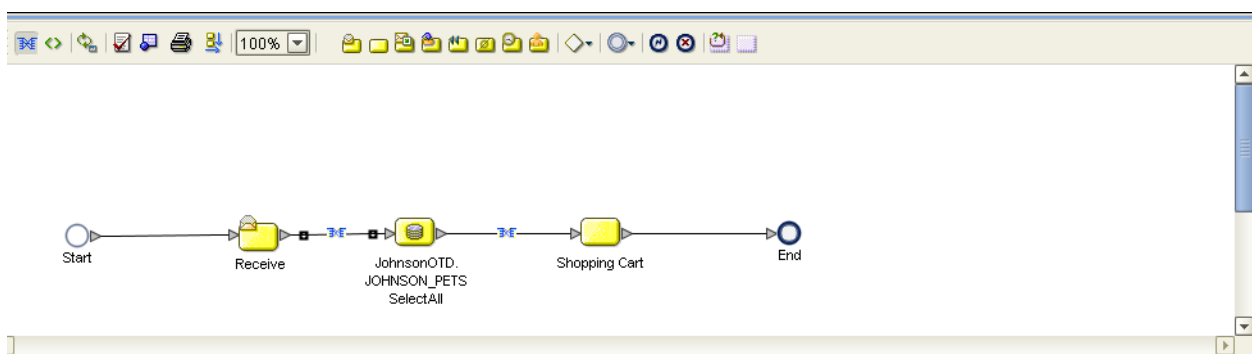
The figure below shows the definition of the where() clause for the SelectAll operation.

Figure 25   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 26   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 27   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 28   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.
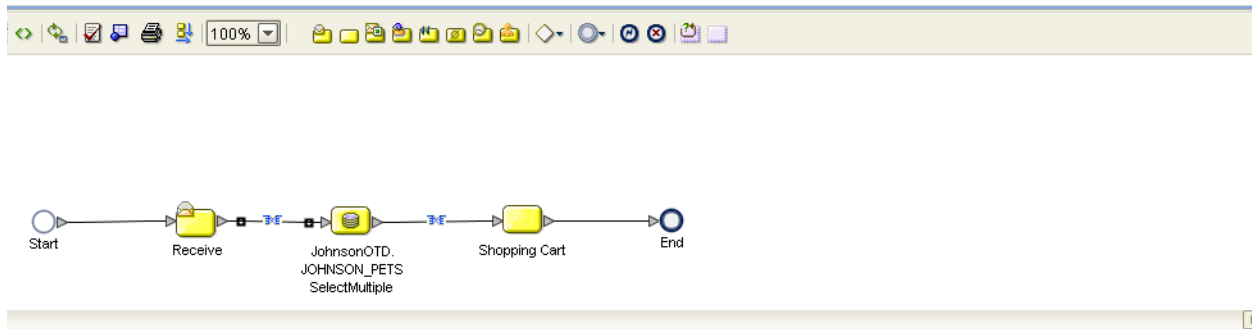
Figure 29   SelectMultiple Output

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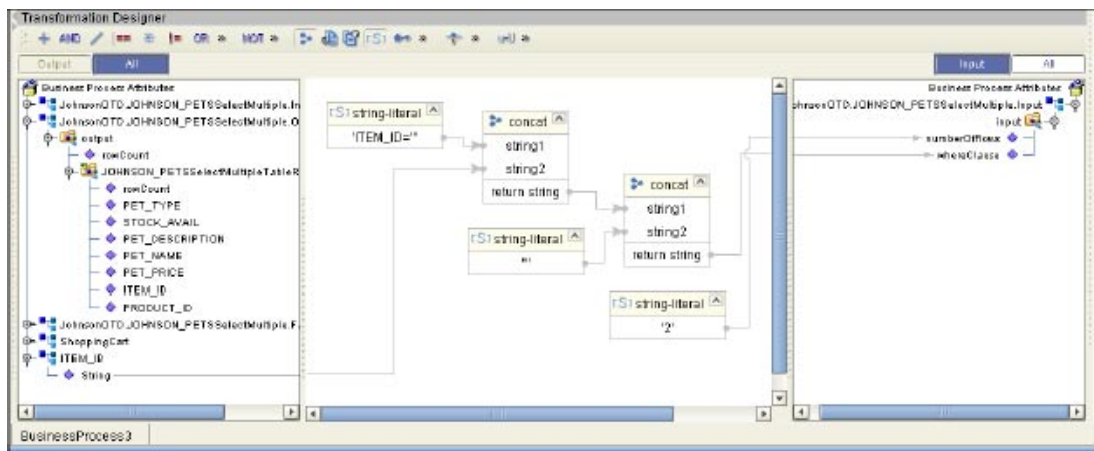
The figure below shows a sample eInsight Business Process using the SelectOne operation. In this process, the SelectOne operation returns the first row where the ITEM_ID matches the specified ITEM_ID to the shopping cart.

Figure 30   SelectOne Sample Business Process

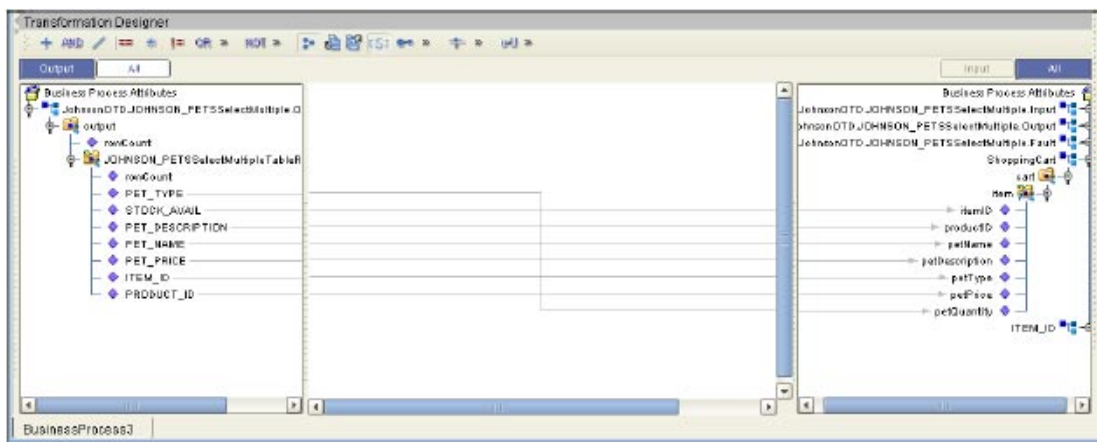The figure below shows the definition of the where() clause for the SelectOne operation.

Figure 31   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.
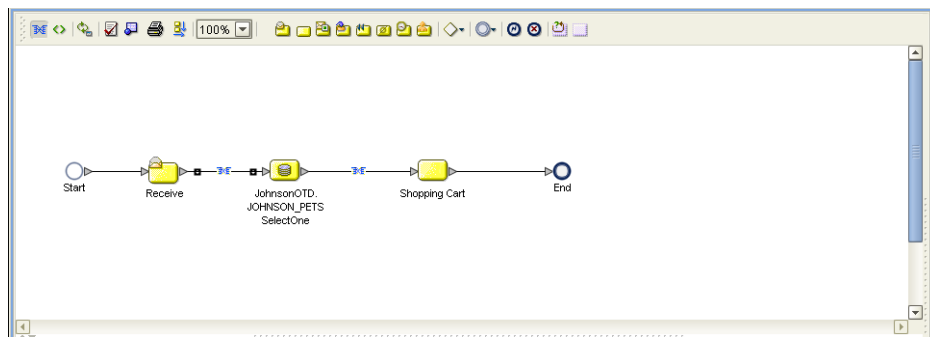
Figure 32   SelectOne Output

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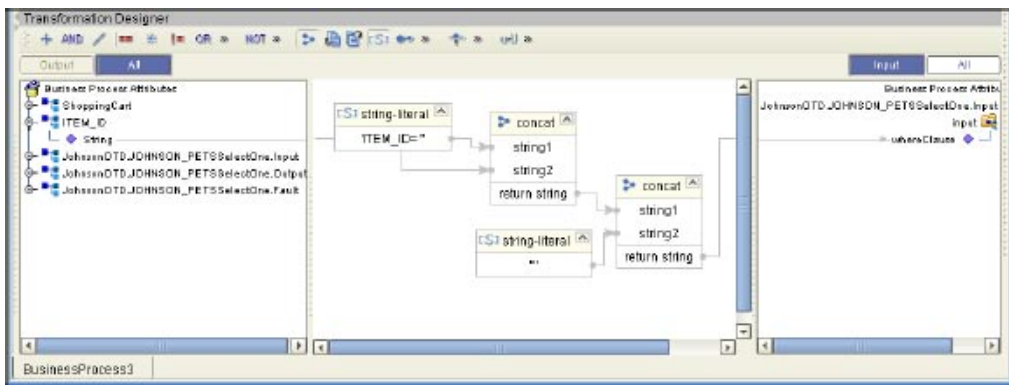
The figure below shows a sample eInsight Business Process using the Insert operation. In this process, the operation inserts a new row into the database to accommodate a new item provided by a vendor.

Figure 33   Insert Sample Business Process
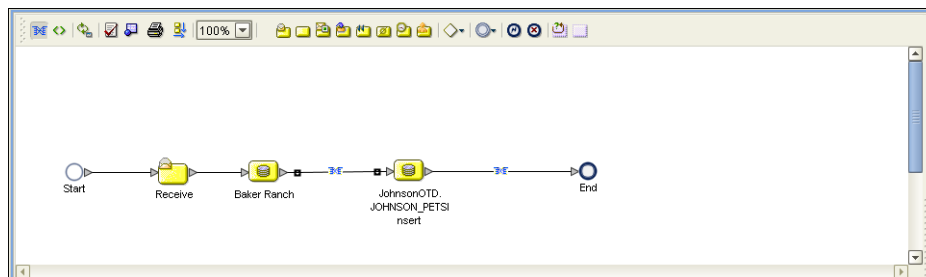
The figure below shows the definition of the input for the Insert operation.

Figure 34   Insert Input

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

Figure 35   Insert Output
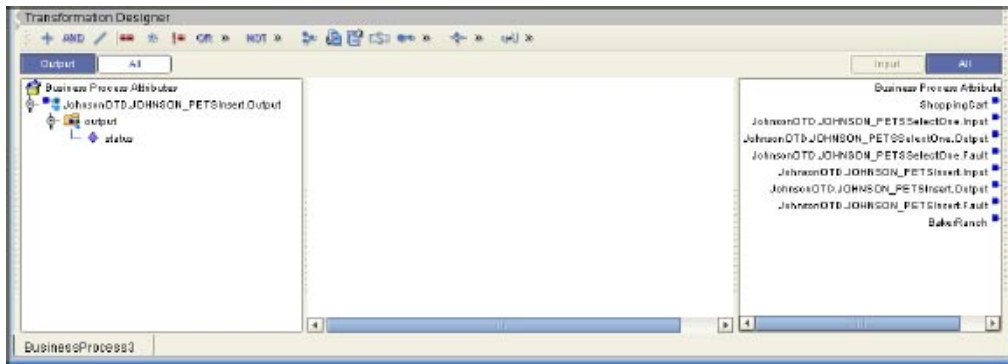
5.2.5 Update

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 36 Update Sample Business Process
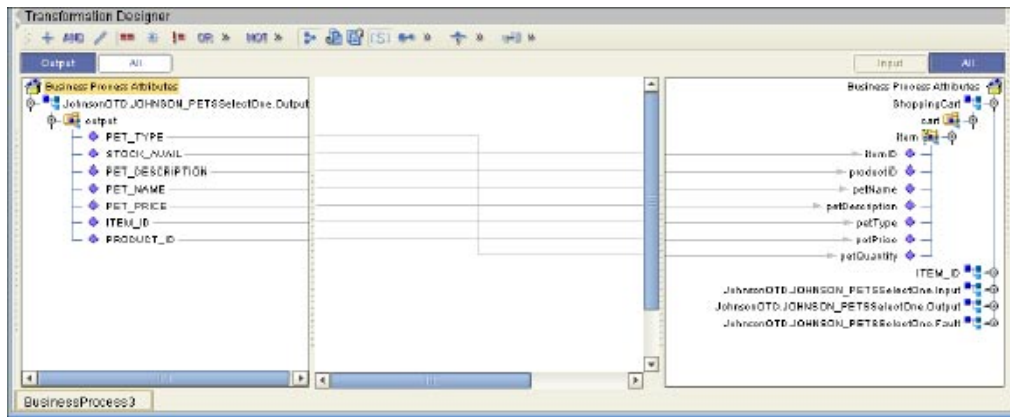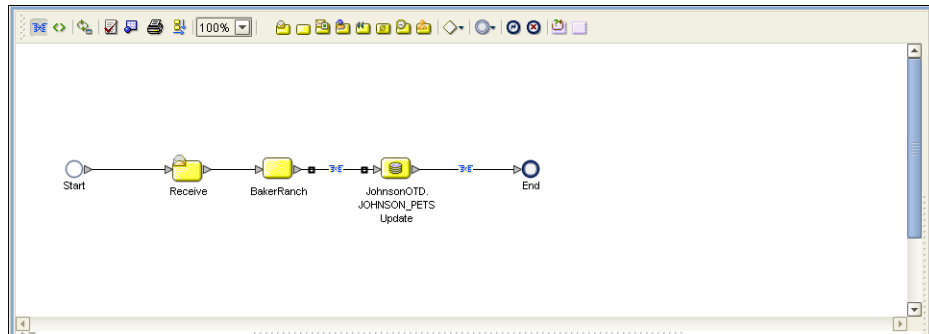

The figure below shows the definition of the where() clause for the Update operation.

Figure 37 Update Input
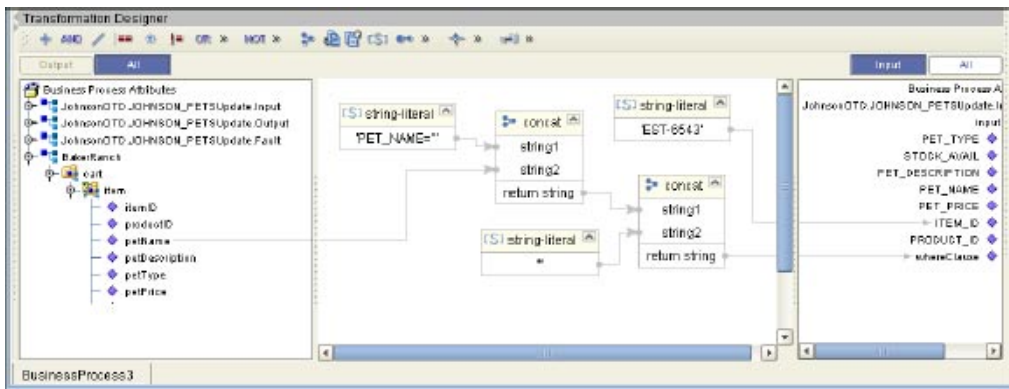
The figure below shows the output of the Update operation, which is a status indicating the number of rows updated.

Figure 38 Update Output
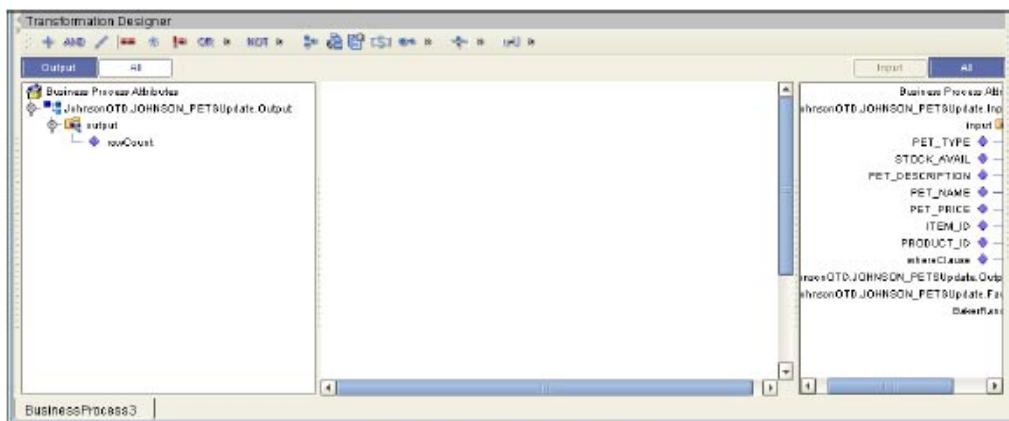
5.2.6 Delete

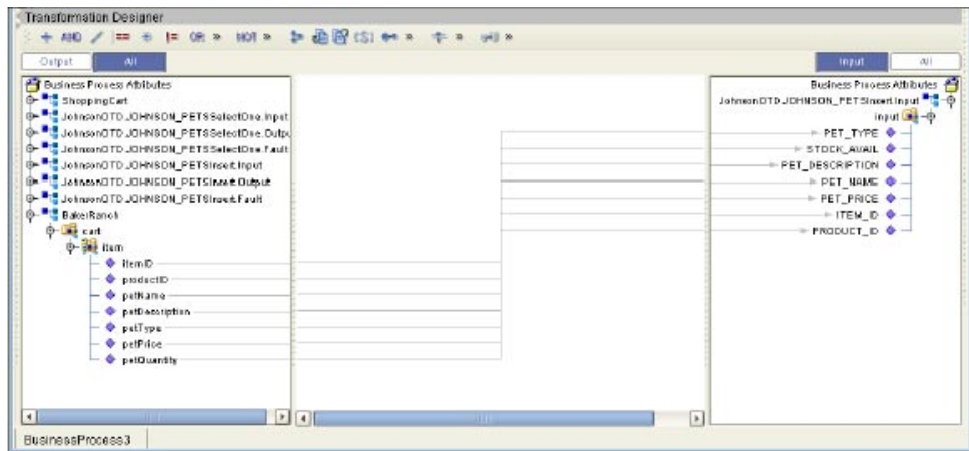The Delete operation deletes rows that match the criteria defined in a where() clause. The output is a status of how many rows where deleted.

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 39 Delete Sample Business Process
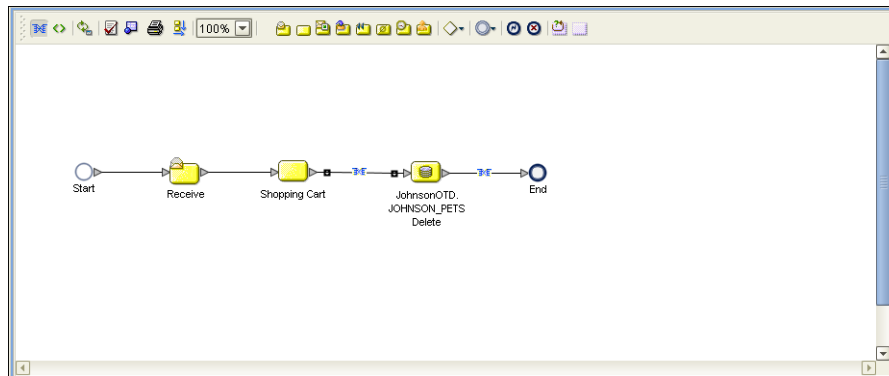
The figure below shows the definition of the where() clause for the Delete operation.

Figure 40 Delete Input
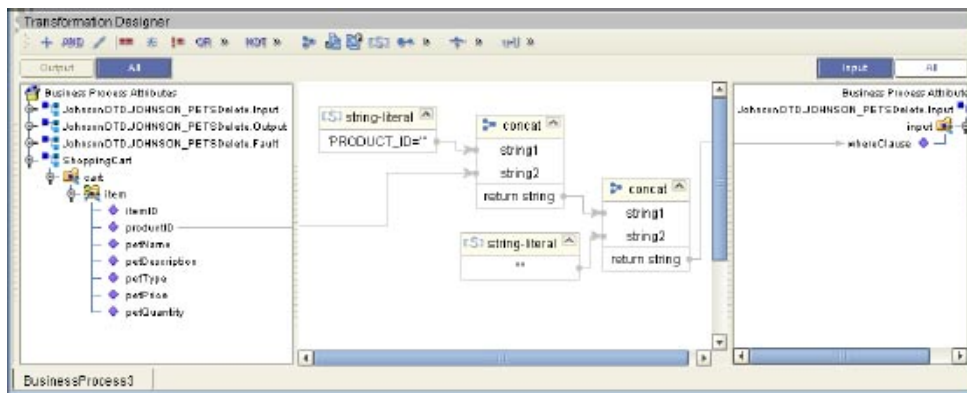
The figure below shows the output of the Delete operation, which is a status indicating the number of rows deleted.

Figure 41   Delete Output
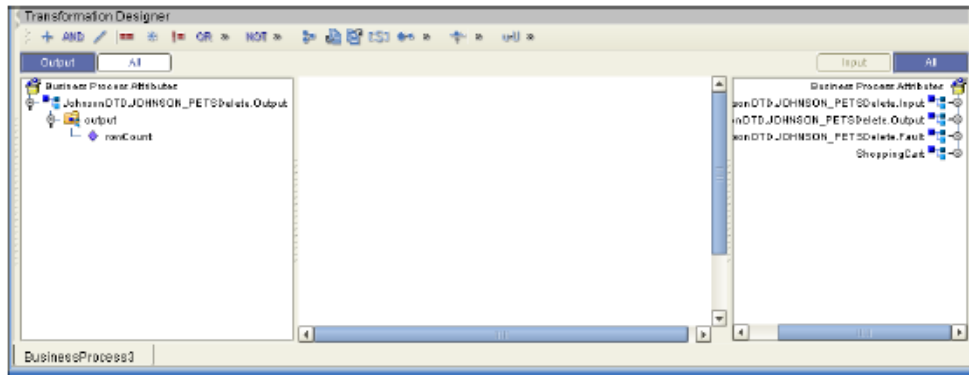
5.3   Using the Sample Project in eGate

To import the sample project **DB_sampleJCE.zip** follow the instructions given in **Importing the Sample Project** on page 47.

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 3   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	4
JOB	JOB	char	8
EDLEVEL	EDLEVEL	smallint	2
SEX	SEX	char	1
BIRTHDATE	BIRTHDATE	date	4
SALARY	SALARY	decimal	9
BONUS	BONUS	decimal	9

Column Name	Mapping	Data Type	Data Length
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 42 Database project flow
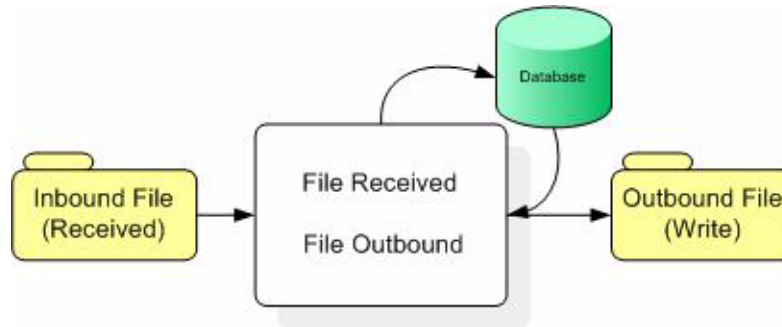
To work with the sample project, follow the instructions given in the *eGate Integrator Tutorial*.

5.3.2 Configuring the eWays

The sample uses an inbound and an outbound File eWay as well as an outbound DB2 eWay. To configure the sample projects eWays, use the following information. For additional information on the DB2 properties, see **Properties of DB2 eWay on Windows or Unix Operating Systems** on page 14.

To configure the Inbound File eWay:

1 On the Connectivity Map canvas, double click the eWay icon located between the **File1** and **Service1**.

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 DB2 eWay:

1 On the Connectivity Map canvas, double click the eWay icon located between the **Service1** and **DB2** database.

2 On the resulting **Templates** window, select the required Outbound or Inbound DB2 connection, and click **OK**.

3 On the Properties window, enter the appropriate configurations for the Outbound DB2 eWay and click **OK**. See **Properties of DB2 eWay on Windows or Unix Operating Systems** on page 14. 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 **Service1** and **File2** 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 is 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 Inbound and an Outbound DB2 eWay, and a Service.

To create the external environment for the Sample project:

1 On the Environment Explorer, right-click the DB2 profile.

2 Select **Properties**. Enter the configuration information required for your Outbound or Inbound DB2 eWay. See **Properties of DB2 eWay on Windows or Unix Operating Systems** on page 14.

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

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 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.

5.4 Common DataType Conversions

Table 4 The DB2 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)	123
Smallint	BigDecimal	**Call a New Constructor BigDecimal**: java.math.BigDecimal(String)	123
Number	BigDecimal	**Call a New Constructor BigDecimal**: java.math.BigDecimal(String)	123
Decimal	BigDecimal	**Call a New Constructor BigDecimal**: java.math.BigDecimal(String)	123
BigInteger	Long	**Long:** java.lang.Long.parseLong(String)	123
Short	Short	**Short**: java.lang.Short.parseShort(String)	123
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 5 The DB2 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
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, Stored Procedures, and Prepared Statements

Tables, Views, Stored Procedures, and Prepared Statements 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 enables 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_TypeToScrollSensitive();
getDBEmp().getDB_EMPLOYEE().select("");
```

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.

Note: *If you want to check if the last value read was SQL NULL or not, you can use the wasNULL() method. It is most useful for native data types like 'int'. A getxxx method should be called before wasNULL() is called.*

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();

//Table_OTD_1.commit
   Table_OTD_1.commit();
}
```

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();
}
//DB_sales_orders_1.commit
   DB_sales_orders_1.commit();

//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.

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( "'" ) );
}
```

5.5.2 Using Clobs

A Clob (Character Large Object) is a LOB datatype that has content consisting of character data in the database character set. The following describes how to Insert, Update, and Select Clobs using:

- Table OTD
- Prepared Statement OTD
- Procedure OTD

Inserting a Clob using a Table OTD

To Insert a Clob using a Table OTD, you must:

1 Invoke "Select" to get the ResultSet.

```
//@map:TEST.select("")
TESTCLOB_1.getTEST().select( "" );
```

2 Move to a new row (to be inserted).

```
//@map:TEST.moveToInsertRow
TESTCLOB_1.getTEST().moveToInsertRow();
```

3 Set the values to the CLOB field.

```
//@map:Copy Text(java.lang.String) to TEXT
TESTCLOB_1.getTEST().setTEXT( input.getText() );
```

4 Insert the row.

```
//@map:TEST.insertRow
TESTCLOB_1.getTEST().insertRow();
```

The Complete JCE Code Appears as:

```
public class TESTCLOB_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,tESTCLOB.TESTCLOBOTD TESTCLOB_1 )
    throws Throwable
    {
        //@map:TEST.select("")
        TESTCLOB_1.getTEST().select( "" );

        //@map:TEST.moveToInsertRow
        TESTCLOB_1.getTEST().moveToInsertRow();

        //@map:Copy Text to TEXT
```

```
                    TESTCLOB_1.getTEST().setTEXT( input.getText() );

                    //@map:TEST.insertRow
                    TESTCLOB_1.getTEST().insertRow();
            }

        }
```

Inserting a Clob using a Prepared Statement OTD

To Insert a Clob using a Prepared Statement OTD, you must:

1 Set the values to the Prepared Statement parameter.

```
//@map:Copy Text (java.lang.String) to Param1 (Clob Column)
InsertClobPrepStat_1.getInsertClobPrepStat().setParam1(
input.getText() );
```

2 Execute the Prepared Statement

```
//@map:InsertClobPrepStat.executeUpdate
InsertClobPrepStat_1.getInsertClobPrepStat().executeUpdate();
```

The Complete JCE Code Appears as:

```
public class CLOB_PREPSTAT_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,insertClobPrepStat.InsertClobPrepStatOTD
InsertClobPrepStat_1 )
    throws Throwable
    {
        //@map:Copy Text to Param1
        InsertClobPrepStat_1.getInsertClobPrepStat().setParam1(
input.getText() );

        //@map:InsertClobPrepStat.executeUpdate
        InsertClobPrepStat_1.getInsertClobPrepStat().executeUpdate();
    }

}
```

Inserting a Clob using a Stored Procedure OTD

To Insert a Clob using a Stored Procedure OTD, you must:

1 Set the values to the Stored Procedure OTD arguments.

```
//@map:Copy Text to CLOBVALUE
TEST_CLOB_STORED_PROC_1.getINSERTCLOB().setCLOBVALUE(
input.getText() );
```

2 Execute the Store procedure.

```
//@map:INSERTCLOB.execute
TEST_CLOB_STORED_PROC_1.getINSERTCLOB().execute();
```

The Complete JCE Code Appears as:

```
public class TESTCLOB_STOREDPROC
{

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

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

    public void receive(
com.stc.connector.appconn.file.FileTextMessage
            input,tEST_CLOB_STORED_PROC.TEST_CLOB_STORED_PROCOTD
            TEST_CLOB_STORED_PROC_1 )
    throws Throwable
    {
        //@map:Copy Text to CLOBVALUE
        TEST_CLOB_STORED_PROC_1.getINSERTCLOB().setCLOBVALUE(
                                            input.getText() );

        //@map:INSERTCLOB.execute
        TEST_CLOB_STORED_PROC_1.getINSERTCLOB().execute();
    }

}
```

Updating a Clob using a Table OTD

To Update a Clob using a Table OTD, you must:

1 Invoke "Select" to get the ResultSet.

```
//@map:TEST.select("")
TESTCLOB_1.getTEST().select( "" );
```

2 Move to the to-be-updated row.

```
//@map:TEST.next
TESTCLOB_1.getTEST().next();
```

3 Set the values to the CLOB field.

```
//@map:Copy Text(java.lang.String) to TEXT
TESTCLOB_1.getTEST().setTEXT( input.getText() );
```

4 Update the row.

```
//@map:TEST.insertRow
TESTCLOB_1.getTEST().updateRow();
```

The Complete JCE Code Appears as:

```
public class TESTCLOB_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,tESTCLOB.TESTCLOBOTD TESTCLOB_1 )
    throws Throwable
    {
        //@map:TEST.select("")
        TESTCLOB_1.getTEST().select( "" );

        //@map:TEST.next
```

```
             TESTCLOB_1.getTEST().next();

             //@map:Copy Text to TEXT
             TESTCLOB_1.getTEST().setTEXT( input.getText() );

             //@map:TEST.insertRow
             TESTCLOB_1.getTEST().updateRow();
        }

    }
```

Updating a CLOB using a Stored Procedure or Prepared Statement OTD

Updating a CLOB using a Stored Procedure or Prepared Statement is similar to the procedures for Inserting a CLOB. For more information, see **"Inserting a Clob using a Prepared Statement OTD" on page 66** and **"Inserting a Clob using a Stored Procedure OTD" on page 66**.

Selecting a Clob using a Table OTD

To Select a Clob using a Table OTD, you must:

1 Select desired Rows.

```
//@map:TEST.select("")
TESTCLOB_1.getTEST().select( "" );
```

2 Scroll to the desired row.

```
//@map:TEST.next
TESTCLOB_1.getTEST().next();
```

3 Declare a variable of type java.sql.Clob.

```
//@map:java.sql.Clob clobValue;
java.sql.Clob clobValue;
```

Note: *The getXXX() method of the OTD returns the Clob as java.lang.Object. This needs to be converted to java.sql.Clob before fetching the actual value from the Field. A local variable must be declared to facilitate conversion.*

To declare a variable of type java.sql.Clob, you must first:

A Open the **Create a Variable** window, as seen in Figure 43.

Figure 43   Create a Local Variable
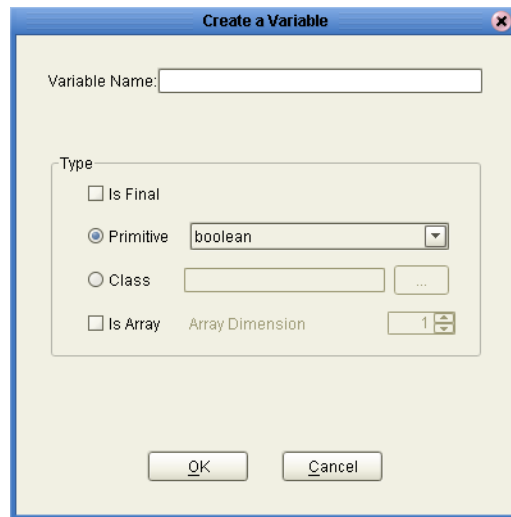
B Select the **Class** button and then click the ellipsis (**...**) button to search for a Class.

C In the **Find Class** window, locate the Clob class and click **OK,** as seen in Figure 44.

Figure 44   Find Class window
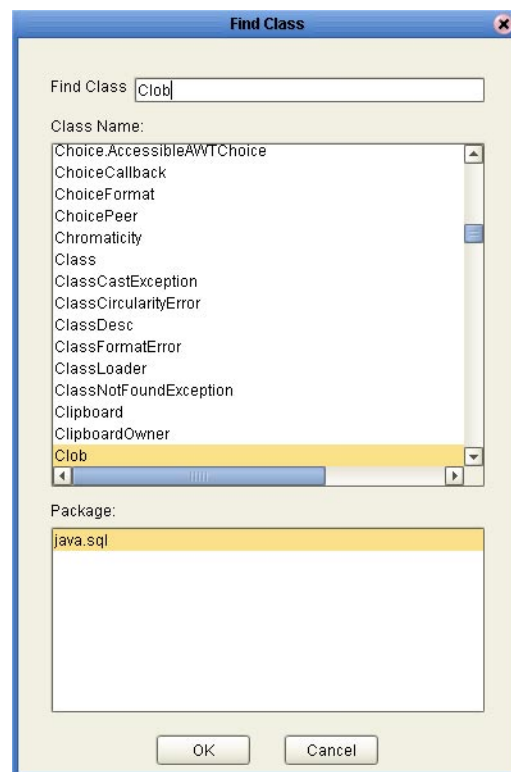
D The java.sql.Clob class appears in the Create a Variable window. Click **OK** to create the new variable, as seen in Figure 45.

Figure 45   Create Local Variable java.sql.Clob
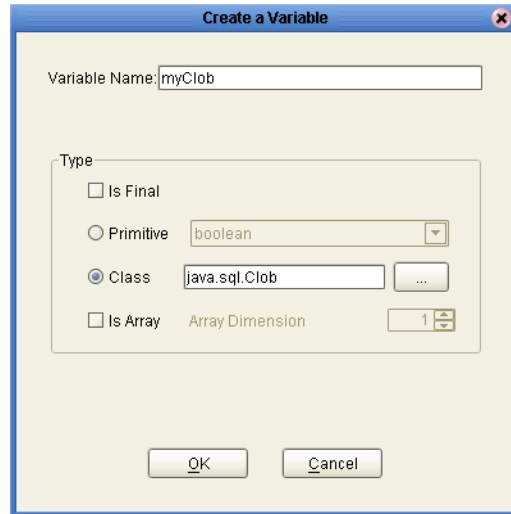
4   Convert (cast) and assign the value of the Clob field to the local variable.

```
//@map:Copy cast TEXT (clob field) to java.sql.Clob to clobValue
clobValue = (java.sql.Clob) TESTCLOB_1.getTEST().getTEXT();
```

Figure 46 displays the Cast window.

Figure 46   Convert using the Cast() Method

Figure 47 displays assigning the value to the variable.

Figure 47   Assigning to the clobValue Variable
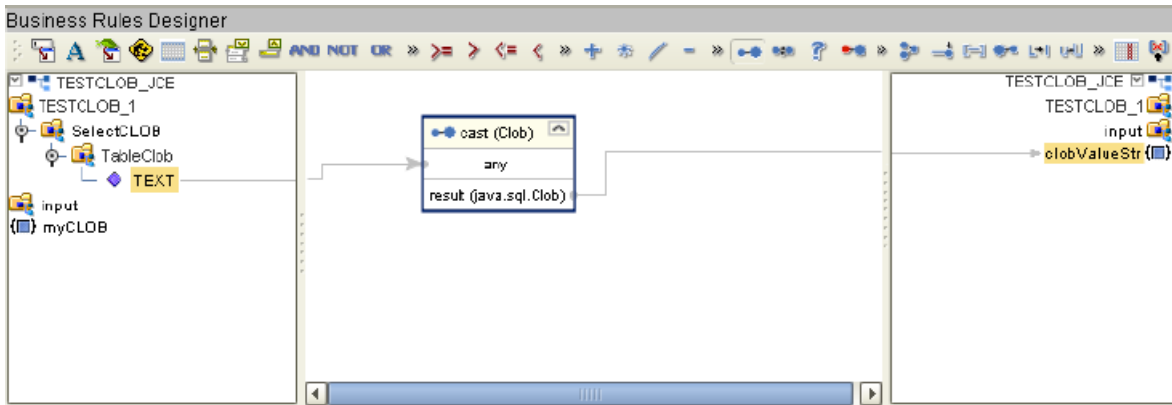
5   Declare the variable of type String.

```
//@map:String clobValueStr;
String clobValueStr;
```

6   Fetch the Clob value to the String variable.

```
//@map:Copy SubString(1 ,cast clobValue.length to int) to
//clobValueStr
clobValueStr = clobValue.getSubString( 1,(int)clobValue.length() );
```

Note: *When using the Business Rules Designer, you must add the cast prior to adding the length() method.*

Figure 48 displays Fetching the Clob value to the String variable.

Figure 48   Copy the Clob Value to a String Variable
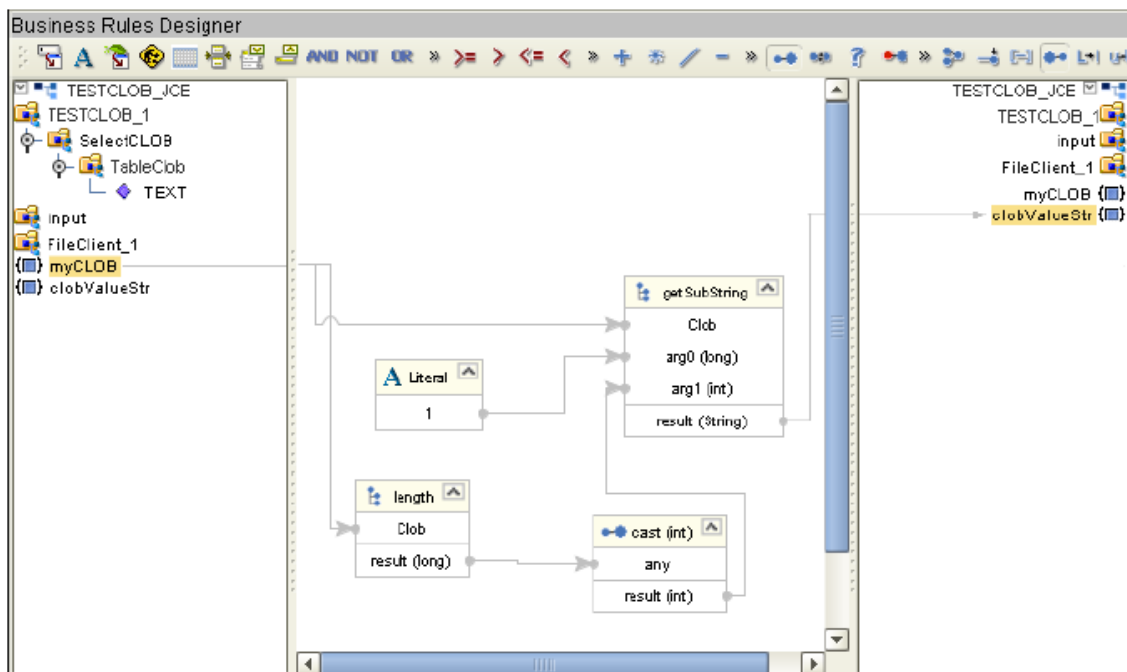
The Complete JCE Code Appears as:

```
public class TESTCLOB_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,tESTCLOB.TESTCLOBOTD TESTCLOB_1 )
    throws Throwable
    {
        //@map:TEST.select("")
        TESTCLOB_1.getTEST().select( "" );

        //@map:TEST.next
        TESTCLOB_1.getTEST().next();

        //@map:java.sql.Clob clobValue;
        java.sql.Clob clobValue;

        //@map:Copy cast TEXT to java.sql.Clob to clobValue
        clobValue = (java.sql.Clob) TESTCLOB_1.getTEST().getTEXT();

        //@map:String clobValueStr;
        String clobValueStr;

        //@map:Copy SubString(1 , cast clobValue.length to
        // int) to clobValueStr
        clobValueStr = clobValue.getSubString( 1,(int)
                                        clobValue.length() );
    }

}
```

Selecting a Clob using a Prepared Statement

To Select a Clob using a Prepared Statement OTD, you must:

1 Execute the query.

```
//@map:SelectClobPrepStat.executeQuery
SelectClobPrepStat_1.getSelectClobPrepStat().executeQuery();
```

2 Declare the variable of type java.sql.Clob.

```
//@map:java.sql.Clob clobValue;
java.sql.Clob clobValue;
```

Note: *The getXXX() method of the OTD resultSet returns the clob as java.lang.Object.*
This needs to be converted to java.sql.Clob before fetching the actual value from the
Field. A local variable must be declared to facilitate conversion. For additional
details, see the figures found in **Selecting a Clob using a Table OTD** *on page 68.*

3 Convert (cast) and assign the value of the Clob field to the local variable.

```
//@map:Copy cast TEXT to java.sql.Clob to clobValue
//clobValue = (java.sql.Clob)
SelectClobPrepStat_1.getSelectClobPrepStat().
get$SelectClobPrepStatResults().getTEXT();
```

4 Declare the variable of type String

```
//@map:String clobValueStr;
String clobValueStr;
```

5 Fetch the clob value to the String variable

```
//@map:Copy SubString(1 ,cast clobValue.length to int) to
//clobValueStr
clobValueStr = clobValue.getSubString( 1,(int)
clobValue.length() );
```

The Complete JCE Code Appears as:

```
public class SELECTCLOB_PREPSTAT_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,selectClobPrepStat.SelectClobPrepStatOTD
                        SelectClobPrepStat_1 )
    throws Throwable
    {
        //@map:SelectClobPrepStat.executeQuery
        SelectClobPrepStat_1.getSelectClobPrepStat().executeQuery();

        //while
        while
(SelectClobPrepStat_1.getSelectClobPrepStat().get$SelectClobPrepStatR
esults().next()) {
            //@map:java.sql.Clob clobValue;
            java.sql.Clob clobValue;

            //@map:Copy cast TEXT to java.sql.Clob to clobValue
            clobValue = (java.sql.Clob)
SelectClobPrepStat_1.getSelectClobPrepStat().get$SelectClobPrepStatRe
sults().getTEXT();

            //@map:String clobValueStr;
            String clobValueStr;

            //@map:Copy SubString(1,cast clobValue.length to int) to
clobValueStr
            clobValueStr = clobValue.getSubString( 1,(int)
clobValue.length() );
        }
    }

}
```

5.5.3 The Stored Procedure

A Stored Procedure OTD represents a database stored procedure. The OTD enables you to execute a stored procedure, with fields corresponding to the arguments of a stored procedure and methods representing the operations that you can apply. Remember that while in the Collaboration Editor you can drag and drop nodes from the OTD into the Collaboration Editor.

Note: *When creating a Package Stored Procedure in the Database Wizard, you must select* ***Use fully qualified names****.*

Note: *Stored Procedure Resultsets are supported in Java collaborations only.*

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();
    }
}
```

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 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.

Update Counts information which is returned from Stored Procedures is often insignificant. You should process returned ResultSet information and avoid looping through all of the Update Counts. The following three methods control exactly what information is 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()** method causes **resultsAvailable()** to return true only if an Update Count is available. The default case of the **enableResultsetsAndUpdateCount()** method allows both ResultSets and Update Counts to be returned.

Note: *Because a Stored Procedure returns a Result set based on the input parameter, the execute method may not detect a result set in certain cases. If this occurs, you must use Manual mode.*

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() 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 that allows for the monitoring of messages. This feature also 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