SeeBeyond ICAN Suite

JDBC/ODBC eWay Intelligent Adapter User's Guide

Release 5.0.2

Version 20041216135552.

Contents

Chapter 1

Introduction 8

About Java Database Connectivity (JDBC) 8
JDBC Drivers 8
Type I: JDBC-ODBC Bridge 9
Type One Driver 9
Type II: Partial Java driver 10
Type Two Driver 11
Pure Java driver for database middleware 11
Type Three Driver 12
Type Four Driver: Direct-to-database pure Java driver 12
About the JDBC/ODBC eWay 14
What's New in This Release 14
About This Document 14
What's in this Document 14
Scope 15
Intended Audience 15
Document Conventions 15
Screenshots 15
Related Documents 16
SeeBeyond Web Site 16
SeeBeyond Documentation Feedback 16

Chapter 2

Installing the eWay 17

Supported Operating Systems 17
System Requirements 18
Installing the eWay Product Files 18
After You Install 19

Chapter 3

Configuring the eWay 20

eWays Properties Overview 20

Connectivity Map Properties 21

Inbound JDBC eWay Properties 21

Pollmilliseconds 21

PreparedStatement 22

Outbound JDBC eWay Properties 22

Description 22

InitialPoolSize 23

LoginTimeOut 23

MaxIdleTime 23

MaxPoolSize 23

MaxStatements 23

MinPoolSize 24

NetworkProtocol 24

PropertyCycle 24

RoleName 24

Outbound JDBC XA eWay Properties 25

Description 25

InitialPoolSize 25

LoginTimeOut 25

MaxIdleTime 26

MaxPoolSize 26

MaxStatements 26

MinPoolSize 26

NetworkProtocol 27

PropertyCycle 27

RoleName 27

Outbound JDBC Non-Transactional eWay Properties 28

Description 28

InitialPoolSize 28

LoginTimeOut 29

MaxIdleTime 29

MaxPoolSize 29

MaxStatements 29

MinPoolSize 29

NetworkProtocol 30

PropertyCycle 30

RoleName 30

Environment Properties 30

Inbound eWay Environment Properties 31

ClassName 31

Password 31

URL 32

User 32

Outbound JDBC eWay Environment Properties 32

ClassName 33

DatabaseName 33

DataSourceName 33

Delimiter 34
Description 34
DriverProperties 34
Password 34
PortNumber 35
ServerName 35
User 35
Outbound JDBC XA eWay Environment Properties 36
ClassName 36
DatabaseName 36
DataSourceName 37
Delimiter 37
Description 37
DriverProperties 37
Password 37
PortNumber 38
ServerName 38
User 38
Outbound JDBC Non-Transactional eWay Environment 39
ClassName 39
DatabaseName 39
DataSourceName 40
Delimiter 40
Description 40
DriverProperties 40
Password 40
PortNumber 41
ServerName 41
User 41

Chapter 4

Using the JDBC/ODBC eWay Database Wizard 42

Select Wizard Type 43
Connect To Database 44
Select Database Objects 44
Select Table/Views 45
Select Procedures 48
Add Prepared Statements 50
Specify the OTD Name 52
Data Types for OTDs 54
Object Type Definition Applications 55
The Table OTD 56
The Query Operation 56
The Insert Operation 57
The Update Operation 58
The Delete Operation 58

The Stored Procedure OTD 59
Executing Stored Procedures 59
Prepared Statement OTD 60
Batch Operations 62

Chapter 5

eWay Sample Projects 63

About the Sample Projects 64
Requirements 64
Sample Projects Files 64
Sample Project Contents 64
JDBC_BPEL_Sample 65
JDBC_JCE_Sample 65
Sample Project Data 66
Sample Projects Drivers 66

Locating the Sample Projects 67

Importing the Sample Projects 67

Running the Sample Projects 68
Setting the Properties 69
Creating the Environment Profile 69
Configuring the Logical Host 70
Deploying a Project 71
Running the Sample 72

Building Business Logic with eInsight 72
Business Process Overview 73
Building Business Processes 74
Using the where() Clause 74
Using Triggers 74
SelectOne 74
Insert 76
Update 78
Delete 79

Additional Business Logic with eInsight 81
SelectAll 81
SelectMultiple 82
Adding Connectivity Maps 83
Building Outbound Connectivity Maps 83

Building Business Logic with eGate 84
Building Collaborations 84
Adding Connectivity Maps 85
Building Outbound Connectivity Maps 85

Alerting and Logging 86

Appendix A

JDBC/ODBC Drivers 87

AS/400 88
OTD Wizard: Database Connection Information 88
Environment Properties 88

Attunity Driver 89
OTD Wizard: Database Connection Information 89
Environment Properties 90

MySQL Connector/J Driver 91
OTD Wizard: Database Connection Information 91
Environment Properties 91

SyBase JConnect V5 92
OTD Wizard: Database Connection Information 92
Environment Properties 92

Sequelink DataDirect Informix ODBC Driver 93
OTD Wizard: Database Connection Information 93
Environment Properties 94

MS Access via MS Access ODBC Driver 94
OTD Wizard: Database Connection Information 95
Environment Properties 95

Data Types for Drivers 96

Installing JDBC/ODBC Drivers 97

Troubleshooting 97

Index 98

Chapter 1

Introduction

This chapter provides an overview of the Java database connectivity (JDBC) and open database connectivity (ODBC) APIs. This chapter also introduces the JDBC/ODBC eWay.

What's In This Chapter

- **About Java Database Connectivity (JDBC)** on page 8
- **About the JDBC/ODBC eWay** on page 14
- **What's New in This Release** on page 14
- **About This Document** on page 14
- **Related Documents** on page 16
- **SeeBeyond Web Site** on page 16
- **SeeBeyond Documentation Feedback** on page 16

1.1 About Java Database Connectivity (JDBC)

Java Database Connectivity (JDBC) is an implementation of the Java programming language that dictates how databases communicate with each other. Through a standardized application programming interface (API), connectivity from database management systems (DBMS) to a wide range of SQL databases is accomplished. By deploying database drivers laced with JDBC technology, it is possible to connect to any database---even in a heterogeneous environment---and access: tables, tabular data, flat files and more. When using JDBC, Java programmers have the ability to: request connections to a database; send queries to the database using SQL statements; and receive results for advanced processing.

In This Section:

- **"JDBC Drivers" on page 8**

1.1.1 JDBC Drivers

To connect with individual databases, JDBC requires drivers for each database. Those drivers come in four varieties. Driver types 1 and 2 are typically intended for programmers that write applications. Driver types 3 and 4 are typically used by

database and middleware vendors. The various driver types are described in the following sections:

- **"Type I: JDBC-ODBC Bridge" on page 9**
- **"Type II: Partial Java driver" on page 10**
- **"Pure Java driver for database middleware" on page 11**
- **"Type Four Driver: Direct-to-database pure Java driver" on page 12**

Type I: JDBC-ODBC Bridge

This combination provides JDBC access via ODBC drivers. ODBC binary code--and in many cases, database client code--must be loaded on each client machine that uses a JDBC-ODBC Bridge. A product called SequeLink from Data Direct Technologies provides a driver that supports some ODBC drivers (for example Microsoft Access).

Type one drivers provide JDBC access via one or more Open Database Connectivity (ODBC) drivers. ODBC, which predates JDBC, is widely used by developers to connect to databases in a non-Java environment.

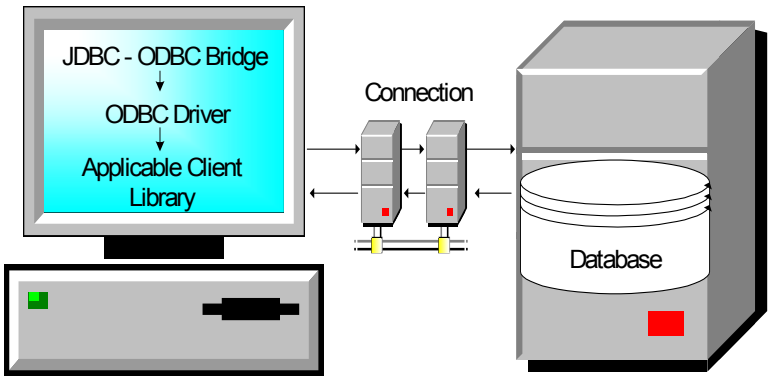
Pros: A good approach for learning JDBC. May be useful for companies that already have ODBC drivers installed on each client machine — typically the case for Windows-based machines running productivity applications. May be the only way to gain access to some low-end desktop databases.

Cons: Not for large-scale applications. Performance suffers because there's some overhead associated with the translation work to go from JDBC to ODBC. Doesn't support all the features of Java. User is limited by the functionality of the underlying ODBC driver.

Type One Driver

A JDBC/ODBC bridge provides JDBC API access through one or more ODBC drivers. Some ODBC native code and in many cases native database client code must be loaded on each client machine that uses this type of driver.

Figure 1 Typical Type 1 Driver Configuration



The pros and cons for using this type of driver are as follows:

Pros

- Allows access to almost any database since the databases ODBC drivers are readily available

Cons

- Performance is degraded since the JDBC call goes through the bridge to the ODBC driver then to the native database connectivity interface. The results are then sent back through the reverse process
- Limited Java feature set
- May not be suitable for a large-scale application

Type II: Partial Java driver

This type of driver converts JDBC calls into calls on the client API for Oracle, Sybase, Informix, DB2, or other DBMS. Note that, like the bridge driver, this style of driver requires that some binary code be loaded on each client machine.

This type of driver converts the calls that a developer writes to the JDBC application programming interface into calls that connect to the client machine's application programming interface for a specific database, such as IBM, Informix, Oracle or Sybase.

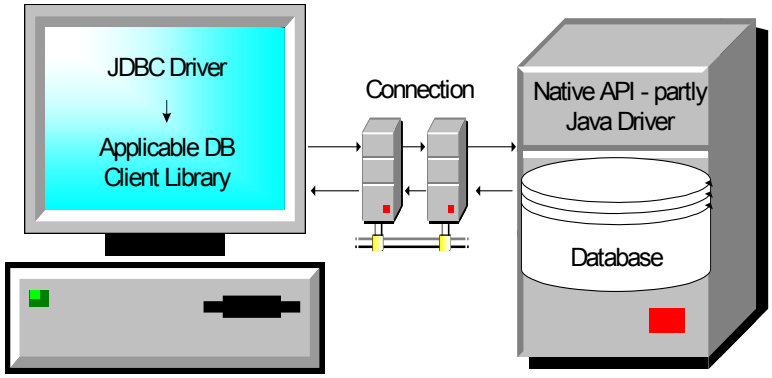
Pros: Performance is better than that of Type 1, in part because the Type 2 driver contains compiled code that's optimized for the back-end database server's operating system.

Cons: User needs to make sure the JDBC driver of the database vendor is loaded onto each client machine. Must have compiled code for every operating system that the application will run on. Best use is for controlled environments, such as an intranet.

Type Two Driver

A native-API partly Java technology-enabled driver converts JDBC calls into calls on the client API for DBMSs. Like the bridge driver, this style of driver requires that some binary code be loaded on each client machine. An example of this type of driver is the Oracle Thick Driver, which is also called OCI.

Figure 2 Typical Type 2 Driver Configuration



The pros and cons for using this type of driver are as follows:

Pros

- Allows access to almost any database since the databases ODBC drivers are readily available
- Offers significantly better performance than the JDBC/ODBC Bridge
- Limited Java feature set

Cons

- Applicable Client library must be installed
- Type 2 driver shows lower performance than type 3 or 4

Pure Java driver for database middleware

This style of driver translates JDBC calls into the middleware vendor's protocol, which is then translated to a DBMS protocol by a middleware server. The middleware provides connectivity to many different databases. which provides connectivity to many different databases.

This driver translates JDBC calls into the middleware vendor's protocol, which is then converted to a database-specific protocol by the middleware server software.

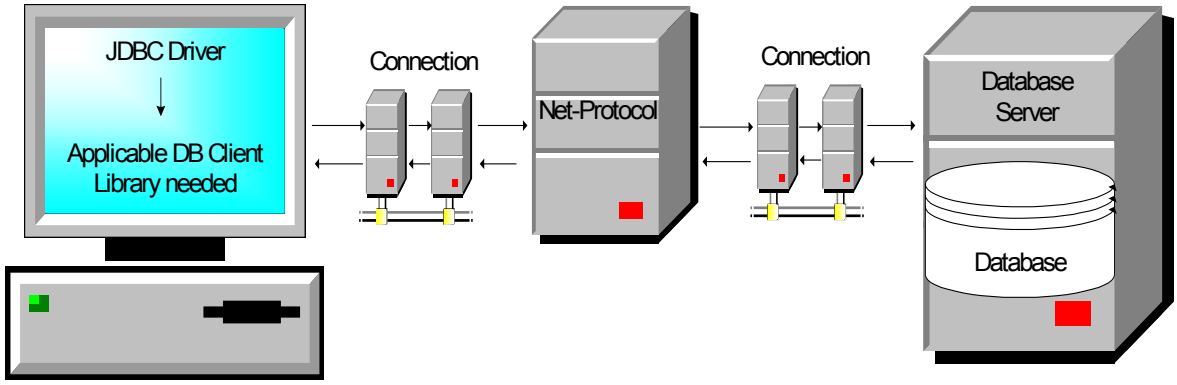
Pros: Better performance than Types 1 and 2. Can be used when a company has multiple databases and wants to use a single JDBC driver to connect to all of them. Server-based, so no need for JDBC driver code on client machine. For performance reasons, the back-end server component is optimized for the operating system that the database is running on.

Cons: Needs some database-specific code on the middleware server. If the middleware must run on different platforms, a Type 4 driver might be more effective.

Type Three Driver

A net-protocol fully Java-enabled driver translates JDBC API calls into a DBMS-independent net protocol which is then translated to a DBMS protocol by a server. This net server middleware is able to connect all of its Java technology-based clients to many different databases.

Figure 3 Typical Type 3 Middleware Driver Configuration



The pros and cons for using this type of driver are as follows:

Pros

- Allows access to almost any database since the databases ODBC drivers are readily available
- Offers significantly better performance than the JDBC/ODBC Bridge and Type 2 Drivers
- Advanced Java feature set
- Scalable
- Caching
- Advanced system administration
- Does not require applicable database client libraries

Cons

- Requires a separate JDBC middleware server to translate specific native-connectivity interface.

Type Four Driver: Direct-to-database pure Java driver

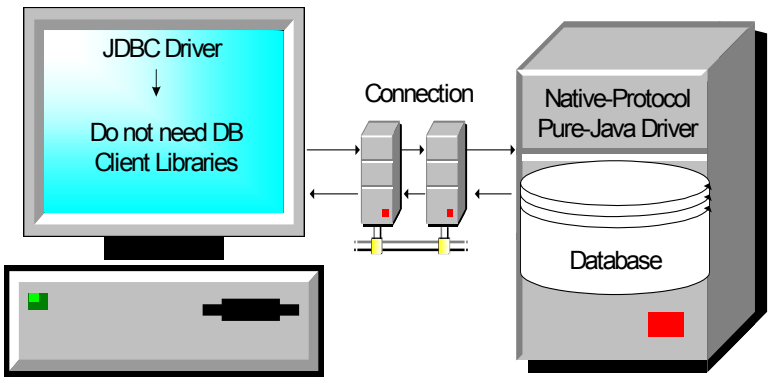
This style of driver converts JDBC calls into a network protocol that sends the converted packets--in a proprietary format--to be used directly by DBMSs; thus, allowing a direct call from the client machine to the DBMS server and providing a practical solution for intranet access. This type of driver has become very popular recently and is supported by most database software vendors. All JDBC drivers from Data Direct Technologies (driver vendor) are Type 4 drivers. Another example of this type of driver is the Oracle Thin driver.

Pros: Better performance than Types 1 and 2. No need to install special software on client or server. Can be downloaded dynamically.

Cons: Not optimized for server operating system, so the driver can't take advantage of operating system features. (The driver is optimized for the database and can take advantage of the database vendor's functionality.) User needs a different driver for each different database.

A native-protocol fully Java technology-enabled driver converts JDBC technology calls into the network protocol used by DBMSs directly. This allows a direct call from the client machine to the DBMS server.

Figure 4 Typical Type 4 Driver Configuration



The pros and cons for using this type of driver are as follows:

Pros

- Allows access to almost any database since the databases ODBC drivers are readily available
- Offers significantly better performance than the JDBC/ODBC Bridge and Type 2 Drivers
- Scalable
- Caching
- Advanced system administration
- Superior performance
- Advance Java feature set
- Does not require applicable database client libraries

Cons

- Each database will require a driver

1.2 About the JDBC/ODBC eWay

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

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

1.3 What's New in This Release

The JDBC eWay 5.0.2 provides support for international platforms.

1.4 About This Document

- **What's in this Document** on page 14
- **Scope** on page 15
- **Intended Audience** on page 15
- **Document Conventions** on page 15
- **Screenshots** on page 15

1.4.1 What's in this Document

This guide contains the following information:

- **Chapter 1 "Introduction"**: 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 eGate eWay.
- **Chapter 3 "Introduction"**: Provides instructions for configuring the eWay to communicate with JDBC drivers.

- **Chapter 4 "Using the JDBC/ODBC eWay Database Wizard"**: Provides instructions for creating Object Type Definitions to be used with the eGate eWay.
- **Chapter 5 "eWay Sample Projects"**: Provides instructions for installing and running the sample Projects.

1.4.2 Scope

This document describes the process of installing, configuring, and running the JDBC 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 JDBC 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 conventions are observed throughout this document.

Table 1 Document Conventions

Text	Convention	Example
Names of buttons, files, icons, parameters, variables, methods, menus, and objects	**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, code samples	`Fixed` font. Variables are shown in ***`bold italic`***.	`bootstrap -p` ***`password`***
Hypertext links	**Blue** text	See **"Document Conventions" on page 15**
Hypertext links for Web addresses (URLs) or email addresses	**Blue underlined** text	**http://www.seebeyond.com** **docfeedback@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 Suite:

- *SeeBeyond Integrated Composite Application Network Suite Primer*
- SeeBeyond ICAN Suite Installation Guide
- *eGate Integrator User's Guide*
- eGate Integrator Tutorial

SeeBeyond ICAN Suite Deployment 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 SeeBeyond Documentation Feedback

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

docfeedback@seebeyond.com

Chapter 2

Installing the eWay

This chapter describes how to install the JDBC/ODBC eWay.

What's in this Chapter

- **Supported Operating Systems** on page 17
- **System Requirements** on page 18
- **Installing the eWay Product Files** on page 18
- **After You Install** on page 19

2.1 Supported Operating Systems

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

- Windows Server 2003, Windows XP, and Windows 2000
- Sun Solaris 8 and 9
- HP Tru64 5.1A
- HP-UX 11.0, 11i (PA-RISC), and11i V2 (11.23)
- IBM AIX 5.1L and 5.2
- Red Hat Linux 8 (Intel x86)
- Red Hat Enterprise Linux AS 2.1 (Intel x86)
- Japanese Windows Server 2003, Windows XP, and Windows 2000
- Japanese Sun Solaris 8 and 9
- Japanese HP-UX 11.0, 11i (PA-RISC), and11i V2 (11.23)
- Japanese IBM AIX 5.1L and 5.2
- Korean Windows Server 2003, Windows XP, and Windows 2000
- Korean Sun Solaris 8 and 9
- Korean HP-UX 11.0, 11i (PA-RISC), and11i V2 (11.23)
- Korean IBM AIX 5.1L and 5.2

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

2.2 System Requirements

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

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

- An eGate Logical Host
- A TCP/IP network connection.
- The appropriate driver type for your database.
- Database drivers installed on the Enterprise Designer machine.
- Database drivers installed on the Logical Host machine.

2.3 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 SeeBeyond *ICAN Suite Installation Guide*, and include the following steps:

- On the Enterprise Manager, select the **JDBCeWay.sar** (to install the eWay) file to upload.
- On the Enterprise Manager, select the **FileeWay.sar** (to install the File eWay, used in the sample Project) file to upload.
- On the Enterprise Manager, install the **JDBCeWayDocs.sar** (to install the documentation, and the sample) file to upload.
- On the Enterprise Manager click on the Documentation tab. Click the document link, or the sample file link. For the sample project, it is recommended that you

extract the file to another file location prior to importing it using the Enterprise Explorer's Import Project tool.

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

Continue installing the eGate Enterprise Designer as instructed.

2.4 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.

Chapter 3

Configuring the eWay

This chapter describes configuring the inbound/outbound eWay in the Connectivity Map as well as the Environment.

What's In This Chapter

- **eWays Properties Overview** on page 20
- **Connectivity Map Properties** on page 21
- **Environment Properties** on page 30

3.1 eWays Properties Overview

eWay properties must be configured from within the Connectivity Map. Select eWays may require additional configuration for the Logical Host Environment. Until you have successfully configured all eWays for your ICAN project, your project cannot be properly executed. The following list identifies the JDBC eWays that must be configured.

eWays Configured in the Connectivity Map

- **Inbound JDBC eWay Properties** on page 21
- **Outbound JDBC eWay Properties** on page 22
- **Outbound JDBC XA eWay Properties** on page 25
- **Outbound JDBC Non-Transactional eWay Properties** on page 28

eWays Configured in the Logical Host Environment

- **Inbound eWay Environment Properties** on page 31
- **Outbound JDBC eWay Environment Properties** on page 32
- **Outbound JDBC XA eWay Environment Properties** on page 36
- **Outbound JDBC Non-Transactional eWay Environment** on page 39

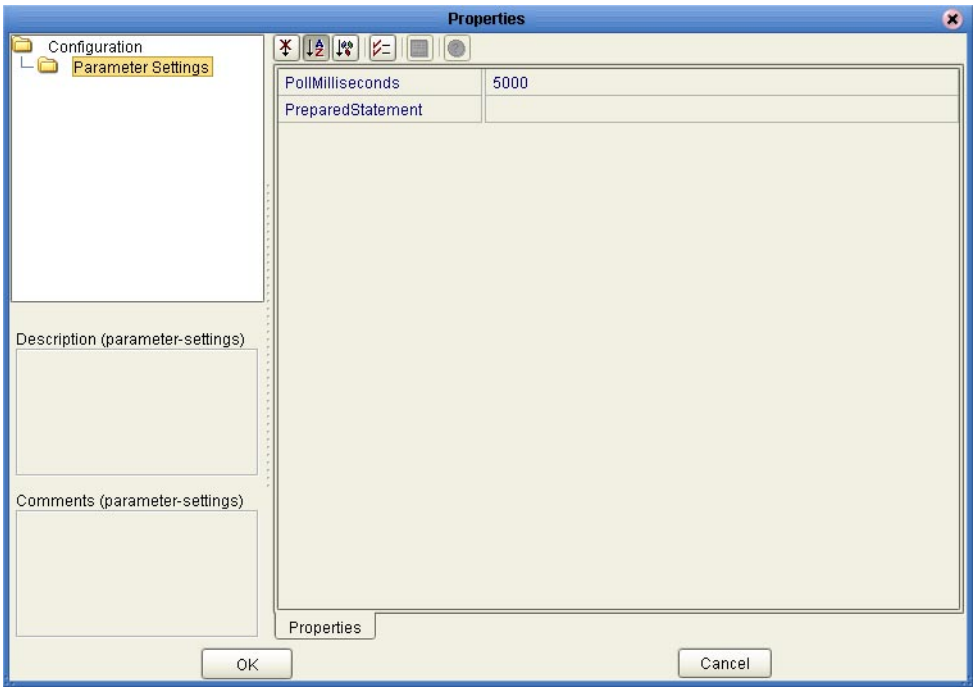
3.2 Connectivity Map Properties

There exists four eWay connection types that JDBC/ODBC eWay implements. The properties that may be configured from the Connectivity Map are listed below:

- **Inbound JDBC eWay Properties** on page 21
- **Outbound JDBC eWay Properties** on page 22
- **Outbound JDBC XA eWay Properties** on page 25
- **Outbound JDBC Non-Transactional eWay Properties** on page 28

3.2.1 Inbound JDBC eWay Properties

Figure 5 Properties of the Inbound JDBC eWay



Pollmilliseconds

Description

Specifies the polling interval in milliseconds.

Required Value

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

PreparedStatement

Description

Prepared Statement used to query against the database.

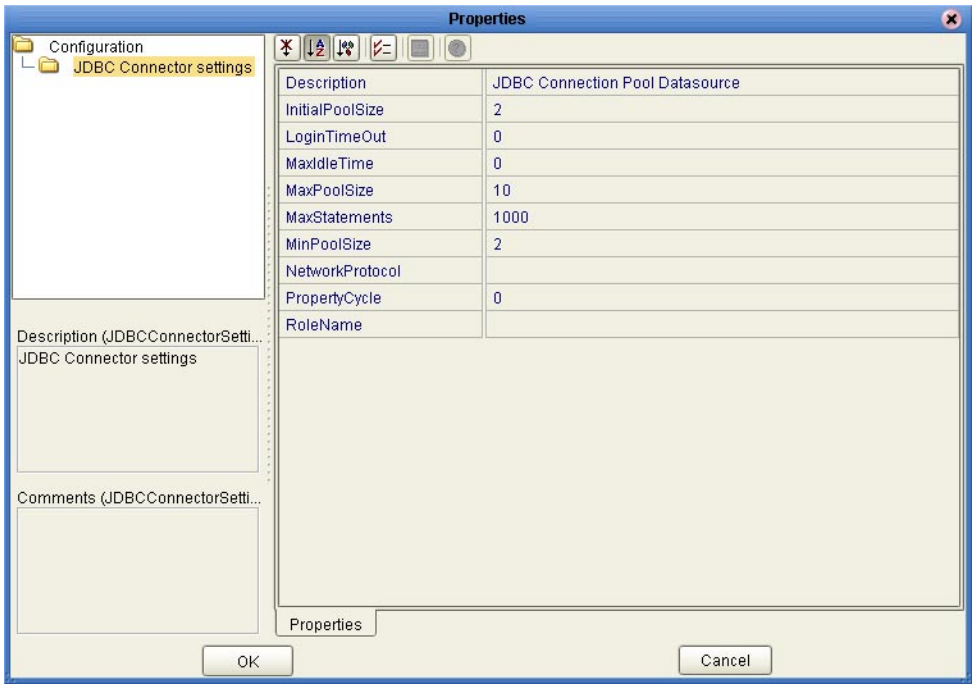
Required Value

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

3.2.2 Outbound JDBC eWay Properties

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

Figure 6 The Outbound JDBC eWay Properties



Description

Description

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

Required Values

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

InitialPoolSize

Description

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

Required Value

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

LoginTimeOut

Description

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

Required Value

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

MaxIdleTime

Description

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

Required Value

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

MaxPoolSize

Description

Specifies the maximum number of physical connections the pool contains.

Required Value

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

MaxStatements

Description

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

Required Value

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

MinPoolSize

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

Required Value

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

NetworkProtocol

Description

Specifies the network protocol used to communicate with the server.

Required Values

The applicable network protocol.

PropertyCycle

Description

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

Required Values

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

RoleName

Description

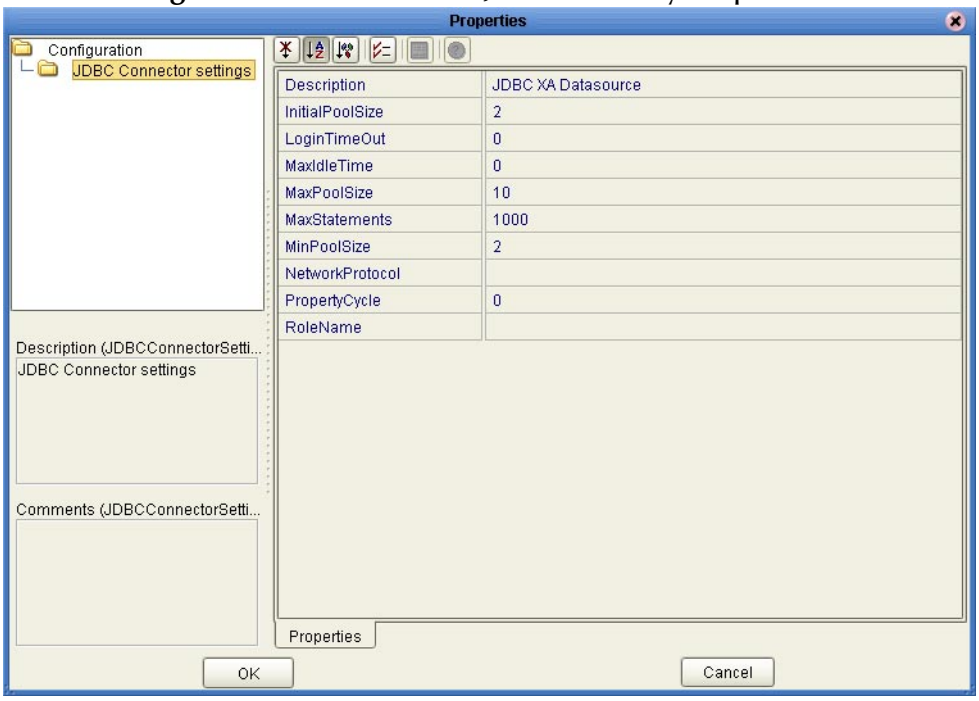
Specifies the initial SQL role name.

Required Values

The applicable initial SQL role name (String).

3.2.3 Outbound JDBC XA eWay Properties

Figure 7 The Outbound JDBC XA eWay Properties



Description

Description

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

Required Values

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

InitialPoolSize

Description

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

Required Value

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

LoginTimeOut

Description

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

Required Value

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

MaxIdleTime

Description

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

Required Value

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

MaxPoolSize

Description

Specifies the maximum number of physical connections the pool contains.

Required Value

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

MaxStatements

Description

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

Required Value

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

MinPoolSize

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

Required Value

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

NetworkProtocol

Description

Specifies the network protocol used to communicate with the server.

Required Values

The applicable network protocol.

PropertyCycle

Description

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

Required Values

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

RoleName

Description

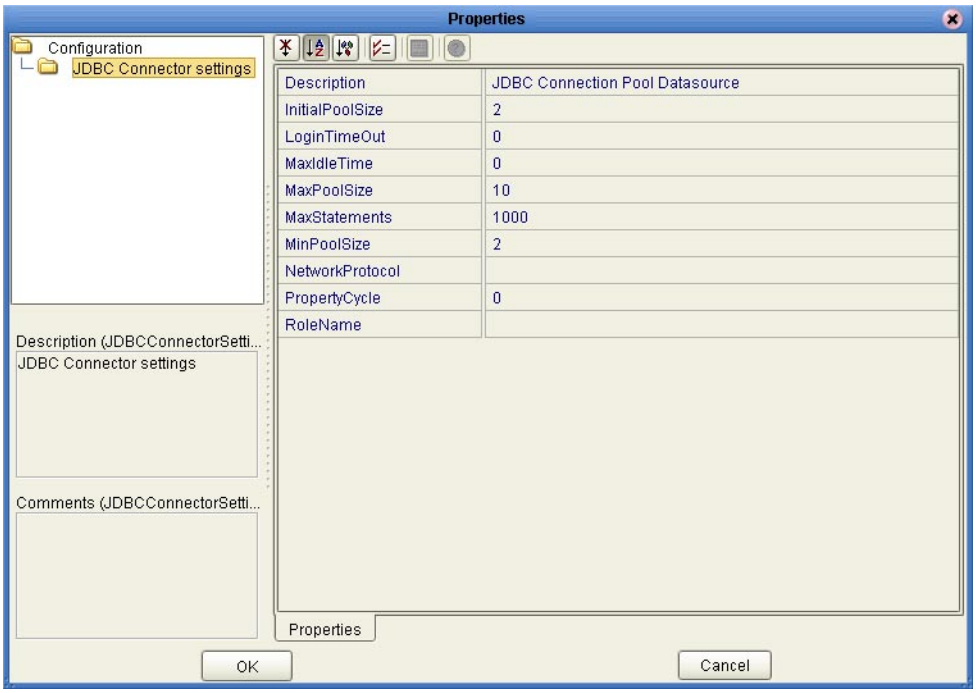
Specifies the initial SQL role name.

Required Values

The applicable initial SQL role name (String).

3.2.4 Outbound JDBC Non-Transactional eWay Properties

Figure 8 Properties of the Outbound JDBC Non-Transactional eWay



Description

Description

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

Required Values

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

InitialPoolSize

Description

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

Required Value

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

LoginTimeOut

Description

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

Required Value

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

MaxIdleTime

Description

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

Required Value

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

MaxPoolSize

Description

The maximum number of physical connections the pool contains.

Required Value

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

MaxStatements

Description

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

Required Value

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

MinPoolSize

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

Required Value

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

NetworkProtocol

Description

Specifies the network protocol used to communicate with the server.

Required Values

The applicable network protocol.

PropertyCycle

Description

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

Required Values

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

RoleName

Description

Specifies the initial SQL role name.

Required Values

The applicable initial SQL role name (String).

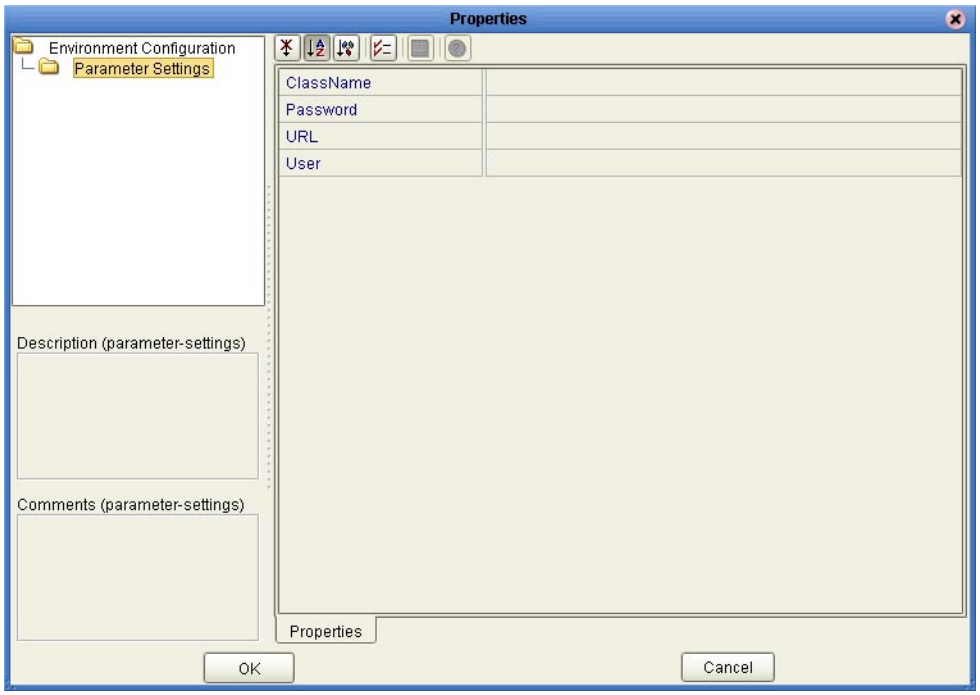
3.3 Environment Properties

There exists four eWay connection types that JDBC/ODBC eWay implements. The properties that may be configured from the Environment Explorer are listed below:

- **Inbound eWay Environment Properties** on page 31
- **Outbound JDBC eWay Environment Properties** on page 32
- **Outbound JDBC XA eWay Environment Properties** on page 36
- **Outbound JDBC Non-Transactional eWay Environment** on page 39

3.3.1 Inbound eWay Environment Properties

Figure 9 Inbound eWay Environment



ClassName

Description

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

Required Values

An applicable class name.

Password

Description

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

Required Values

The correct password assigned to the User name.

URL

Description

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

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

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

Required Values

The applicable JDBC URL.

User

Description

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

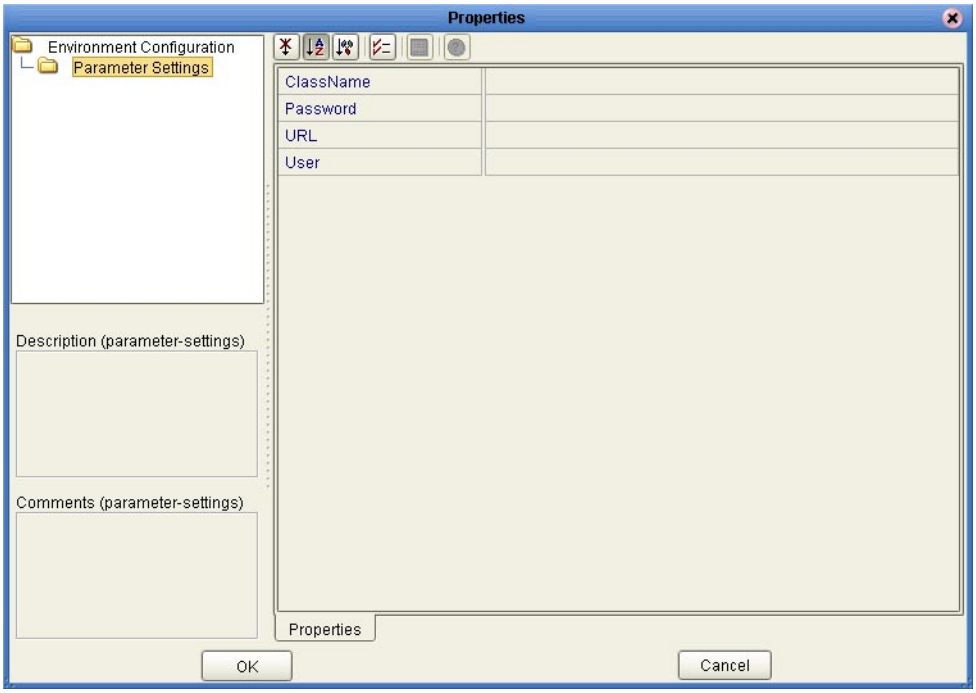
Required Values

An applicable user name.

3.3.2 Outbound JDBC eWay Environment Properties

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

Figure 10 Outbound JDBC eWay Environment Configuration



ClassName

Description

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

Required Values

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

DatabaseName

Description

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

Required Values

An applicable database name.

DataSourceName

Description

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

Required Value

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

Delimiter

Description

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

Required Value

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

Description

Description

Provides a description of the database.

Required Value

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

DriverProperties

Description

Specifies the driver properties. You must 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. You must ensure that the driver is installed on both the Logical Host machine and the Enterprise Designer machine.

Required Value

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

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

Password

Description

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

Required Values

The correct password assigned to the User name.

PortNumber

Description

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

Required Values

The appropriate port number. For example, 4100.

ServerName

Description

Specifies the host name of the external database server.

Required Values

An applicable server name.

User

Description

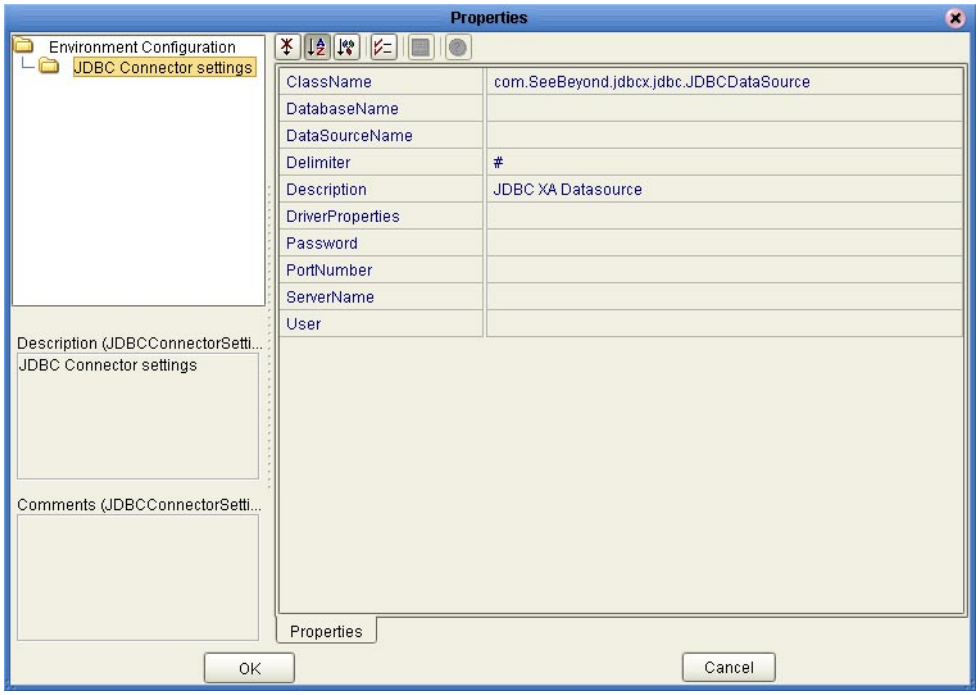
Specifies an available user name with authority to access the database. This is used in conjunction with the **Password** property.

Required Values

An applicable user name.

3.3.3 Outbound JDBC XA eWay Environment Properties

Figure 11 Outbound JDBC XA eWay Environment Properties



ClassName

Description

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

Required Values

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

DatabaseName

Description

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

Required Values

An applicable database name.

DataSourceName

Description

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

Required Value

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

Delimiter

Description

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

Required Value

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

Description

Description

Provides a description of the database.

Required Value

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

DriverProperties

Description

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

Required Value

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

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

Password

Description

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

Required Values

The correct password assigned to the User name.

PortNumber

Description

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

Required Values

The appropriate port number. For example, 4100.

ServerName

Description

Specifies the host name of the external database server.

Required Values

An applicable server name.

User

Description

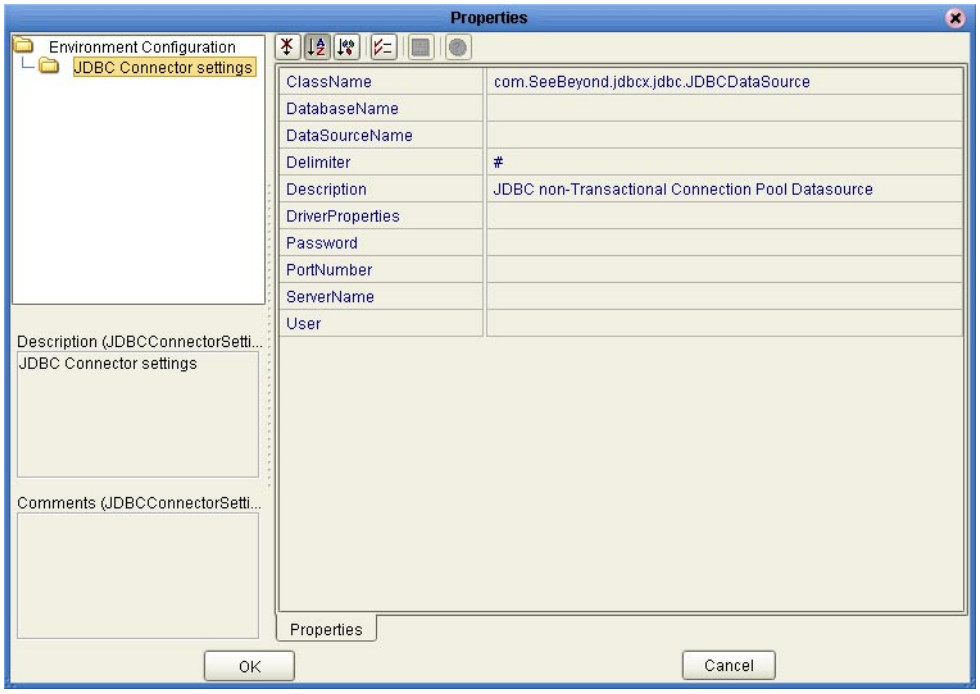
Specifies an available user name with authority to access the database. This is used in conjunction with the **Password** property.

Required Values

An applicable user name.

3.3.4 Outbound JDBC Non-Transactional eWay Environment

Figure 12 Outbound JDBC Non-Transactional eWay Environment Properties



ClassName

Description

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

Required Values

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

DatabaseName

Description

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

Required Values

An applicable database name.

DataSourceName

Description

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

Required Value

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

Delimiter

Description

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

Required Value

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

Description

Description

Provides a description of the database.

Required Value

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

DriverProperties

Description

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

Required Value

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

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

Password

Description

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

Required Values

The correct password assigned to the User name.

PortNumber

Description

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

Required Values

The appropriate port number. For example, 4100.

ServerName

Description

Specifies the host name of the external database server.

Required Values

An applicable server name.

User

Description

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

Required Values

An applicable user name.

Chapter 4

Using the JDBC/ODBC eWay Database Wizard

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

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

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

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

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

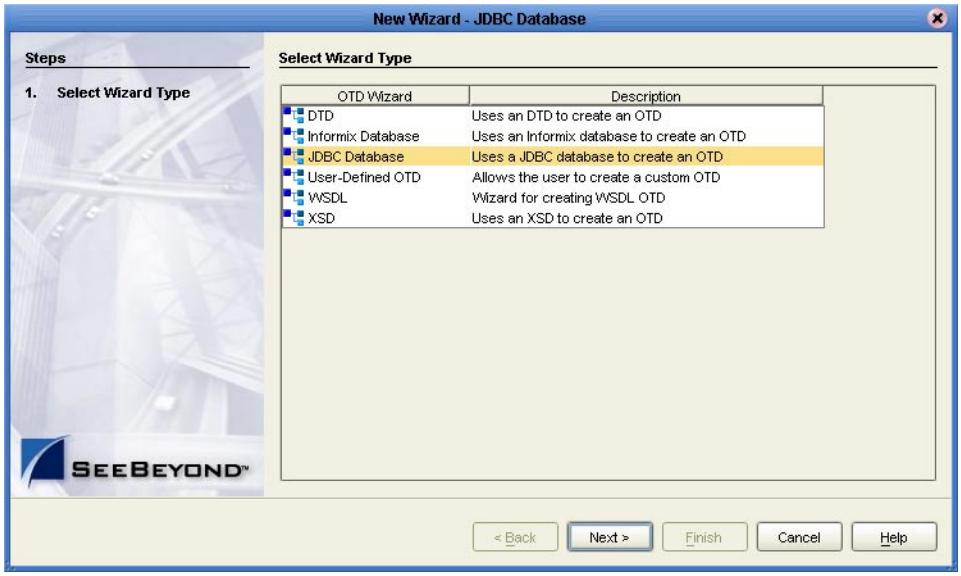
What's In This Chapter:

- **Select Wizard Type** on page 43
- **Connect To Database** on page 44
- **Select Database Objects** on page 44
- **Select Table/Views** on page 45
- **Select Procedures** on page 48
- **Add Prepared Statements** on page 50
- **Specify the OTD Name** on page 52
- **Data Types for OTDs** on page 54
- **Object Type Definition Applications** on page 55

4.1 Select Wizard Type

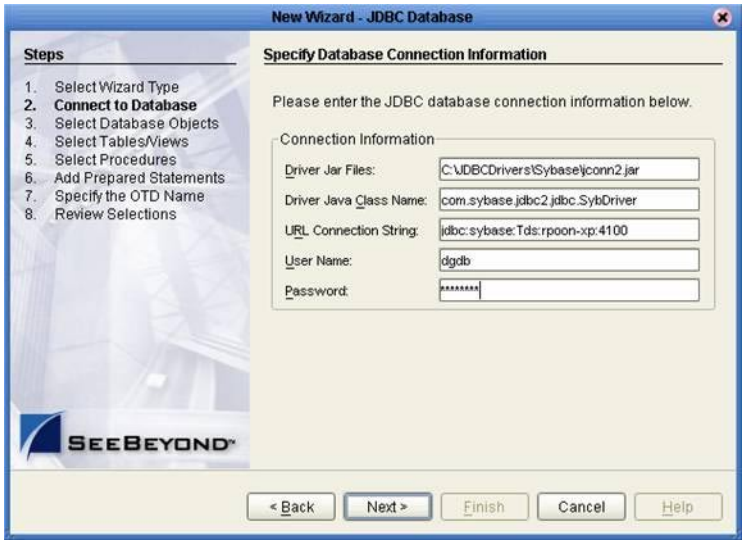
1. On the Enterprise Explorer, right click on the project and select **New > Create an Object Type Definition** from the shortcut menu.
2. The **Select Wizard Type** window appears, displaying the available **OTD** wizards (see Figure 13).

Figure 13 OTD Wizard Selection



3. From the list, select the JDBC Database OTD and click **Next**. The **Specify Database Connection Information** window appears (see Figure 14).

Figure 14 Database Connection Information



4.2 Connect To Database

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

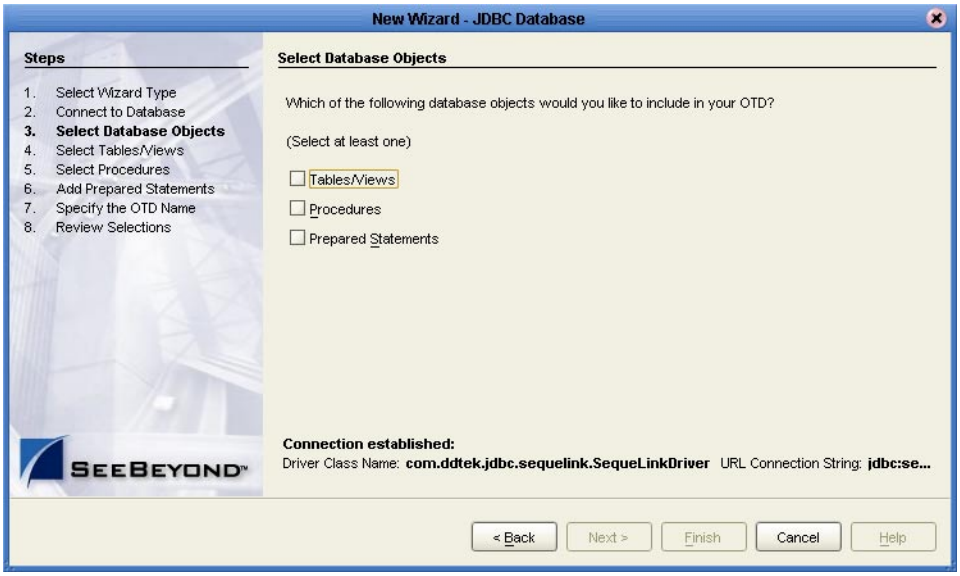
- **Driver Jar Files** – the absolute path to driver Jar files, separated by semicolons.
- **Driver Java Class Name** – the driver class name.
- **URL Connection String** – the URL connection string.
- **User Name** – the user name required to log into the database.
- **Password** – the password required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

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

4.3 Select Database Objects

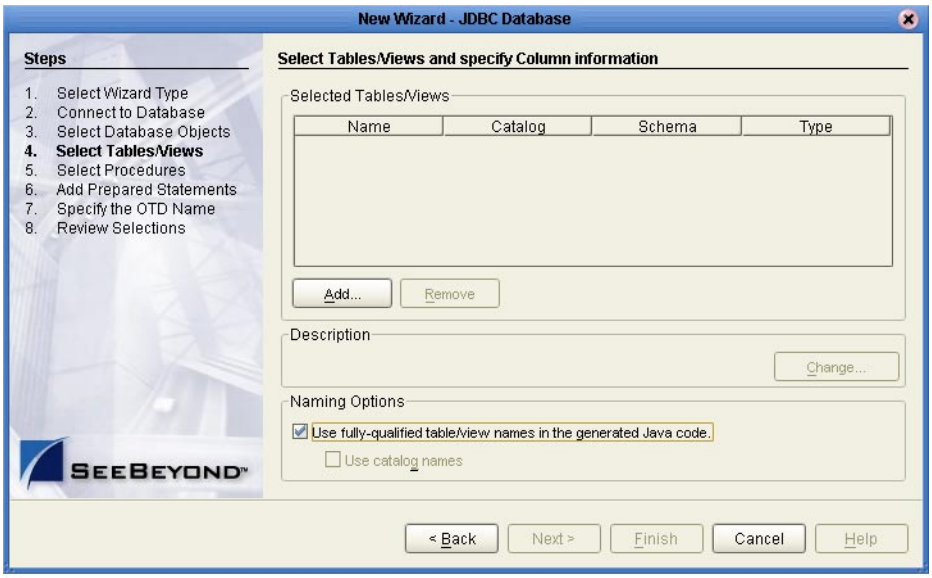
1 On the Select Database Objects window, select **Tables/Views**, **Procedures**, and **Prepared Statements** check boxes (see Figure 15).

Figure 15 Select Database Objects



2 Click **Next**. The **Select Tables/Views** window appears. See **Figure 16 on page 45**.

Figure 16 Select Tables/Views



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

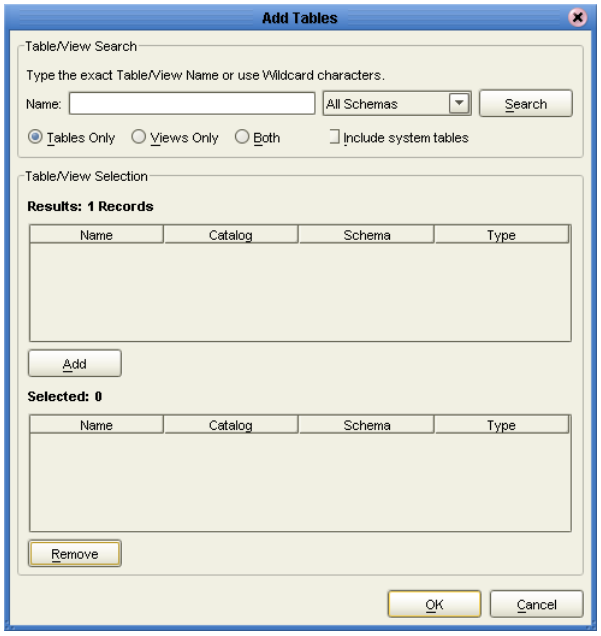
Note: *Not all drivers provide metadata information such as column names and data types. If your table does not have column names and data types, add them before saving the OTD.*

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

4.4 Select Table/Views

1 On the Select Tables/Views window, click the **Add** button. The **Add Tables** window appears. See **Figure 17 on page 46**.

Figure 17 Add Tables Window

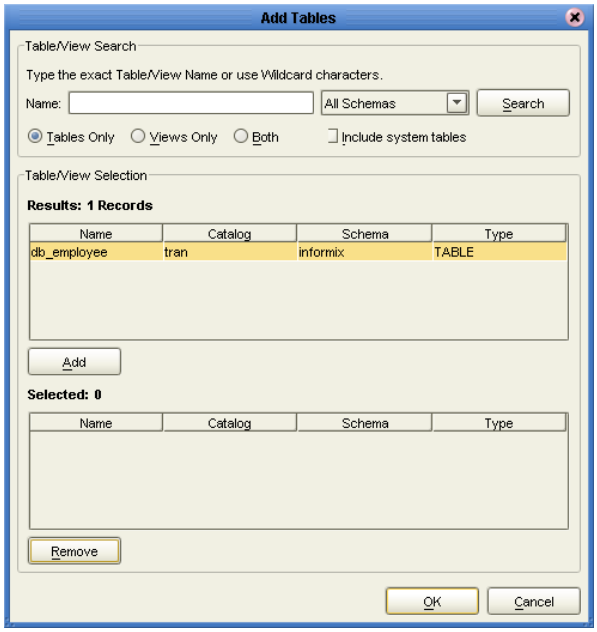


2 In the **Add Tables** window, select whether your selection criteria will include table data, view only data, both, and/or system tables.

3 From the **Table/View Name** drop down list, select the location of your database table and click **Search** (see Figure 18).

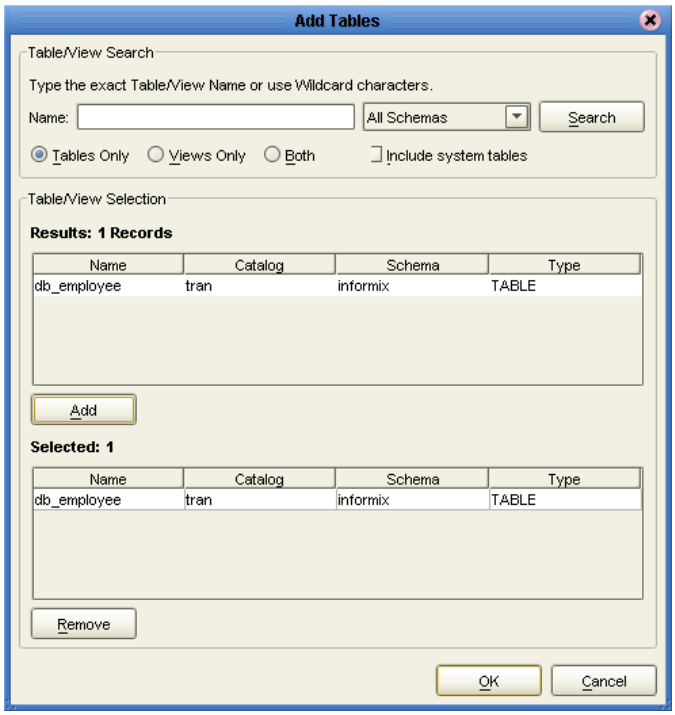
Note: *Click Search to find the desired table or tables. You can also use wildcard characters to search for a table or view. Available wildcard characters include the "?", "_", and "*". For example, you can use "AB?CD", "AB_CD", or "AB*CD". However, do not use "%". Using this character results in nothing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

Figure 18 Database Wizard - All Schemes



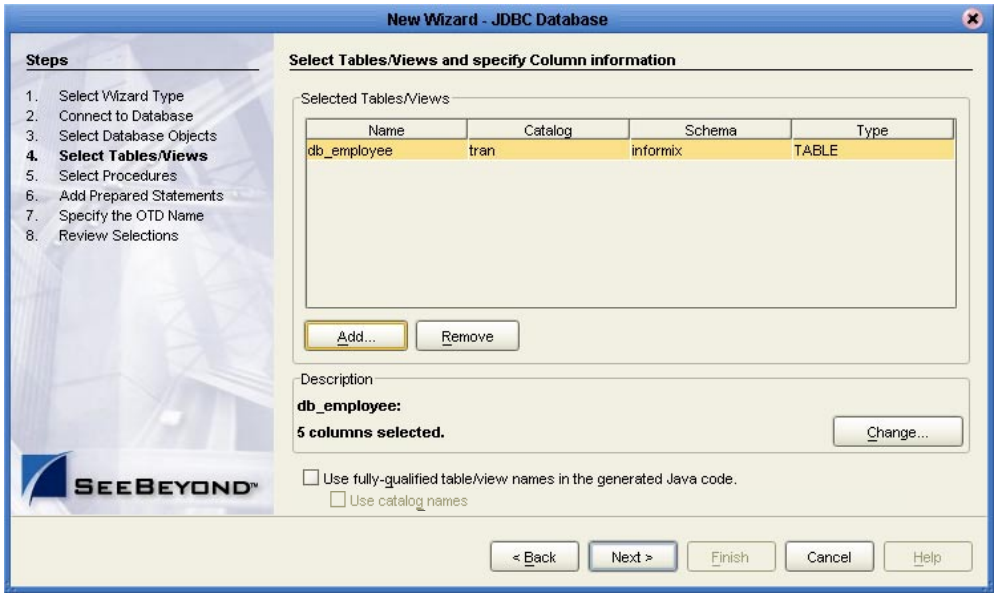
4 Select a table and click **OK**. The selected table is added to the **Selected Tables/Views** window. See Figure 19.

Figure 19 Selected Tables/Views window with a table selected



5 On the Selected Tables/Views window, review your selected tables. To make changes to the selected Table or view, click **Change**, or click **Next** if no additional changes are required (see Figure 20).

Figure 20 Selected table and column window

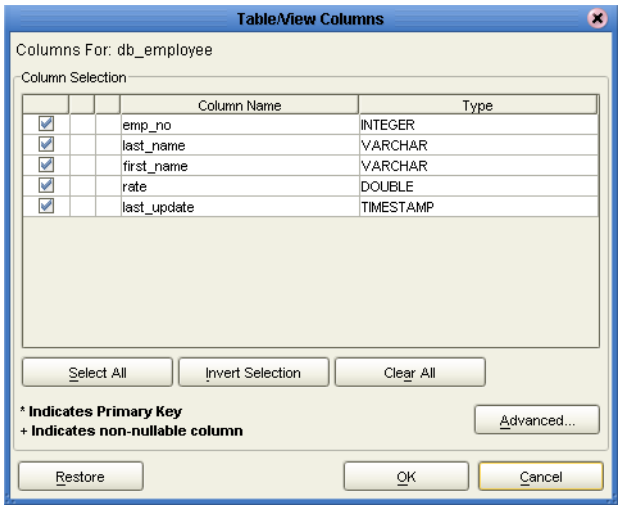


6 If you click **Change**, the Table/View Columns window appears, allowing you to select or deselect any table columns. You can also change the data type for each table by highlighting the data type and selecting a different data type from the drop-down list. Double check the data types the driver is returning. If they are not correct, change them to the appropriate types (see Figure 21).

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 21 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.

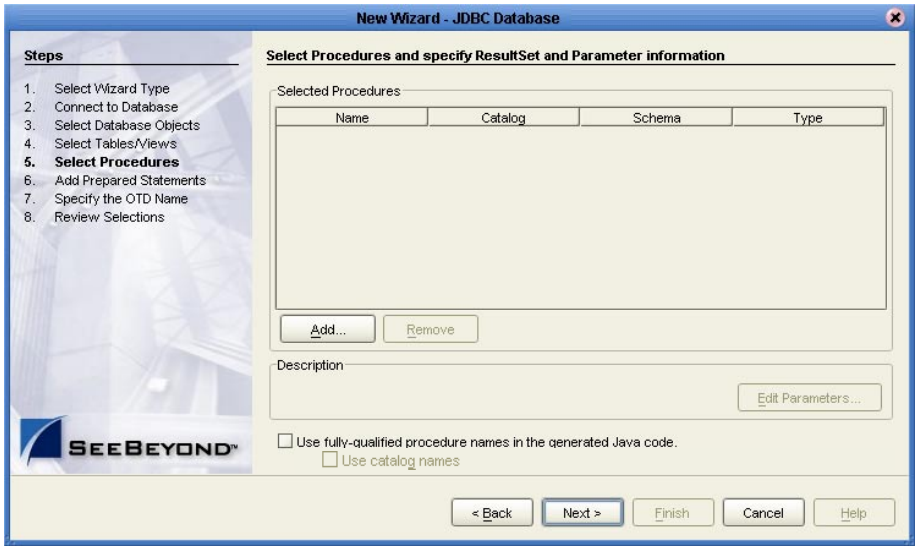
8 When using Prepared Statement packages, select **Use fully qualified table/view names in the generated code**. See **Figure 22 on page 49**.

4.5 Select Procedures

Note: *Only stored procedures are supported. Stored procedures with ResultSet are not supported.*

1 On the **Select Procedures and specify ResultSet and Parameter Information** window, click **Add** (see Figure 22).

Figure 22 Select Procedures window

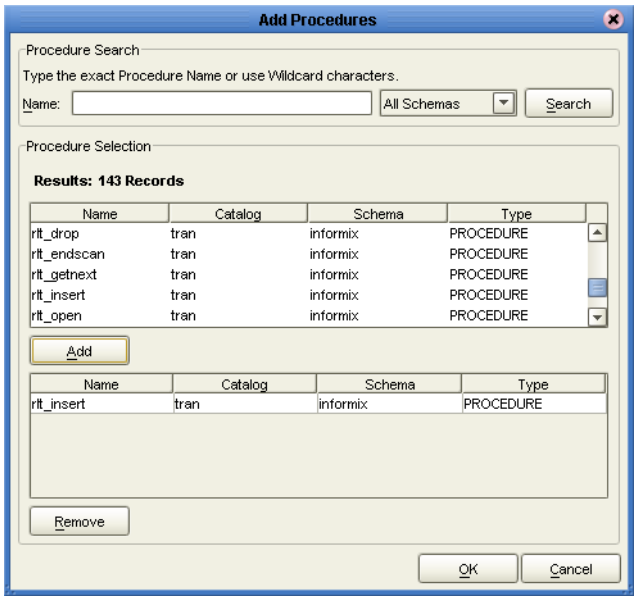


2 On the **Select Procedures** window, enter the name of a Procedure or select a table from the drop down list. Click **Search**. Wildcard characters can also be used.

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

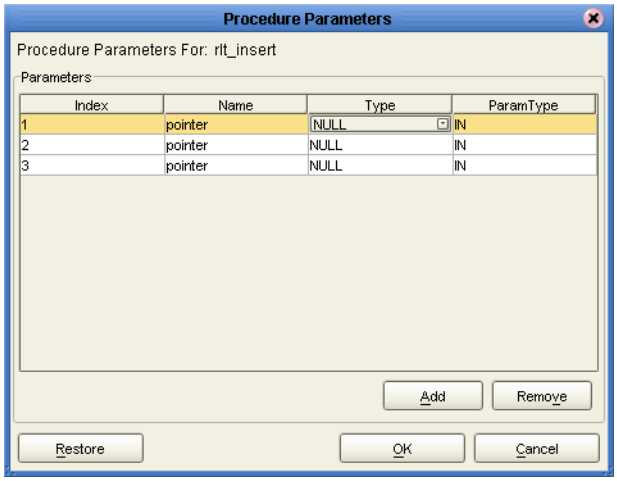
3 In the resulting **Procedure Selection** list box, select a Procedure (see Figure 23). Click **OK**.

Figure 23 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 24).

Figure 24 Procedure Parameters



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

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

4.6 Add Prepared Statements

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

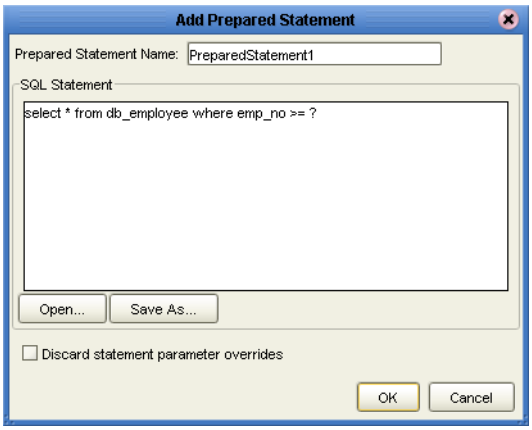
You can use prepared statements to perform insert, update, delete, and query operations. A prepared statement uses a question mark (?) as a place holder for input. **For example**: insert into EMP_TAB(Age, Name, Dept No) values(?, ?, ?).

Note: *If the prepared statement performs either the insert, update, or delete operations, the* ***executeUpdate()*** *method is created when the OTD is built. If the prepared statement performs only the Select operation, the* ***executeQuery()*** *method is created when the OTD is built.*

1 On the **Add Prepared Statements** window, click **Add**. The **Add Prepared Statement** window appears (see Figure 25).

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

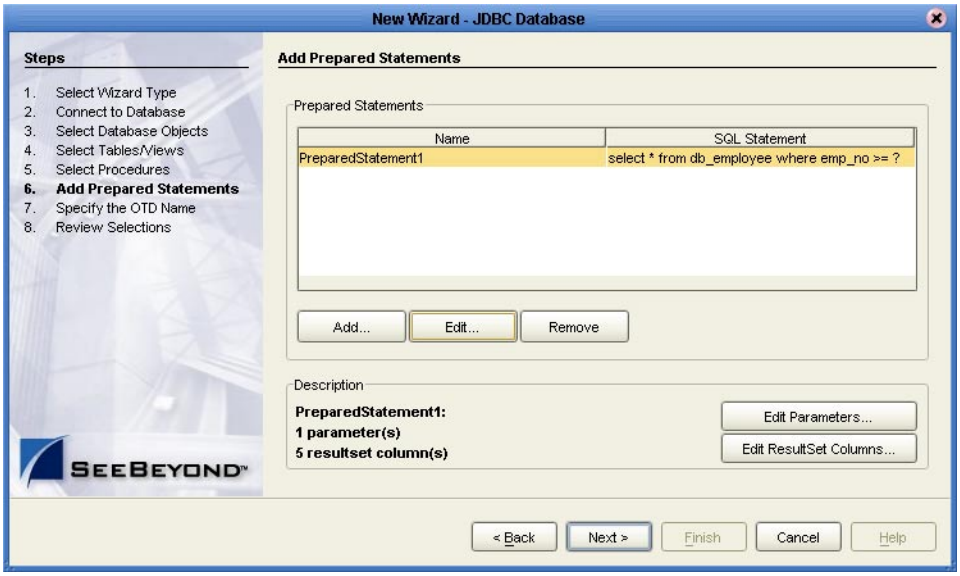
Figure 25 Prepared SQL Statement



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

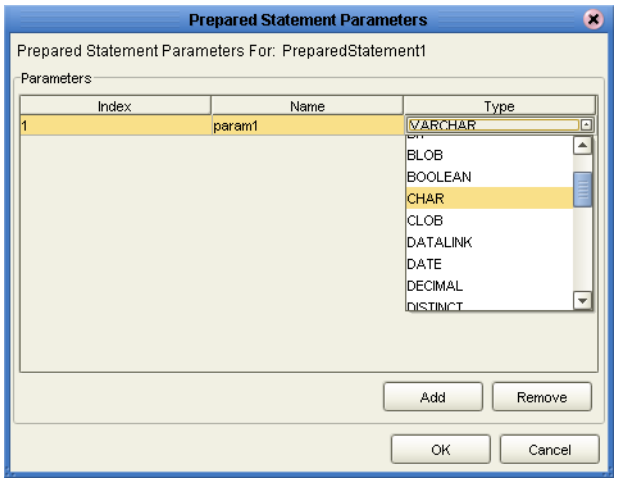
On the **Add Prepared Statement** window, the name you assigned to the Prepared Statement appears (seeFigure 26).

Figure 26 Add Prepared Statement window



4 To edit the parameters, click **Edit Parameters**. You can change the audiotape by clicking in the **Type** field and selecting a different type from the list (see Figure 27).

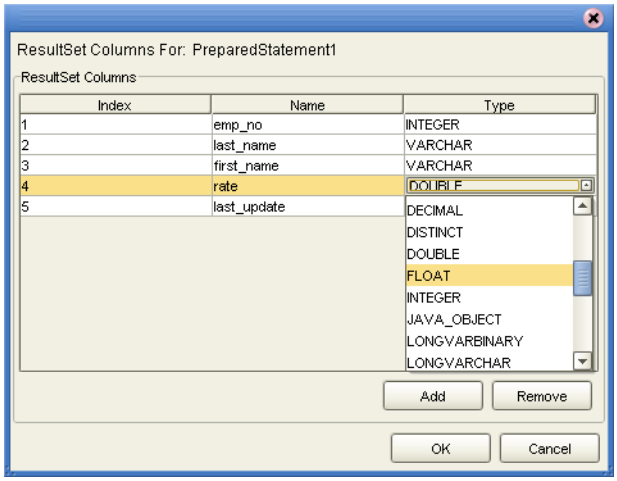
Figure 27 Edit the Prepared Statement Parameters.



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

6 To edit the ResultSet Columns, click **Edit ResultSet Columns**. Both the Name and Type are editable. Click **OK** to save your changes and exit the window (see **Figure 28 on page 52**.

Figure 28 ResultSet Columns



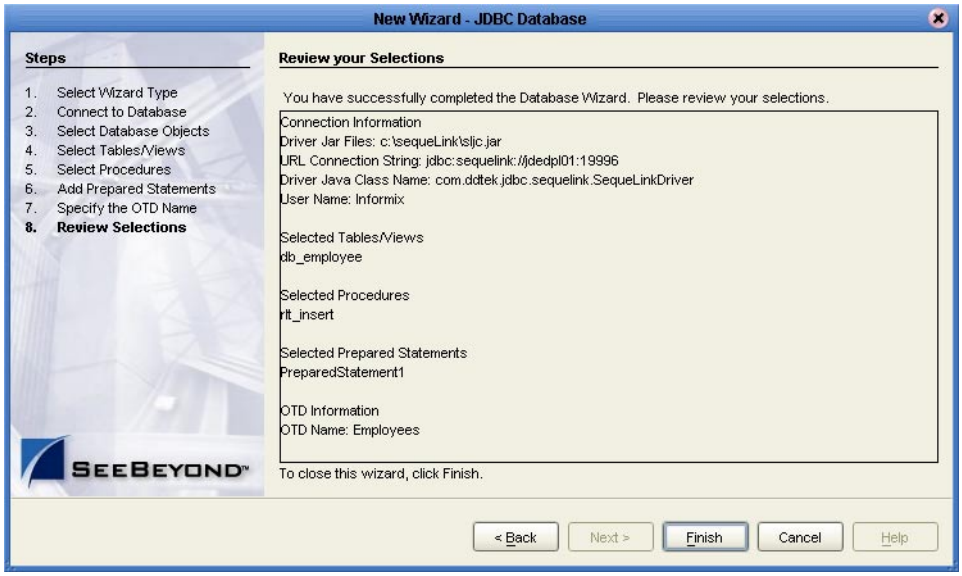
4.7 Specify the OTD Name

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 29**.

Figure 29 Naming an OTD

2 Review the summary of the OTD (see **Figure 30 on page 53**). Click **Back** if necessary, to return to a previous screen to make any corrections**.**

Figure 30 Database Wizard - Summary



If you are satisfied with the OTD information, click **Finish** to begin generating the OTD. When complete, the resulting **OTD** appears in the OTD Editor.

4.8 Data Types for OTDs

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

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

Data Type	OTD/Java Data Type	Java Method or New Constructor to Use (Default: Java Method)	Sample Data
Int	BigDecimal	**Call a New Constructor BigDecimal**: java.math.BigDecimal(String)	123
Smallint	BigDecimal	**Call a New Constructor BigDecimal**: java.math.BigDecimal(String)	123
Number	BigDecimal	**Call a New Constructor BigDecimal**: java.math.BigDecimal(String)	123
Decimal*	BigDecimal	**Call a New Constructor BigDecimal**: java.math.BigDecimal(String)	147.78
Real	Double	**Double**: java.lang.Double.parseDouble(String)	147.78
Float	Double	**Double**: java.lang.Double.parseDouble(String)	147.78
Double	Double	**Double**: java.lang.Double.parseDouble(String)	147.78
Date	TimeStamp	**TimeStamp**: java.sql.TimeStamp.valueOf(String)	2003-08-11 11:47:39.0
Varchar2	String	Direct Assign	Any character
Char	String	Direct Assign	Any character
Long Char	String	Direct Assign	Any character
Raw	Byte[]	**String**: java.lang.String.getBytes()	Any character
Long Raw	Byte[]	**String**: java.lang.String.getBytes()	Any character
CLOB	Clob	See Appendix	Any character

Table 3 Sample Select Operation Audiotape Conversion (Text/String output data)

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

4.9 Object Type Definition Applications

Tables, Views, and Stored Procedures, and Prepared Statements are manipulated through OTDs. This section describes how to use OTDs in one of the four common implementation scenarios in which you will likely deploy a JDBC/ODBC OTD. Depending on the driver you use, some of these OTDs may not be supported.

Note: *Some drivers do not support updatable ResultSets for Insert, Update and Delete. Use a Prepared Statement instead.*

Related Topics:

- **The Table OTD** on page 56

- **The Stored Procedure OTD** on page 59
- **Prepared Statement OTD** on page 60
- **Batch Operations** on page 62

4.9.1 The Table OTD

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

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

- SetConcurrencytoUpdatable
- SetConcurrencytoReadOnly
- SetScrollTypetoForwardOnly
- SetScrollTypetoScrollSensitive
- SetScrollTypetoInsensitive

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

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

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

Table Topics:

- **The Query Operation** on page 56
- **The Insert Operation** on page 57
- **The Update Operation** on page 58
- **The Delete Operation** on page 58

The Query Operation

To perform a query operation on a table

1. Execute the **select()** method with the "**where**" clause specified if necessary.
2. Loop through the ResultSet using the **next()** method.
3. Process the return record within a **while()** loop.

For example:

```
package SelectSales;
public class Select
{

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

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

}
```

The Insert Operation

To perform an insert operation on a table

1. Execute the **insert()** method. Assign a field.
2. Insert the row by calling **insertRow()**

   This example inserts an employee record.

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

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

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

//@map:Copy java.lang.Float.parseFloat(Rate) to Rate
```

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

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

}
```

The Update Operation

To perform an update operation on a table

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

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

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

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

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

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

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

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

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

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

The Delete Operation

To perform a delete operation on a table

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

2 Move to the row that you want to delete.

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

In this example DELETE an employee.

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

4.9.2 The Stored Procedure OTD

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

Stored Procedure Topics:

- **Executing Stored Procedures** on page 59

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();
    }

}
```

4.9.3 Prepared Statement OTD

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

You can use prepared statements to perform insert, update, delete and query operations. A prepared statement uses a question mark (?) as a place holder for input.

For example:

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

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

To Insert

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


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


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


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


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


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


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


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


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

}
```

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

To Update

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


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


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

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


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

    }
```

4.9.4 Batch Operations

While the Java API used by SeeBeyond does not support traditional bulk insert or update operations, there is an equivalent feature that can achieve comparable results, with better performance. This is the "Add Batch" capability. The only modification required is to include the **addBatch()** method for each SQL operation and then the **executeBatch()** call to submit the batch to the database server. Batch operations apply only to Prepared Statements.

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

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

Chapter 5

eWay Sample Projects

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

What's in This Chapter

- **About the Sample Projects** on page 64
- **Locating the Sample Projects** on page 67
- **Importing the Sample Projects** on page 67
- **Running the Sample Projects** on page 68
- **Building Business Logic with eInsight** on page 72
- **Additional Business Logic with eInsight** on page 81
- **Building Business Logic with eGate** on page 84
- **Alerting and Logging** on page 86

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

5.1 About the Sample Projects

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

More About the Sample Projects:

- **Requirements** on page 64
- **Sample Projects Files** on page 64
- **Sample Project Contents** on page 64
- **Sample Project Data** on page 66
- **Sample Projects Drivers** on page 66

5.1.1 Requirements

The JDBC eWay sample projects require the installation of the JDBC eWay. Furthermore, you must have access to the JDBC drivers that the eWay requires to communicate with the desired database architectures. The JDBC eWay project samples also require the following eWays to be installed:

- File eWay
- JDBC eWay

5.1.2 Sample Projects Files

The compressed JDBC_Sample_Projects.zip file contains the following files:

- **JDBC_JCE_Sample.zip** – demonstrates how to use the insert, update, delete, and select employee records from the db_employee table using eGate Integrator.
- **JDBC_JCE_Sample.zip** – demonstrates how to insert, update, delete, and select employee records from the db_employee table using eInsight's BPEL business process.
- **select.txt** – input file used for the select Collaboration.
- **trigger.txt** – input file used for all BPEL business processes.
- **update.txt** – input file used for the update Collaboration.
- **insert.txt** – input file used for the insert Collaboration.
- **delete.txt** – input file used for the delete Collaboration.
- **preparedstatement.txt** – input file used for the prepared statement Collaboration.

5.1.3 Sample Project Contents

Each Project contains the following:

- Input data

- Connectivity Maps
- Collaborations Definitions
- Inbound and outbound Collaborations

The sample projects provide a project that allows you to browse its configurations to learn how inbound and outbound JDBC Projects are designed. The Projects do not include ICAN Environments and Deployment Profiles necessary to deploy the sample Projects. To learn how to complete the Projects for deployment, refer to **"Running the Sample" on page 72**.

JDBC_BPEL_Sample

The **JDBC_BPEL_Sample** sample Project uses input files to pass data into business process. There are four Collaborations that demonstrate the Insert, Update, Delete, and Select operations. Results are written out to an output file while the input files are contained in the sample Project folder.

Business Processes used in the Project:

- **bpelTableInsert** – inserts a new row for a "John Smith" that contains a rate of "125" and time stamp of "2004-02-06 11:50:00".
- **bpelTableUpdate** – updates any record with a last name like "Smith" to have a last name of "Doe" and a new rate of "33.88".
- **bpelTableDelete** – deletes the table db_employee.
- **bpelTableSelect** – selects all employees with an employee number greater than "100" and writes the results of the employee number and last name to a text file.
- **bpelPreparedStatement** – uses a prepared statement to select all records from the db_employee table, and write the last names to a text file.

JDBC_JCE_Sample

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

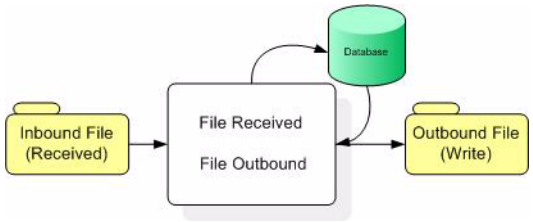
Collaborations used in the sample Project include:

- **jceTableInsert** – inserts a new row for a "John Smith" that contains a rate of "125" and time stamp of "2004-02-06 11:50:00".
- **jceTableUpdate** – updates any record with a last name like "Smith" to have a last name of "Doe" and a new rate of "33.88".
- **jceTableDelete** – deletes the table db_employee.
- **jceTableSelect** – selects all employees with an employee number greater than "100" and writes the results of the employee number and last name to a text file.

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

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

Figure 31 Database project flow



Additional instructions on working with sample Projects are provided in the *eGate Integrator Tutorial*.

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

5.1.4 Sample Project Data

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

Table 4 Sample Project data – db_employee table

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

5.1.5 Sample Projects Drivers

Sample Projects included with this eWay were built using the DataDirect Technology SequeLink driver (not included). You must get this or a different driver from a third-party vendor and install and configure the eWay as per the vendors specification, using the vendor's specific driver jar file and appropriate driver specific parameters such as driver class name, URL connection string, etc.

When using any driver to build OTDs, be sure to include the absolute path in the Database Connection Information window. See **Figure 14 on page 43**.

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

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

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

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

5.2 Locating the Sample Projects

The eWay sample Projects are included in the **JDBCeWayDocs.sar**. This file is uploaded separately from the JDBC eWay sar file during installation. For information, refer to **"Installing the eWay" on page 16**.

Once you have uploaded the **JDBCeWayDocs.sar** to the Repository and you have downloaded the sample Projects (**JDBC_ODBC_eWay_Sample.zip**) using the **DOCUMENTATION** tab in the Enterprise Manager, the sample resides in the folder you specified during the download.

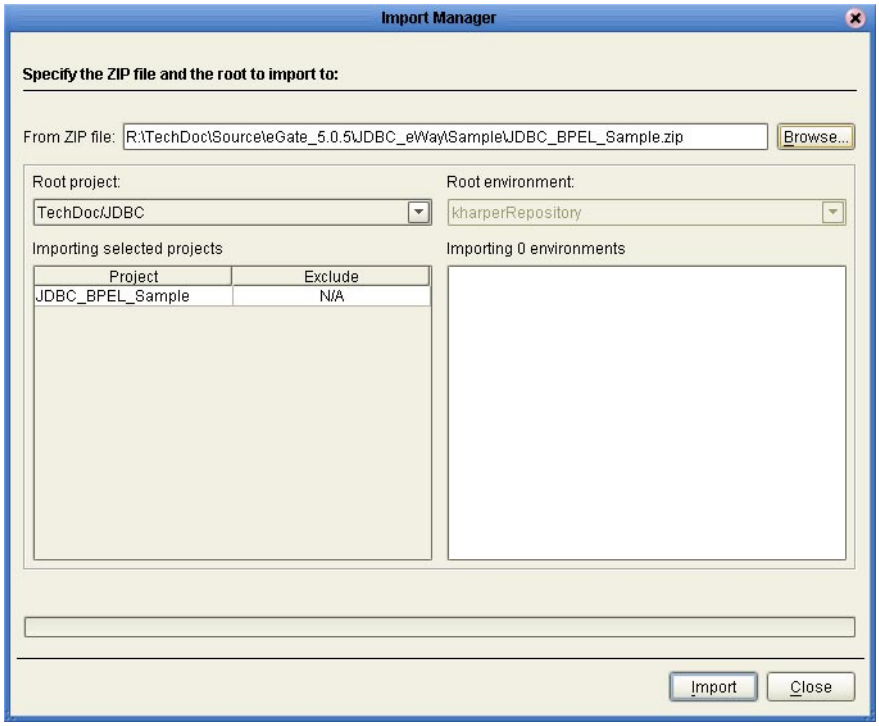
5.3 Importing the Sample Projects

You can import the JDBC sample Projects as described below. To find out where the Projects reside, refer to **"Locating the Sample Projects" on page 67**.

To import the sample Projects

1. Unzip the **JDBC_ODBC_eWay_Sample.zip** file. This creates the following zip files:
   - **JDBC_BPEL_Sample.zip** (for the eGate Project)
   - **JDBC_JCE_Sample.zip**(for the eGate Project)
2. In the **Project Explorer** tab of the Enterprise Designer, right-click the Repository and click **Import**. The **Import Manager** dialog box appears.
3. Click **Browse** and navigate to the folder where you unzipped the sample zip file.
4. Click the desired sample file. The **Import Manager** dialog box appears similar to the following:

Figure 32 Import Manager Dialog Box



5 Click **Import**. A dialog box confirms that the Project import was successful.

6 Click **OK** and click **Close**.

You can now explore the Connectivity Maps, the OTDs, and the business logic for the Collaborations or Business Processes.

5.4 Running the Sample Projects

Steps required to run a sample Project include:

- **Setting the Properties** on page 69
- **Creating the Environment Profile** on page 69
- **Deploying a Project** on page 71
- **Running the Sample** on page 72

The sample Projects do not include the Environments, Deployment Profiles, and the physical configurations for the eWays needed to deploy the Projects. To deploy the Projects, do the following after import:

1 Configure the eWay properties– see **"Building Business Logic with eGate" on page 84**.

2 Create the Environment Profile – see **"Creating the Environment Profile" on page 69**.

3 Configure the eWay Environment Properties - see **"Environment Properties" on page 30** for assistance.

4 Configure the logical host - see **"Configuring the Logical Host" on page 70**.

5 Deploy the Project - see **"Deploying a Project" on page 71**.

6 Run the Sample Project - see **"Running the Sample" on page 72**

5.4.1 Setting the Properties

The samples uses both inbound and outbound File eWays, as well as an outbound JDBC eWay. Use the following information to configure the sample Project eWays. For additional information on the eWay properties, see **Configuring the eWay** on page 20.

To Configure File eWays:

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

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

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

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

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

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

To Configure the Outbound JDBC eWay:

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

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

5.4.2 Creating the Environment Profile

The procedure below describes how you create an eGate Environment for the JDBC sample Projects. For detailed information about creating Environments, refer to the *eGate Integrator User's Guide*.

To create Environments for the sample Projects

1 In the Environment Explorer tab of the Enterprise Designer, right-click the Repository and click **New Environment**.

2 Right-click the Environment and click **New File External System** to add a File eWay. The list below shows which systems to add for which Business Process/ Collaboration:

- Inbound: one inbound File eWay and one outbound File eWay
- Outbound: one inbound File eWay and one outbound File eWay

3 Right-click the Environment and click **New JDBC External System** to add an JDBC eWay. The list below shows which systems to add for which Business Process/ Collaboration:

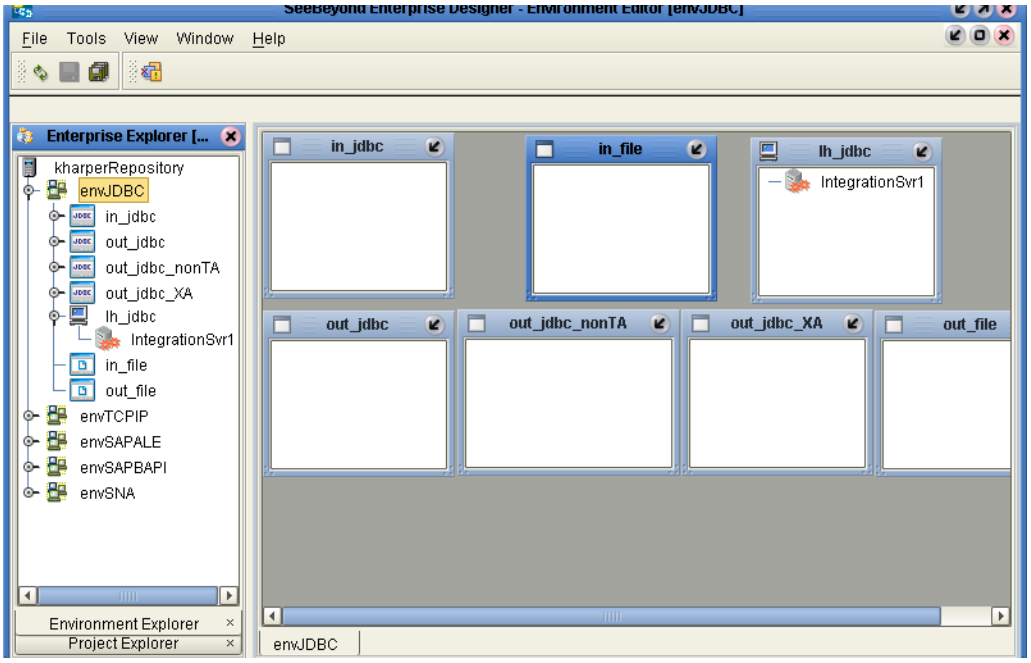
- Inbound: one inbound JDBC eWay and one outbound JDBC eWay
- Outbound: one inbound JDBC eWay and one outbound JDBC eWay

4 Right-click the Environment and click **New Logical Host**.

5 Right-click the Logical Host and click **New SeeBeyond Integration Server.**

Figure 33 shows the completed Environment.

Figure 33 Environment for Sample Projects



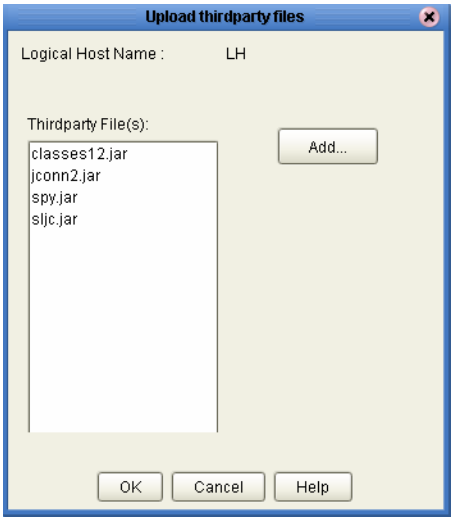
5.4.3 Configuring the Logical Host

Before you can execute any projects created with the JDBC eWay, you must add the JDBC drivers to the logical host.

1 If the logical host is not already installed, download and install [but do not bootstrap) the logical host as described in the *ICAN Suite Installation Guide*. Do not

2 Right-click the targeted logical host name for your JDBC eWay project and select: **Upload File...**

3 Select the path and file name for the JDBC/ODBC driver file. The Upload Third Party Files window appears.



4 Confirm that the files to be uploaded are correct and click **OK** to continue.

5.4.4 Deploying a Project

Once you have created the Environment and added its components, you can create the Deployment Profiles for the sample. The procedure below describes how to create Deployment Profiles for the inbound and outbound Collaborations.

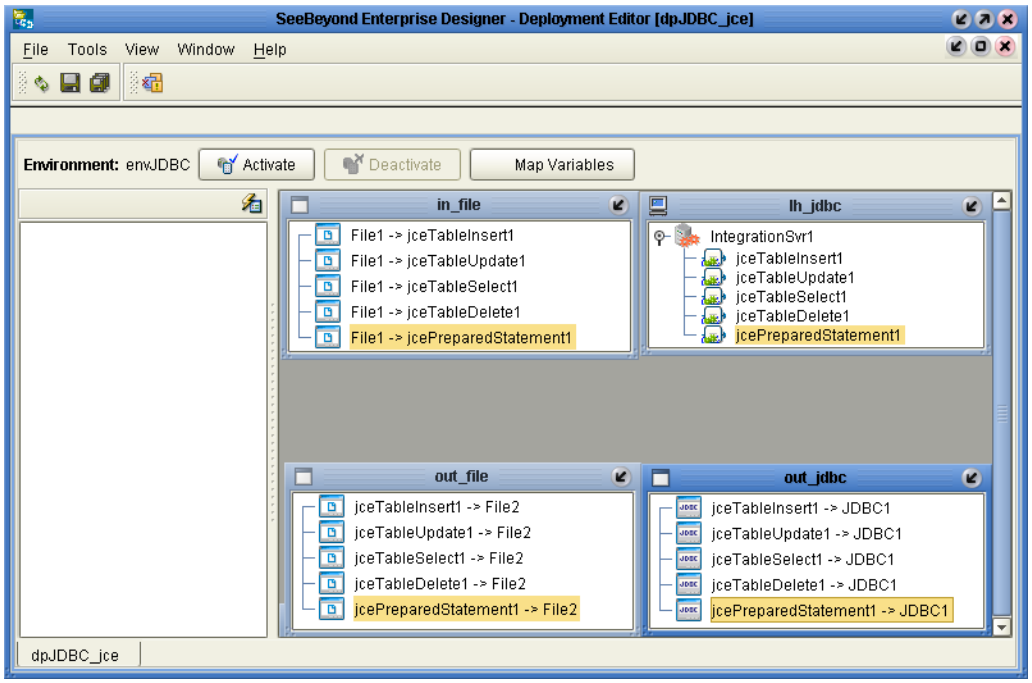
To create Deployment Profiles for sample Projects

1 In the Project Explorer tab of the Enterprise Designer, right-click the Project and click **New Deployment Profile**.

2 Enter a name for the inbound Deployment Profile, and select the Environment you created for the sample.

3 Double-click the inbound Deployment Profile. Drag the Project components to the Environment component as shown in Figure 34.

Figure 34 Deployment Profile



5.4.5 Running the Sample

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

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

5.5 Building Business Logic with eInsight

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

This section describes how to build the JDBC Collaborations:

- **Business Process Overview** on page 73
- **Building Business Processes** on page 74
- **Adding Connectivity Maps** on page 85
- **Building Outbound Connectivity Maps** on page 85

To see an example of JDBC Collaborations and Connectivity Maps, import the JDBC sample Projects as described in **"eWay Sample Projects" on page 63**.

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*.

5.5.1 Business Process Overview

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

The following operations are available:

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

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

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

Note: *For information on how to create a database OTD, refer to* **Using the JDBC/ ODBC eWay Database Wizard** *on page 42.*

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.

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

5.5.2 Building Business Processes

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

Sample Project Operations:

- **SelectOne** on page 74
- **Insert** on page 76
- **Update** on page 78
- **Delete** on page 79

Using the where() Clause

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

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

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

Using Triggers

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

SelectOne

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

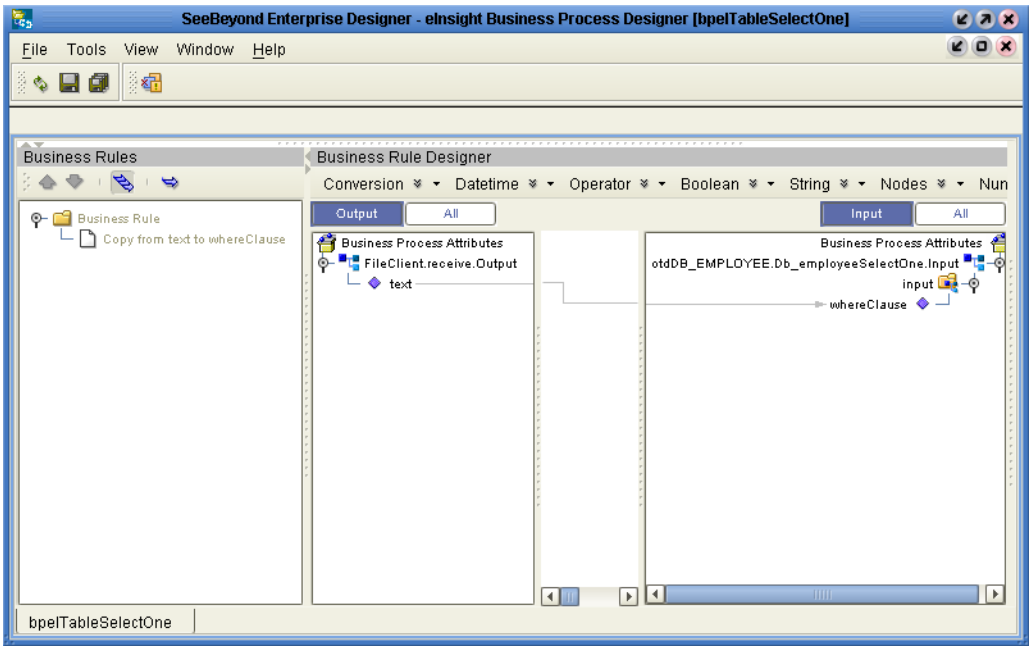
This business process describes the account retrieval process seen in **Figure 35**.

Figure 35 Employee Name Retrieval

Business Process:

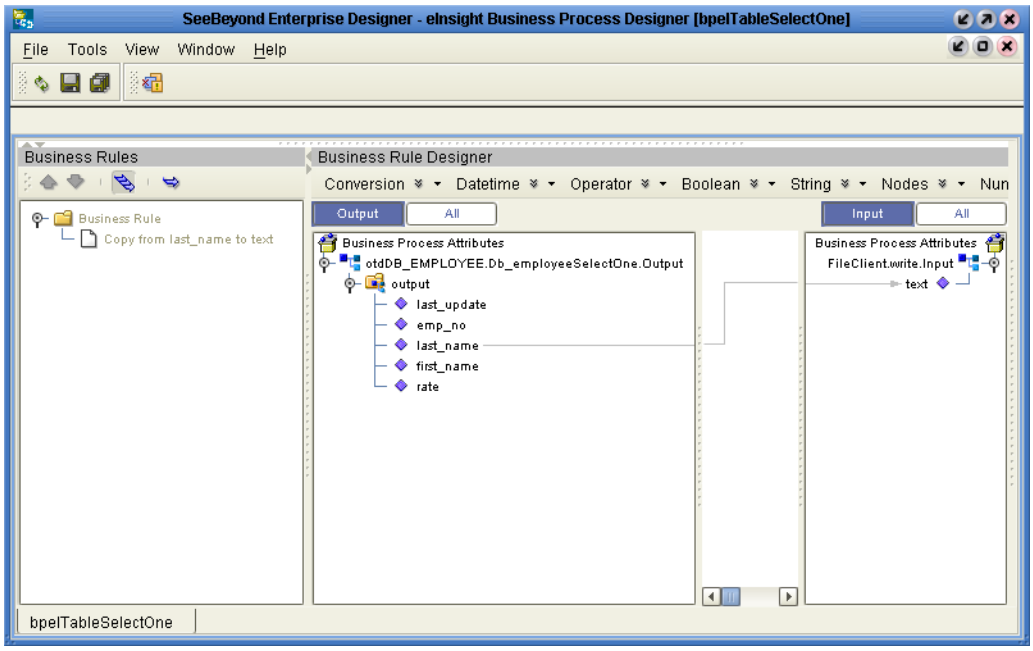
1. The File eWay subscribes to an external directory and activates a trigger (emp_no > 0) to query the database.
2. A where() clause loops through the database and queries the first record matching the criteria.

Figure 36 BP_GetEmployee Business Process - Find Name



3. The employee's last name (last_name) is written into an input file, before passing onto the Outbound File eWay.

Figure 37 BP_GetEmployee Business Process - Send Name



Insert

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

Figure 38 shows a sample eInsight Business Process using the Insert operation.

Figure 38 Insert Sample Business Process

Figure 39 shows the input data used for the Insert operation.

Figure 39 Insert Input

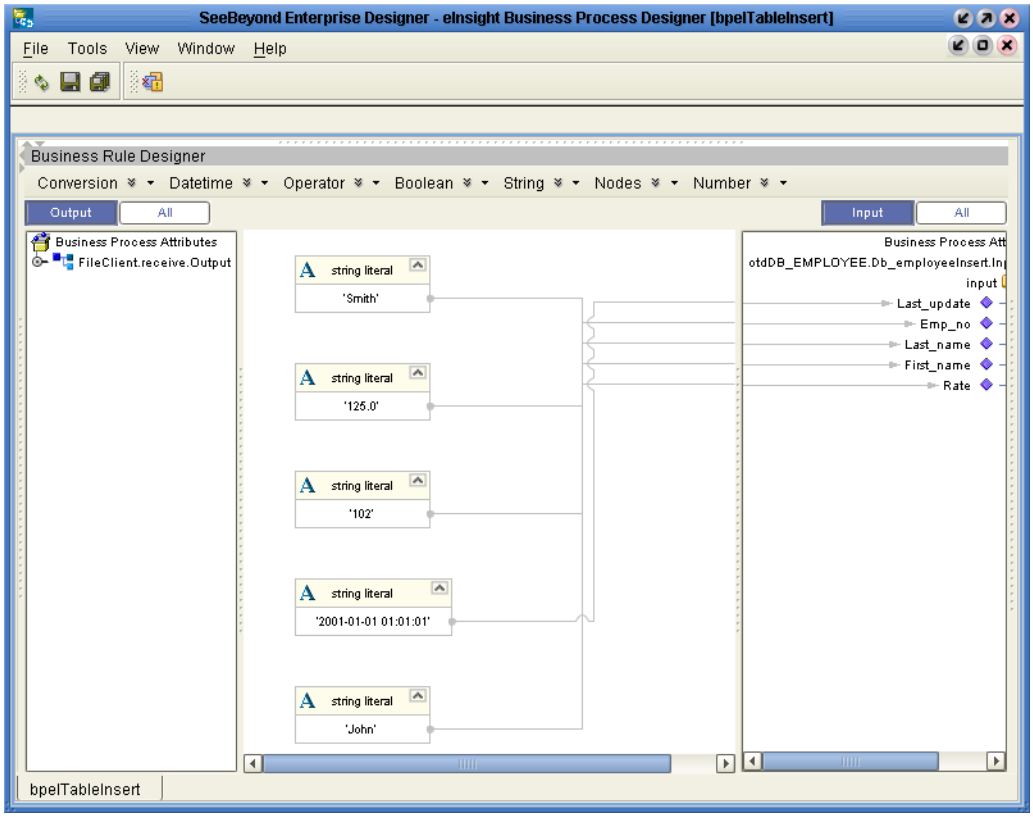
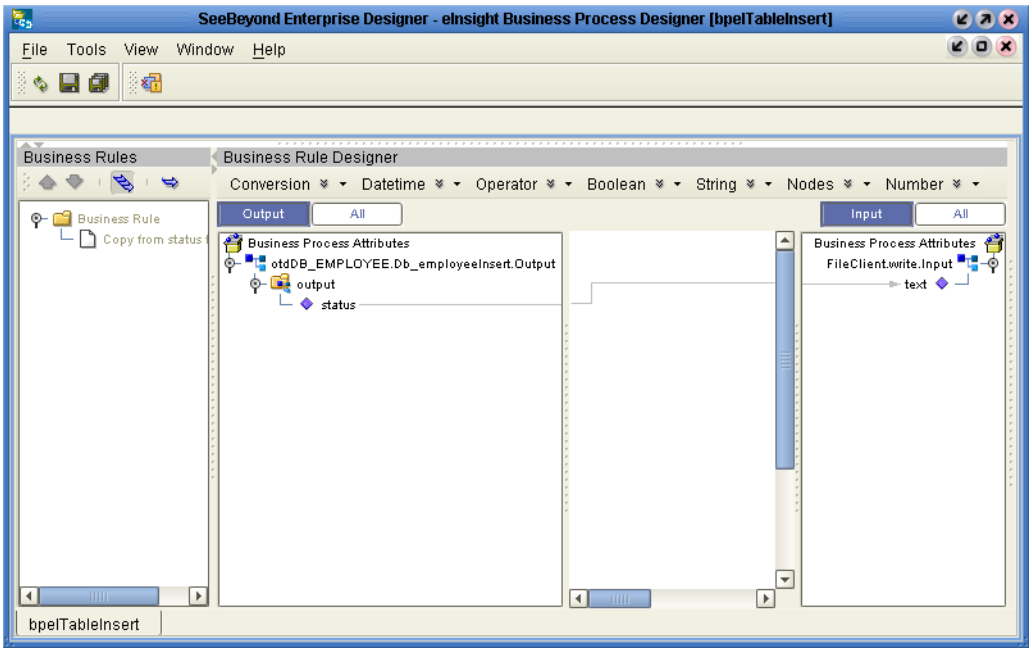


Figure 40 shows the output of the Insert operation, which is a status indicating the number of rows created.

Figure 40 Insert Output



Update

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

Figure 41 shows a sample eInsight Business Process using the Update operation. In this process, the operation updates a rate for a certain employee.

Figure 41 Update Sample Business Process

Figure 42 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 42 Update Input

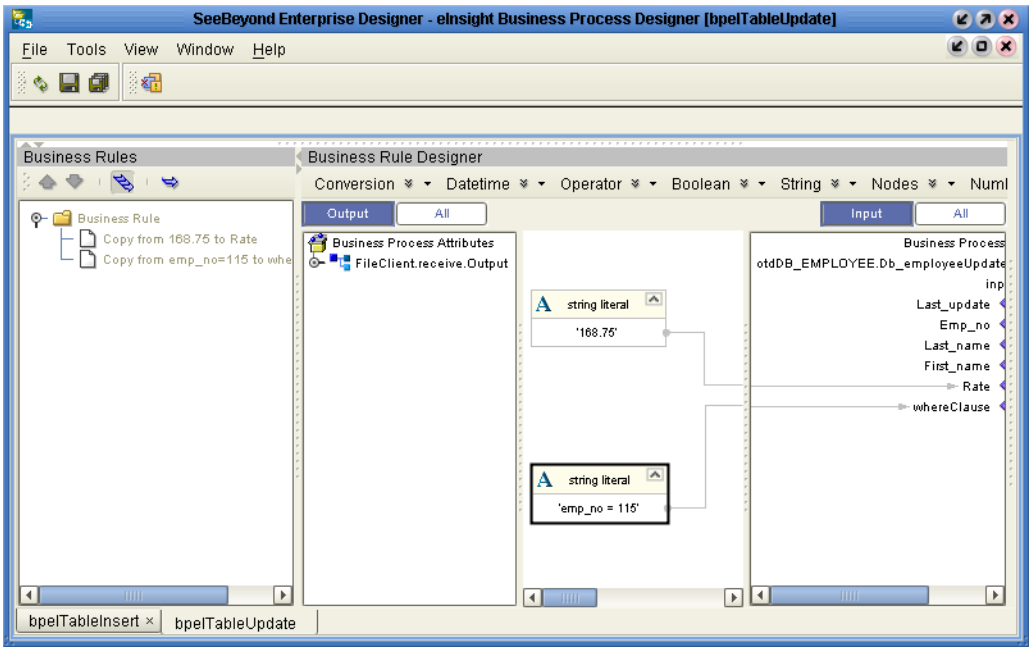
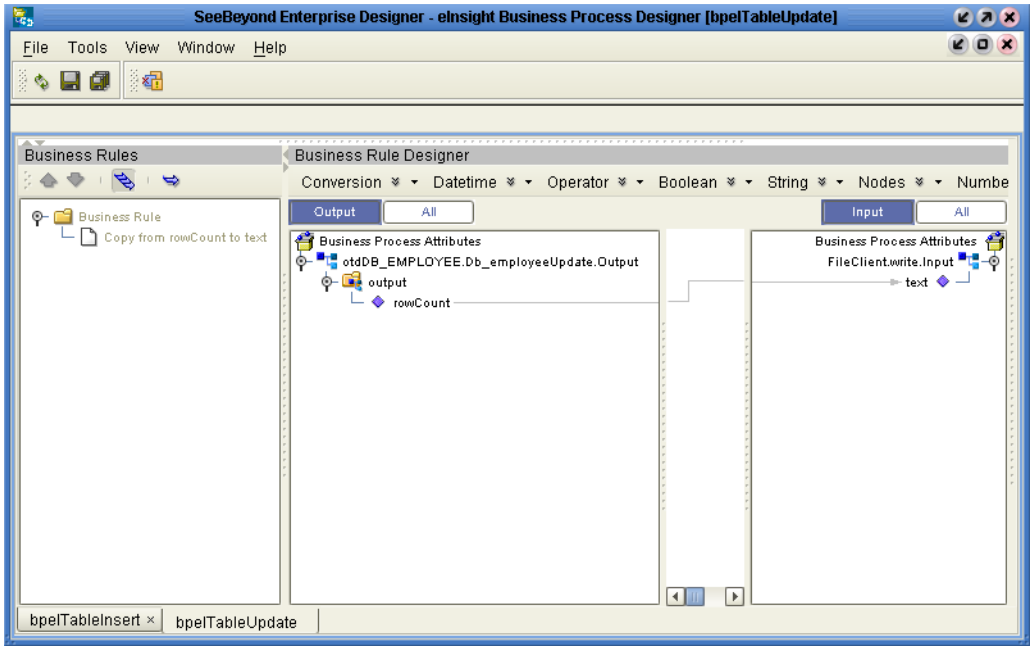


Figure 43 shows the output of the Update operation, which is a status indicating the number of rows updated.

Figure 43 Update Output



Delete

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

Figure 44 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 44 Delete Sample Business Process

Figure 45 shows the definition of the where() clause for the Delete operation. In this example, employee 115 is deleted.

Figure 45 Delete Input

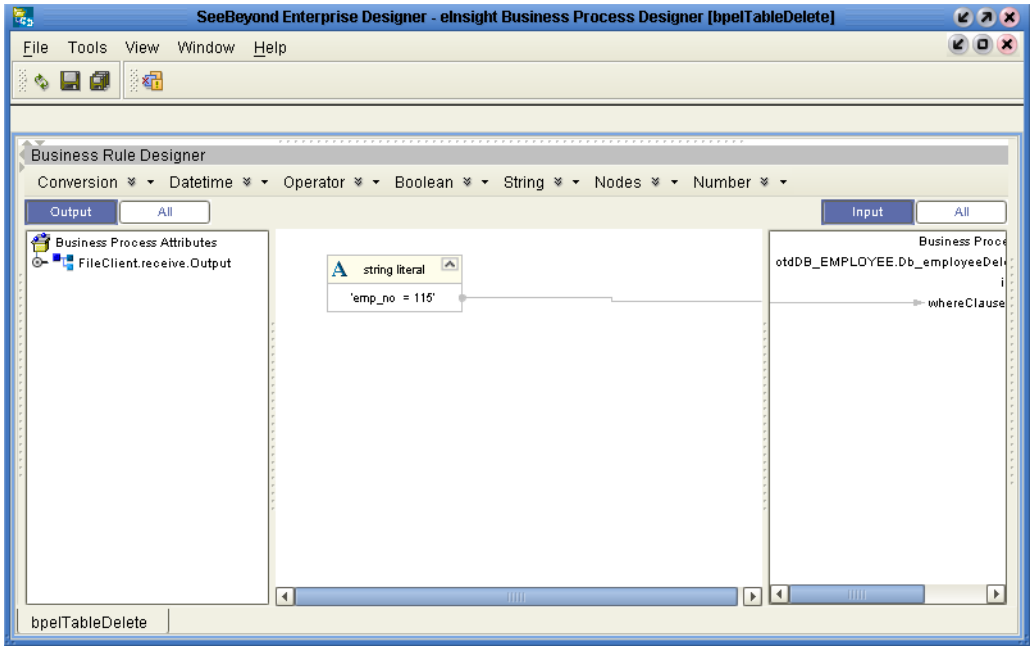
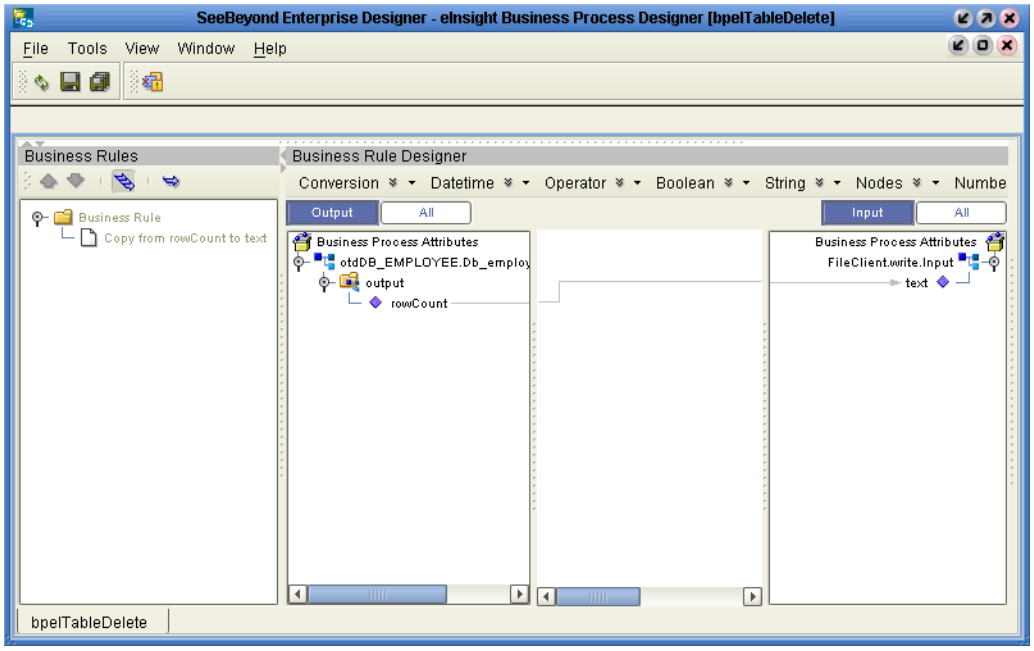


Figure 46 shows the output of the Delete operation, which is a status indicating the number of rows deleted.

Figure 46 Delete Output



5.6 Additional Business Logic with eInsight

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

BPEL Topics

- **SelectAll** on page 81
- **SelectMultiple** on page 82

5.6.1 SelectAll

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

Figure 47 shows a sample eInsight Business Process using the SelectAll operation. In this process, the SelectAll operation returns all rows where the emp_no is >=0.

Figure 47 SelectAll Sample Business Process

Figure 48 shows the definition of the where() clause for the SelectAll operation.

Figure 48 SelectAll Input

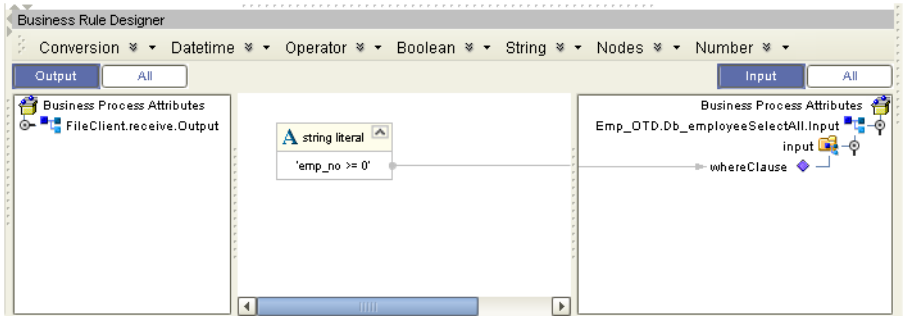
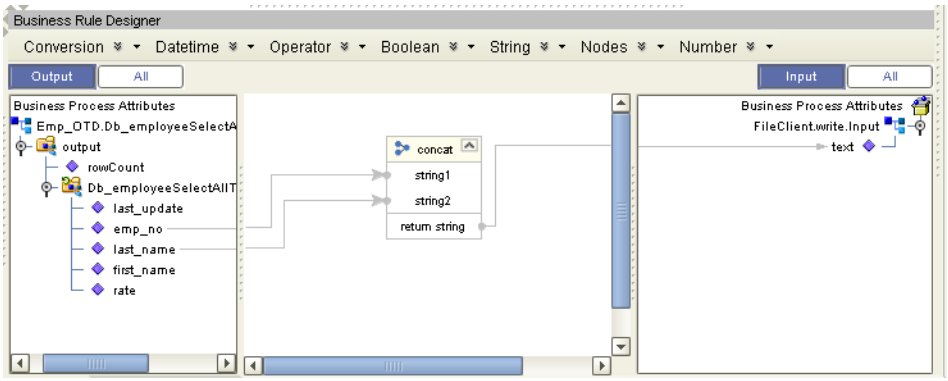


Figure 49 shows the output for the SelectAll operation. For each row selected during the operation, the employee number and last name are concatenated and passed out as text using the FileClient.write.

Figure 49 SelectAll Output



5.6.2 SelectMultiple

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

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

Figure 50 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 50 SelectMultiple Sample Business Process

Figure 51 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 51 SelectMultiple Input

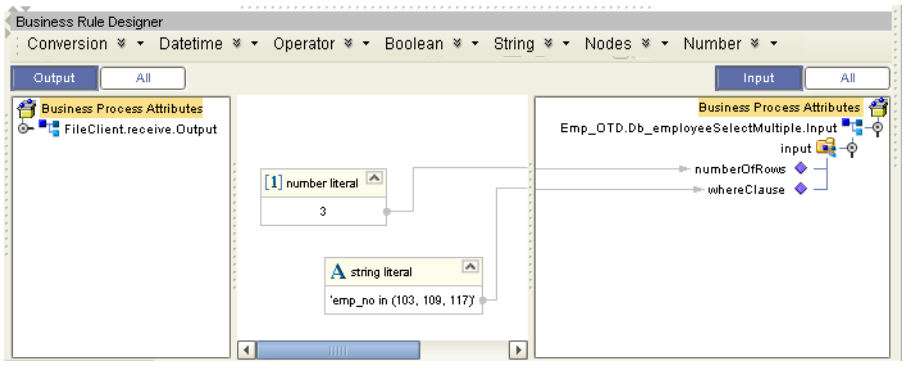
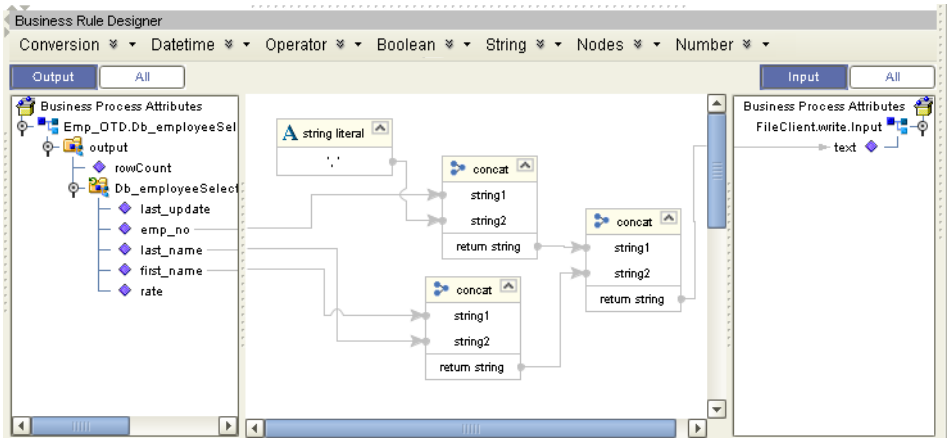


Figure 52 shows the output for the SelectMultiple operation. For each row selected during the operation, the employee number, first and last name are returned as text using the FileClient.write.

Figure 52 SelectMultiple Output



5.6.3 Adding Connectivity Maps

To add Connectivity Maps

- In the **Project Explorer** tab of the Enterprise Designer, right-click the Project for which you intend to create a Connectivity Map, click **New**, and then **Connectivity Map**.

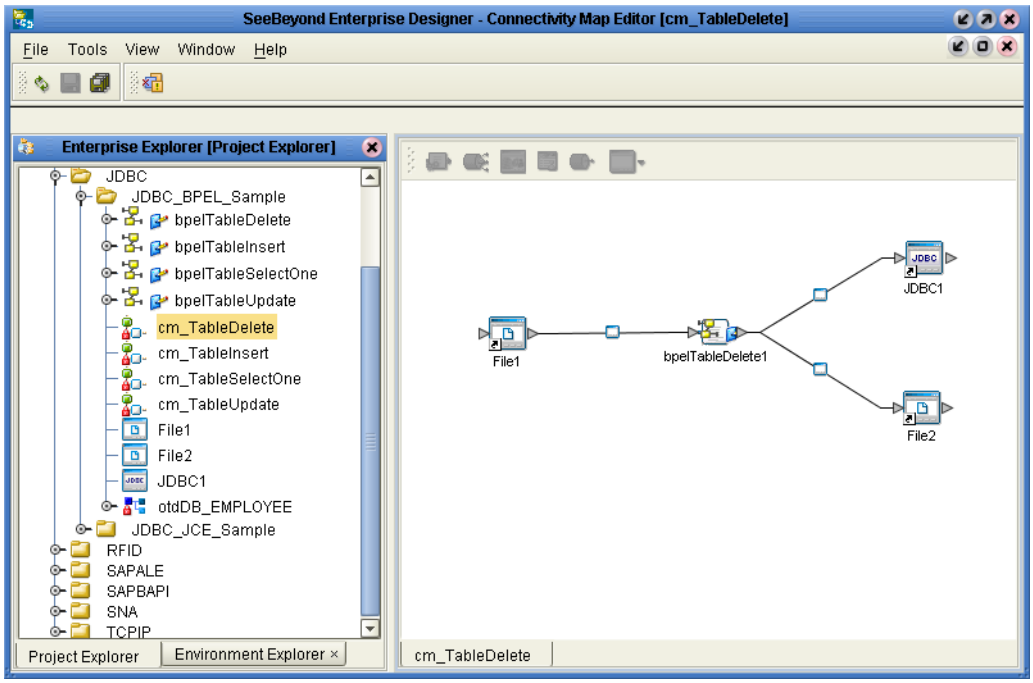
5.6.4 Building Outbound Connectivity Maps

To build outbound JDBC Connectivity Maps

1. From the Connectivity Map, click the External Applications icon then select **JDBC External Application**.
2. Add other components such as other eWays and Collaborations to the Connectivity Map.
3. Drag the outbound Collaboration from the **Project Explorer** tab into the Collaboration icon in the Connectivity Map.
4. Link and configure all components. For details, refer to the *eGate Integrator User's Guide*.

The figure below shows an example of an outbound JDBC Connectivity Map. To explore the Connectivity Map for an actual Project, import the JDBC sample Project as described in **"Importing the Sample Projects" on page 67**.

Figure 53 Outbound JDBC Connectivity Map



5.7 Building Business Logic with eGate

This section describes how to build the JDBC Collaborations:

- **Building Collaborations** on page 84
- **Adding Connectivity Maps** on page 85
- **Building Outbound Connectivity Maps** on page 85

To see an example of JDBC Collaborations and Connectivity Maps, import the JDBC sample Projects as described in **"eWay Sample Projects" on page 63**.

5.7.1 Building Collaborations

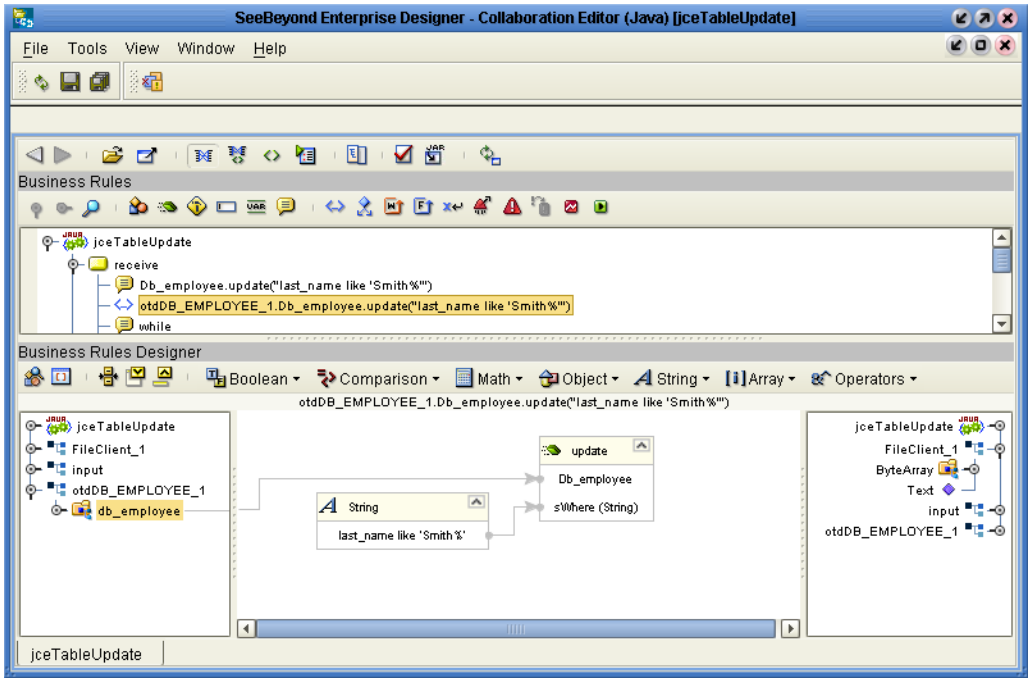
After you have built the OTDs as described in **"Using the JDBC/ODBC eWay Database Wizard" on page 42**, you are ready to build Collaboration Definitions.

To build Collaborations

1. In the **Project Explorer** tab of the Enterprise Designer, right-click the Project, click **New**, and then **Collaboration Definition (Java)**.
2. Complete the **Collaboration Definition** wizard. For details about this wizard, refer to the *eGate Integrator User's Guide*.

3 In the **Collaboration Editor** window, create the source code and the data mappings for the Collaboration. For details, refer to the *eGate Integrator User's Guide*. For information about JDBC methods, refer to the supplied javadocs.

Figure 54 Outbound Collaboration



5.7.2 Adding Connectivity Maps

To add Connectivity Maps

- In the **Project Explorer** tab of the Enterprise Designer, right-click the Project for which you intend to create a Connectivity Map, click **New**, and then **Connectivity Map**.

5.7.3 Building Outbound Connectivity Maps

To build outbound JDBC Connectivity Maps

1 From the Connectivity Map, click the External Applications icon then select **JDBC External Application**.

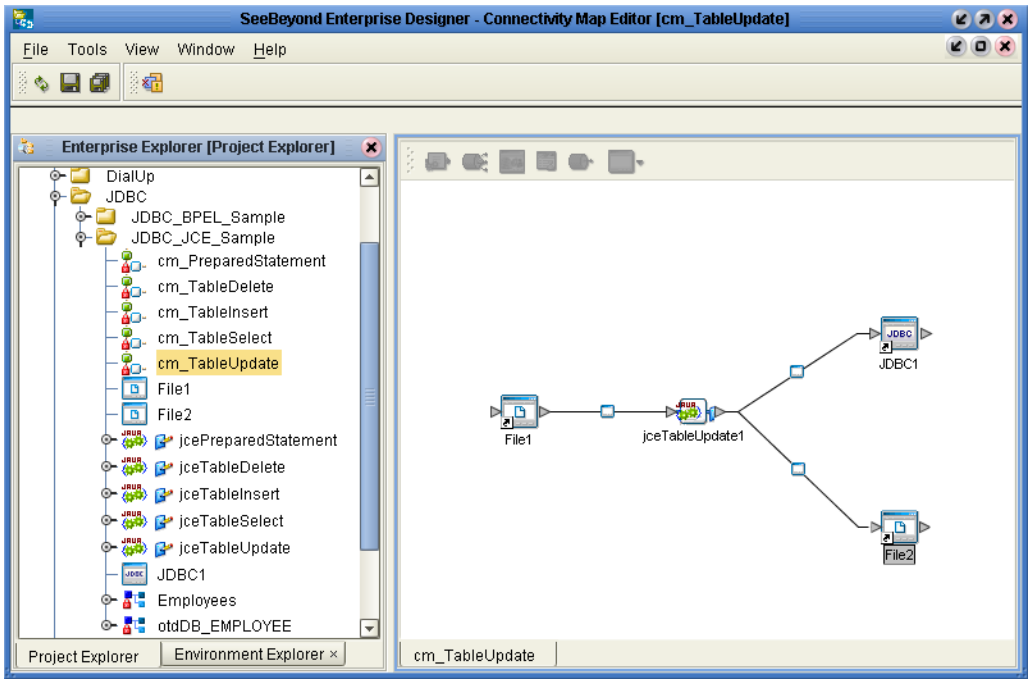
2 Add other components such as other eWays and Collaborations to the Connectivity Map.

3 Drag the outbound Collaboration from the **Project Explorer** tab into the Collaboration icon in the Connectivity Map.

4 Link and configure all components. For details, refer to the *eGate Integrator User's Guide*.

The figure below shows an example of an outbound JDBC Connectivity Map. To explore the Connectivity Map for an actual Project, import the JDBC sample Project as described in **"Importing the Sample Projects" on page 67**.

Figure 55 Outbound JDBC Connectivity Map



6.8 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***.

Appendix A

JDBC/ODBC Drivers

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

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

This appendix provides database configuration information and environment properties specifications for specific JDBC/ODBC drivers. You should use the information listed in the included tables to define values for required input parameters.

While any standards compliant JDBC/ODBC database driver may be used, the drivers covered in this chapter are used more frequently. For runtime, only drivers that support Connection Pool Data Source and XA Data Source are supported; Connection Pool Data Source takes advantage of the Integration Service's connection pooling in order to improve performance. For the OTD Wizard, the regular driver class will work; however, not all drivers support all metadata discovery methods---some of which are needed to build the OTD. Check with your driver vendor for what is supported.

What's In This Chapter:

- **AS/400** on page 88
- **Attunity Driver** on page 89
- **MySQL Connector/J Driver** on page 91
- **SyBase JConnect V5** on page 92
- **Sequelink DataDirect Informix ODBC Driver** on page 93
- **MS Access via MS Access ODBC Driver** on page 94
- **Data Types for Drivers** on page 96
- **Installing JDBC/ODBC Drivers** on page 97
- **Troubleshooting** on page 97

Refer to the supplied *readme.txt* file for information regarding the location of the required driver files necessary to connect using the JDBC/ODBC eWay.

6.9 AS/400

The AS/400 driver does not support Updatable Resultsets; however, it does allow standard Insert and Update operations when used with the prepared statement feature:

```
Insert into employee (empno)(?);
```

Important: *Remember to ensure that the input parameter data types match the data types specified in the database table targeted by the prepared statements. Optionally, you may perform the data conversion in the collaboration.*

Configuration Properties:

- **OTD Wizard: Database Connection Information** on page 88
- **Environment Properties** on page 88

6.9.1 OTD Wizard: Database Connection Information

To connect to AS/400, use the information provided this table to complete the Connect to Database step of the JDBC/ODBC OTD Wizard. To access DB2, SeeBeyond recommends the use of the DB2 eWay Intelligent Adapter or the DB2 Connect eWay Intelligent Adapter.

Table 5 AS/400 Database Connection Information.

Parameter	Value
Driver Jar Files	jt400.jar
Driver Java Class Name	com.ibm.as400.access.AS400JDBCDriver
URL Connection String	jdbc:as400://<server-name>:<server-port>/ ***Note:*** *NOTE: Default server port is 446.*
User Name	Login name of the account used to access the AS/400 database.
Password	Password associated with the login account name used to connect to the AS/400 database.

6.9.2 Environment Properties

Use this table to configure the environment properties for the specified JDBC/ODBC driver.

Table 6 AS/400 Database Connection Information.

Parameter	Value
ClassName	com.ibm.as400.access.AS400JDBCConnectionPoolDataDource
DatabaseName	
DataSourceName	
Delimiter	#
Description	JDBC Connection Pool Datasource
DriverProperties	
Password	Password associated with the login account name used to connect to the database.
PortNumber	<server-port> ***Note:*** *NOTE: Default server port is 446.*
ServerName	Server name of the machine hosting the database the database.
User	Login name of the account used to access the database.

6.10 Attunity Driver

When using the Attunity driver with the JDBC eWay, you must use the

- **OTD Wizard: Database Connection Information** on page 89
- **Environment Properties** on page 90

6.10.1 OTD Wizard: Database Connection Information

To connect to Attunity, use the information provided in this table to complete the Connect to Database step of the JDBC/ODBC OTD Wizard.

Table 7 Attunity Database Connection Information.

Parameter	Value
Driver Jar Files	nvjdbc2.jar
Driver Java Class Name	com.attunity.jdbc.NvDriver

Table 7 Attunity Database Connection Information.

Parameter	Value
URL Connection String	jdbc:attconnect://<server-name>;DefTdpName=<database-logical-name>;OneTdpMode=1 ***Note:*** *The* <database-logical-name> is created in the Attunity server.
User Name	Leave password field blank. Value configured when the database entry is created in the Attunity Server.
Password	Leave password field blank. Value configured when the database entry is created in the Attunity Server.

6.10.2 Environment Properties

Use this table to configure the environment properties for the specified JDBC/ODBC driver.

Table 8 Attunity Database Connection Information.

Parameter	Value
ClassName	com.attunity.jdbc.NvXADataSource
DatabaseName	
DataSourceName	
Delimiter	#
Description	JDBC Connection Pool Datasource
DriverProperties	setDefTdpName#<database-logical-name>##setWorkspace#Navigator##
Password	Leave password field blank. Value configured when the database entry is created in the Attunity Server.
PortNumber	<server-port> ***Note:*** *NOTE: Default server port is 2551.*
ServerName	Server name of the machine hosting the database the database.
User	Leave password field blank. Value configured when the database entry is created in the Attunity Server.

6.11 MySQL Connector/J Driver

- **OTD Wizard: Database Connection Information** on page 91
- **Environment Properties** on page 91

6.11.1 OTD Wizard: Database Connection Information

To connect to MYSQL, use the information provided in this table to complete the Connect to Database step of the JDBC/ODBC OTD Wizard.

Table 9 MySQL Connector/J Driver Database Connection Information.

Parameter	Value
Driver Jar Files	mysql-connector-java-3.0.11-stable-bin.jar
Driver Java Class Name	com.mysql.jdbc.Driver
URL Connection String	jdbc:mysql://<server-name>:<server-port>/<database-name> ***Note:*** *NOTE: Default server port is 3306*
User Name	Login name of the account used to access the database.
Password	Password associated with the login account name used to connect to the database.

6.11.2 Environment Properties

Use this table to configure the environment properties for the specified JDBC/ODBC driver.

Table 10 MYSQL Environment Properties.

Parameter	Value
ClassName	ccom.mysql.jdbc.jdbc2.optional.MysqlConnectionPoolDataSource
DatabaseName	<database-name>
DataSourceName	
Delimiter	#
Description	JDBC Connection Pool Datasource
DriverProperties	jdbc:mysql://<server-name>:<server-port>/<database-name>
Password	Password associated with the login account name used to connect to the database.

Table 10 MYSQL Environment Properties.

Parameter	Value
PortNumber	<server-port> ***Note:*** *NOTE: Default server port is 3306.*
ServerName	Server name of the machine hosting the database the database.
User	Login name of the account used to access the database.

6.12 SyBase JConnect V5

OTD Wizard: Database Connection Information on page 92

Environment Properties on page 92

6.12.1 OTD Wizard: Database Connection Information

To connect to Sybase, use the information provided in this table to complete the Connect to Database step of the JDBC/ODBC OTD Wizard. To access Sybase, SeeBeyond recommends the use of the Sybase eWay Intelligent Adapter.

Table 11 Sybase Database Connection Information.

Parameter	Value
Driver Jar Files	jconn2.jar
Driver Java Class Name	com.sybase.jdbc2.jdbc.SybDriver
URL Connection String	jdbc:sybase:Tds:<server-name>:<server-port> ***Note:*** *NOTE: Default server port is 4100.*
User Name	Login name of the account used to access the database.
Password	Password associated with the login account name used to connect to the database.

6.12.2 Environment Properties

Use this table to configure the environment properties for the specified JDBC/ODBC driver.

Table 12 Sybase Environment Properties.

Parameter	Value
ClassName	com.sybase.jdbc2.jdbc.SybConnectionPoolDataSource
DatabaseName	<database-name>
DataSourceName	
Delimiter	#
Description	JDBC Connection Pool Datasource
DriverProperties	setURL#jdbc:sybase:Tds:<server-name>:<server-port>
Password	Password associated with the login account name used to connect to the database.
PortNumber	<server-port> ***Note:*** *NOTE: Default server port is 4100.*
ServerName	Server name of the machine hosting the database the database.
User	Login name of the account used to access the database.

6.13 Sequelink DataDirect Informix ODBC Driver

OTD Wizard: Database Connection Information on page 93

Environment Properties on page 94

6.13.1 OTD Wizard: Database Connection Information

These settings describe how to use the DataDirect Sequelink JDBC/ODBC bridge with the JDBC/ODBC eWay. This information demonstrates how Sequelink can be used to interface with the ODBC driver. To connect to an Informix database, SeeBeyond recommends the use of the Informix eWay Intelligent Adapter.

Table 13 Informix Database Connection Information.

Parameter	Value
Driver Jar Files	sljc.jar
Driver Java Class Name	com.ddtek.jdbc.sequelkink.SequeLinkDriver
URL Connection String	jdbc:sequelink://<server-name>:<server-port> ***Note:*** *NOTE: Default server port is 19996.*

Table 13 Informix Database Connection Information.

Parameter	Value
User Name	Login name of the account used to access the database.
Password	Password associated with the login account name used to connect to the database.

6.13.2 Environment Properties

Use this table to configure the environment properties for the specified JDBC/ODBC driver.

Table 14 Informix Environment Properties.

Parameter	Value
ClassName	com.ddtek.jdbcx.sequelkink.SequeLinkDataSource
DatabaseName	
DataSourceName	
Delimiter	#
Description	JDBC Connection Pool Datasource
DriverProperties	
Password	Password associated with the login account name used to connect to the database.
PortNumber	<server-port> ***Note:*** *NOTE: Default server port is 19996.*
ServerName	Server name of the machine hosting the database the database.
User	Login name of the account used to access the database.

6.14 MS Access via MS Access ODBC Driver

- **OTD Wizard: Database Connection Information** on page 95
- **Environment Properties** on page 95

6.14.1 OTD Wizard: Database Connection Information

To connect to Microsoft Access, via the Microsoft Access ODBC driver, use the information provided in this table to complete the Connect to Database step of the JDBC/ODBC OTD Wizard.

Table 15 MS Access Database Connection Information.

Parameter	Value
Driver Jar Files	sljc.jar
Driver Java Class Name	com.ddtek.jdbc.sequelink.SequeLinkDriver
URL Connection String	jdbc:sequelink://<server-name>:<server-port> ***Note:*** *NOTE: Default server port is 19996.*
User Name	Login name of the account used to access the database.
Password	Password associated with the login account name used to connect to the database.

6.14.2 Environment Properties

Use this table to configure the environment properties for the specified JDBC/ODBC driver.

Table 16 MS Access Environment Properties.

Parameter	Value
ClassName	com.ddtek.jdbcx.sequelkink.SequeLinkDataSource
DatabaseName	
DataSourceName	
Delimiter	#
Description	JDBC Connection Pool Datasource
DriverProperties	
Password	Password associated with the login account name used to connect to the database.
PortNumber	<server-port> ***Note:*** *NOTE: Default server port is 19996.*
ServerName	Server name of the machine hosting the database the database.
User	Login name of the account used to access the database.

6.15 Data Types for Drivers

Table 17 lists common data types---the list is not all inclusive---that may be available for a particular JDBC/ODBC driver. Check your driver documentation for more information:

Table 17 Sample of Standard Data Types Supported by the eWay

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

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

6.16 Installing JDBC/ODBC Drivers

The database drivers specified in your projects need to be installed on both the Enterprise Designer machine and the Logical host machine. When installing the drivers on the Enterprise Designer machine, you must specify the absolute path to the driver. When installing the drivers on the Logical Host, it is not required to specify the absolute path to the driver if you place the driver into the Logical Host `stcis` directory: *ican50\logicalhost\stcis\lib*. If you do not place your drivers into this directory on your logical host, you must specify the absolute path to the driver.

For procedures on how to install database drivers, see **"Configuring the Logical Host" on page 70**.

6.17 Troubleshooting

Refer to the following when troubleshooting Driver issues.

- The ReceiveOne operation in BPEL is not supported when using inbound functions with some drivers.
- Some drivers do not support Updatable ResultSets. If you find this to be the case, use a Prepared Statement to Update, Insert, and Delete data.
- Not all drivers provide metadata information such as column names and data types. If your table does not have column names and data types, add them before saving the OTD.

Index

A

AS/400 **88**
  driver **88**
Attunity **89**

B

Batch Operations **62**
bpelPreparedStatement **65**
bpelTableDelete **65**
bpelTableInsert **65**
bpelTableSelect **65**
bpelTableUpdate **65**
building
  Collaborations **84**

C

Char **96**
Clob **96**
Collaborations, building **84**
Connectivity Map
  Inbound JDBC eWay Properties **21**
  Outbound JDBC eWay Properties **22**
  Outbound JDBC Non-Transactional eWay Properties **28**
  Outbound JDBC XA eWay Properties **25**
Connectivity Maps
  adding, eGate **83**, **85**
  outbound, eGate **83**, **85**
conventions, document **15**

D

Data Types
  drivers **96**
DataDirect **66**, **93**
DataSourceName **33**, **37**, **40**
Date **97**
Delete **58**
delete.txt **64**
Delimiter **34**, **37**, **40**
Deployment Profiles, creating **71**
Description **34**, **37**, **40**
document conventions **15**
driver class, JDBC **22**, **25**, **28**, **31**, **33**, **36**, **39**
DriverProperties **34**, **37**, **40**
Drivers **87**
drivers
  AS/400 **88**

E

Environment Properties **30**
  DataSourceName **33**, **37**, **40**
  Delimiter **34**, **37**, **40**
  Description **34**, **37**, **40**
  DriverProperties **34**, **37**, **40**
  Inbound eWay Environment Properties **31**
  Outbound JDBC eWay Environment Properties **32**
  Outbound JDBC Non-Transactional eWay Environment **39**
  Outbound JDBC XA eWay Environment Properties **36**
  Password **34**, **37**, **40**
  PortNumber **35**, **38**, **41**
  ServerName **35**, **38**, **41**
  User **35**, **38**, **41**
Environments
  creating **69**
eWay Properties
  Connectivity Map **21**
  Inbound eWay Environment Properties **31**
  Inbound JDBC eWay Properties **21**
  Outbound JDBC eWay Environment Properties **32**
  Outbound JDBC eWay Properties **22**
  Outbound JDBC Non-Transactional eWay Environment **39**
  Outbound JDBC Non-Transactional eWay Properties **28**
  Outbound JDBC XA eWay Environment Properties **36**
  Outbound JDBC XA eWay Properties **25**

F

finding sample Projects **67**

I

importing sample Projects **67**
Inbound Environment Properties
  Password **31**
Inbound eWay Environment Properties **31**
Inbound JDBC eWay Properties **21**

Informix 93
InitialPoolSize 23, 25, 28
Insert 57
insert.txt 64

J

JAR file
  upload 70
jcePreparedStatement 66
jceTableDelete 65
jceTableInsert 65
jceTableSelect 65
jceTableUpdate 65
JConnect 92
JDBC
  driver class 22, 25, 28, 31, 33, 36, 39
JDBC_BPEL_Sample 65
JDBC_JCE_Sample 65
JDBC_JCE_Sample.zip 64

L

Logical Host
  configure 70
LoginTimeOut 23, 25, 29
Long 96

M

MaxIdleTime 23, 26, 29
MaxPoolSize 23, 26, 29
MaxStatements 23, 26, 29
MinPoolSize 24, 26, 29
MS Access 94
MySQL 91

N

NetworkProtocol 24, 27, 30
Number 96

O

organization of information, document 14
outbound
  Connectivity Maps (eGate) 83, 85
Outbound JDBC eWay Environment Properties 32
Outbound JDBC eWay Properties 22
Outbound JDBC Non-Transactional eWay Environment 39
Outbound JDBC Non-Transactional eWay Properties 28
Outbound JDBC XA eWay Environment Properties 36
Outbound JDBC XA eWay Properties 25
Outbound Properties
  InitialPoolSize 23, 25, 28
  LoginTimeOut 23, 25, 29
  MaxIdleTime 23, 26, 29
  MaxPoolSize 23, 26, 29
  MaxStatements 23, 26, 29
  MinPoolSize 24, 26, 29
  NetworkProtocol 24, 27, 30
  PropertyCycle 24, 27, 30
  RoleName 24, 27, 30

P

Password 31, 34, 37, 40
PortNumber 35, 38, 41
Prepared Statement 60
Prepared Statements 50
preparedstatement.txt 64
Properties
  Environment 30
Property settings, Environment
  DataSourceName 33, 37, 40
  Delimiter 34, 37, 40
  Description 34, 37, 40
  DriverProperties 34, 37, 40
  Password 34, 37, 40
  PortNumber 35, 38, 41
  ServerName 35, 38, 41
  User 35, 38, 41
Property settings, Inbound Environment
  Password 31
Property settings, Outbound
  InitialPoolSize 23, 25, 28
  LoginTimeOut 23, 25, 29
  MaxIdleTime 23, 26, 29
  MaxPoolSize 23, 26, 29
  MaxStatements 23, 26, 29
  MinPoolSize 24, 26, 29
  NetworkProtocol 24, 27, 30
  PropertyCycle 24, 27, 30
  RoleName 24, 27, 30
PropertyCycle 24, 27, 30

Q

Query 56

R

Raw 96

RoleName **24**, **27**, **30**

S

sample Projects
    Deployment Profiles **71**
    Environments **69**
    finding **67**
    importing **67**
Screenshots **15**
select.txt **64**
SequeLink **66**
Sequelink **93**
ServerName **35**, **38**, **41**
Stored Procedure **59**
Supported Operating Systems **17**, **54**
supporting documents **16**
SyBase **92**
System Requirements **18**

T

trigger.txt **64**
Troubleshooting **97**

U

Update **58**
update.txt **64**
upload
    JAR file **70**
User **35**, **38**, **41**

V

VarChar2 **96**