SeeBeyond ICAN Suite

Informix eWay Intelligent Adapter User's Guide

Release 5.0.1

Version 20040429150513.

eWay Contents

Chapter 1

Introduction

This document describes how to install and configure the eWay Intelligent Adapter for Informix.

This Chapter Includes:

1.1 Overview

The Informix eWay enables eGate Integrator Projects to exchange data with an external Informix Dynamic Server (IDS) database. IDS is a best-of-breed, general-purpose online transaction processing (OLTP) database that is designed for enterprise and workgroup platforms. This user's guide describes how to install and configure the Informix eWay.

1.2 Supported Operating Systems

The Informix eWay is available on the following operating systems:

- Windows XP, Windows 2000, and Windows Server 2003
- Solaris 8 and 9
- AIX 5.1L and 5.2
- HP-UX 11.0 and HP-UX 11.i (PA-RISC)
- Red Hat Linux 8 (Intel)
- Red Had Linux Advanced Server 2.1 (Intel)

Although the Informix eWay, the Repository, and Logical Hosts run on the platforms listed above, the Enterprise Designer requires the Windows operating system. Enterprise Manager can run on any platform that supports Internet Explorer 6.0.

1.3 System Requirements

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

1.4 External System Requirements

The Informix eWay supports the following software for external systems running eGate Projects.

- Informix Dynamic Server (IDS), version 9.2, 9.3, and 9.4.

Chapter 2

Installing the Informix eWay

This chapter describes how to install the Informix eWay.

2.1 Installing the Informix 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.

Note: *Refer to the ICAN Installation Guide for additional installation instructions.*

Required Files

During the procedures for uploading files to the eGate Repository using the Enterprise Manager, select and upload the following files:

- **InformixeWay.sar** (to install the Informix eWay)
- **FileeWay.sar** (to install the File eWay, used in the sample Projects)
- **InformixeWayDocs.sar** (to install the Informix eWay documentation)

Sample Projects

The Informix eWay also includes sample Projects for demonstration purposes.

To Download Sample Projects:

1. In the Enterprise Manager, click the **DOCUMENTATION** tab.
2. Click **Informix eWay**.
3. In the right window pane, click Download Sample, and select a location to save the .zip file.

Additional information on importing and using sample Projects are found in **Locating, Importing, and Using Sample Projects** on page 46.

Chapter 4

Setting Properties of the Informix eWay

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

This Chapter Includes:

4.1 Creating and Configuring an Informix eWay

All eWays contain a unique set of default configuration parameters. After the eWays are established and an Informix External System is created in the Project's Environment, the eWay parameters are modified for your specific system. The Informix eWay configuration parameters are modified from two locations:

- From the **Connectivity Map**—which contains parameters specific to the Informix eWay, and may vary from other eWays (of the same type) in the Project.
- From the **Environment Explorer** tree—which contains global parameters that commonly apply to all eWays (of the same type) in the Project. Saved parameters are shared by all eWays in the Informix External System Properties window.

Note: *You must set configuration parameters for the Informix eWay in both locations.*

4.2 Configuring the eWay Connectivity Map Properties

When you connect an External Application to a Collaboration, Enterprise Designer automatically assigns the appropriate eWay to the link. Each eWay is supplied with a template containing default configuration properties that are accessible on the Connectivity Map.

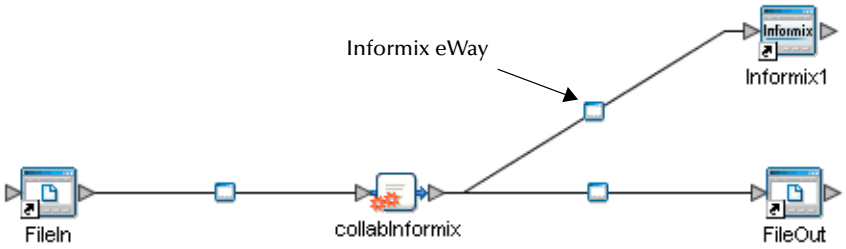
Templates supplied with the Informix eWay include:

- Outbound Informix eWay
- Inbound Informix eWay
- Outbound Informix XA eWay
- Outbound Informix non-Transactional eWay

To configure the eWay properties:

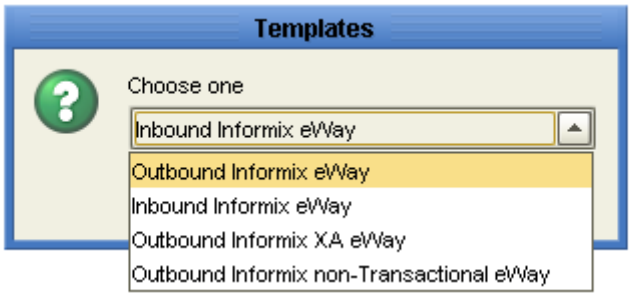
1 On the Enterprise Designer's Connectivity Map (see Figure 1), double-click the inbound Informix eWay icon. The Templates window appears.

Figure 1 Connectivity Map with Components



2 Select a parameter from the list and click **OK** .

Figure 2 Template window

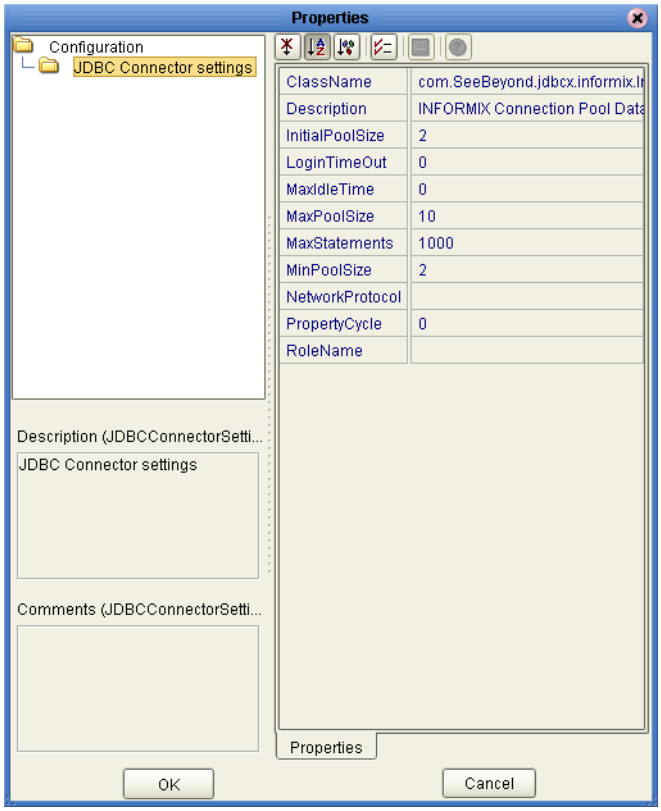


3 The Configuration properties window opens, displaying the default properties for the eWay.

4.2.1. Configuring the Outbound eWay Properties

The Outbound eWay Properties include outbound parameters used by the external database.

Figure 3 Outbound 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.SeeBeyond.jdbcx.informix.InformixDataSource.**

Description

Description

Enter a description for the database.

Required Value

A valid string. The default is **INFORMIX Connection Pool Datasource**.

InitialPoolSize

Description

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

Required Value

A valid numeric value. The default is **2**.

LoginTimeOut

Description

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

Required Value

A valid numeric value. The default is **0**.

MaxIdleTime

Description

Specifies 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. The default is **0**.

MaxPoolSize

Description

Specifies 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. The default is **10**.

MaxStatements

Description

Specifies 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. The default is **1000**.

MinPoolSize

Specifies 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. The default is **2**.

NetworkProtocol

Description

Specifies the network protocol used to communicate with the server.

Required Values

Any valid string.

PropertyCycle

Description

Determines 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. The default is **0**.

RoleName

Description

Specifies an initial SQL role name.

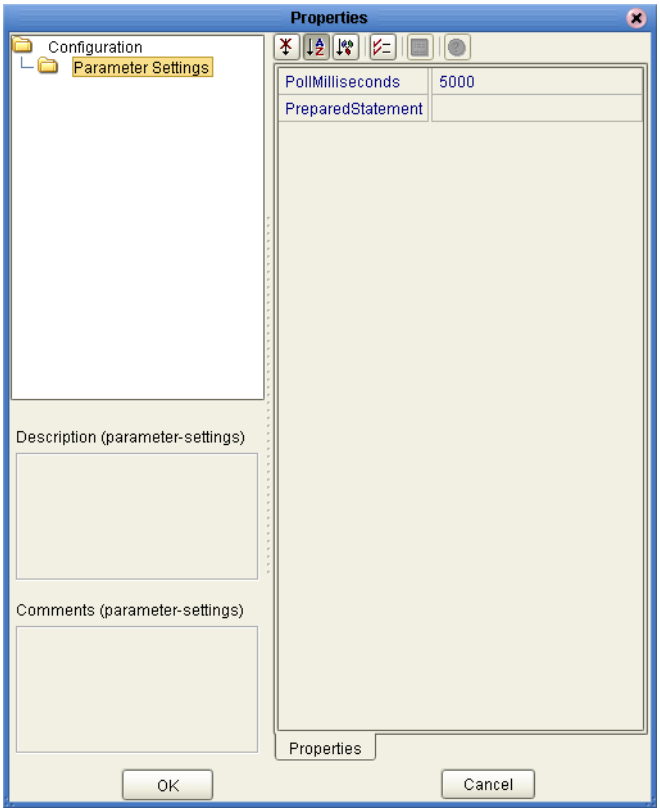
Required Values

Any valid string.

4.2.2. Configuring the Inbound eWay Properties

The Inbound eWay Properties include inbound parameters used by the external database.

Figure 4 Properties of the Inbound eWay



Pollmilliseconds

Description

Specifies the polling interval in milliseconds.

Required Value

A valid numeric value. The default is **5000** (5 seconds).

PreparedStatement

Description

Specifies the Prepared Statement used to query against the database.

Required Value

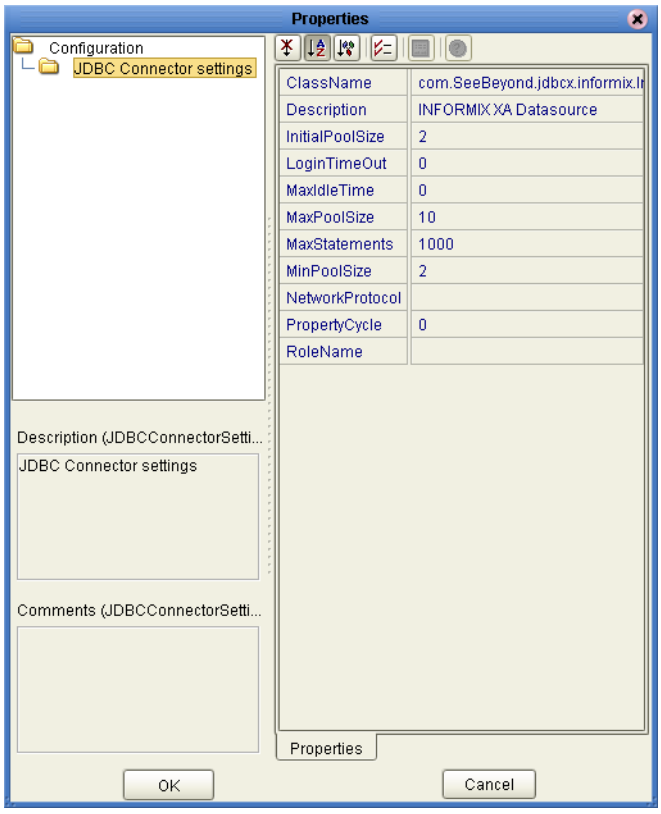
The Prepared Statement must be the same Prepared Statement you created using the Database OTD Wizard. Only **SELECT** Statement is allowed. Additionally, no place holders should be specified. There should not be any "?" in the Prepared Query.

Note: *The SQL used to create the Prepared Statement must exactly match the PreparedStatement listed in the Properties window.*

4.2.3. Configuring the Outbound Informix XA eWay Properties

The Outbound XA eWay Properties include inbound parameters used by the external database. Informix supports XA, a data source that provides connections that can participate in a distributed transaction. XA is a two-phase commit protocol that forms part of the JDBC 2.0 Standard Extension.

Figure 5 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.SeeBeyond.jdbcx.informix.InformixDataSource**.

Description

Description

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

Required Value

A valid string.

InitialPoolSize

Description

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

Required Value

A valid numeric value. The default is **2**.

LoginTimeOut

Description

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

Required Value

A valid numeric value. The default is **0**.

MaxIdleTime

Description

Specifies 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. The default is **0**.

MaxPoolSize

Description

Specifies 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. The default is **10**.

MaxStatements

Description

Specifies 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. The default is **1000**.

MinPoolSize

Description

Specifies 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. The default is **2**.

NetworkProtocol

Description

Specifies the network protocol used to communicate with the server.

Required Values

Any valid string.

PropertyCycle

Description

Specifies 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. The default is **0**.

RoleName

Description

Specifies an initial SQL role name.

Required Values

Any valid string.

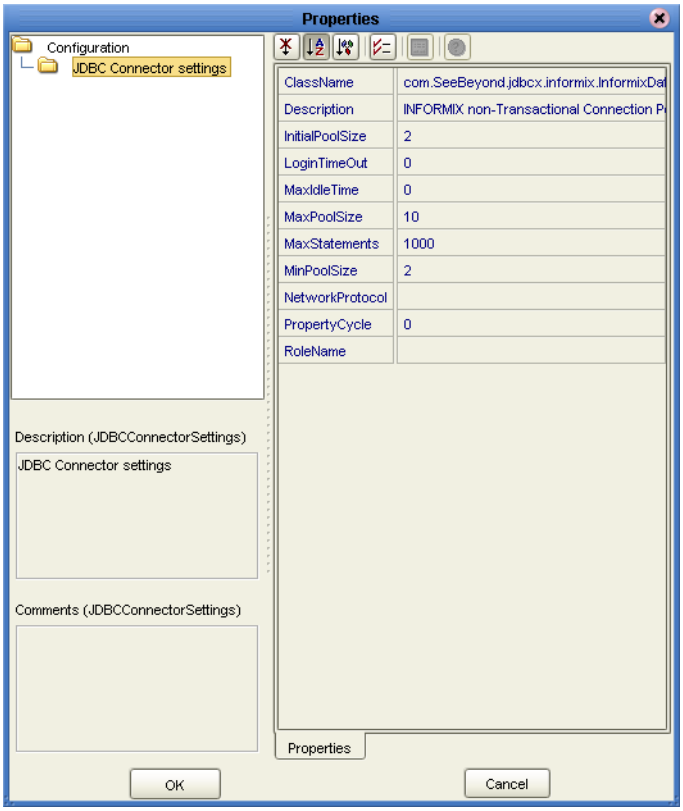
4.2.4. Configuring the Outbound Informix non-Transactional eWay Properties

You can create Informix databases with or without logging enabled. If logging is disabled, then Non-Transactional mode must be used. Because data logs are not retained during Non-Transactional execution of SQL calls, data recovery is not possible during accidental or unscheduled shut-down of the database server.

Disabled logging also prevents transactions—enclosed in BEGIN-Tran and END-Tran statements—from occurring. This means that Non-Transactional mode cannot be used in XA (2 phase commit) transactions.

The Outbound non-Transactional eWay Properties listed in Figure 6 include inbound parameters used by the external database.

Figure 6 Outbound non-Transactional 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.SeeBeyond.jdbcx.informix.InformixDataSource**.

Description

Description

Enter a description for the database. The default is **INFORMIX non-Transactional Connection Pool 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. The default is **2**.

LoginTimeOut

Description

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

Required Value

A valid numeric value. The default is **0**.

MaxIdleTime

Description

Specifies 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. The default is **0**.

MaxPoolSize

Description

Specifies 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. The default is **10**.

MaxStatements

Description

Specifies 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. The default is **1000**.

MinPoolSize

Description

Specifies 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. The default is **2**.

NetworkProtocol

Description

Specifies the network protocol used to communicate with the server.

Required Values

Any valid string.

PropertyCycle

Description

Specifies 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. The default is **0**.

RoleName

Description

Specifies an initial SQL role name.

Required Values

Any valid string.

4.3 Configuring the Environment Properties

The eWay Environment Configuration properties contain parameters that define how the eWay connects to and interacts with other eGate components within the Environment. When you create a new Informix External System, you select the type of External System required.

Available External Systems include:

- Inbound Informix eWay
- Outbound Informix eWay
- Outbound Informix XA eWay
- Outbound Informix non-Transactional eWay

To Configure the Environment Properties:

1. In Enterprise Explorer, click the Environment Explorer tab.
2. Expand the Environment created for the Informix Project and locate the Informix External System.

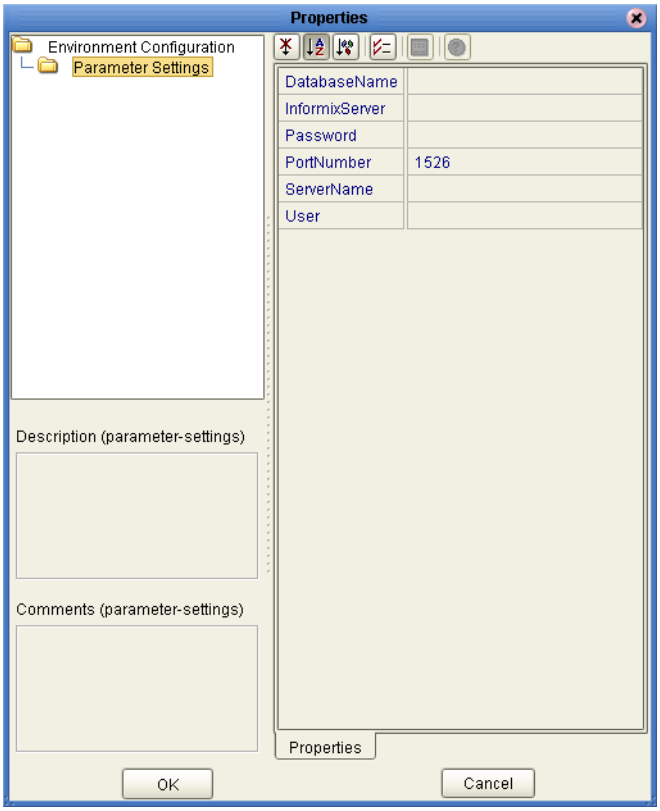
Note: *For more information on creating an Environment, see the eGate Integrator Tutorial.*

3. Right-click the External System created for the Informix Project and select Properties from the list box. The Environment Configuration Properties window appears.
4. Click on any folder to display the default configuration properties for that section.
5. Click on any property field to make it editable.
6. After modifying the configuration properties, click **OK** to save the changes.

4.3.1. Inbound Informix eWay External System Properties

Before deploying your eWay, you will need to set the Environment properties. This section describes the External System properties used by the Inbound Informix eWay.

Figure 7 Inbound Informix eWay Environment Configuration



DatabaseName

Description

Specifies the name of the database instance.

Required Values

Any valid string.

InformixServer

Description

Specifies the name of the Informix server being used.

Required Values

Any valid string.

Password

Description

Specifies the password used to access the database.

Required Values

Any valid string.

PortNumber

Description

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

Required Values

A valid port number. The default is **1526**.

ServerName

Description

Specifies the host name of the external database server.

Required Values

Any valid string.

User

Description

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

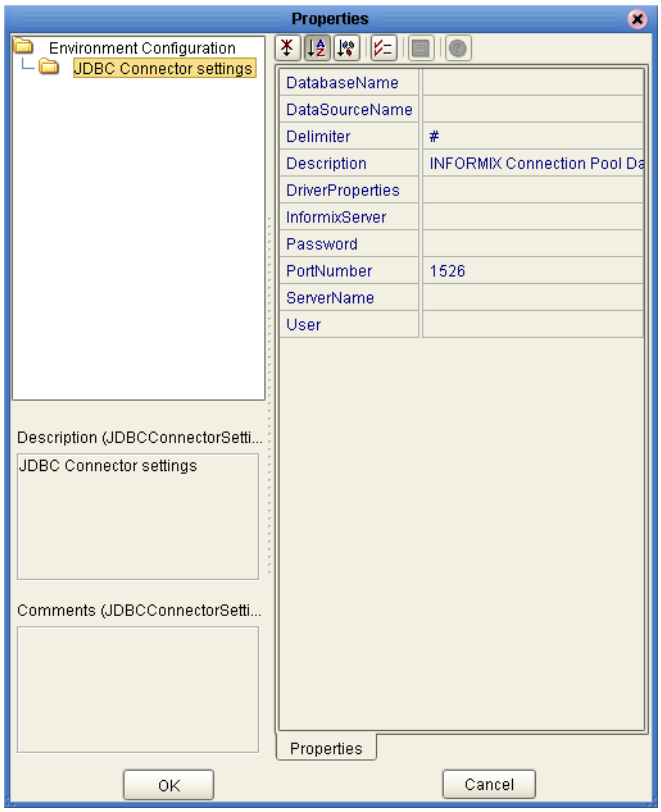
Required Values

Any valid string.

4.3.2. Outbound Informix eWay External System Properties

Before deploying your eWay, you will need to set the Environment properties. This section describes the External System properties used by the Outbound Informix eWay.

Figure 8 Outbound Informix eWay Environment Configuration



DatabaseName

Description

Specifies the name of the database instance.

Required Values

Any valid string.

DataSourceName

Description

Provides 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

Specifies the delimiter character used in the DriverProperties prompt.

Required Value

The default is #

Description

Description

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

Required Value

A valid string.

DriverProperties

Description

Specifies the driver properties used for this eWay. 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#<url>##".

If you are using Spy Log. Optional:

"setURL#jdbc:SeeBeyond:informix://<host>:1526;DatabaseName=<database>##setInformixServer#<informixserver>##setSpyAttributes#log=(file)c:/temp/spy.log;logTName=yes##".

InformixServer

Description

Specifies the name of the Informix database being used.

Required Values

Any valid string.

Password

Description

Specifies the password used to access the database.

Required Values

Any valid string.

PortNumber

Description

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

Required Values

A valid port number. The default is **1526**.

ServerName

Description

Specifies the host name of the external database server.

Required Values

Any valid string.

User

Description

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

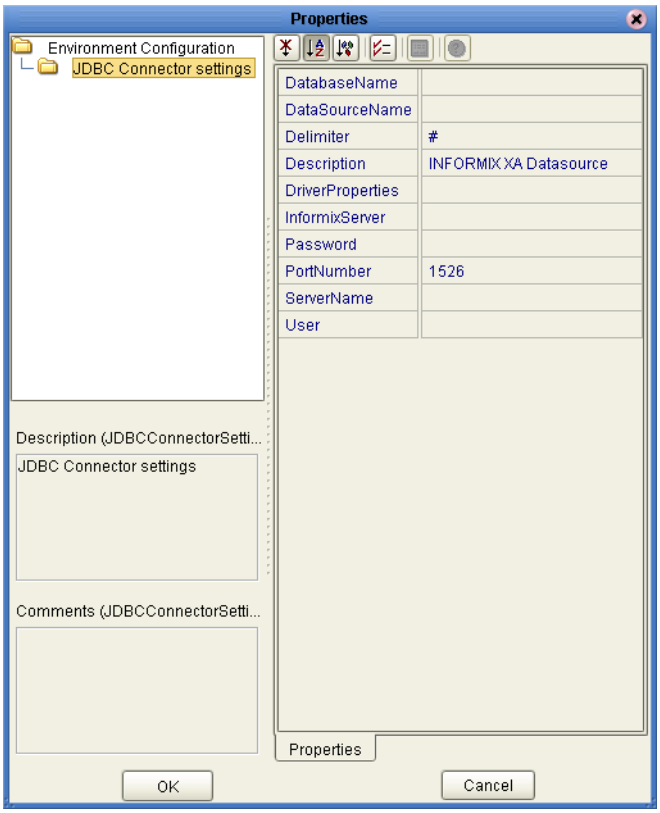
Required Values

Any valid string.

4.3.3. Outbound Informix XA eWay External System Properties

Before deploying your eWay, you will need to set the Environment properties. This section describes the External System properties used by the Outbound Informix XA eWay.

Figure 9 Outbound Informix XA eWay Environment Configuration



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

Specifies the delimiter character used in the DriverProperties prompt.

Required Value

The default is **#**

Description

Description

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

Required Value

A valid string.

DriverProperties

Description

Specifies the driver properties used in this eWay. 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#<url>##".

If you are using Spy Log. Optional:

"setURL#jdbc:SeeBeyond:informix://<host>:1526;DatabaseName=<database>##setInformixServer#<informixserver>##setSpyAttributes#log=(file)c:/temp/spy.log;logTName=yes##".

InformixServer

Description

Specifies the name of the Informix server being used.

Required Values

Any valid string.

Password

Description

Specifies the password used to access the database.

Required Values

Any valid string.

PortNumber

Description

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

Required Values

A valid port number. The default is **1526**.

ServerName

Description

Specifies the host name of the external database server.

Required Values

Any valid string.

User

Description

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

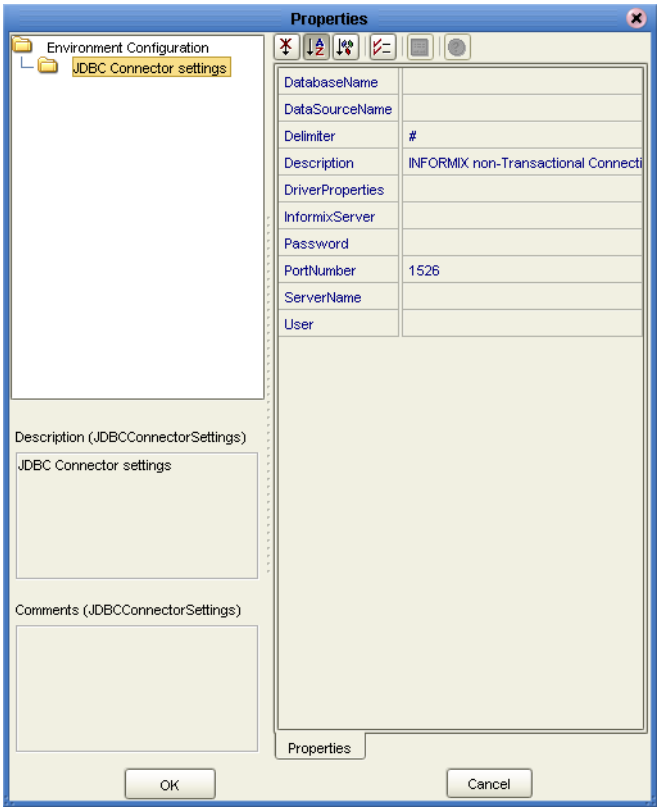
Required Values

Any valid string.

4.3.4. Outbound Informix non-Transactional eWay External System Properties

Before deploying your eWay, you will need to set the Environment properties. This section describes the External System properties used by the Outbound Informix non-Transactional eWay.

Figure 10 Outbound Informix non-Transactional eWay Environment Configuration



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 **INFORMIX non-Transactional Connection Pool 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#<url>##”.

If you are using Spy Log. Optional:

“setURL#jdbc:SeeBeyond:informix://<host>:1526;DatabaseName=<database>##setInformixServer#<informixserver>##setSpyAttributes#log=(file)c:/temp/spy.log;logTName=yes##”.

InformixServer

Description

Specifies the name of the Informix server being used.

Required Values

Any valid string.

Password

Description

Specifies the password used to access the database.

Required Values

Any valid string.

PortNumber

Description

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

Required Values

A valid port number. The default is **1526**.

ServerName

Description

Specifies the host name of the external database server.

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 5

Using the OTD Database Wizard

This chapter describes how to build and use Object Type Definitions (OTDs) using the Informix eWay Database Wizard.

5.1 Creating the OTD

OTDs contain the data structure and rules that define an object. The OTD Wizard creates OTDs based on any combination of Tables, Prepared SQL Statements, and stored procedures.

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 Java methods used in this eWay, see **Using eWay Java Methods** on page 75.

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

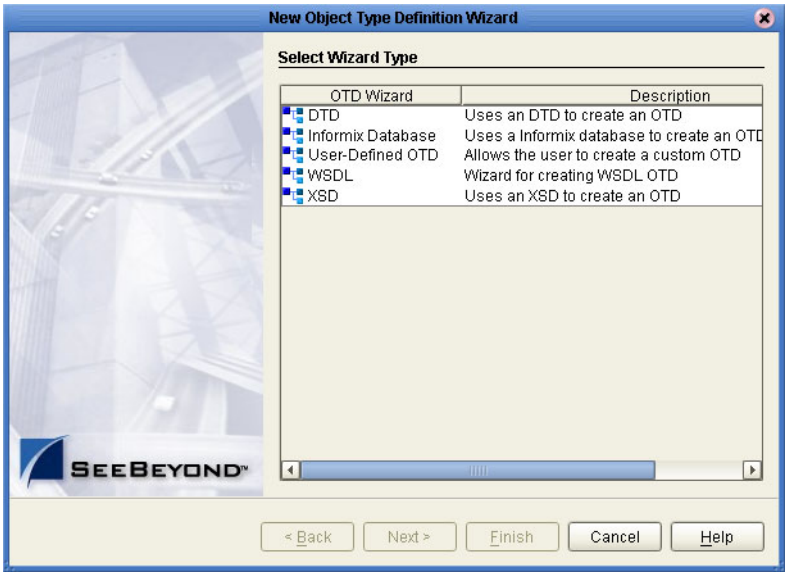
Steps required to create an OTD include:

Select Wizard Type

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

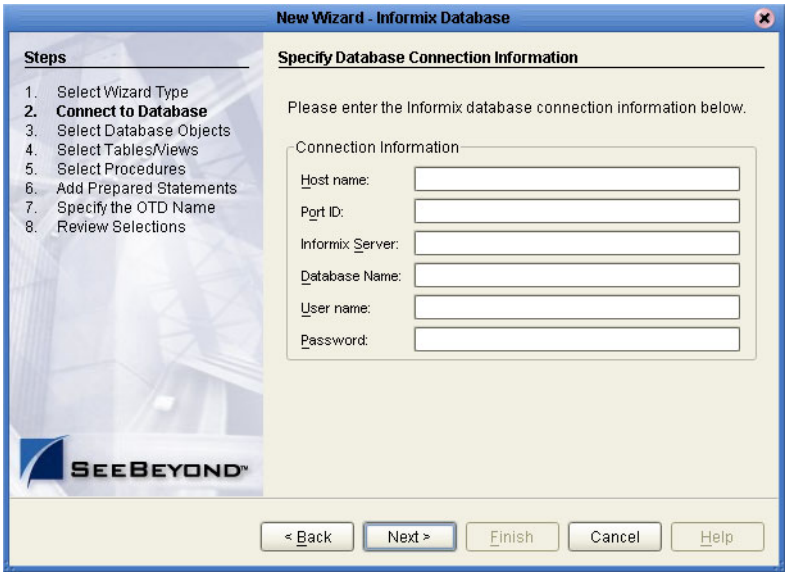
2 The **Select Wizard Type** window appears, displaying the available **OTD** wizards.

Figure 11 OTD Wizard Selection



3 From the list, select the Informix Database OTD and click **Next**. The **Specify Database Connection** Information window appears.

Figure 12 Database Connection Information



Connect To Database

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

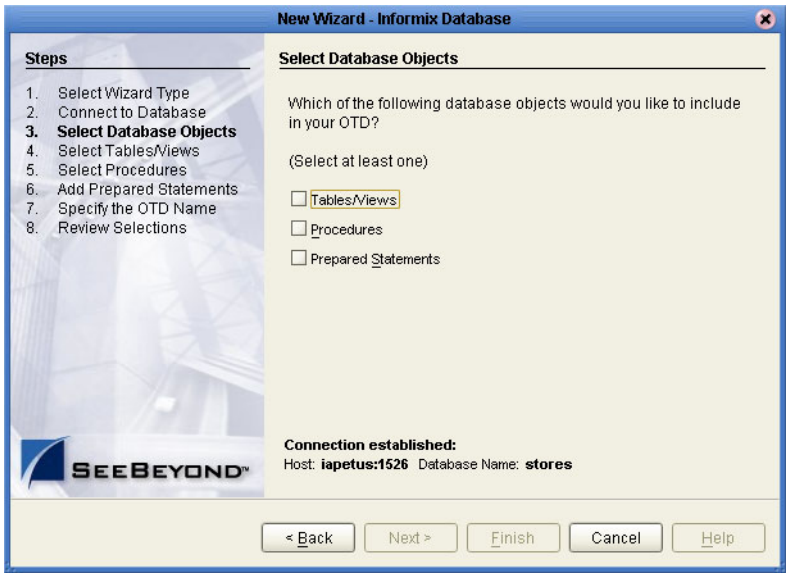
- **Host name** – the name of the host you are connecting.
- **Port ID** – the host port number (1526 is the default).
- **Informix Server** – the name of the Informix server.
- **Database name** – the name of the database you are connecting
- **User name** – your user ID.
- **Password** – your password.

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

Select Database Objects

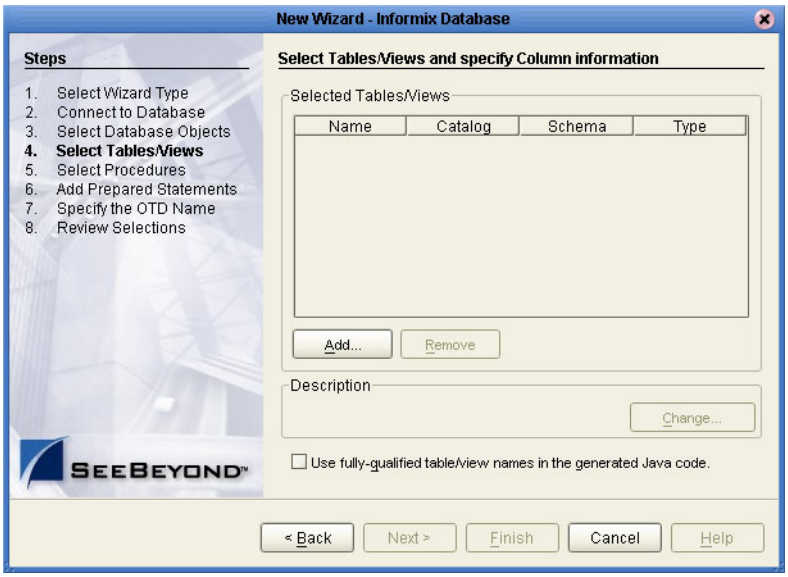
1 On the Select Database Objects window, select **Tables/Views**, **Procedures**, and **Prepared Statements** checkboxes.

Figure 13 Select Database Objects



2 Click **Next**. The **Select Tables/Views** window appears. See **Figure 14 on page 36**.

Figure 14 Select Tables/Views

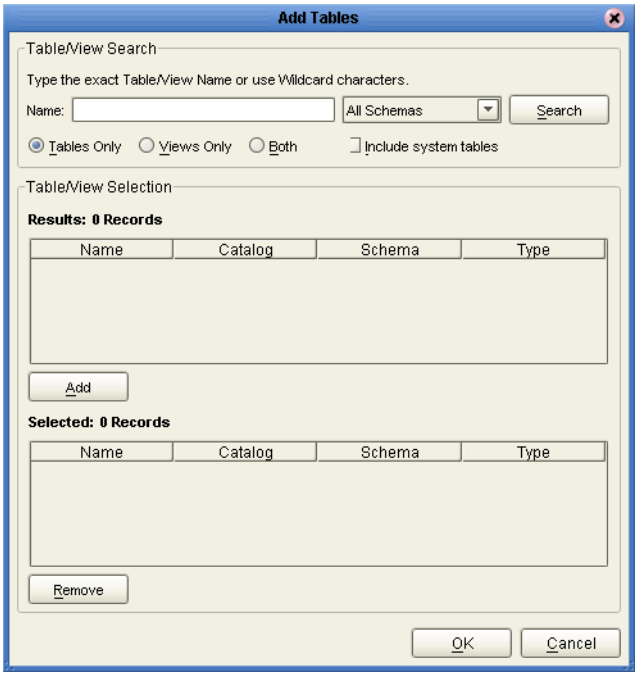


Note: *Views are read-only and are for informational purposes only.*

Select Table/Views

1 On the Select Tables/Views window, click the **Add** button. The **Add Tables** window appears.

Figure 15 Add Tables Window

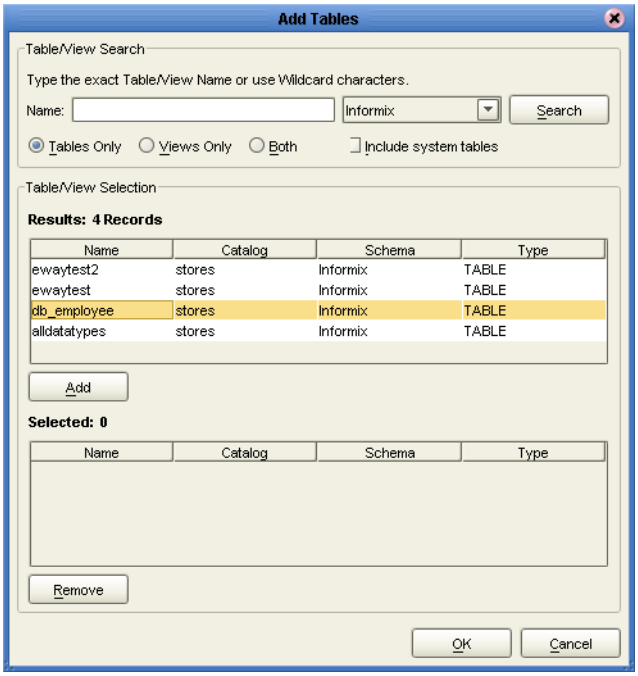


2 In the **Add Tables** window, select whether your selection criteria will include table data, view only data, both, and/or system tables.

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

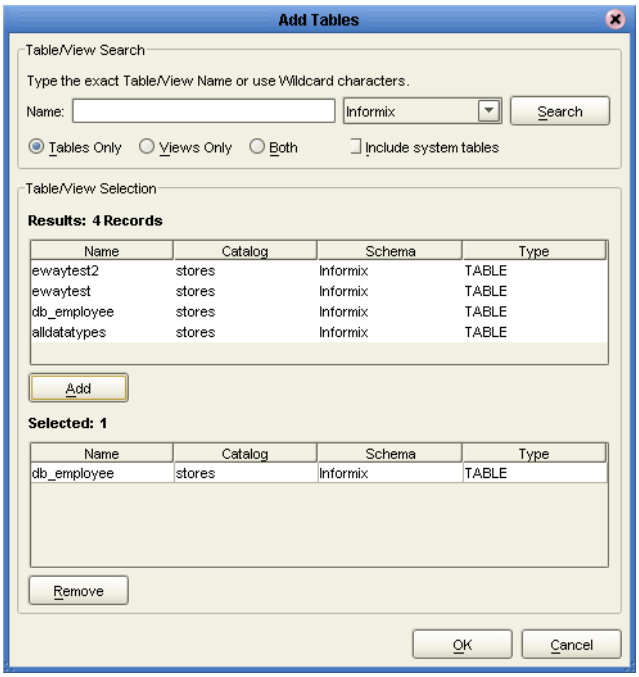
Note: *Click Search to find the desired table or tables. You can also use wildcard characters to search for a table or view. Available wildcard characters include "?", "_", and "*". For example, you can use "AB?CD", "AB_CD", or "AB*CD". However, do not use "%". Using this character results in nothing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

Figure 16 Database Wizard - All Schemes



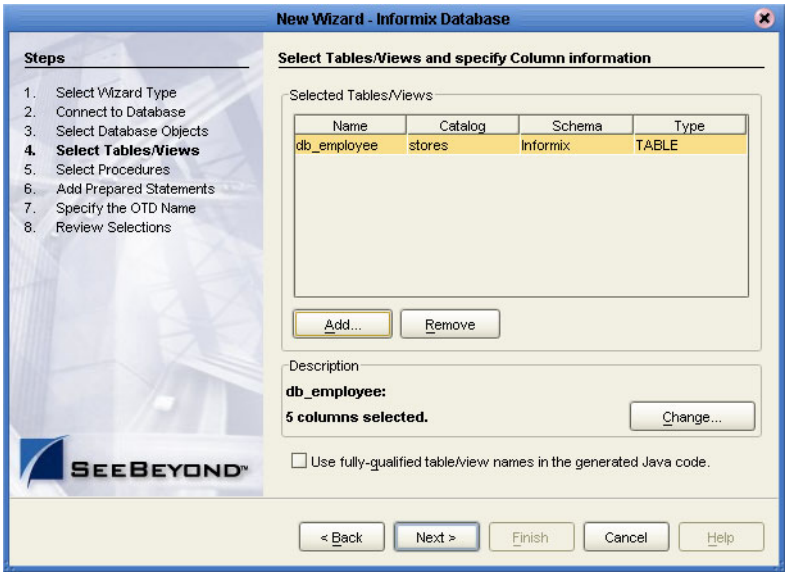
4 Select a table and click **OK**. The selected table is added to the **Selected Tables/ Views** window. See **Figure 17**.

Figure 17 Selected Tables/Views window with a table selected



5 On the Selected Tables/Views window, review your selected tables. To make changes to the selected Table or view, click **Change**, or click **Next** if no additional changes are required.

Figure 18 Selected table and column window

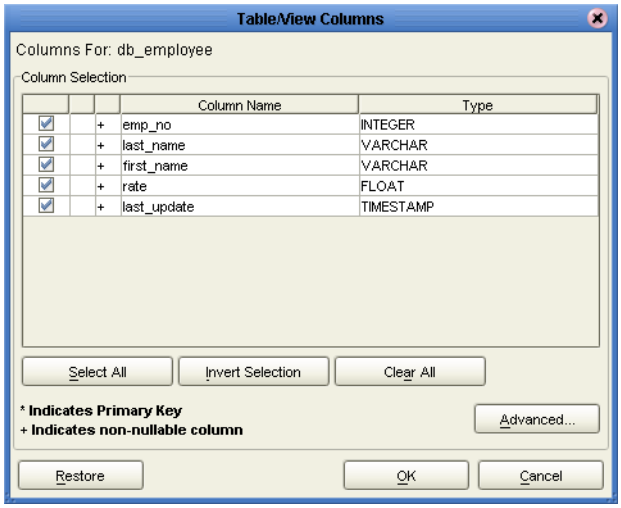


6 If you click **Change**, the Table/View Columns window appears, allowing you to select or deselect any table columns. You can also change the data type for each table by highlighting the data type and selecting a different data type from the drop-down list.

The buttons in this window include:

- **Select All** – allows you to select all columns.
- **Invert Selection** – allows you to invert the order of the selected columns.
- **Clear All** – allows you to deselect all columns.
- **Advanced** – allows you to perform advanced operations with the columns. See the *eGate Integrator User's Guide* for details.
- **Restore Metadata** – allow you to restore the data to its original state before you made any changes via the wizard; returns you to the **Specify Database Connection** window.

Figure 19 Table/View Columns window

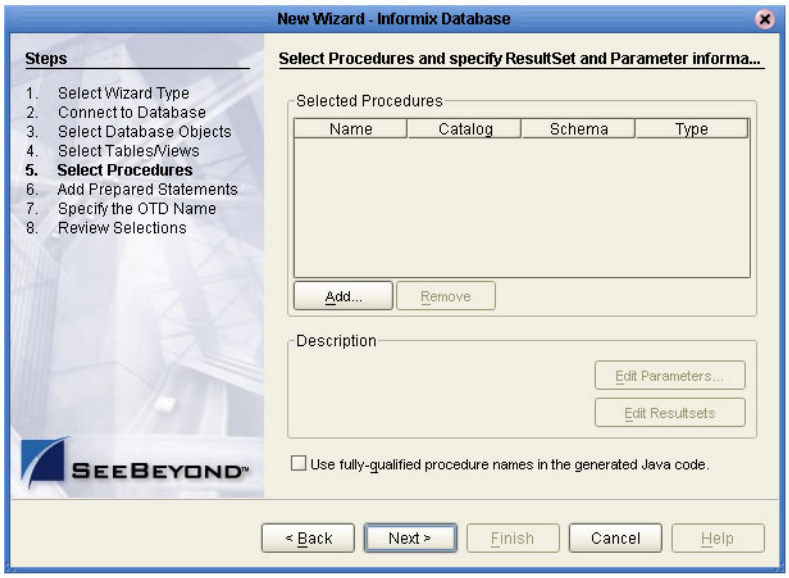


7 When you are finished using this window, click **OK** to save your changes and return to the Select Tables/Views window.

Select Procedures

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

Figure 20 Select Procedures window

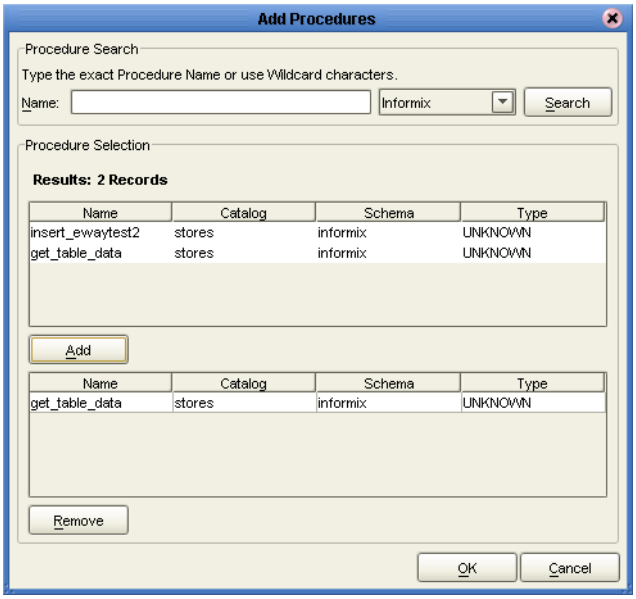


2 On the **Select Procedures** window, enter the name of a Procedure or select a table from the drop down list. Click **Search**. Wildcard characters can also be used.

Note: *You must use lower case schema names when calling stored procedures.*

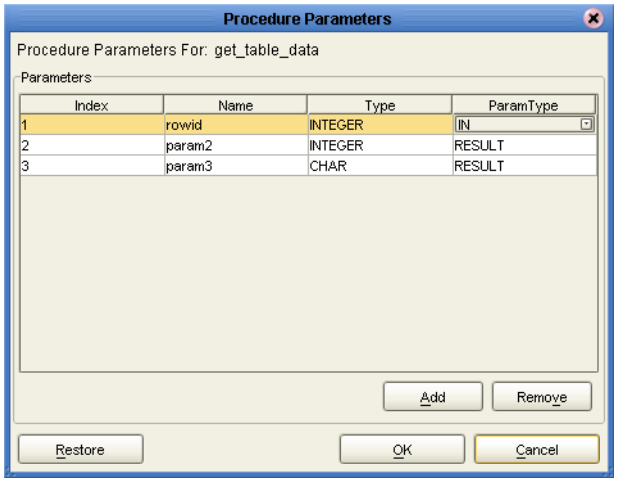
3 In the resulting **Procedure Selection** list box, select a Procedure. Click **OK**.

Figure 21 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.

Figure 22 Procedure Parameters

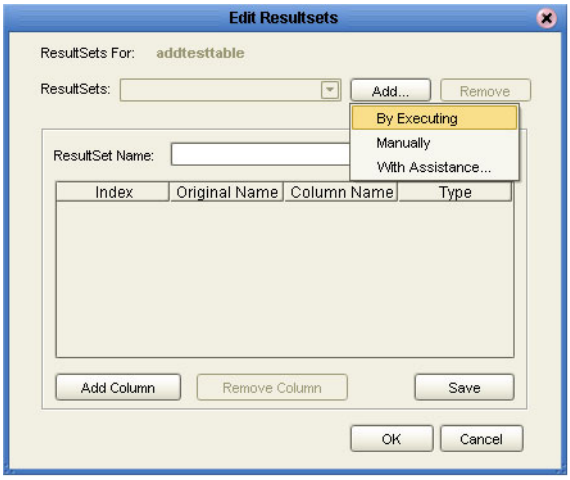


5 To restore the data type, click **Restore**. When finished, click **OK**.

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 23 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.

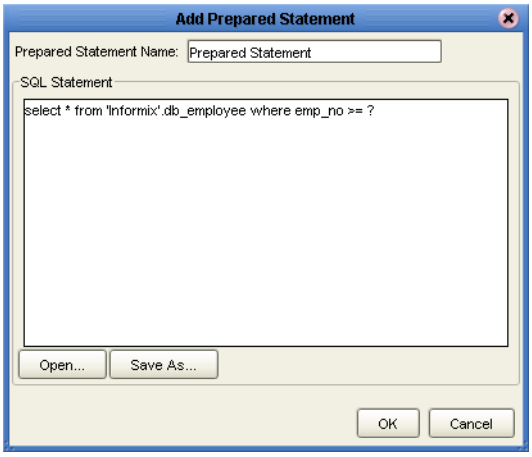
8 On the **Select Procedures and specify Resultset and Parameter Information** window click **Next** to continue.

Add Prepared Statements

1 On the **Add Prepared Statements** window, click **Add**. The **Add Prepared Statement** window appears.

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

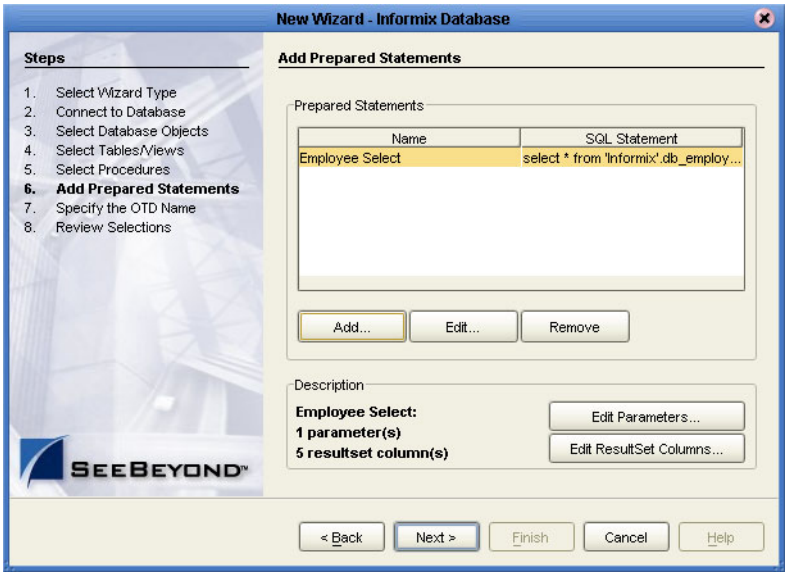
Figure 24 Prepared SQL Statement



3 Click **Save As** to save the statement, or click the **OK** button to exit the window.

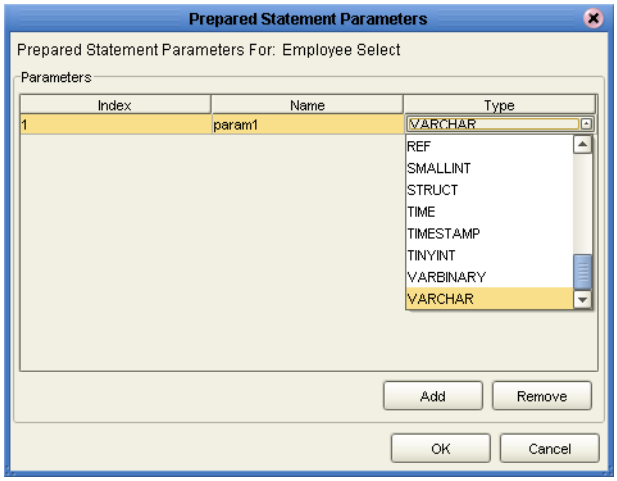
On the **Add Prepared Statement** window, the name you assigned to the Prepared Statement appears.

Figure 25 Add Prepared Statement window



4 To edit the parameters, click **Edit Parameters**. You can change the datatype by clicking in the **Type** field and selecting a different type from the list.

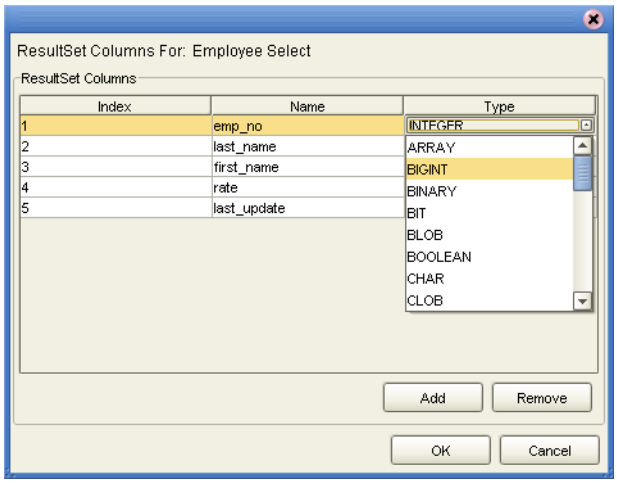
Figure 26 Edit the Prepared Statement Parameters



5 Click **Add** if you want to add additional parameters to the Statement or highlight a row and click **Remove** to remove it. Click **OK** to save your changes and exit the window.

6 To edit the Resultset Columns, click **Edit Resultset Columns**. Both the **Name** and **Type** are editable. Click **OK** to save your changes and exit the window.

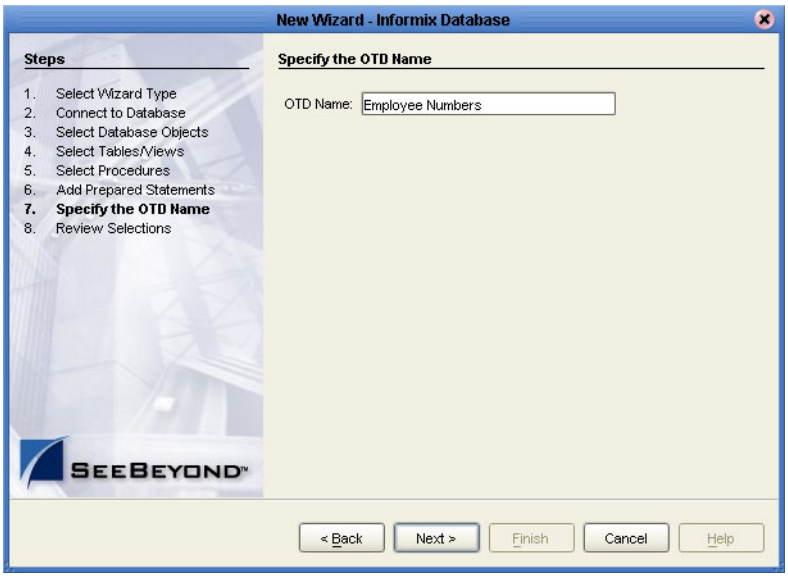
Figure 27 ResultSet Columns



Specify the OTD Name

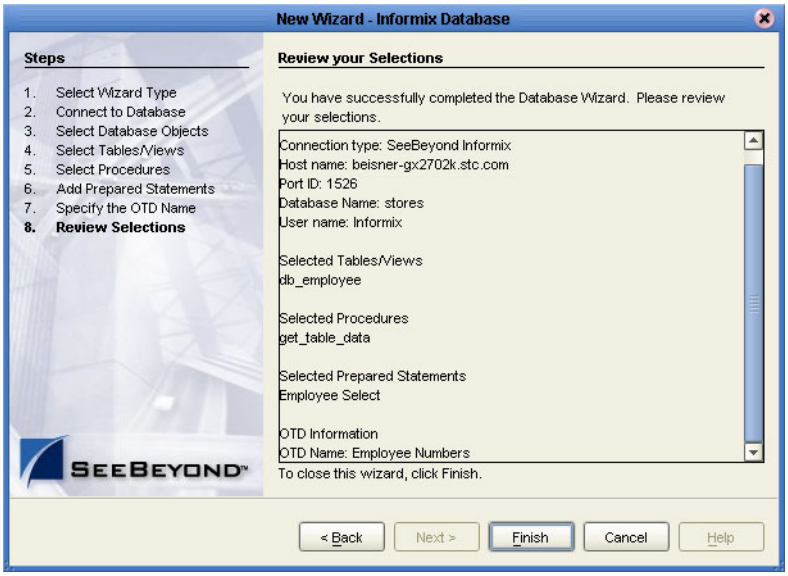
1 On the **Specify the OTD Name** window, enter a name for the OTD. The OTD contains the selected tables and the package name of the generated classes. See **Figure 28**.

Figure 28 Naming an OTD



2 Review the summary of the OTD. If your information is correct, click **Finish** to generate the OTD. The resulting **OTD** editor appears with the newly generated OTD. See **Figure 29**.

Figure 29 Database Wizard - Summary



Chapter 6

Locating, Importing, and Using Sample Projects

This chapter describes how to use the sample Projects included in the installation CD-ROM package.

Chapter Topics Include:

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

6.1 Sample Projects Overview

Sample Projects are designed to provide an overview of the basic functionality of the Informix eWay by demonstrating how to pass information between eGate and the Informix database.

Sample Projects Include:

Informix_BPEL_Sample – describes how to retrieve the last name and employee number of all employees in the Informix database, using eInsight's BPEL business process.

Informix_JCE_Sample – describes how to retrieve the last name and employee number of all employees in the Informix database, using eGate Integrator.

Sample Data Used:

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

Table 1 Sample project data

Column Name	Mapping	Data Type	Data Length
EMP_NO	employee_no	integer	10
LAST_NAME	employee_lname	varchar	30
FIRST_NAME	employee_fname	varchar	30
LAST_UPDATE	update_date	datetime - year to second	16
RATE	rate	float	53

6.2 Locating and Importing the Sample Projects

The eWay sample Projects are included in the **InformixeWayDocs.sar**. This file is uploaded separately from the Informix eWay SAR file during installation. For more information, refer to **"Installing the Informix eWay" on page 8**.

Once you have uploaded the **InformixeWayDocs.sar** to the Repository, you can begin downloading the sample Projects from the **DOCUMENTATION** tab on Enterprise Manager, to a folder of your choosing.

Before using the sample Project, you must first import it into the SeeBeyond Enterprise Designer using the Enterprise Designer Project Import utility.

To Import the Sample Project:

1. From the Enterprise Designer's Project Explorer pane, right-click the Repository and select **Import**.
2. In the **Import Manager** window, browse to the directory that contains the sample Project zip file.
3. Select the sample file and then click **Open**.
4. Click the **Import** button. If the import was successful, then click the **OK** button on the **Import Status** window.

6.3 Running the Sample Projects

Steps required to run a sample Project include:

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

6.3.1 Setting the Properties

Sample Projects use both an inbound and an outbound File eWay, as well as an outbound Informix eWay. Use the following information to configure the sample Project eWays. For additional information on the eWay properties, see **Configuring the eWay Connectivity Map Properties** on page 10.

To Configure the File eWays:

1. On the Connectivity Map, double-click the **Inbound File eWay**.
2. The **Properties** window for the Inbound File eWay opens. Modify any parameter settings necessary for your system, then change the Directory and Input file name to match the location and name of the sample data file.
3. Click **OK** to close the **Properties** window.
4. On the Connectivity Map, double-click the Outbound File eWay. The **Properties** window for the Outbound File eWay opens. Modify any parameter settings necessary for your system, including the target Directory and Output file name.
5. Click **OK** to close the Properties window.

To Configure the Outbound Informix eWay:

1. On the Connectivity Map, double-click the **Informix** eWay.
2. The **Properties** window for the Informix eWay opens. Modify the parameter settings necessary for your system, then click **OK** to close the Properties window.

6.3.2. Creating the Environment Profile

An eGate Environment represents the physical system required to implement a Project. A typical Environment contains several components, including Logical Hosts, Integration Servers, Message Servers, and External Systems. Environments are created using the Enterprise Designer's Environment Explorer.

To Create a New Environment:

1. On the Environment Explorer, select and right-click the eWay profile.
2. Select **Properties**, and enter the configuration information required for the eWay. See **Configuring the Environment Properties** on page 21.

6.3.3 Deploying the Project

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

6.3.4. Running the Sample

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

After completing the process, the Output file in the target directory—that was previously configured in the Outbound File eWay—contains all records retrieved from the database in text format.

6.4 Using the Sample Project in eInsight

This section describes how to use the **Informix_BPEL_Sample** Project with the ICAN Suite's eInsight Business Process Manager and the Web Services interface. This section *does not* explain how to create a Project using an eInsight business process. For these instructions, refer to the *"eInsight Business Process Manager User's Guide"*.

Before running a sample Project, you must:

- Import the sample Project
- Create an Environment for the sample Project
- Configure the eWay properties for your specific system (see **Creating and Configuring an Informix eWay** on page 9)
- Create a Deployment Profile

6.4.1 eInsight Engine and eGate Components

You can deploy an eGate component as an Activity in an eInsight Business Process. Once you have associated the desired component with an Activity, the eInsight engine can invoke it using a Web Services interface. Examples of eGate components that can interface with eInsight in this way are:

- Java Messaging Service (JMS)
- Object Type Definitions (OTDs)
- An eWay
- Collaborations

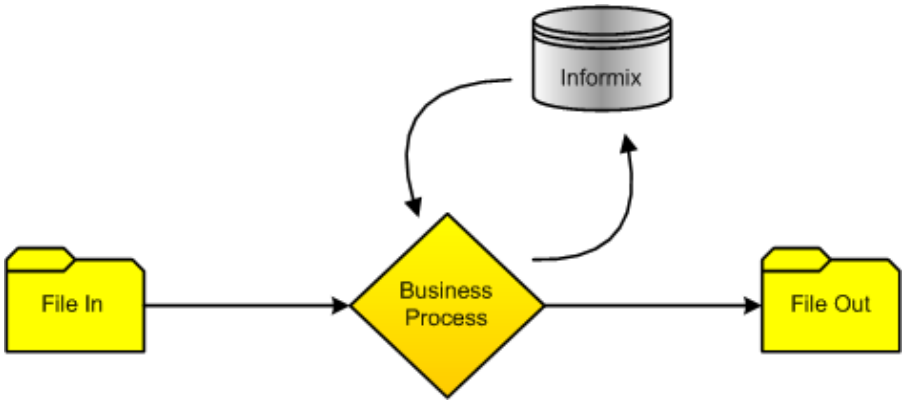
Using the eGate Enterprise Designer and eInsight, you can add an Activity to a Business Process, then associate that Activity—such as an eWay—with an eGate component. When eInsight runs the Business Process, it automatically invokes that component via its Web Services interface.

6.4.2 The Informix_BPEL_Sample Project

The **Informix_BPEL_Sample** sample Project describes how to retrieve the last name and employee number of all employees in the Informix database, using eInsight's BPEL business process engine. In this sample, specific employee information is retrieved by passing specific conditions into an OTD Collaboration which then queries the database and extracts results in the form of an output file.

Figure 30 illustrates the business process used by the sample Project.

Figure 30 Sample Project Data Exchange



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

BP_GetEmployee

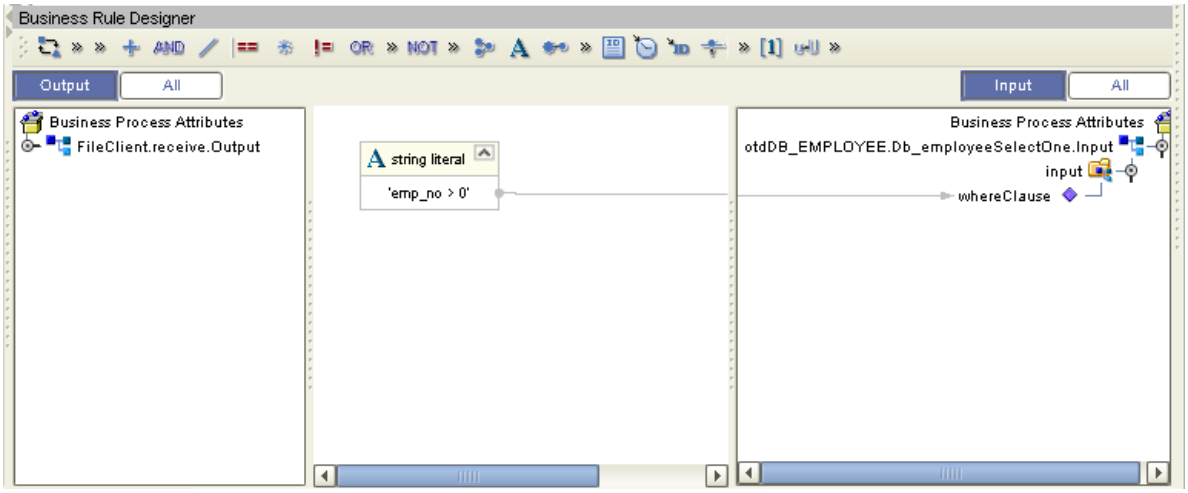
This business process describes the account retrieval process seen in Figure 31.

Figure 31 Employee Number Retrieval

Business Process:

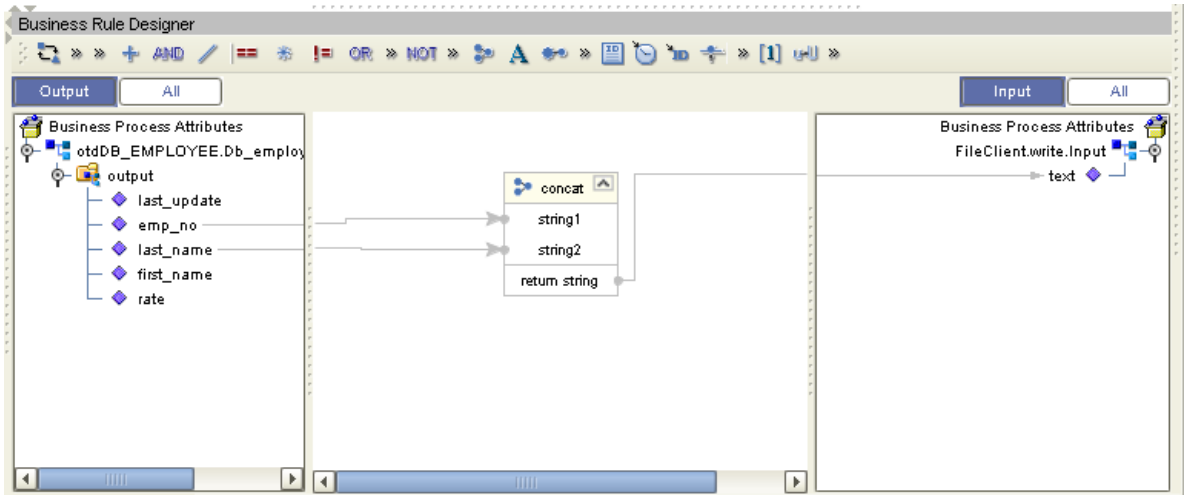
1. The File eWay subscribes to an external directory and activates a trigger to query the database.
2. A string literal **'emp_no > 0'** is passed into a whereClause which loops through and queries the Informix database for the first record matching the criteria.

Figure 32 BP_GetEmployee Business Process - Find Account



3 Both employee number (emp_no) and last name (last_name) are concatenated and written into an input file, before passing onto the Outbound File eWay.

Figure 33 BP_GetEmployee Business Process - Send Account



6.5 Working With Other Business Process Activities

You can associate an eInsight Business Process Activity with the eWay, both during the system design phase and during run time. To make this association, select the desired operation, found under the Project OTD in Enterprise Explorer and drag it onto the eInsight Business Process canvas.

The following operations are available:

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

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

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

Note: *For information on how to create an Informix Database OTD, refer to* **Creating the OTD** *on page 33.*

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

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

6.5.1 whereClause()

A BPEL whereClause() statement may be joined by AND/OR with conditions of "=", "!=", "<>", "<", ">", "<=", ">=".

For example:

whereClause such as where column2=2 AND column1=1 OR column3=3 is valid

6.5.2 SelectAll

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

Figure 34 shows a sample eInsight Business Process using the SelectAll operation. In this process, the SelectAll operation returns all rows where the emp_no is not null.

Figure 34 SelectAll Sample Business Process

Figure 35 shows the definition of the **where()** clause for the SelectAll operation.

Figure 35 SelectAll Input

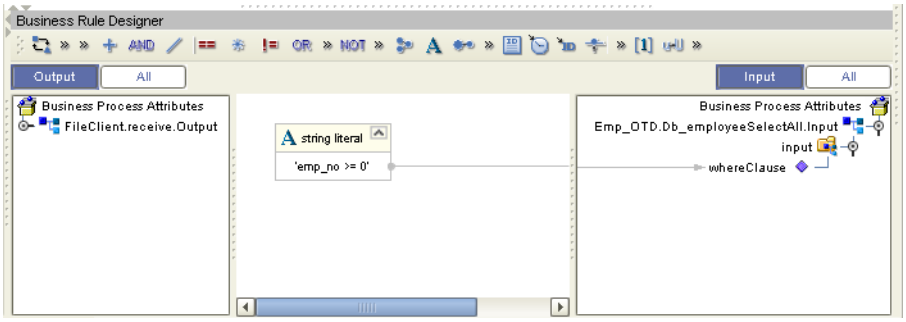
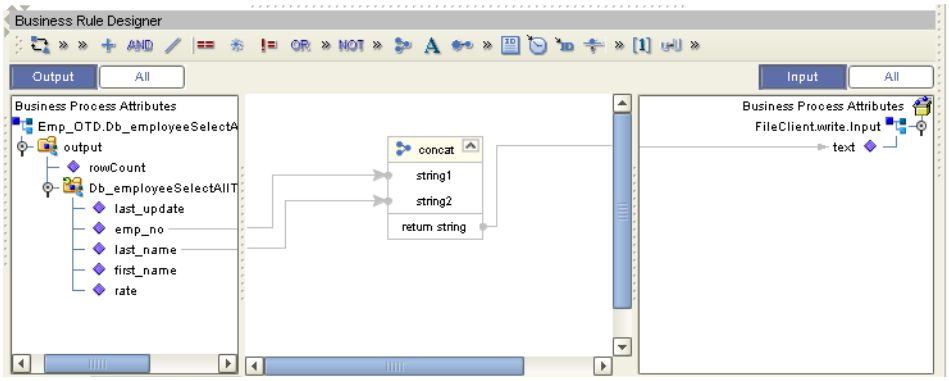


Figure 36 on page 55 shows the output for the SelectAll operation. For each row selected during the operation, the employee number and last name are concatenated and passed out as text using the FileClient.write.

Figure 36 SelectAll Output



6.5.3 SelectMultiple

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

Figure 37 shows a sample eInsight Business Process using the SelectMultiple operation. In this process, the SelectMultiple operation returns three rows specified in the **where()** clause.

Figure 37 SelectMultiple Sample Business Process

Figure 38 shows three rows requested using the **where()** clause. In this example, we are requesting everything from the table where emp_no is equal to 103, 109, and 117.

Figure 38 SelectMultiple Input

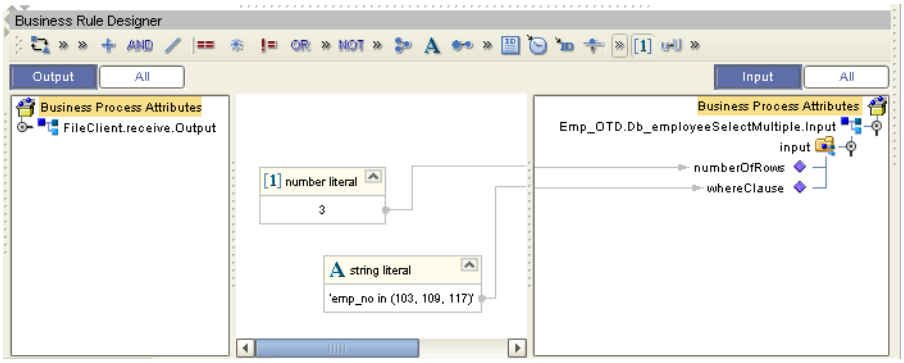
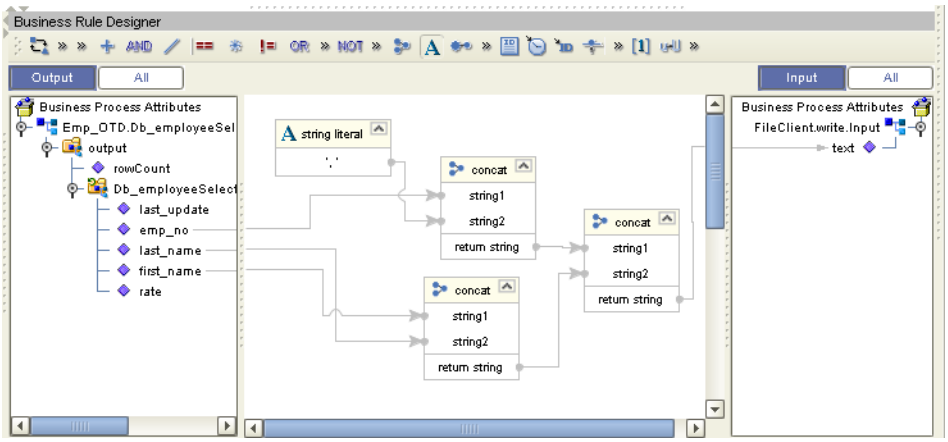


Figure 39 shows the output for the SelectMultiple operation. For each row selected during the operation, the employee number, first and last name are returned as text using the FileClient.write.

Figure 39 SelectMultiple Output



6.5.4 SelectOne

The input to a **SelectOne** operation uses a **where()** clause to define the criteria for selecting, and returning the first row in the database that meets required conditions.

Figure 40 shows a sample eInsight Business Process using the SelectOne operation. In this process, the **SelectOne** operation returns the first row where rate is >= 100.

Figure 40 SelectOne Sample Business Process

Figure 41 on page 57 shows the definition of the **where()** clause for the SelectOne operation.

Figure 41 SelectOne Input

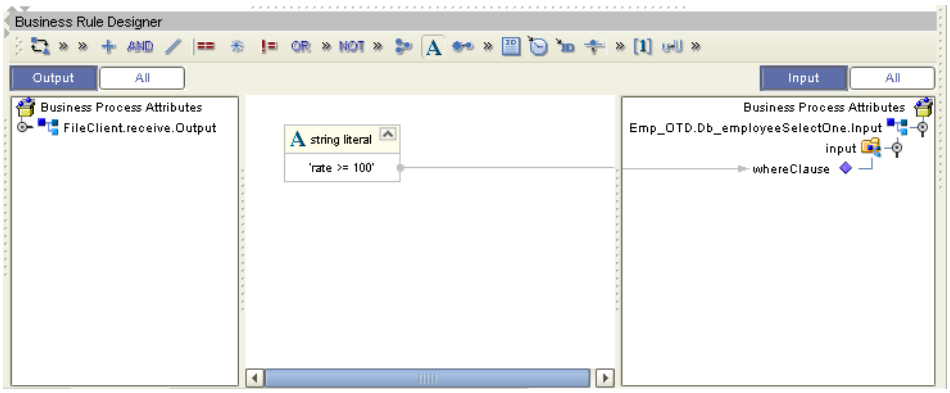
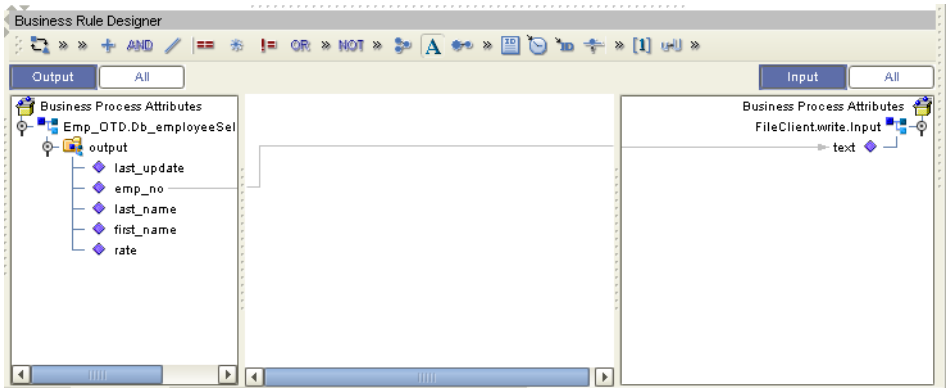


Figure 42 shows the definition of the output for the SelectOne operation. The text output shows the employee number (emp_no) of the first row selected during the operation.

Figure 42 SelectOne Output



6.5.5 Insert

The Insert operation inserts a row. The input to an Insert operation is a **where()** clause. The **where()** clause defines the criteria for the new row. In this operation, the first row that fits the criteria is returned.

Figure 43 shows a sample eInsight Business Process using the Insert operation. In this process, the operation inserts a new row into the database to accommodate the provided data.

Figure 43 Insert Sample Business Process

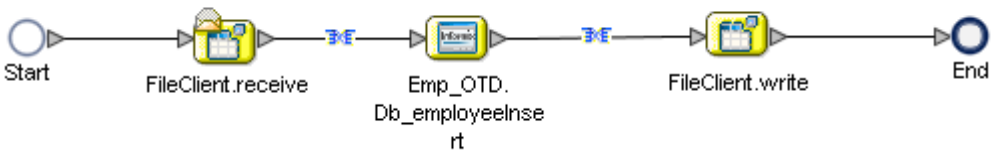


Figure 44 shows the input data used for the Insert operation. The emp_no is not included since an index is automatically created for the new row.

Figure 44 Insert Input

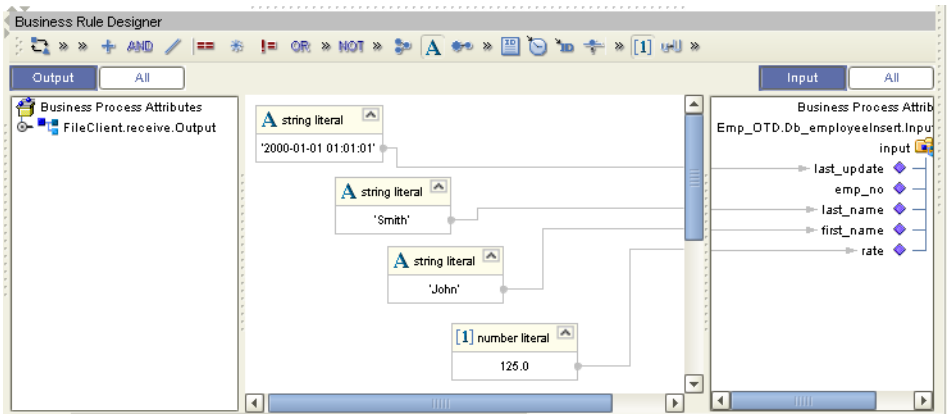
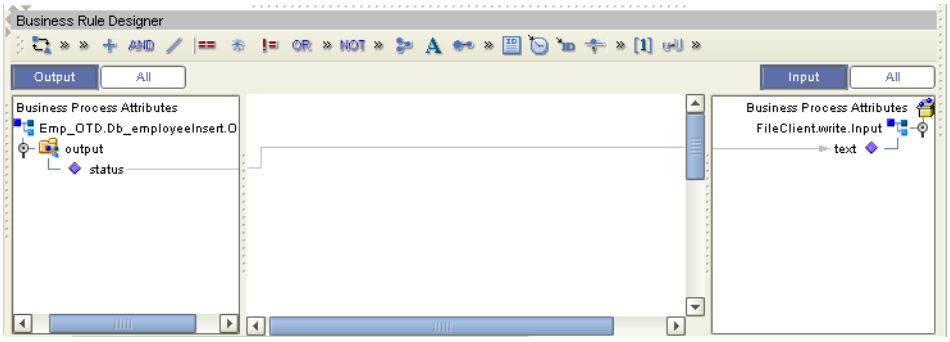


Figure 45 shows the output of the Insert operation, which is a status indicating the number of rows created.

Figure 45 Insert Output



6.5.6 Update

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

Figure 46 shows a sample eInsight Business Process using the Update operation. In this process, the operation updates a rate for a certain employee.

Figure 46 Update Sample Business Process

Figure 47 shows the definition of the **where()** clause for the Update operation. In this example, employee number 115 is updated to have a new rate of 168.75.

Figure 47 Update Input

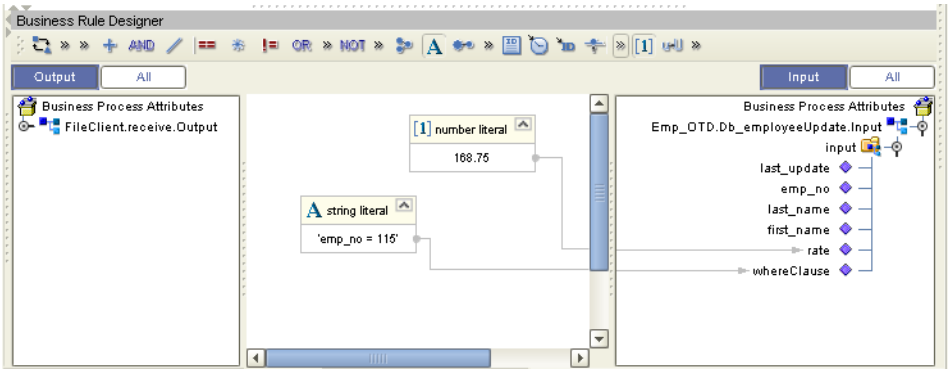
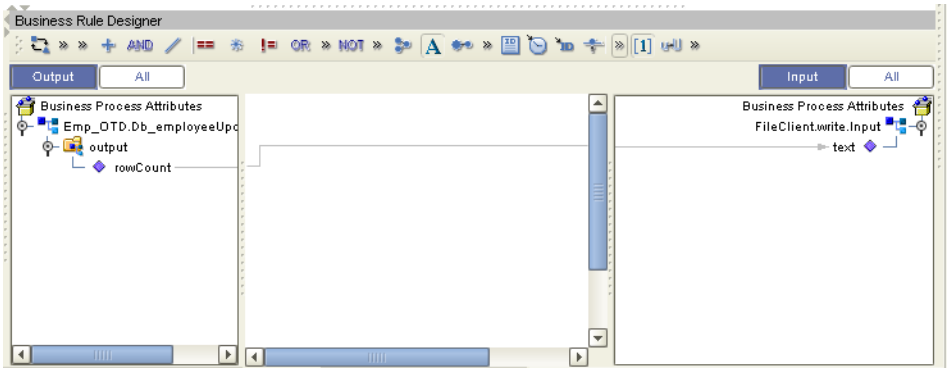


Figure 48 shows the output of the Update operation, which is a status indicating the number of rows updated.

Figure 48 Update Output



6.5.7 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.

Figure 49 shows a sample eInsight Business Process using the Delete operation. In this process, the operation deletes a row that matches a certain employee number.

Figure 49 Delete Sample Business Process

Figure 50 shows the definition of the **where()** clause for the Delete operation. In this example, employee 115 is deleted.

Figure 50 Delete Input

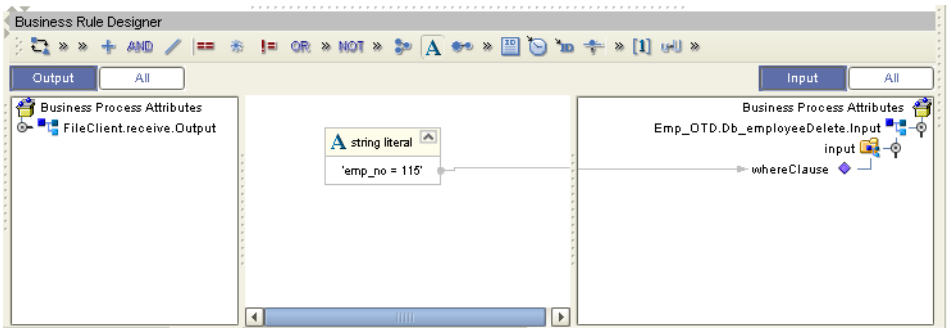
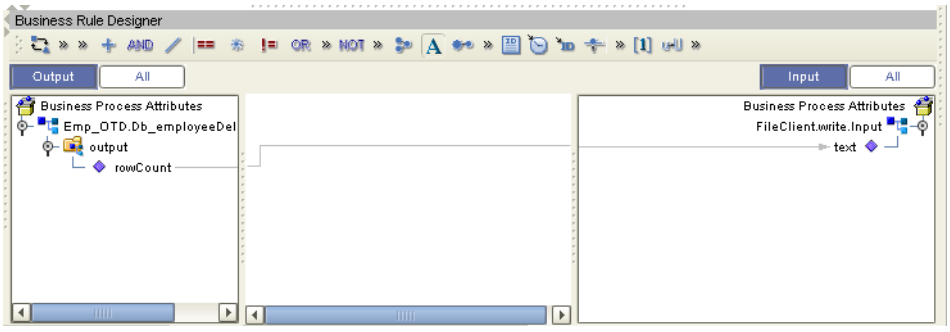


Figure 51 shows the output of the Delete operation, which is a status indicating the number of rows deleted.

Figure 51 Delete Output



6.6 Using the Sample Project in eGate

This section describes how to use the **Informix_JCE_Sample** Project with eGate Integrator. This section does not provide an explanation of how to *create* a Project. For information on building a project, you should refer to the *"eGate Integrators User's Guide"*.

Before running a sample Project, you must:

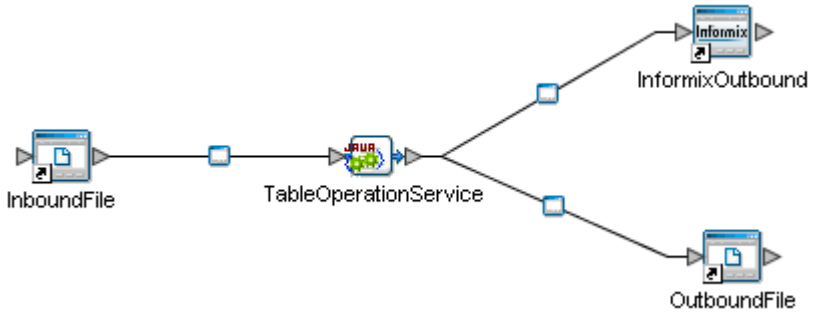
- Import the sample Project
- Create an Environment for the sample Project
- Configure the eWay properties for your specific system (see **Creating and Configuring an Informix eWay** on page 9)
- Create a Deployment Profile

6.6.1. Working with the Sample Project in eGate

The eGate sample Project **Informix_JCE_Sample** includes four Collaboration Definitions (Java) that describe how to delete, insert, select, and update the sample database. The Connectivity Map contains an inbound and outbound File eWay connections, as well as an outbound Informix eWay connection. To use sample operations, simply drag and drop one of the four Collaborations onto the TableOperatioService and reapply the connections.

The Connectivity Map for this sample Project appears as follows:

Figure 52 Connectivity Map Project Flow



6.6.2 Sample Project Collaboration Definitions (Java)

The Collaboration Definitions used in the **Informix_JCE_Sample** include:

- jceTableDelete
- jceTableInsert
- jceTableSelect
- jceTableUpdate

jceTableDelete Collaboration Definition

The jceTableDelete Collaboration Definition uses the Delete operation to select and delete from the first_name column, any names that begin like "John". The text input "Table Deletion Completed" is also added as a string literal input to confirm the completion.

jceTableInsert Collaboration Definition

The jceTableInsert Collaboration Definition uses the Insert operation to add a new row to the Db_employee table by inserting a new employee, “John Smith”. The text input "Table Insert Completed" is also added as a string literal, confirming the completion before writing to an output file.

jceTableSelect Collaboration Definition

The jceTableSelect Collaboration Definition uses the Select operation to retrieve the employee number and last name of the first employee with an employee number greater than zero. Matching employee numbers are concatenated with the corresponding last name and written to an output file.

jceTableUpdate Collaboration Definition

The jceTableSelect Collaboration Definition uses the Update operation to update any employee with a last name like "Smith" to the new name "Doe" with a new rate of "33.88". The text input "Table Update Completed" is also added as a string literal confirming the completion before writing to an output file.

6.7 Supported Data Types

Informix eWay supports the following data types:

Figure 53 Data Types Supported by Informix

Data Type	Description	OTD/Java Data Type
BOOLEAN	Stores Boolean values true and false.	boolean
BYTE	Stores any kind of binary data.	byte[]
CHAR(n)	Stores single-byte or multibyte sequences of characters, including letters, numbers, and symbols; collation is code-set dependent.	java.lang.String
CHARACTER VARYING(m,r)	Stores single-byte and multibyte sequences of characters, including letters, numbers, and symbols of varying length (ANSI compliant); collation is code-set dependent.	java.lang.String
DATE	Stores calendar date.	java.sql.Date
DATETIME	Stores calendar date combined with time of day.	java.sql.Timestamp
DECIMAL	Stores numbers with definable scale and precision.	java.math.BigDecimal
DOUBLE PRECISION	Behaves the same way as FLOAT.	double
FLOAT(n)	Stores double-precision floating-point numbers corresponding to the double data type in C.	double
LVARCHAR	Stores single-byte and multibyte sequences of characters, including letters, numbers, and symbols; collation is locale dependant.	java.lang.String
INTEGER	Stores whole numbers from -2,147,483,647 to +2,147,483,647.	int
INT8	Stores an 8-byte integer value. These whole numbers can be in the range -(263 - 1) to 263 -1.	long
MONEY(p,s)	Stores currency amount.	java.math.BigDecimal
NCHAR(n)	Stores single-byte and multibyte sequences of characters, including letters, numbers, and symbols; collation is locale dependent.	java.lang.String

Data Type	Description	OTD/Java Data Type
NVARCHAR(m,r)	Stores single-byte and multibyte sequences of characters, including letters, numbers, and symbols of varying length; collation is locale dependent.	java.lang.String
SERIAL	Stores sequential integers.	int
SERIAL8	Stores large sequential integers; has same range as INT8.	long
SMALLFLOAT	Stores single-precision floating-point numbers corresponding to the float data type in C.	float
SMALLINT	Stores whole numbers from -32,767 to +32,767.	short
TEXT	Stores any kind of text data.	java.lang.String
VARCHAR(m,r)	Stores multibyte strings of letters, numbers, and symbols of varying length to a maximum of 255 bytes; collation is code-set dependent.	java.lang.String

6.8 Converting Datatypes in Informix eWay

When working with data in the Informix eWay OTDs, use the following conversion chart (Figure 54) as a guide for working with Insert and Update operations.

Figure 54 Insert and Update Datatype Conversions

Data Type	Java Method or New Constructor to Use(Default: Java Method)	Sample Data
BOOLEAN	**Boolean:** valueOf(java.lang.String)	True/False
BYTE	**Byte:** valueOf(java.lang.String)	n/a
CHAR(n)	**String:** valueOf(java.lang.Object)	Any Character
CHARACTER VARYING(m,r)	**String:** valueOf(java.lang.Object)	Any Character
DATE	**Date:** valueOf(java.lang.String)	2004-12-31
DATETIME	**Timestamp:** valueOf(long)	2004-12-31 00:00:00
DECIMAL	**BigDecimal:** valueOf(long)	123.45

Data Type	Java Method or New Constructor to Use(Default: Java Method)	Sample Data
DOUBLE PRECISION	**Double:** ParseDouble(java.lang.String)	123.45
FLOAT(n)	**Double:** ParseDouble(java.lang.String)	123.45
INTEGER	**Integer:** ParseInt(java.lang.String)	123
INT8	**Long:** parseLong(java.lang.String)	123
LVARCHAR	**String:** java.lang.String	Any Character
MONEY(p,s)	**BigDecimal:** valueOf(long)	123.45
NCHAR(n)	**String:** valueOf(java.lang.Object)	Any Character
NVARCHAR(m,r)	**String:** valueOf(java.lang.Object)	Any Character
SERIAL	**Integer:** ParseInt(java.lang.String)	123.45
SERIAL8	**Long:** parseLong(java.lang.String)	123
SMALLFLOAT	**Float:** parseFloat(java.lang.String)	123.45
SMALLINT	**Short:** parseShort(java.lang.String)	123
TEXT	**String:** valueOf(java.lang.Object)	Any Character
VARCHAR(m,r)	**String:** valueOf(java.lang.Object)	Any Character

When working with data in the Informix eWay OTDs, use the following conversion chart (Figure 55) as a guide for working with Select operations.

Figure 55 Select Datatype Conversions

Data Type	Java Method or New Constructor to Use(Default: Java Method)	Sample Data
BOOLEAN	**String:** valueOf(boolean)	True/False
CHAR(n)	**Direct Assign**	Any Character
CHARACTER VARYING(m,r)	**Direct Assign**	Any Character
DATE	**String:** valueOf(java.lang.Object)	2004-12-31
DATETIME	**String:** valueOf(java.lang.Object)	2004-12-31 00:00:00
DECIMAL	**String:** valueOf(java.lang.Object)	123.45
DOUBLE PRECISION	**Double:** valueOf(double)	123.45
FLOAT(n)	**String:** valueOf(double)	123.45
INTEGER	**String:** valueOf(int)	123
INT8	**String:** valueOf(long)	123
LVARCHAR	**Direct Assign**	Any Character
MONEY(p,s)	**String:** valueOf(java.lang.Object)	123.45
NCHAR(n)	**Direct Assign**	Any Character
NVARCHAR(m,r)	**Direct Assign**	Any Character
SERIAL	**String:** valueOf(int)	123
SERIAL8	**String:** valueOf(long)	123
SMALLFLOAT	**String:** valueOf(float)	123.45
SMALLINT	**Short:** toString(short)	123
TEXT	**Direct Assign**)	Any Character
VARCHAR(m,r)	**Direct Assign**	Any Character

6.9 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.

6.9.1 The Table

A table OTD represents a database table. It consists of fields and methods. Fields correspond to the columns of a table while methods are the operations that you can apply to the OTD. This allows you to perform query, update, insert, and delete SQL operations in a table.

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

- SetConcurrencytoUpdatable
- SetConcurrencytoReadOnly
- SetScrollTypetoForwardOnly
- SetScrollTypetoScrollSensitive
- SetScrollTypetoInsensitive

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

```
getDBEmp().setConcurToUpdatable();
getDBEmp().setScroll_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.

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() ) );

//@map:Insert Row
   Table_OTD_1.getDB_EMPLOYEE().insertRow();

}
```

The Update Operation

To perform an update operation on a table

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

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

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

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

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

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

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

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

}
```

The Delete Operation

To perform a delete operation on a table

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

In this example DELETE an employee.

```
//@map:Db_employee.delete("first_name like 'John%'")
otdDB_EMPLOYEE_1.getDb_employee().delete("first_name like 'John%'");
```

6.9.2 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" inbound parameter (INLOCALID). This inbound parameter is generated by the DataBase Wizard and is represented in the resulting OTD as nodes. Within the Transformation Designer, you can drag values from the input parameters, and execute the call.

Below are the steps for executing the Stored Procedure:

1 Specify the input values.

2 Execute the Stored Procedure.

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();
    }

}
```

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

DataDirect driver for Infomix requires the following syntax when defining a Stored Procedure ResultSet:

```
CREATE PROCEDURE user_defined (arg0 datatype0, ..., argN datatypeN)
RETURNING returntype AS return_param_name0, ..., returntype AS
return_param_nameN;
...
END PROCEDURE;

CREATE FUNCTION user_defined (arg0 datatype0, ..., argN datatypeN)
RETURNING returntype AS return_param_name0, ..., returntype AS
return_param_nameN;
...
END FUNCTION;
```

The DataDirect driver must use the AS keyword in the returning clause of a Stored Procedure ResultSet to name a returned parameter. The AS keyword is only available in 9.4 version of Informix server, so the Stored Procedure ResultSet is only supported in Informix 9.4.

ResultSets generated in the OTD must match the return parameter name and type in the returning clause in the Stored Procedure definition. The Informix OTD Wizard automatically generates an OTD when using the By Executing mode, but when using the Manual or With Assistance mode, the Original Names must match the returning parameter names defined by the AS keyword. Using the Manual mode requires adding column items manually, but using the With Assistance mode might populate erroneous data in the Original Name column, which must be manually changed to match the return parameter names.

Informix 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.

6.10 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***.

Chapter 1

Using eWay Java Methods

The Informix eWay exposes various Java methods to add extra functionality, making it easier to set, and get information in the Informix eWay OTDs. For a complete list of the Java methods within the classes listed below, refer to the **Javadoc**.

- InformixApplicationConnection
- InformixApplicationConnectionFactory
- InformixCallableStatementAgent
- InformixConnector
- InformixCPManagedConnectionFactoryExt
- InformixNTManagedConnectionFactoryExt
- InformixPreparedStatementAgent
- InformixPreparedStatementResultSet
- InformixSession
- InformixTableResultSet
- InformixXAManagedConnectionFactoryExt

You can find the **Javadoc** in the **InformixeWayDocs.sar** file. Open the **index.html** file to review the Java methods described within the classes listed above.

Index

R

S

U