

SeeBeyond ICAN Suite

DB2 Universal Database eWay Intelligent Adapter User's Guide

Release 5.0.7



The information contained in this document is subject to change and is updated periodically to reflect changes to the applicable software. Although every effort has been made to ensure the accuracy of this document, SeeBeyond Technology Corporation (SeeBeyond) assumes no responsibility for any errors that may appear herein. The software described in this document is furnished under a License Agreement and may be used or copied only in accordance with the terms of such License Agreement. Printing, copying, or reproducing this document in any fashion is prohibited except in accordance with the License Agreement. The contents of this document are designated as being confidential and proprietary; are considered to be trade secrets of SeeBeyond; and may be used only in accordance with the License Agreement, as protected and enforceable by law. SeeBeyond assumes no responsibility for the use or reliability of its software on platforms that are not supported by SeeBeyond.

SeeBeyond, e*Gate, e*Way, and e*Xchange are the registered trademarks of SeeBeyond Technology Corporation in the United States and/or select foreign countries. The SeeBeyond logo, SeeBeyond Integrated Composite Application Network Suite, eGate, eWay, eInsight, eVision, eXchange, eView, eIndex, eTL, ePortal, eBAM, and e*Insight are trademarks of SeeBeyond Technology Corporation. The absence of a trademark from this list does not constitute a waiver of SeeBeyond Technology Corporation's intellectual property rights concerning that trademark. This document may contain references to other company, brand, and product names. These company, brand, and product names are used herein for identification purposes only and may be the trademarks of their respective owners.

© 2005 SeeBeyond Technology Corporation. All Rights Reserved.

This work is confidential and proprietary information of SeeBeyond and must be maintained in strict confidence.

Version 20050526153338.

Contents

Chapter 1

Introducing the DB2 Universal Database eWay	7
About DB2 Universal Database	7
About the DB2 Universal Database eWay	8
What's New in This Version	8
About This Document	8
What's in This Document	8
Scope	9
Intended Audience	9
Document Conventions	9
Screenshots	9
Related Documents	10
SeeBeyond Web Site	10
Feedback	10

Chapter 2

Installing the eWay	11
Supported Operating Systems	11
WebLogic and WebSphere Application Server Support	11
System Requirements	12
Supported External Applications	12
Installing the eWay Product Files	12
After You Install	13

Chapter 3

Configuring the DB2 eWay	14
Properties of DB2 eWay on Windows or Unix Operating Systems	14
Setting the Properties of the Inbound DB2 eWay on Windows or Unix Operating Systems	15
PollMilliseconds	15
PreparedStatement	15

Setting the Environment Properties of the Inbound DB2 eWay on Windows or Unix Operating Systems	16
DatabaseName	16
Password	16
PortNumber	16
ServerName	17
User	17
Setting the Properties of the Outbound DB2 eWay on Windows or Unix Operating Systems	17
ClassName	18
Description	18
InitialPoolSize	18
LoginTimeout	18
MaxIdleTime	18
MaxPoolSize	19
MaxStatements	19
MinPoolSize	19
NetworkProtocol	19
PropertyCycle	19
RoleName	20
Setting the Environment Properties of the Outbound DB2 eWay on Windows or Unix Operating Systems	20
DatabaseName	20
Delimiter	21
Description	21
DriverProperties	21
Password	21
PortNumber	22
ServerName	22
User	22
Properties of the DB2 eWay Connecting to z/OS or an AS/400 Operating System	22
Setting the Properties of the Inbound DB2 eWay Connecting to z/OS or and AS/400 Operating System	23
PollMilliseconds	23
PreparedStatement	23
Setting the Environment Properties of the Inbound DB2 eWay Connecting to z/OS or an AS/400 Operating Systems	24
CollectionId	24
LocationName	24
Password	24
PortNumber	25
ServerName	25
User	25
Setting the Properties of the Outbound DB2 eWay Connecting to z/OS or an AS/400 Operating System	26
ClassName	26
Description	26
InitialPoolSize	27
LoginTimeout	27
MaxIdleTime	27
MaxPoolSize	27
MaxStatements	27
MinPoolSize	28
NetworkProtocol	28

PropertyCycle	28
RoleName	28
Setting the Environment Properties of the Outbound DB2 eWay Connecting to z/OS or an AS/400	
Operating Systems	28
CollectionID	29
Delimiter	29
Description	29
DriverProperties	30
LocationName	30
Any valid string.	30
Password	30
PortNumber	30
ServerName	31
User	31

Chapter 4

DB2 Wizard Operation **32**

Using the Database OTD Wizard	32
Select Wizard Type	32
Connect to Database	33
Select Database Objects	34
Select Table/Views	35
Select Procedures	39
Add Prepared Statements	41
Specify the OTD Name	43

Chapter 5

Implementing the DB2 eWay **45**

eInsight Engine and eGate Components	45
Using the Sample Project in eInsight	45
The Business Process	46
SelectAll	48
SelectMultiple	49
SelectOne	51
Insert	52
Update	53
Delete	55
Using the Sample Project in eGate	56
Working with the Sample Project in eGate	56
Configuring the eWays	57
Creating an External Environment	58
Deploying a Project	58
Running the Sample	58
Common DataType Conversions	58
Using OTDs with Tables, Views, Stored Procedures, and Prepared Statements	60

The Table	60
The Query Operation	61
The Insert Operation	62
The Update Operation	63
The Delete Operation	63
Using Clobs	64
Inserting a Clob using a Table OTD	64
Inserting a Clob using a Prepared Statement OTD	65
Inserting a Clob using a Stored Procedure OTD	65
Updating a Clob using a Table OTD	66
Updating a CLOB using a Stored Procedure or Prepared Statement OTD	67
Selecting a Clob using a Table OTD	67
Selecting a Clob using a Prepared Statement	71
The Stored Procedure	72
Executing Stored Procedures	73
Manipulating the ResultSet and Update Count Returned by Stored Procedure	74
Editing Existing OTDs	77
Alerting and Logging	78

Appendix A

Support for WebSphere Application Server	79
Uploading the Application Server Interface	79
Creating an EAR File	80
Deploying an EAR File	81
Configuring the WebSphere Application Server	81
Creating the Topic or Queue	81
Creating a Connection Factory	83
Installing the Application	84

Introducing the DB2 Universal Database eWay

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

What's in This Chapter

- [“About DB2 Universal Database” on page 7](#)
- [“About the DB2 Universal Database eWay” on page 8](#)
- [“About This Document” on page 8](#)
- [“Related Documents” on page 10](#)
- [“SeeBeyond Web Site” on page 10](#)
- [“Feedback” on page 10](#)

1.1 About DB2 Universal Database

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

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

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

1.2 About the DB2 Universal Database eWay

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

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

1.3 What's New in This Version

The DB2 Universal Database (UDB) eWay includes:

- The DB2 eWay includes an editable OTD feature that allows a user to edit the OTD instead of rebuilding it from scratch. For more information on editing existing OTDs refer to [Editing Existing OTDs](#) on page 77.
- Added support for DB2 version V5R2 and V5R3 when connecting to DB2 running on an AS/400 operating system.

1.4 About This Document

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

1.4.1 What's in This Document

This document includes the following chapters:

- **Chapter 1 “Introducing the DB2 Universal Database eWay”:** Provides an overview description of the product as well as high-level information about this document.
- **Chapter 2 “Installing the eWay”:** Describes the system requirements and provides instructions for installing the DB2 Universal Database eWay.
- **Chapter 3 “Configuring the DB2 eWay”:** Provides instructions for configuring the eWay to communicate with your legacy systems.
- **Chapter 4 “DB2 Wizard Operation”:** Provides information about .sag files and using the DB2 Universal Database wizard.
- **Chapter 5 “Implementing the DB2 eWay”:** Provides instructions for installing and running the sample Projects.
- **Appendix A “Support for WebSphere Application Server”** Provides information on deploying an Enterprise Archive (EAR) file to the WebSphere™ Application

Server, including information on installing the WebSphere Application Server interface and configuring the selected Application server to deploy the EAR file.

1.4.2 Scope

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

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

1.4.3 Intended Audience

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

1.4.4 Document Conventions

The following writing conventions are observed throughout this document.

Table 1 Writing Conventions

Text	Convention	Example
Button, file, icon, parameter, variable, method, menu, and object names.	Bold text	<ul style="list-style-type: none">Click OK to save and close.From the File menu, select Exit.Select the logicalhost.exe file.Enter the timeout value.Use the getClassname() method.Configure the Inbound File eWay.
Command line arguments and code samples	Fixed font. Variables are shown in <i>bold italic</i> .	<code>bootstrap -p <i>password</i></code>
Hypertext links	Blue text	http://www.seebeyond.com

1.4.5 Screenshots

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

1.5 Related Documents

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

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

1.6 SeeBeyond Web Site

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

<http://www.seebeyond.com>

1.7 Feedback

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

docfeedback@seebeyond.com

Installing the eWay

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

What's in This Chapter

- [“Supported Operating Systems” on page 11](#)
- [“System Requirements” on page 12](#)
- [“Supported External Applications” on page 12](#)
- [“Installing the eWay Product Files” on page 12](#)
- [“After You Install” on page 13](#)

2.1 Supported Operating Systems

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

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

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

2.1.1 WebLogic and WebSphere Application Server Support

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

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

These are limited to outbound mode using Java Collaborations. For additional information see the *eGate Integrator User's Guide* and "[Support for WebSphere Application Server](#)" on page 79.

2.2 System Requirements

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

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

2.3 Supported External Applications

The DB2 eWay supports the following software on external systems:

- DB2 Universal Database (UDB) version 8.2 for Windows/UNIX.
- DB2 Universal Database (UDB) version 7.1 when connecting to DB2 running on an z/OS operating system when using DataDirect drivers.
- DB2 Universal Database (UDB) version V5R1, V5R2, and V5R3 when connecting to DB2 running on an AS/400 operating system.
- Driver support for DataDirect Drivers JDBC 3.4

2.4 Installing the eWay Product Files

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

The installation process includes installing the following components:

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

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

- On the Enterprise Manager, select the **DB2eWay.sar** (to install the DB2 eWay) file to upload.
- On the Enterprise Manager, select the **FileeWay.sar** (to install the File eWay, used in the sample Project) file to upload.
- On the Enterprise Manager, install the **DB2eWayDocs.sar** (to install the documentation and the sample) file to upload.
- On the Enterprise Manager under the Documentation tab, click on the document link or the sample file link. It is recommended that you extract the sample project file to another location prior to importing it, using the Enterprise Explorer's Import Project tool.
- For additional information on how to use eGate, please see the *eGate Integrator Tutorial*.

Continue installing the eGate Enterprise Designer as instructed.

2.5 After You Install

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

Configuring the DB2 eWay

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

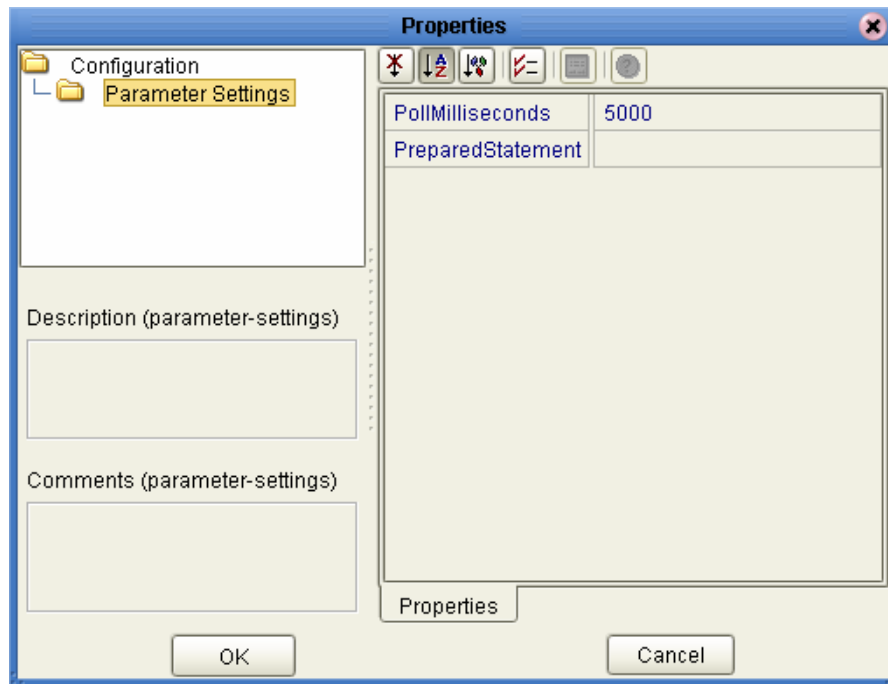
What's in This Chapter

- [“Setting the Properties of the Inbound DB2 eWay on Windows or Unix Operating Systems” on page 15](#)
- [“Setting the Environment Properties of the Inbound DB2 eWay on Windows or Unix Operating Systems” on page 16](#)
- [“Setting the Properties of the Outbound DB2 eWay on Windows or Unix Operating Systems” on page 17](#)
- [“Setting the Environment Properties of the Outbound DB2 eWay on Windows or Unix Operating Systems” on page 20](#)
- [“Setting the Properties of the Inbound DB2 eWay Connecting to z/OS or an AS/400 Operating System” on page 23](#)
- [“Setting the Environment Properties of the Inbound DB2 eWay Connecting to z/OS or an AS/400 Operating Systems” on page 24](#)
- [“Setting the Properties of the Outbound DB2 eWay Connecting to z/OS or an AS/400 Operating System” on page 26](#)
- [“Setting the Environment Properties of the Outbound DB2 eWay Connecting to z/OS or an AS/400 Operating Systems” on page 28](#)

3.1 Properties of DB2 eWay on Windows or Unix Operating Systems

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

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



PollMilliseconds

Description

Specify the polling interval between database queries in milliseconds.

Required Values

A valid numeric value. The default is 5000.

PreparedStatement

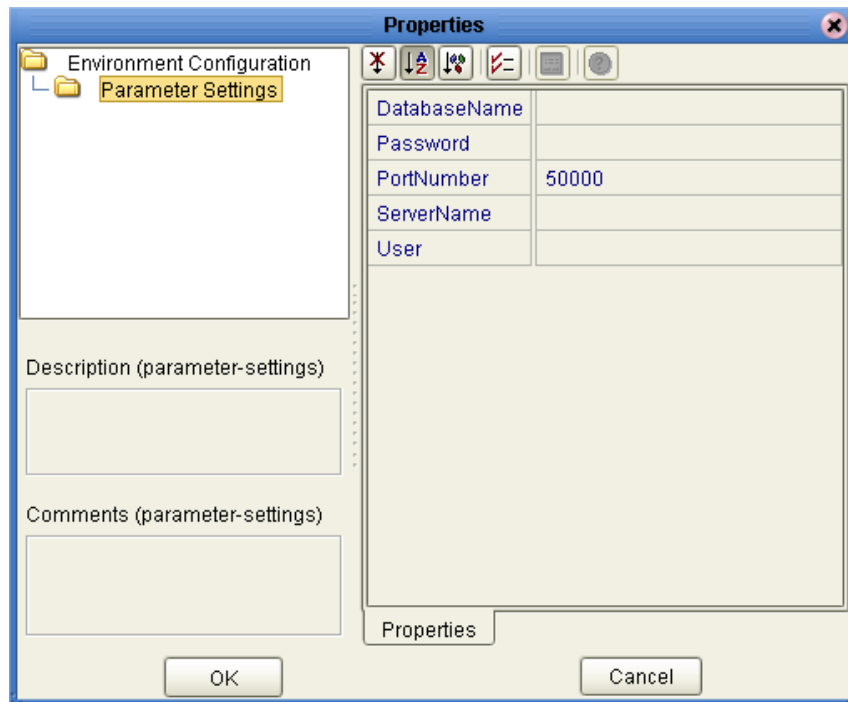
Description

Specify the Prepared Statement used to query the database.

Required Values

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

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



DatabaseName

Description

Specify the name of the database instance.

Required Values

Any valid string.

Password

Description

Specify the password used to access the database.

Required Values

Any valid string.

PortNumber

Description

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

Required Values

A valid port number. The default is 50000.

ServerName

Description

Specify the host name of the external database server.

Required Values

Any valid string.

User

Description

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

Required Values

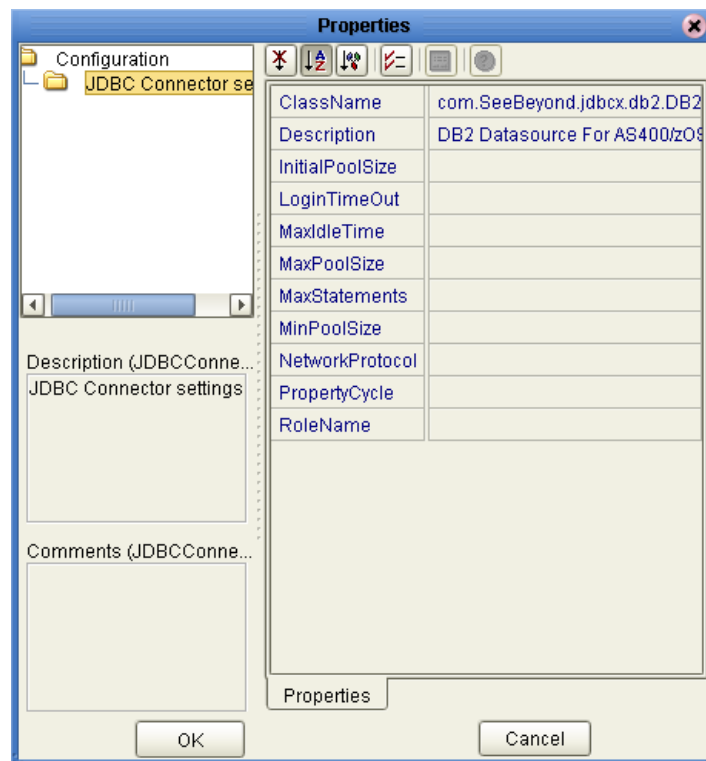
Any valid string.

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

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

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

Figure 1 The eWay Properties



ClassName

Description

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

Required Values

A valid class name.

The default is `com.SeeBeyond.jdbcx.db2.DB2DataSource`.

Description

Description

Enter a description for the database.

Required Values

A valid string.

InitialPoolSize

Description

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

Required Values

A valid numeric value.

LoginTimeout

Description

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

Required Values

A valid numeric value.

MaxIdleTime

Description

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

Required Values

A valid numeric value.

MaxPoolSize

Description

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

Required Values

A valid numeric value.

MaxStatements

Description

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

Required Values

A valid numeric value.

MinPoolSize

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

Required Values

A valid numeric value.

NetworkProtocol

Description

The network protocol used to communicate with the server.

Required Values

Any valid string.

PropertyCycle

Description

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

Required Values

A valid numeric value.

RoleName

Description

An initial SQL role name.

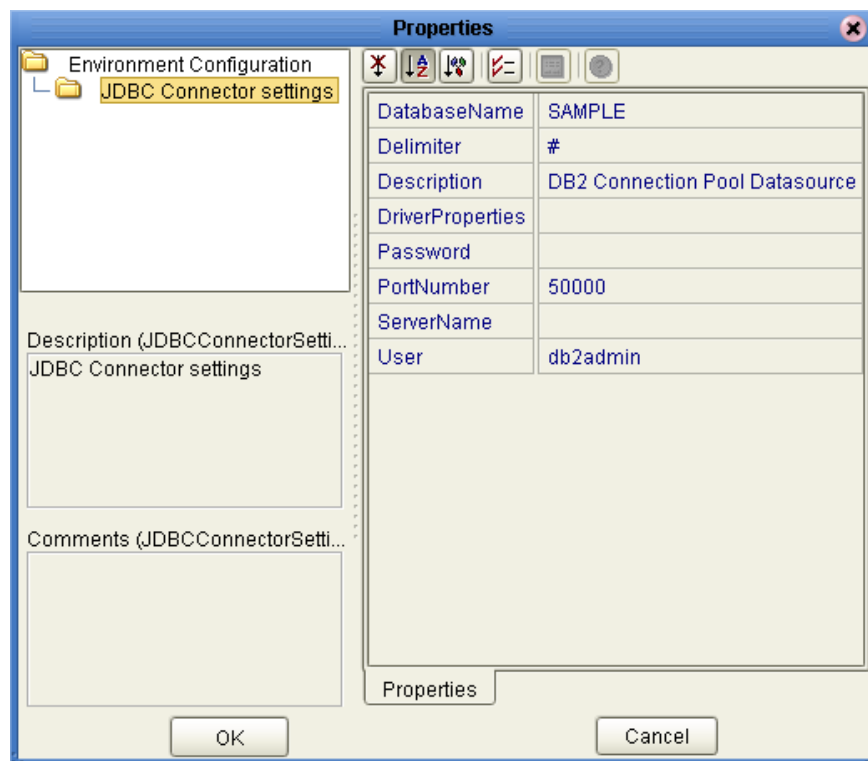
Required Values

Any valid string.

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

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

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



DatabaseName

Description

Specify the name of the database instance.

Required Values

Any valid string.

Delimiter

Description

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

Required Values

The default is #.

Description

Description

Enter a description for the database.

Required Values

A valid string.

DriverProperties

Description

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

Required Values

Any valid delimiter.

Valid delimiters are: “<method-name-1>#<param-1>#<param-2>##.....<param-n>##<method-name-2>#<param-1>#<param-2>#.....<param-n>##.....##”.

For example: to execute the method setURL, give the method a String for the URL “setURL#<url>##”.

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

Windows & Unix example:

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

zOS & AS/400 example:

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

Password

Description

Specify the password used to access the database.

Required Values

Any valid string.

PortNumber

Description

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

Required Values

A valid port number. The default is 50000.

ServerName

Description

Specify the host name of the external database server.

Required Values

Any valid string.

User

Description

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

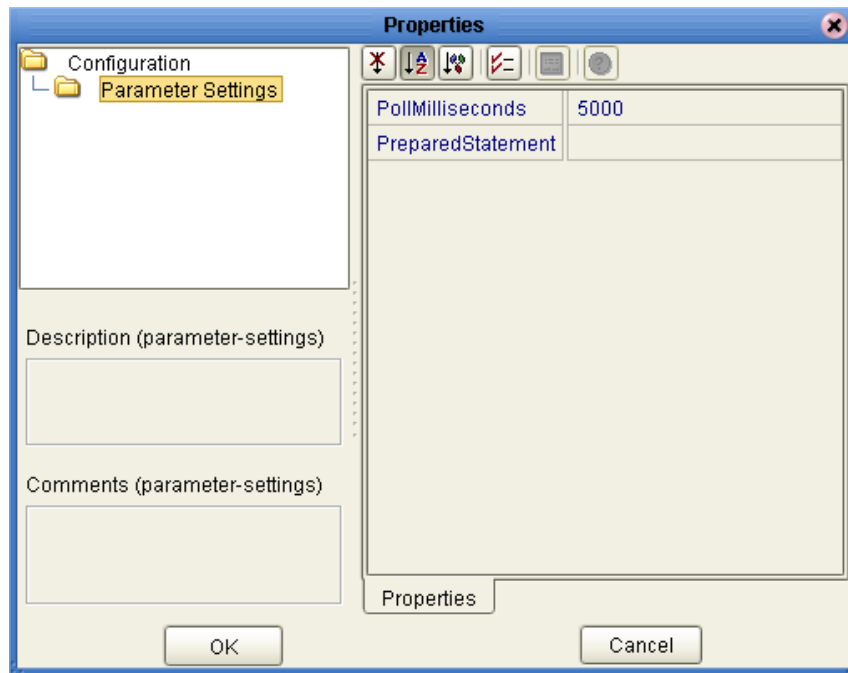
Required Values

Any valid string.

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

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

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



PollMilliseconds

Description

Specify the polling interval between database queries in milliseconds.

Required Values

A valid numeric value. The default is 5000.

PreparedStatement

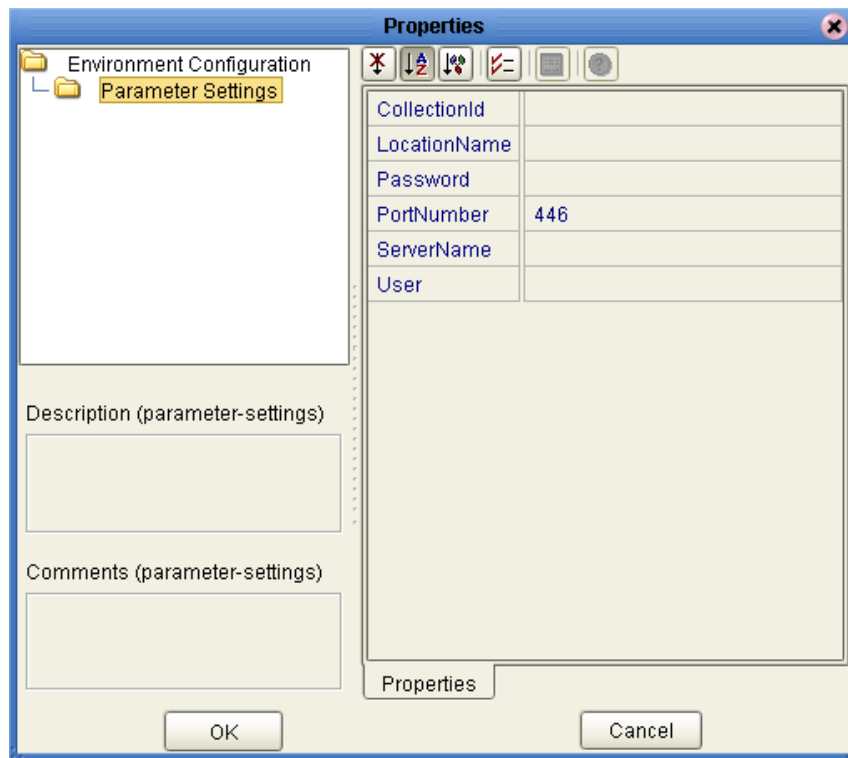
Description

Specify the Prepared Statement used to query the database.

Required Values

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

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



CollectionId

Description

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

Required Values

Any valid String.

LocationName

Description

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

Required Values

Any valid String.

Password

Description

Specify the password used to access the database.

Required Values

Any valid String.

PortNumber

Description

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

Required Values

Any valid String. The default port number is 446.

ServerName

Description

Specify the name of the database server being used.

Required Values

Any valid String.

User

Description

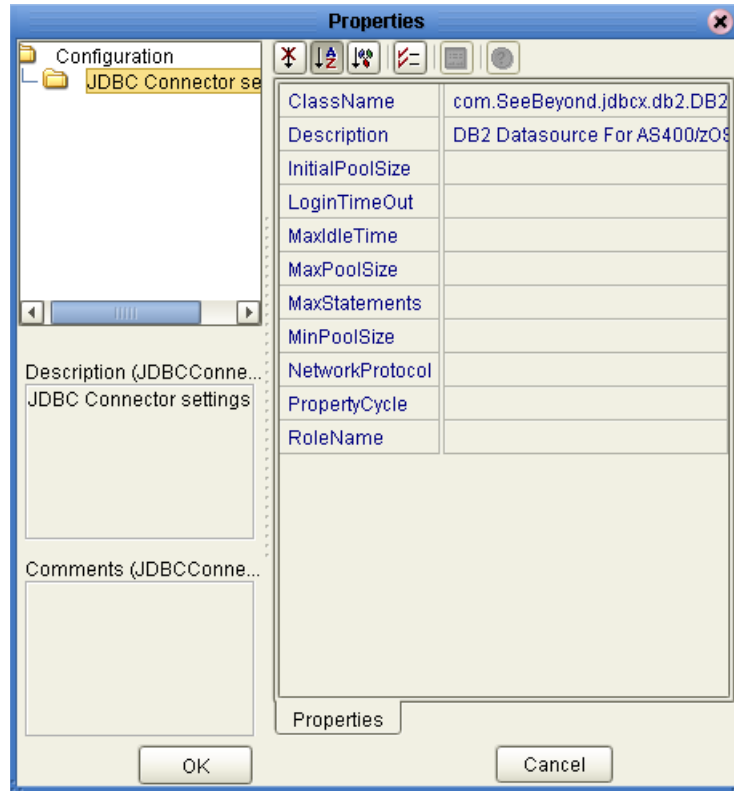
Specify the user name the eWay uses to connect to the database.

Required Values

Any valid String.

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

Figure 3 Outbound DB2 eWay Connecting to an AS400



ClassName

Description

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

Required Values

A valid class name.

The default is **com.SeeBeyond.jdbcx.db2.DB2DataSource**.

Description

Description

Enter a description for the database.

Required Values

A valid string.

InitialPoolSize

Description

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

Required Values

A valid numeric value.

LoginTimeOut

Description

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

Required Values

A valid numeric value.

MaxIdleTime

Description

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

Required Values

A valid numeric value.

MaxPoolSize

Description

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

Required Value

A valid numeric value.

MaxStatements

Description

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

Required Values

A valid numeric value.

MinPoolSize

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

Required Values

A valid numeric value.

NetworkProtocol

Description

The network protocol used to communicate with the server.

Required Values

Any valid string.

PropertyCycle

Description

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

Required Values

A valid numeric value.

RoleName

Description

An initial SQL role name.

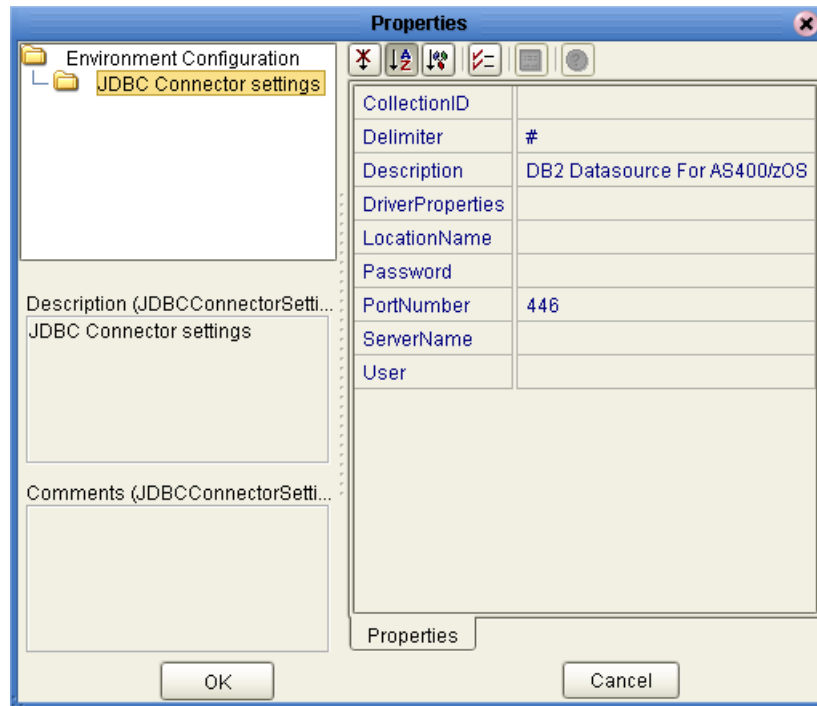
Required Values

Any valid string.

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

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

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



CollectionID

Description

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

Required Values

Any valid string.

Delimiter

Description

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

Required Values

The default is #.

Description

Description

Enter a description for the database.

Required Values

A valid string.

DriverProperties

Description

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

Required Values

Any valid delimiter.

Valid delimiters are: “<method-name-1>#<param-1>#<param-2>##.....<param-n>##<method-name-2>#<param-1>#<param-2>#.....<param-n>##.....##”.

For example: to execute the method `setURL`, give the method a String for the URL “`setURL#<url>##`”.

If you are using Spy Log. Optional:

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

LocationName

Description

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

Required Values

Any valid string.

Password

Description

Specify the password used to access the database.

Required Values

Any valid string.

PortNumber

Description

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

Required Values

A valid port number. The default is 446.

ServerName

Description

Specify the host name of the external database server.

Required Values

Any valid string.

User

Description

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

Required Values

Any valid string.

DB2 Wizard Operation

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

What's in This Chapter

- “Select Wizard Type” on page 32
- “Connect to Database” on page 33
- “Select Database Objects” on page 34
- “Select Table/Views” on page 35
- “Select Procedures” on page 39
- “Add Prepared Statements” on page 41
- “Specify the OTD Name” on page 43

4.1 Using the Database OTD Wizard

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

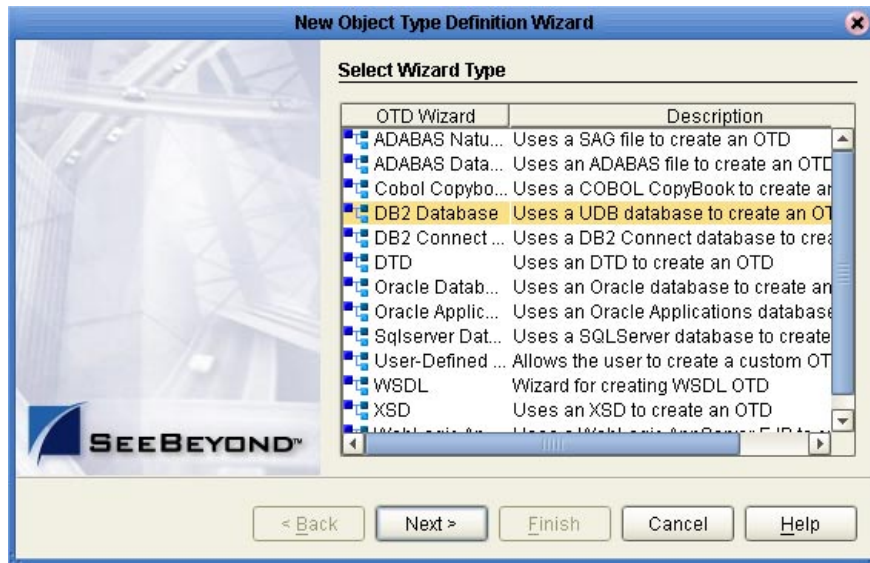
Field nodes are added to the OTD based on the Tables in the external data source. Java method and parameter nodes are added to provide the appropriate JDBC functionality. For more information about Java methods, refer to your JDBC developer's reference.

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

Select Wizard Type

- 1 On the Enterprise Explorer, right click on the project and select **Create an Object Type Definition** from the shortcut menu.
- 2 From the OTD Wizard Selection window, select the **DB2 Database** and click **Next**. See Figure 5.

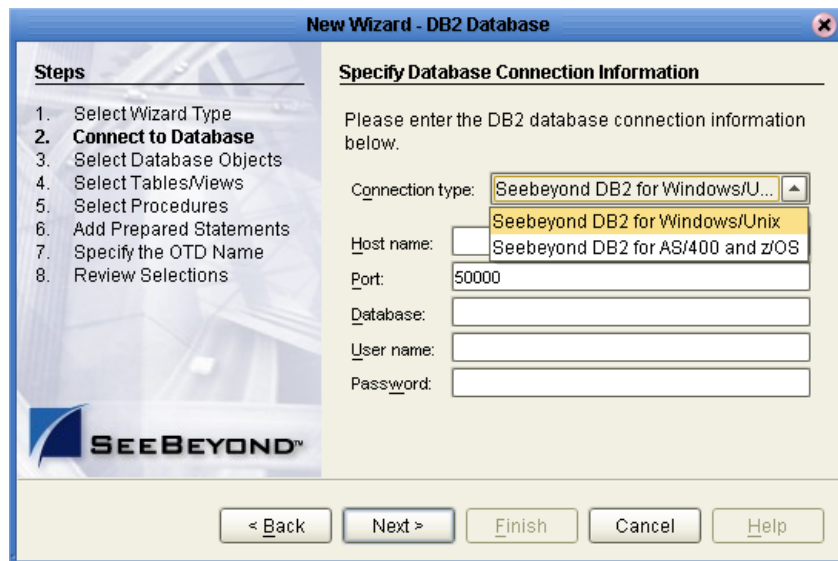
Figure 5 OTD Wizard Selection



Connect to Database

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

Figure 6 Database Connection Information



- 2 Specify the applicable connection information for your database including:
 - ♦ **Host Name** - The server where DB2 resides.
 - ♦ **Port** - The port number of DB2.
 - ♦ **Location** (AS/400 and z/OS specific) - The name of the DB2 subsystem. To find the location of the DB2 subsystem, use the Database Query Tool to issue the following query: `select current server from sysibm.sysdummy1`.
 - ♦ **Database** (Windows/Unix specific) - The name of the database instance.
 - ♦ **Collection** (AS/400 and z/OS specific) - The name that identifies a group of packages.
 - ♦ **User Name** - The user name that the eWay uses to connect to the database.
 - ♦ **Password** - The password used to access the database.

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

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

Packages

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

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

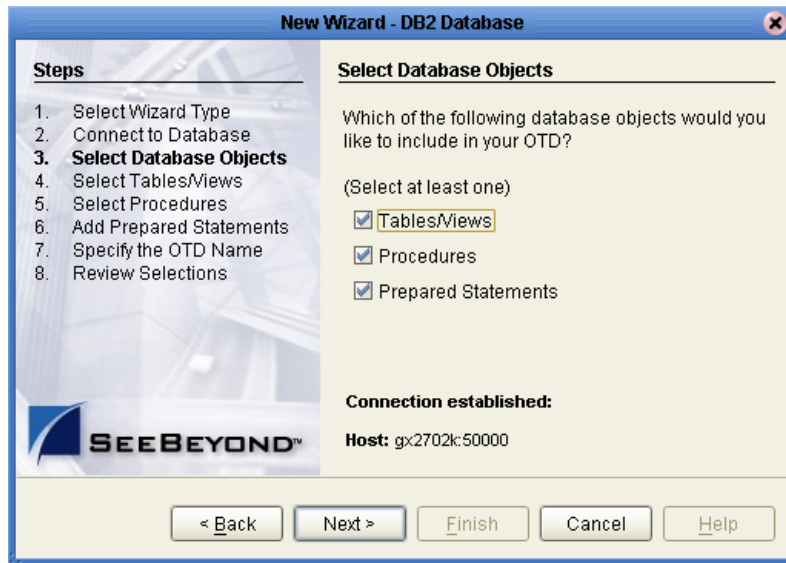
The driver creates SQL packages on the database including: DDJC330A, DDJC330B, DDJC330C, etc. When connecting, the driver queries a system table to determine whether the default packages exist on the system. If none exist, the driver creates them.

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

Select Database Objects

- 1 When selecting Database Objects, you can select any combination of **Tables**, **Views**, **Procedures**, or **Prepared Statements** to be included in the .otd file (see Figure 7).

Figure 7 Select Database Objects



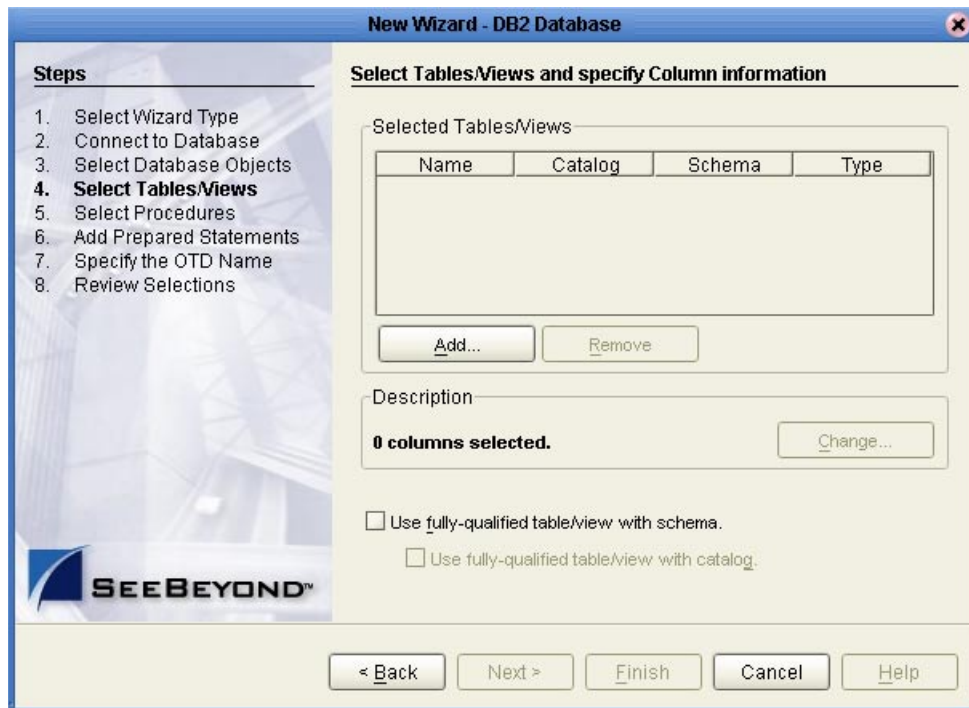
- 2 Click **Next** to continue. The **Select Tables/Views** window appears.

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

Select Table/Views

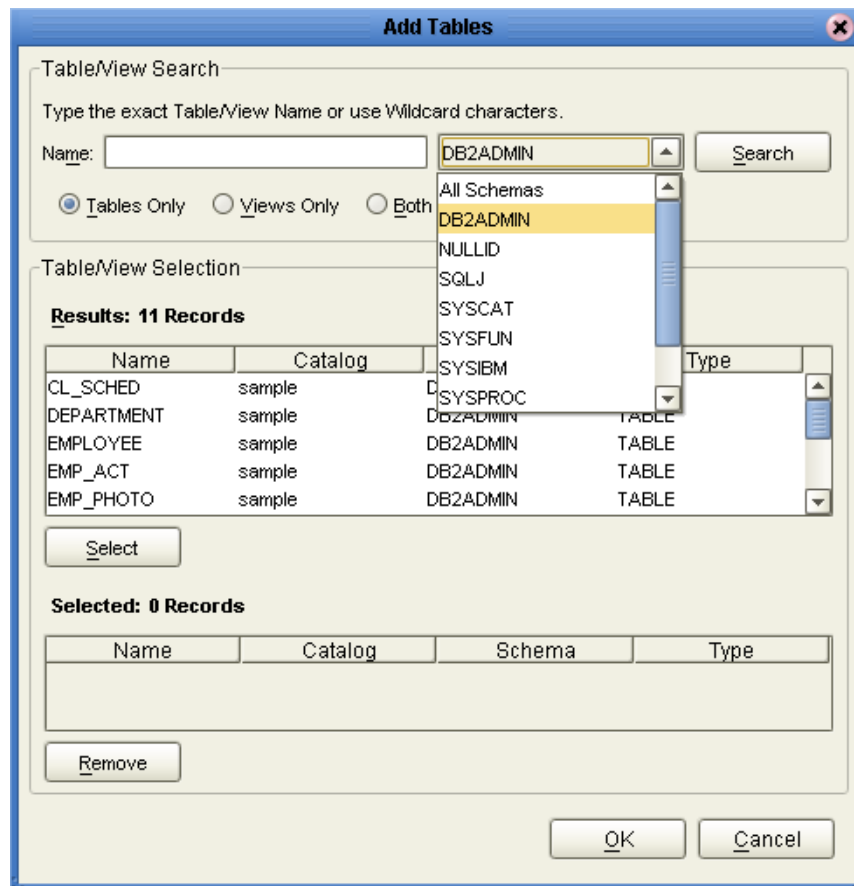
- 1 In the **Select Tables/Views** window, click **Add** (see Figure 8).

Figure 8 Select Tables/Views



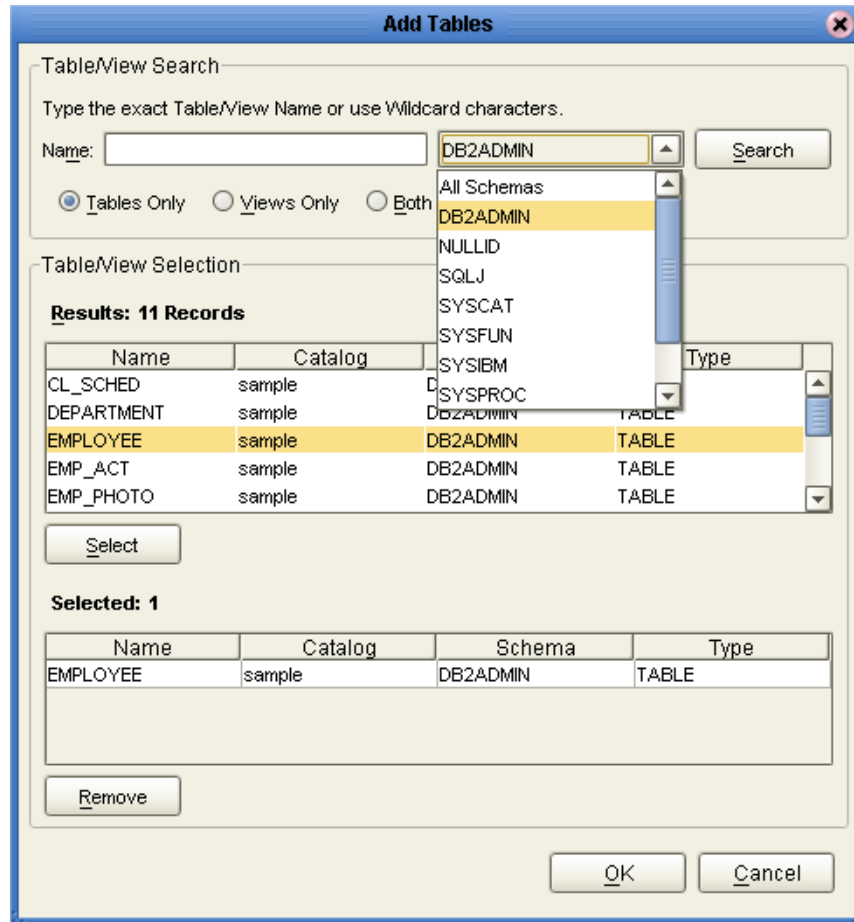
- 2 In the **Add Tables** window, select the type of criteria to be used for your search, consisting of table data, view only data, or both. You can include system tables in your search by selecting the checkbox.
- 3 From the **Table/View Name** drop down list, select the location of your database table and click **Search** (see Figure 9). You can search for **Table/View Names** by entering a table name. The use of wildcard characters of '?', and '*' as part of your Table/View name search allow for greater search capabilities. For example, "AB?CD" or "AB*CD".

Figure 9 Database Wizard - All Schemes



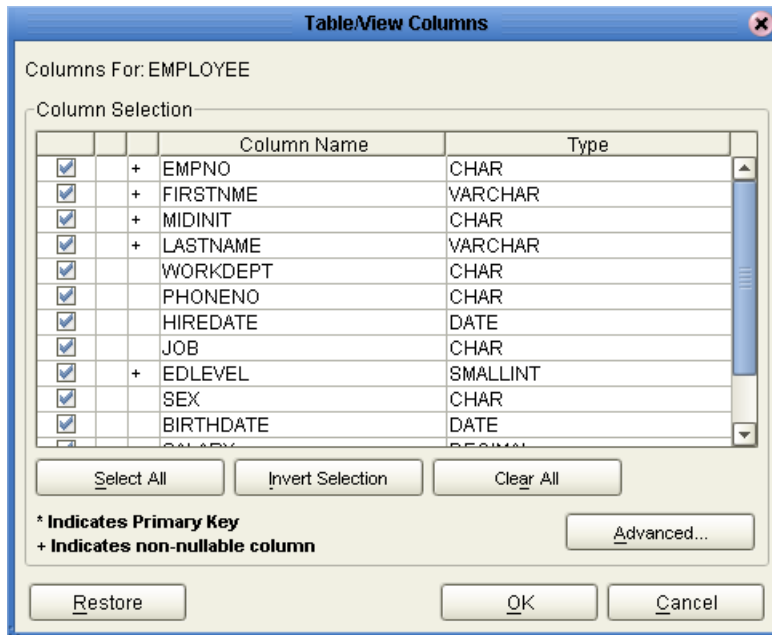
- 4 Select the table of choice and click **OK**. The table selected is added to the **Selected** window (see Figure 10).

Figure 10 Selected Tables/Views window with a table selected



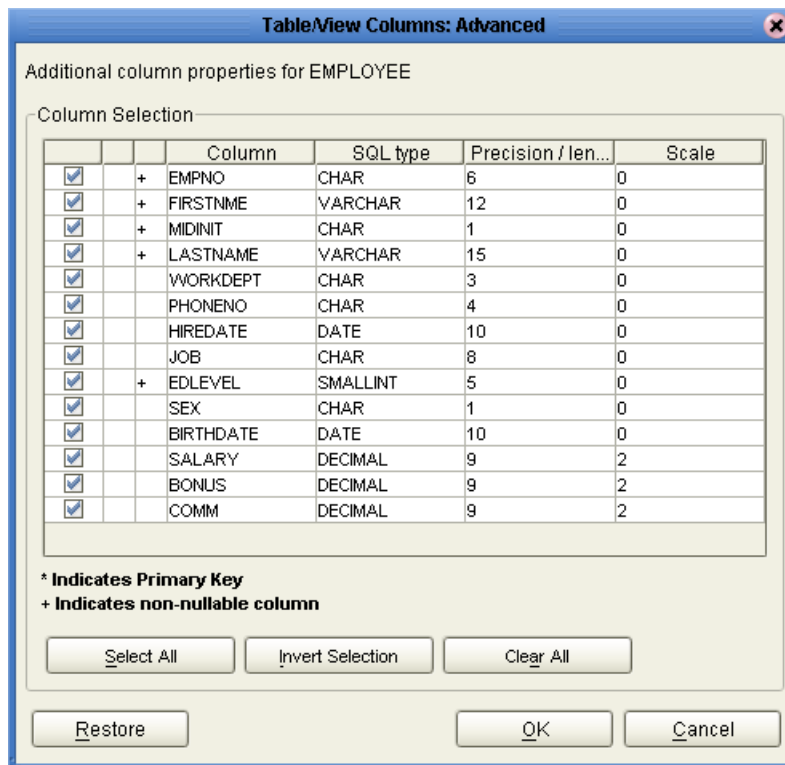
- 5 On the **Selected Tables/Views** window, review the table(s) you have selected. To make changes to the selected Table or View, click **Change**. If you do not wish to make any additional changes, click **Next** to continue.
- 6 If you clicked **Change** on the **Selected Tables/Views** window, you can select or deselect your table columns on the **Table/View Columns** window. You can also change the data type for each table by highlighting the data type and selecting a different one from the drop down (see Figure 11).

Figure 11 Table/View Columns



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

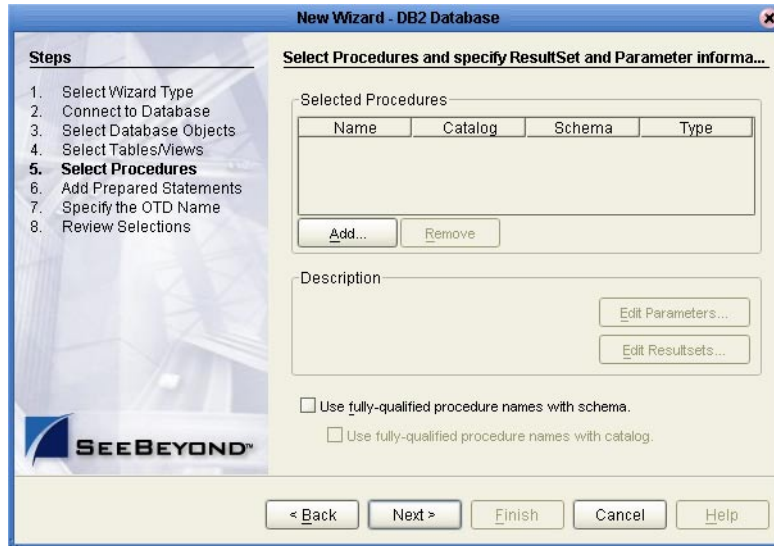
Figure 12 Table/View Columns – Advanced



Select Procedures

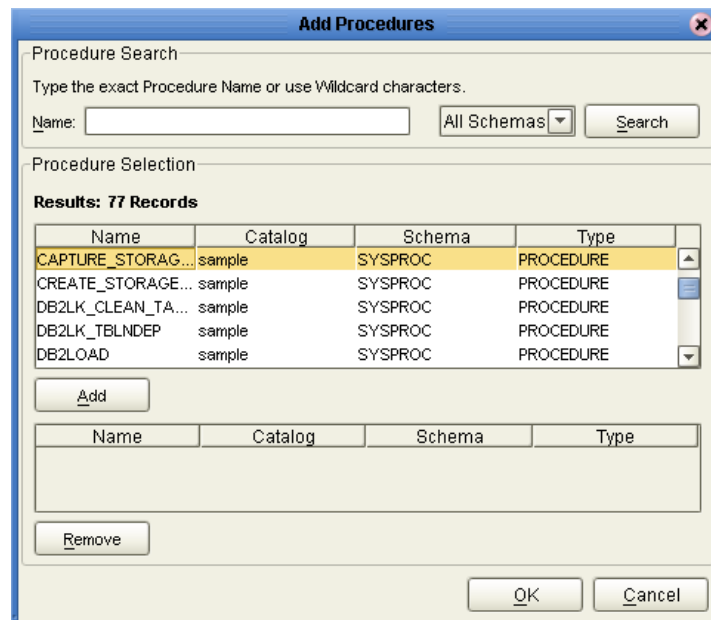
- 1 On the **Select Procedures and specify Resultset and Parameter Information** window, click **Add**.

Figure 13 Select Procedures and specify Resultset and Parameter Information



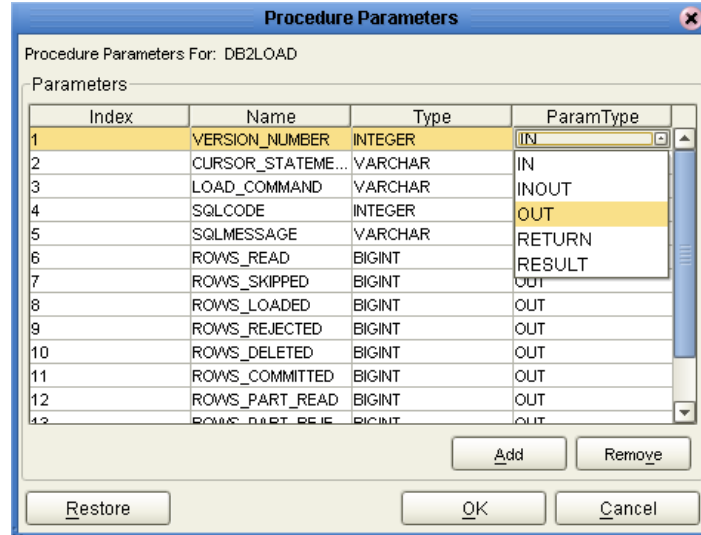
- 2 On the **Select Procedures** window, enter the name of a Procedure or select a table from the drop down list. Click **Search**. Wildcard characters can also be used.
- 3 In the resulting **Procedure Selection** list box, select a Procedure. Click **OK**.

Figure 14 Add Procedures



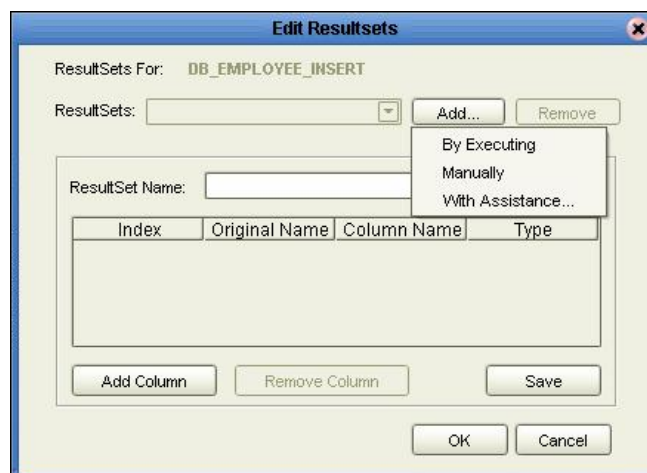
- 4 On the **Select Procedures and specify Resultset and Parameter Information** window click **Edit Parameters** to make any changes to the selected Procedure (see Figure 15).

Figure 15 Procedure Parameters



- 5 To restore the data type, click **Restore**. When finished, click **OK**.
- 6 To select how you would like the OTD to generate the nodes for the Resultset click **Edit Resultsets**.
- 7 Click **Add** to add the type of Resultset node you would like to generate.

Figure 16 Edit Resultset



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

The "**By Executing**" mode executes the specified Stored Procedure with default values to generate the ResultSet(s). Depending on the business logic of the Stored Procedure, zero or more ResultSets can be returned from the execution. If there are multiple ResultSets and the "**By Executing**" mode does not return all ResultSets, one will use the other modes to generate the ResultSet nodes.

The "**With Assistance**" mode enables users to specify a query and execute it to generate the ResultSet node. To facilitate this operation, the DBWizard tries to retrieve the content of the specified Stored Procedure and display it. However, content retrieval is not supported by all types of Stored Procedures. We can roughly classify Stored Procedures into two types: SQL and external. SQL Stored Procedures are created using CREATE PROCEDURE SQL statements while external Stored Procedures are created using host languages (e.g. Java). Since external Stored Procedures do not store their execution plans in the database, content retrieval is impossible. When using the "**Assist**" mode, highlight the execute statement up to and including the table name(s) before executing the query.

The "**Manually**" mode is the most flexible way to generate the result set nodes. It enables users to specify the node name, original column name and data type manually. One drawback of this method is that users need to know the original column names and data types. This is not always possible. For example, the column name of 3*C in this query.

```
SELECT A, B, 3*C FROM table T
```

is generated by the database. In this case, the "**With Assistance**" mode is a better choice.

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

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

Add Prepared Statements

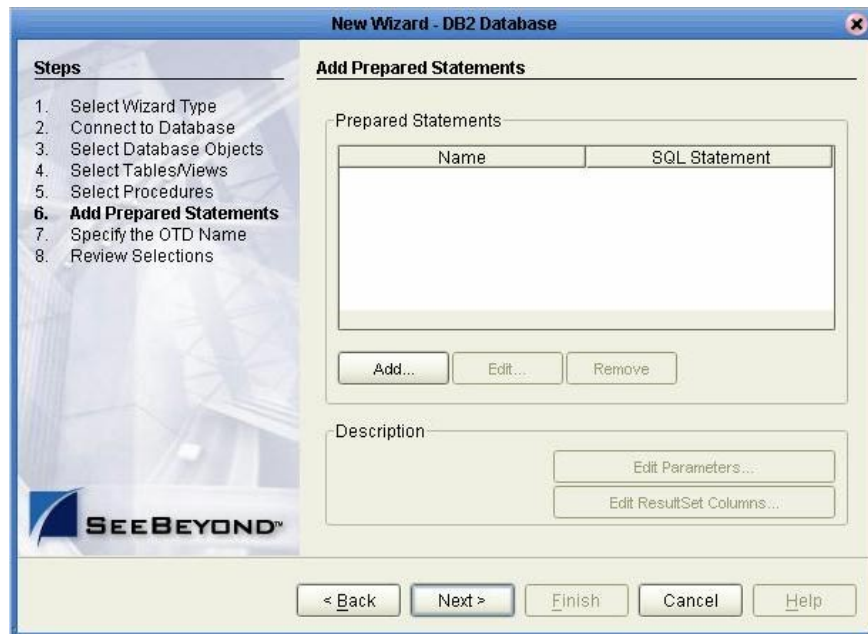
A Prepared Statement OTD represents a SQL statement that has been compiled. Fields in the OTD correspond to the input values that users need to provide.

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

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

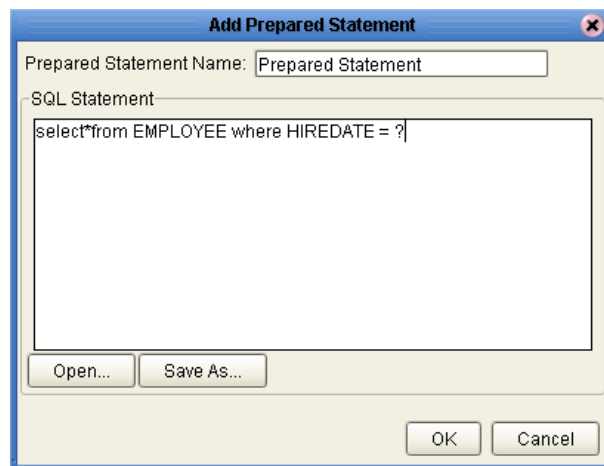
- 1 On the **Add Prepared Statements** window, click **Add**.

Figure 17 Prepared Statement



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

Figure 18 Prepared SQL Statement

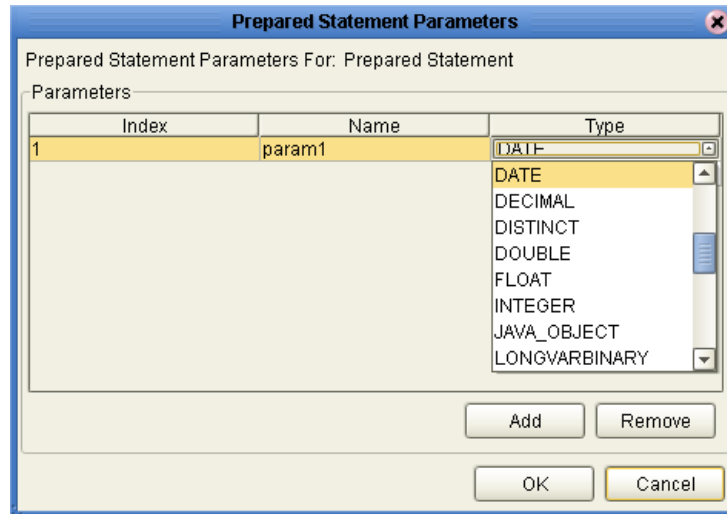


- 3 On the **Add Prepared Statement** window, the name you assigned to the Prepared Statement appears. To edit the parameters, click **Edit Parameters**. You can change the datatype by clicking in the **Type** field and selecting a different type from the list.

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

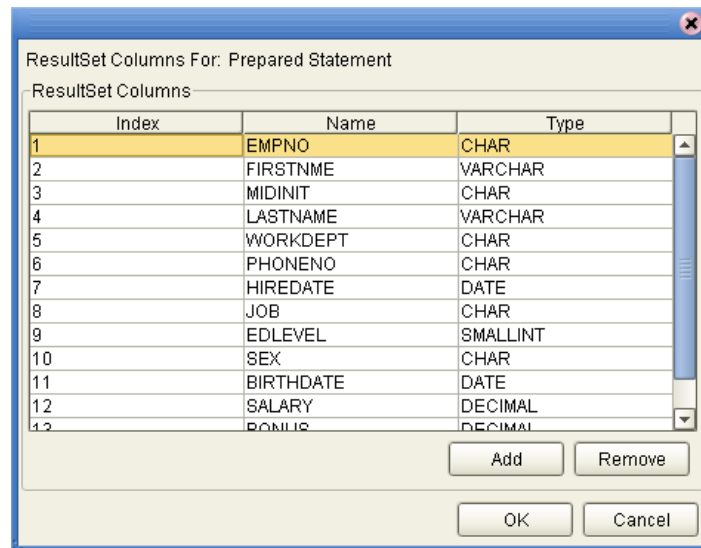
- Click **Add** if you want to add additional parameters to the Statement or highlight a row and click **Remove** to remove it. Click **OK** (see Figure 19).

Figure 19 Edit the Prepared Statement Parameters



- To edit the Resultset Columns, click **Edit Resultset Columns**. Both the Name and Type are editable. Click **OK** (see Figure 20).

Figure 20 ResultSet Columns

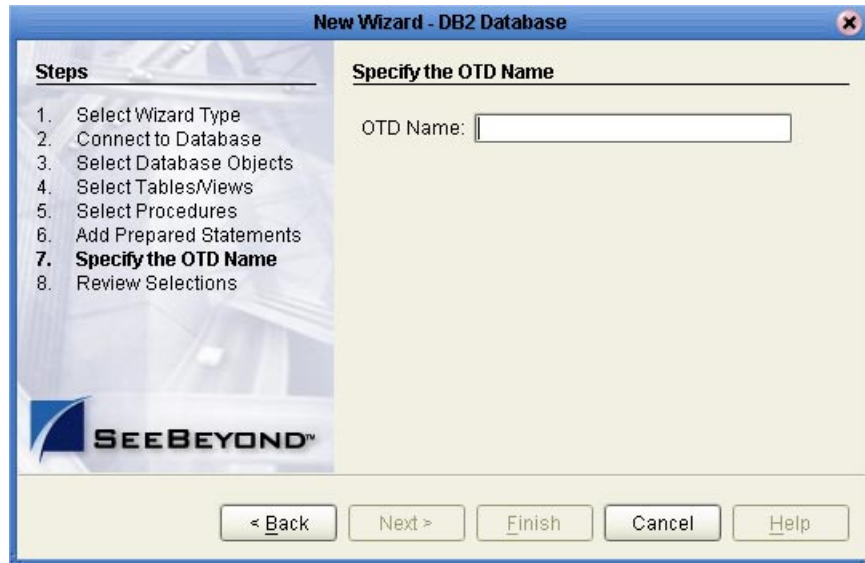


- On the **Add Prepared Statements** window, click **OK**.

Specify the OTD Name

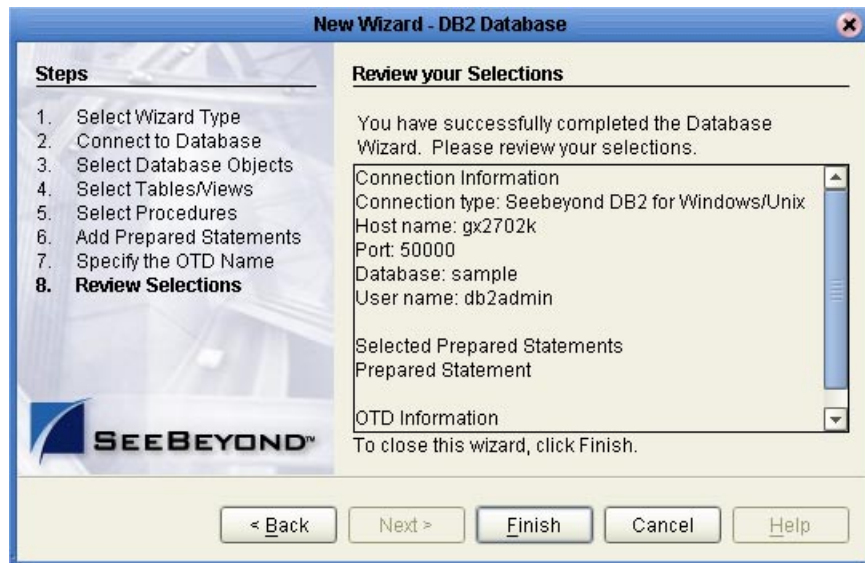
- Enter a name for the OTD. The OTD contains the selected tables and the package name of the generated classes (see Figure 21).

Figure 21 Naming an OTD



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

Figure 22 Database Wizard - Summary



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

Implementing the DB2 eWay

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

What's in This Chapter

- [“eInsight Engine and eGate Components” on page 45](#)
- [“Using the Sample Project in eInsight” on page 45](#)
- [“Using the Sample Project in eGate” on page 56](#)
- [“Common Data Type Conversions” on page 58](#)
- [“Using OTDs with Tables, Views, Stored Procedures, and Prepared Statements” on page 60](#)
- [“Editing Existing OTDs” on page 77](#)
- [“Alerting and Logging” on page 78](#)

5.1 eInsight Engine and eGate Components

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

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

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

5.2 Using the Sample Project in eInsight

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

Project Import utility. Import the **DB2_sampleBPEL.zip** file contained in the eWay sample folder on the installation CD-ROM.

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

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

Importing the Sample Project

- 1 From Enterprise Designer's Project Explorer pane, right-click the Repository and select **Import**.
- 2 In the Import Manager window, browse to the directory that contains the sample Project zip files.

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

- ♦ DB2_sampleBPEL.zip
- ♦ DB2_sampleJCE.zip

- 3 Select a sample Project zip file and click **Import**.
- 4 After importing the file, click **OK** on the Import Status window. You can now import another zip file, or click **Close** to exit the Import Manager window.

The Business Process

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

Table 2 Sample Project Data

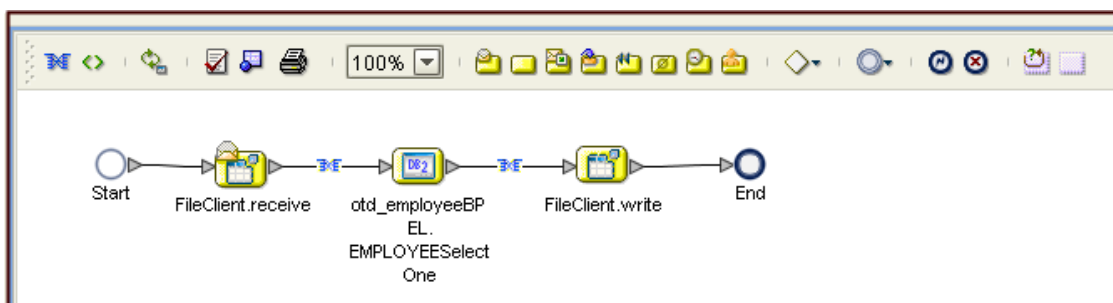
Column Name	Mapping	Data Type	Data Length
EMPNO	EMPNO	char	6
FIRSTNME	FIRSTNME	varchar	12
MIDINIT	MIDINIT	char	1
LASTNAME	LASTNAME	varchar	15
WORKDEPT	WORKDEPT	char	3
PHONENO	PHONENO	char	4
HIREDATE	HIREDATE	date	4
JOB	JOB	char	8
EDLEVEL	EDLEVEL	smallint	2
SEX	SEX	char	1
BIRTHDATE	BIRTHDATE	date	4

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

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

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

Figure 23 Sample Project Business Process



You can associate an eInsight Business Process Activity with the eWay, both during the system design phase and during run time. To make this association, select the desired **receive** or **write** operation under the eWay in the Enterprise Explorer and drag it onto the eInsight Business Process canvas. The following operations are available:

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

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

At run time, the eInsight engine invokes each step in the order that you defined in the Business Process. Using the engine’s Web Services interface, the Activity in turn invokes the DB2 eWay. You can open a file specified in the eWay and view its contents before and after the Business Process is executed.

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

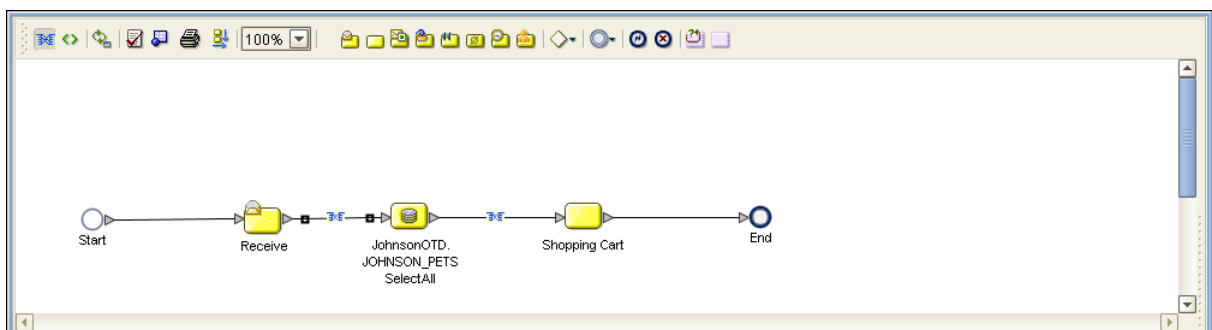
eInsight Operation	Input	Output
SelectAll	where() clause (optional)	Returns all rows that fit the condition of the where() clause
SelectMultiple	number of rows where() clause. Optional	Returns the number of rows specified that fit the condition of the where() clause
SelectOne	number of rows where() clause.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

5.2.1 SelectAll

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

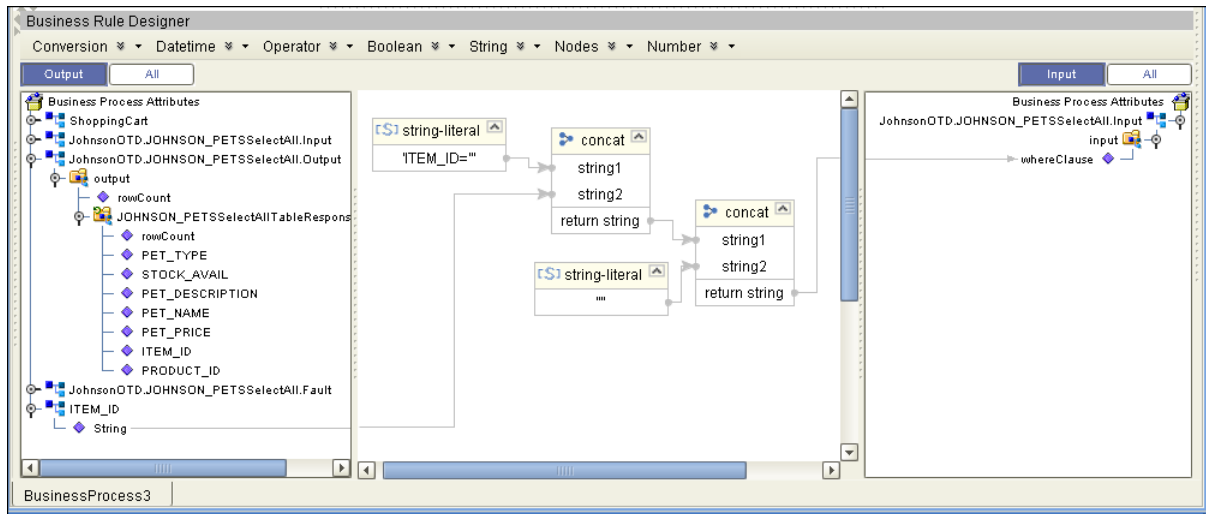
The figure below shows a sample eInsight Business Process using the SelectAll operation. In this process, the SelectAll operation returns all rows where the ITEM_ID matches the selected ITEM_ID to the shopping cart.

Figure 24 SelectAll Sample Business Process



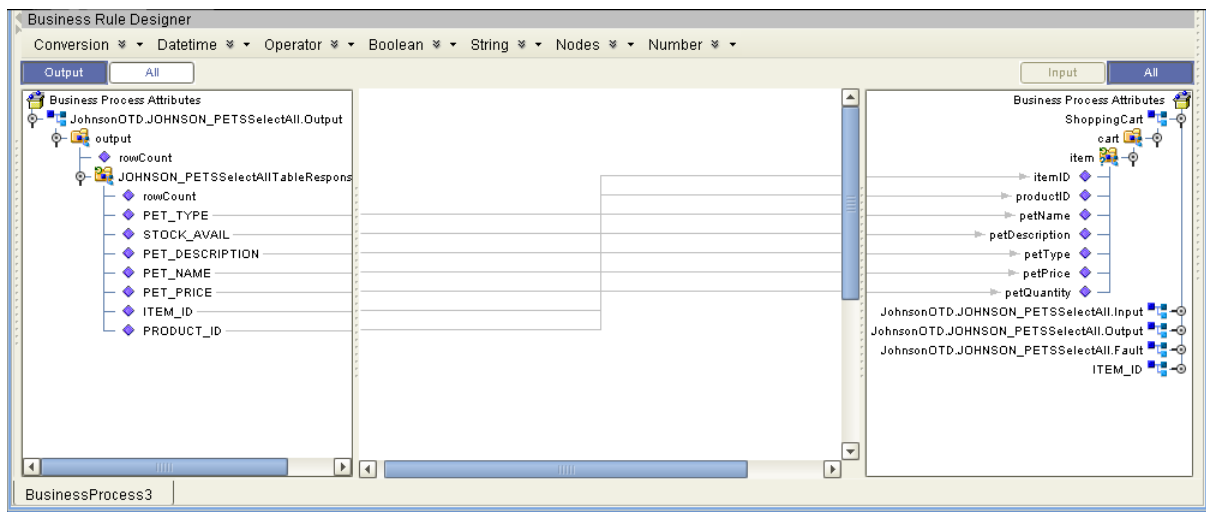
The figure below shows the definition of the where() clause for the SelectAll operation.

Figure 25 SelectAll Input



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

Figure 26 SelectAll Output

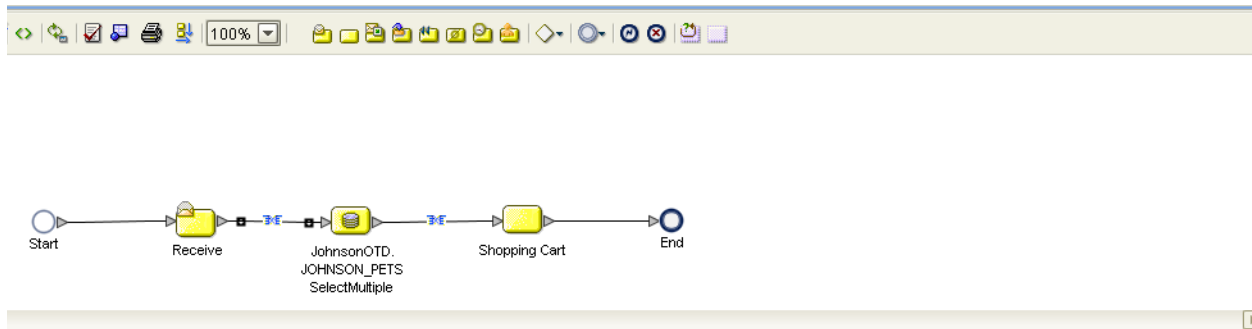


5.2.2 SelectMultiple

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

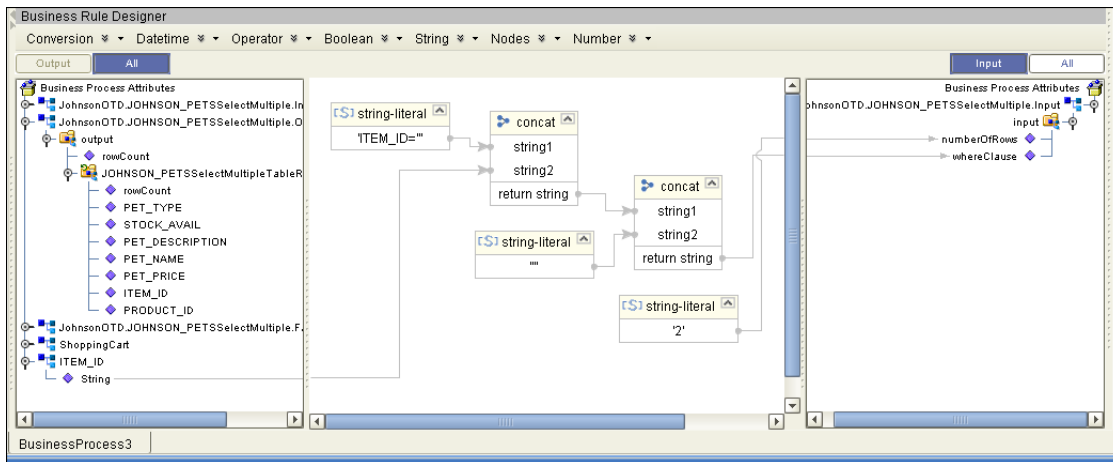
The figure below shows a sample eInsight Business Process using the SelectMultiple operation. In this process, the SelectMultiple operation returns the first two rows where the ITEM_ID matches the selected ITEM_ID to the shopping cart.

Figure 27 SelectMultiple Sample Business Process



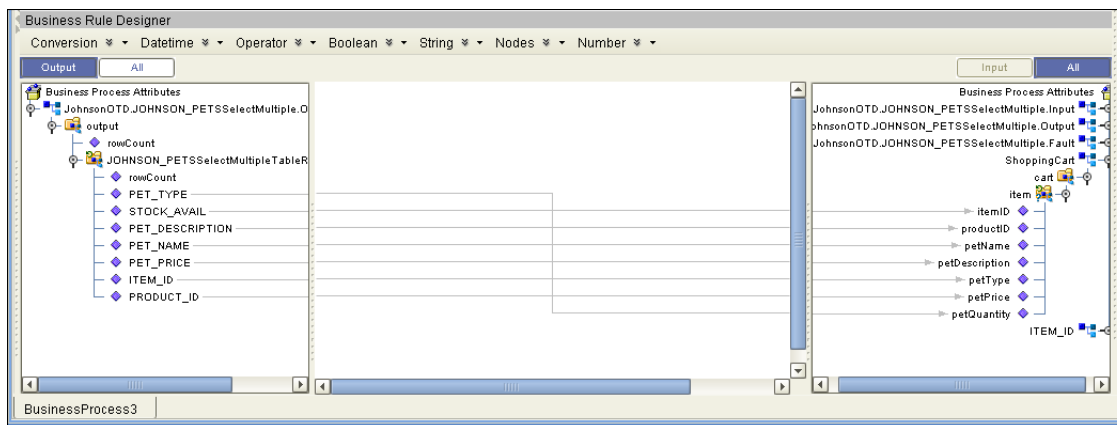
The figure below shows the definition of the number of rows and where() clause into the input for the SelectMultiple operation. You could also use an empty string or Item_ID='123'.

Figure 28 SelectMultiple Input



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

Figure 29 SelectMultiple Output

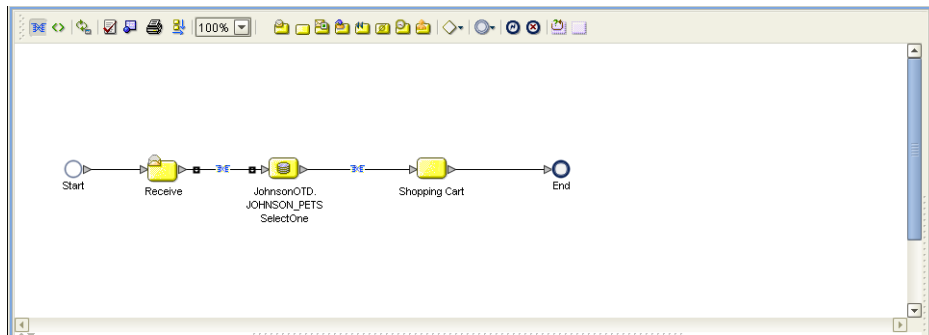


5.2.3 SelectOne

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

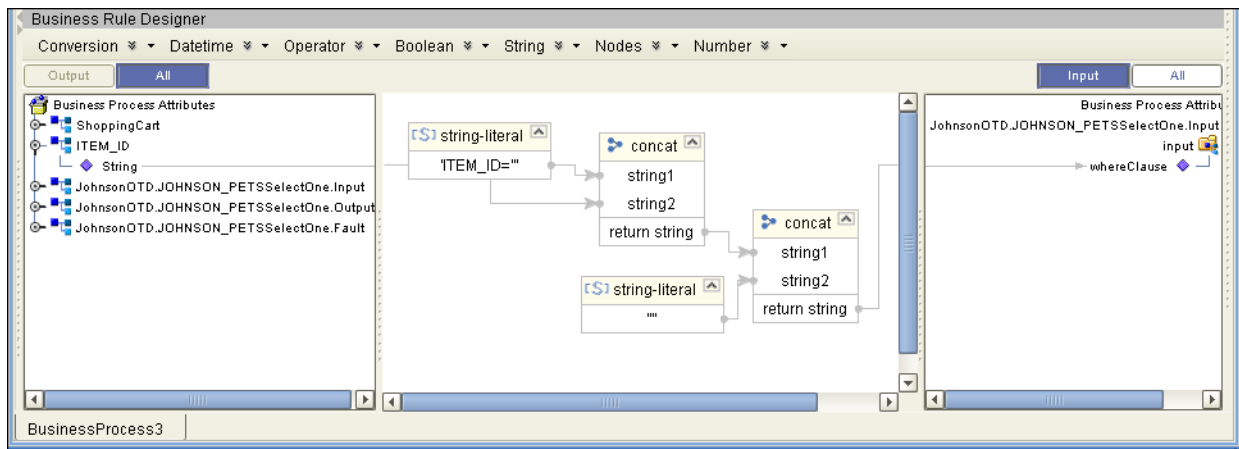
The figure below shows a sample eInsight Business Process using the SelectOne operation. In this process, the SelectOne operation returns the first row where the ITEM_ID matches the specified ITEM_ID to the shopping cart.

Figure 30 SelectOne Sample Business Process



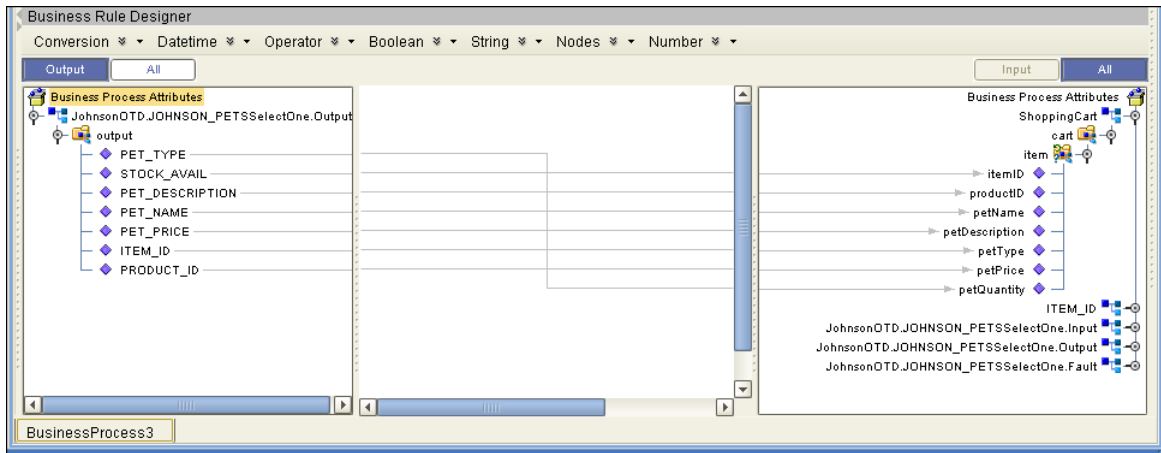
The figure below shows the definition of the where() clause for the SelectOne operation.

Figure 31 SelectOne Input



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

Figure 32 SelectOne Output

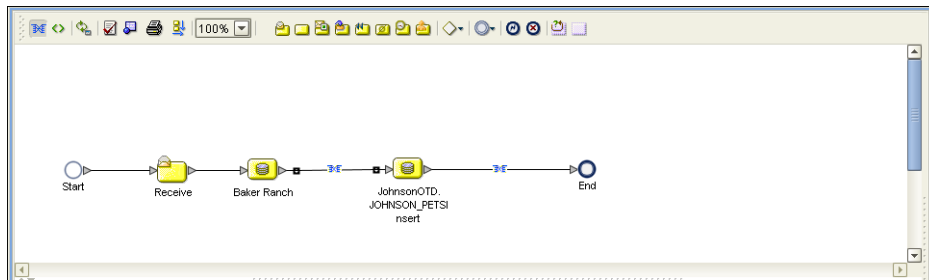


5.2.4 Insert

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

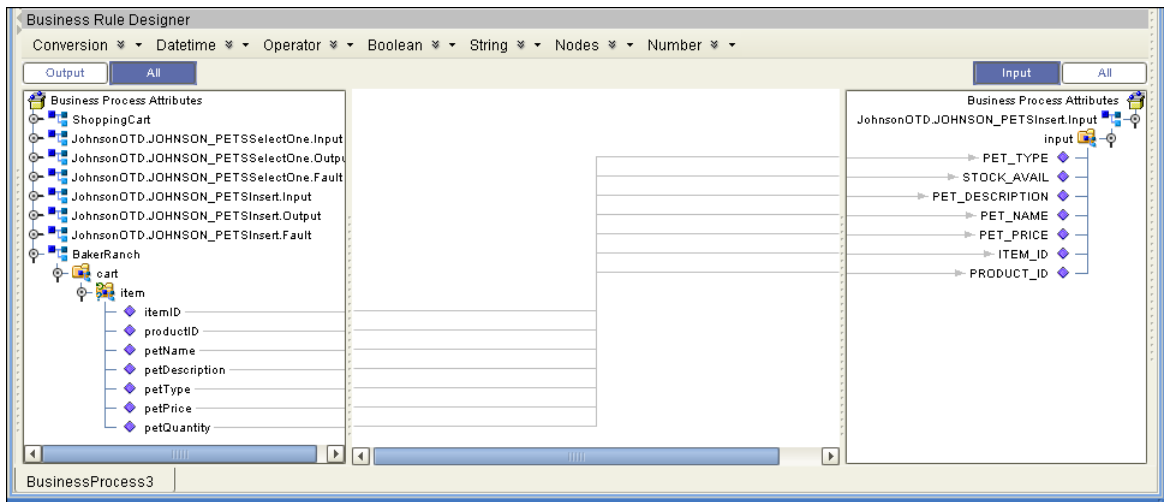
The figure below shows a sample eInsight Business Process using the Insert operation. In this process, the operation inserts a new row into the database to accommodate a new item provided by a vendor.

Figure 33 Insert Sample Business Process



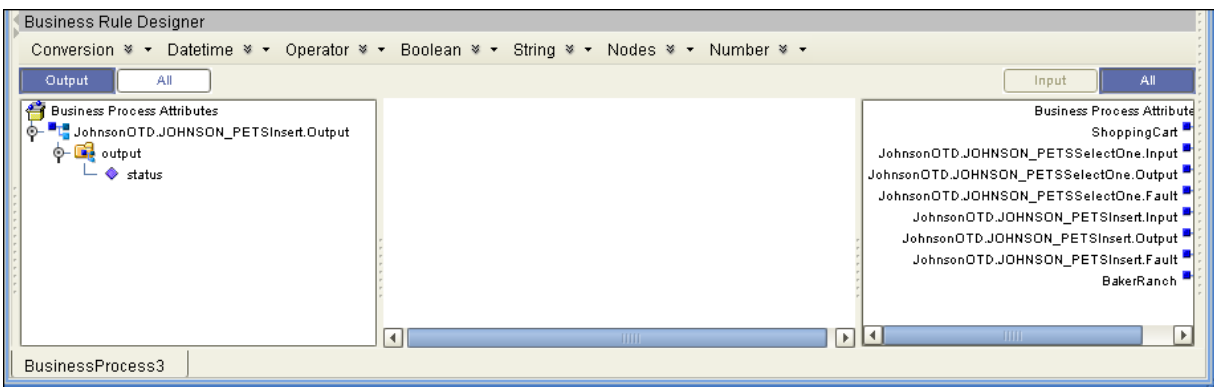
The figure below shows the definition of the input for the Insert operation.

Figure 34 Insert Input



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

Figure 35 Insert Output

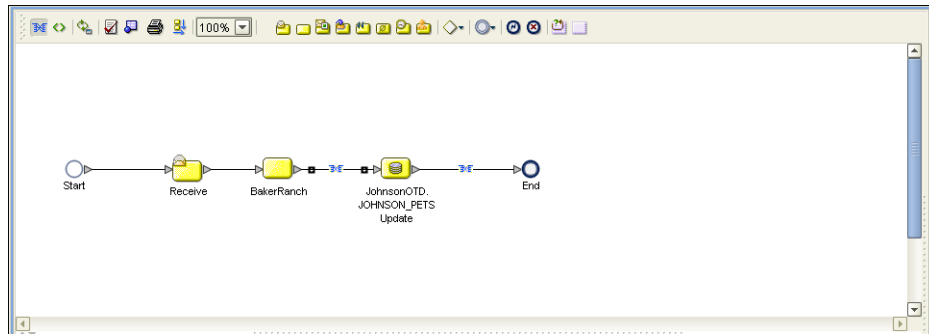


5.2.5 Update

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

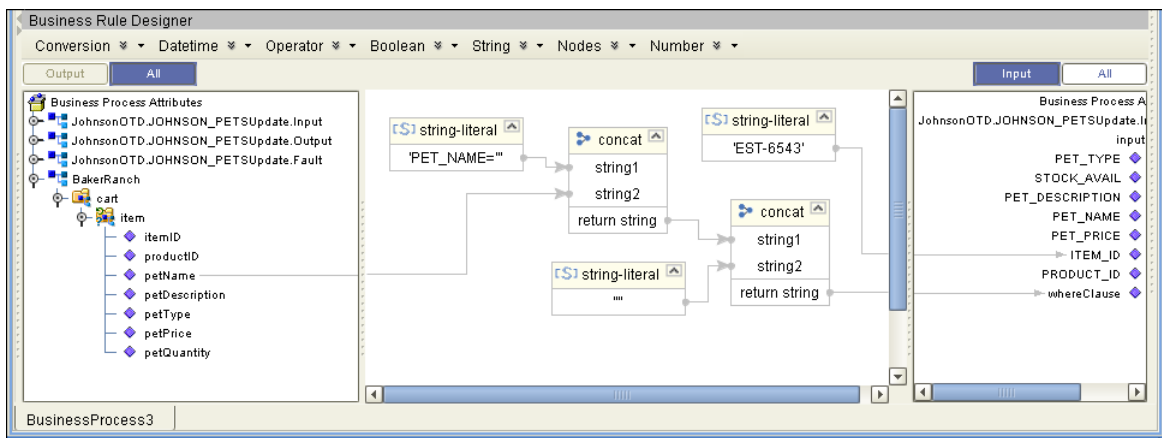
The figure below shows a sample eInsight Business Process using the Update operation. In this process, the operation updates the ITEM_ID for all items with a certain name to ESR_6543.

Figure 36 Update Sample Business Process



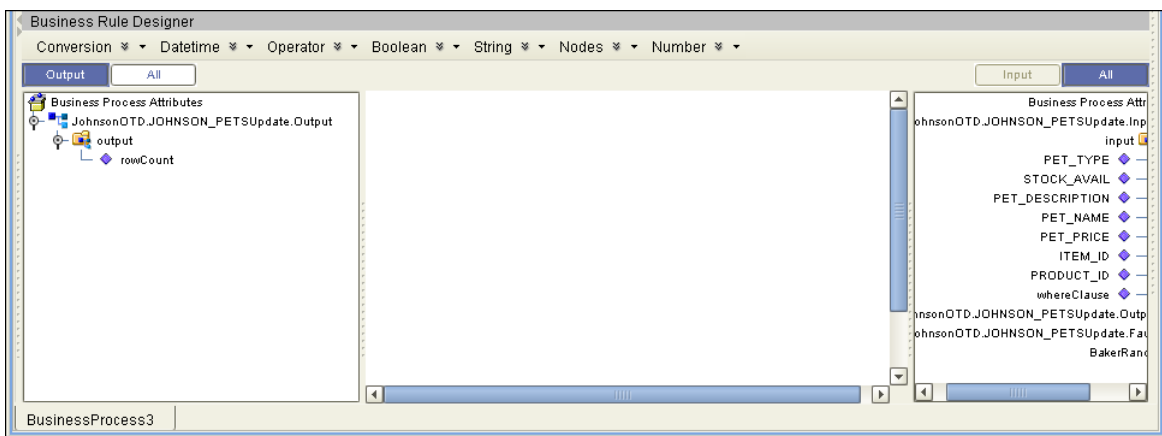
The figure below shows the definition of the where() clause for the Update operation.

Figure 37 Update Input



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

Figure 38 Update Output



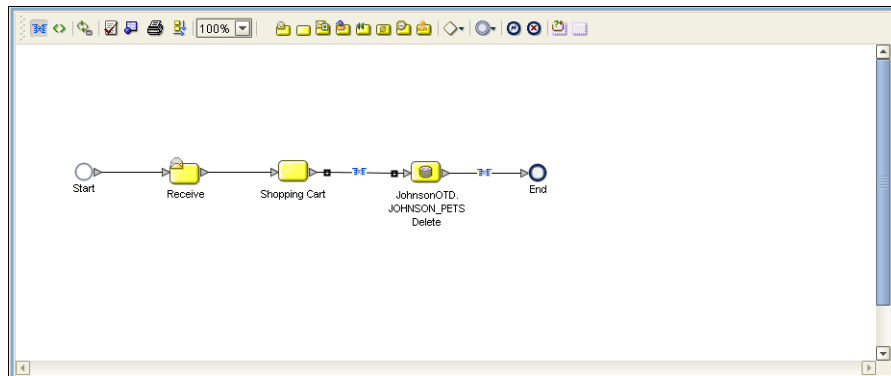
5.2.6 Delete

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

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

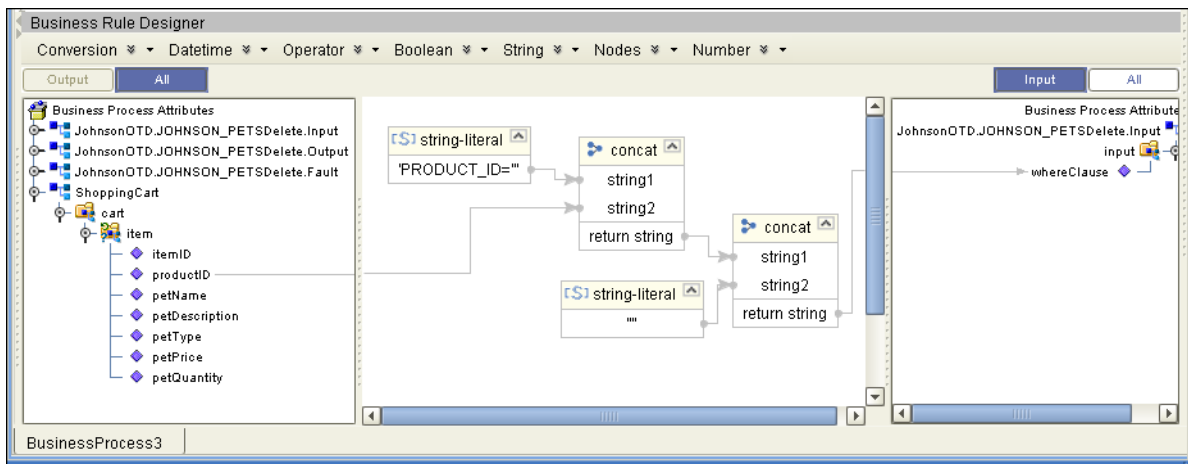
Note: If a where() clause is not defined, all rows will be deleted.

Figure 39 Delete Sample Business Process



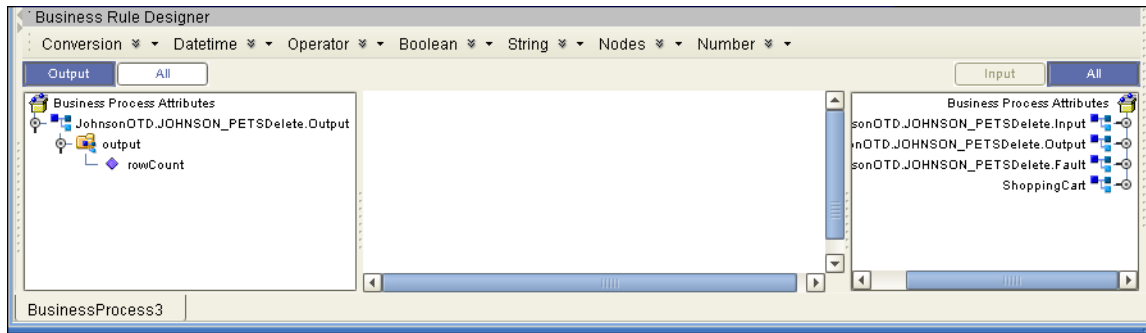
The figure below shows the definition of the where() clause for the Delete operation.

Figure 40 Delete Input



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

Figure 41 Delete Output



5.3 Using the Sample Project in eGate

To import the sample project **DB_sampleJCE.zip** follow the instructions given in [Importing the Sample Project](#) on page 46.

5.3.1 Working with the Sample Project in eGate

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

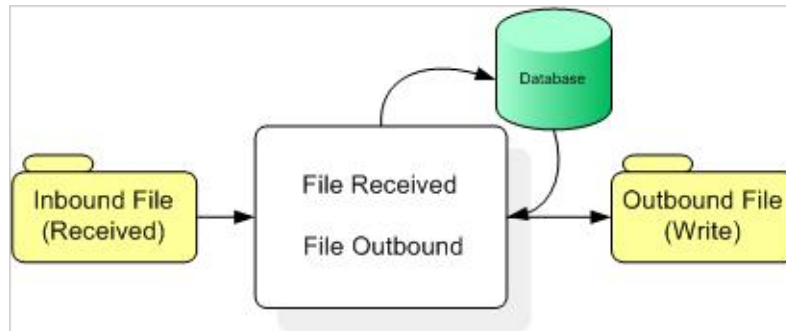
The data used for this projects is within a table called DBEmployee. The table contains the following columns:

Table 3 Sample Project Data

Column Name	Mapping	Data Type	Data Length
EMPNO	EMPNO	char	6
FIRSTNME	FIRSTNME	varchar	12
MIDINIT	MIDINIT	char	1
LASTNAME	LASTNAME	varchar	15
WORKDEPT	WORKDEPT	char	3
PHONENO	PHONENO	char	4
HIREDATE	HIREDATE	date	4
JOB	JOB	char	8
EDLEVEL	EDLEVEL	smallint	2
SEX	SEX	char	1
BIRTHDATE	BIRTHDATE	date	4
SALARY	SALARY	decimal	9
BONUS	BONUS	decimal	9
COMM	COMM	decimal	9

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

Figure 42 Database project flow



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

5.3.2 Configuring the eWays

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

To configure the Inbound File eWay:

- 1 On the Connectivity Map canvas, double click the eWay icon located between the **File1** and **Service1**.
- 2 On the resulting **Templates** window, select **Inbound File eWay** and click **OK**.
- 3 On the **Properties** window, enter the appropriate configurations for the Inbound File eWay. See the *File eWay User's Guide* for information on how to specifically configure the File eWay. For this sample, the default settings are used.
- 4 When you have completed your selections, click **OK**.

To configure the DB2 eWay:

- 1 On the Connectivity Map canvas, double click the eWay icon located between the **Service1** and **DB2** database.
- 2 On the resulting **Templates** window, select the required Outbound or Inbound DB2 connection, and click **OK**.
- 3 On the Properties window, enter the appropriate configurations for the Outbound DB2 eWay and click **OK**. See [Properties of DB2 eWay on Windows or Unix Operating Systems](#) on page 14. For this sample, the default settings are used.
- 4 When you have completed your selections, click **OK**.

To configure the Outbound File eWay:

- 1 On the Connectivity Map canvas, double click the eWay icon located between **Service1** and **File2** eWay.
- 2 On the resulting **Templates** window, select **Outbound File eWay** and click **OK**.
- 3 On the **Properties** window, enter the appropriate configurations for the Outbound File eWay. See the *File eWay User's Guide* for information on how to specifically configure the File eWay. For this sample, change the Directory field to **<valid path to the directory where the output file is stored>**. The Output File Name to **Output1.dat**. For the remaining parameters, the default settings are used.
- 4 When you have completed your selections, click **OK**.

5.3.3 Creating an External Environment

To review the components of the Sample project, there is an Inbound and an Outbound File eWay, an Inbound and an Outbound DB2 eWay, and a Service.

To create the external environment for the Sample project:

- 1 On the Environment Explorer, right-click the DB2 profile.
- 2 Select **Properties**. Enter the configuration information required for your Outbound or Inbound DB2 eWay. See [Properties of DB2 eWay on Windows or Unix Operating Systems](#) on page 14.
- 3 When you have completed your selections, click **OK**.

5.3.4 Deploying a Project

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

5.3.5 Running the Sample

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

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

5.4 Common Data Type Conversions

Table 4 The DB2 eWay Insert or Update Operations for Text/String Input Data

DB2 Data Type	OTD/Java Data Type	Java Method or New Constructor to Use (Default: Java Method)	Sample Data
Int	Int	Integer java.lang.Integer.parseInt(String)	123

DB2 Data Type	OTD/Java Data Type	Java Method or New Constructor to Use (Default: Java Method)	Sample Data
Smallint	BigDecimal	Call a New Constructor BigDecimal: java.math.BigDecimal(String)	123
Number	BigDecimal	Call a New Constructor BigDecimal: java.math.BigDecimal(String)	123
Decimal	BigDecimal	Call a New Constructor BigDecimal: java.math.BigDecimal(String)	123
BigInteger	Long	Long: java.lang.Long.parseLong(String)	123
Short	Short	Short: java.lang.Short.parseShort(String)	123
Real	Float	Float: java.lang.Float.parseFloat(String)	2454.56
Float	Double	Double: java.sql.Double.parseDouble (String)	2454.56
Double	Double	Double: java.sql.Double.parseDouble (String)	2454.56
Timestamp	Timestamp	TimeStamp: java.sql.TimeStamp.valueOf (String)	2003-09-04 23:55:59
Time	Time	Time: java.sql.Time.valueOf(String)	11:15:33
Date	Date	Date: java.sql.Date.valueOf(String)	2003-09-04
Varchar2	String	Direct Assign	Any character
Char	String	Direct Assign	Any character

Table 5 The DB2 eWay Select Operations for Text/String Output Data

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

DB2 Data Type	OTD/Java Data Type	Java Method or New Constructor to Use (Default: Java Method)	Sample Data
Number	BigDecimal	BigDecimal: java.math.BigDecimal.toString()	123
Decimal	BigDecimal	BigDecimal: java.math.BigDecimal.toString()	123
Short	Short	Short: java.lang.Short.toString(short)	123
Real	Float	Float: java.lang.Float.toString(Float)	2454.56
Float	Double	Double: java.sql.Double.parseDouble(String)	2454.56
Double	Double	Double: java.sql.Double.parseDouble(String)	2454.56
Timestamp	Timestamp	TimeStamp: java.sql.Timestamp.toString()	2003-09-04 23:55:59
Time	Time	Time: java.sql.Time.toString()	11:15:33
Date	TimeStamp	Date: java.sql.Date.toString()	2003-09-04
Varchar2	String	Direct Assign	Any character
Char	String	Direct Assign	Any character

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

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

5.5.1 The Table

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

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

- `SetConcurrencyToUpdateable`
- `SetConcurrencytoReadOnly`
- `SetScrollTypeToForwardOnly`
- `SetScrollTypeToScrollSensitive`
- `SetScrollTypeToInsensitive`

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

```
getDBEmp().setConcurToUpdateable();
getDBEmp().setScroll_TypeToScrollSensitive();
getDBEmp().getDB_EMPLOYEE().select("");
```

The Query Operation

To perform a query operation on a table

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

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

For example:

```
package SelectSales;

public class Select
{
    public com.stc.codegen.logger.Logger logger;
    public com.stc.codegen.alerter.Alerter alerter;

    public void receive(
com.stc.connector.appconn.file.FileTextMessage
input, com.stc.connector.appconn.file.FileApplication
FileClient_1, db_employee.Db_employeeOTD
db_employee_1, employeeDb.Db_employee employeeDb_db_employee_1 )
    throws Throwable
    {
        //@map:Db_employee.select(Text)
        db_employee_1.getDb_employee().select( input.getText() );

        //while
        while (db_employee_1.getDb_employee().next()) {
            //@map:Copy EMP_NO to Employee_no
            employeeDb_db_employee_1.setEmployee_no(
java.lang.Integer.toString(
db_employee_1.getDb_employee().getEMP_NO() ) );

            //@map:Copy LAST_NAME to Employee_lname
```

```

        employeeedb_db_employee_1.setEmployee_lname(
db_employee_1.getDb_employee().getLAST_NAME() );

        //@map:Copy FIRST_NAME to Employee_fname
        employeeedb_db_employee_1.setEmployee_fname(
db_employee_1.getDb_employee().getFIRST_NAME() );

        //@map:Copy RATE to Rate
        employeeedb_db_employee_1.setRate(
java.lang.Double.toString(
db_employee_1.getDb_employee().getRATE() ) );

        //@map:Copy LAST_UPDATE to Update_date
        employeeedb_db_employee_1.setUpdate_date(
db_employee_1.getDb_employee().getLAST_UPDATE().toString() );
    }
    //@map:Copy employeeedb_db_employee_1.marshallToString to
Text
    FileClient_1.setText(
employeeedb_db_employee_1.marshallToString() );

    //@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(
employeeedb_with_top_db_employee_1.getEmployee_no() ) );

//@map:Copy Employee_lname to Employee_Lname
insert_DB_1.getInsert_new_employee().setEmployee_Lname(
employeeedb_with_top_db_employee_1.getEmployee_lname() );

//@map:Copy Employee_fname to Employee_Fname
insert_DB_1.getInsert_new_employee().setEmployee_Fname(
employeeedb_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(
employeeedb_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(
employeeedb_with_top_db_employee_1.getUpdate_date() ) );
Table_OTD_1.getDB_EMPLOYEE().insertRow();

//Table_OTD_1.commit

```

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

The Update Operation

To perform an update operation on a table

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

```
//SalesOrders_with_top_SalesOrders_1.unmarshalFromString(Text)  
SalesOrders_with_top_SalesOrders_1.unmarshalFromString(  
    input.getText() );  
  
//SALES_ORDERS.update("SO_num =99")  
DB_sales_orders_1.getSALES_ORDERS().update( "SO_num ='01'" );  
  
//while  
while (DB_sales_orders_1.getSALES_ORDERS().next()) {  
  
//Copy SalesOrderNum to SO_num  
DB_sales_orders_1.getSALES_ORDERS().setSO_num(  
    SalesOrders_with_top_SalesOrders_1.getSalesOrderNum() );  
  
//Copy CustomerName to Cust_name  
DB_sales_orders_1.getSALES_ORDERS().setCust_name(  
    SalesOrders_with_top_SalesOrders_1.getCustomerName() );  
  
//Copy CustomerPhone to Cust_phone  
DB_sales_orders_1.getSALES_ORDERS().setCust_phone(  
    SalesOrders_with_top_SalesOrders_1.getCustomerPhone() );  
  
//SALES_ORDERS.updateRow  
DB_sales_orders_1.getSALES_ORDERS().updateRow();  
}  
//DB_sales_orders_1.commit  
DB_sales_orders_1.commit();  
  
//Copy "Update completed" to Text  
FileClient_1.setText( "Update completed" );  
  
//FileClient_1.write  
FileClient_1.write();  
}
```

The Delete Operation

To perform a delete operation on a table

- 1 Execute the **delete()** method.
In this example DELETE an employee.

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

5.5.2 Using Clobs

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

- Table OTD
- Prepared Statement OTD
- Procedure OTD

Inserting a Clob using a Table OTD

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

- 1 Invoke "Select" to get the ResultSet.

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

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

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

- 3 Set the values to the CLOB field.

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

- 4 Insert the row.

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

The Complete JCE Code Appears as:

```
public class TESTCLOB_JCE
{
    public com.stc.codegen.logger.Logger logger;
    public com.stc.codegen.alerter.Alerter alerter;

    public void receive(
com.stc.connector.appconn.file.FileTextMessage
        input, tTESTCLOB.TESTCLOBOTD TESTCLOB_1 )
    throws Throwable
    {
        //@map:TEST.select("")
        TESTCLOB_1.getTEST().select( "" );

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

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

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


Inserting a Clob using a Prepared Statement OTD

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

- 1 Set the values to the Prepared Statement parameter.

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

- 2 Execute the Prepared Statement

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

The Complete JCE Code Appears as:

```
public class CLOB_PREPSTAT_JCE
{
    public com.stc.codegen.logger.Logger logger;
    public com.stc.codegen.alerter.Alerter alerter;
    public void receive(
com.stc.connector.appconn.file.FileTextMessage
        input,insertClobPrepStat.InsertClobPrepStatOTD
InsertClobPrepStat_1 )
        throws Throwable
    {
        //@map:Copy Text to Param1
        InsertClobPrepStat_1.getInsertClobPrepStat().setParam1(
input.getText() );
        //@map:InsertClobPrepStat.executeUpdate
        InsertClobPrepStat_1.getInsertClobPrepStat().executeUpdate();
    }
}
```

Inserting a Clob using a Stored Procedure OTD

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

- 1 Set the values to the Stored Procedure OTD arguments.

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

- 2 Execute the Store procedure.

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

The Complete JCE Code Appears as:

```
public class TESTCLOB_STOREDPROC
{
    public com.stc.codegen.logger.Logger logger;
    public com.stc.codegen.alerter.Alerter alerter;
```

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

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

Updating a Clob using a Table OTD

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

- 1 Invoke "Select" to get the ResultSet.

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

- 2 Move to the to-be-updated row.

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

- 3 Set the values to the CLOB field.

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

- 4 Update the row.

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

The Complete JCE Code Appears as:

```
public class TESTCLOB_JCE
{
    public com.stc.codegen.logger.Logger logger;
    public com.stc.codegen.alerter.Alerter alerter;

    public void receive(
com.stc.connector.appconn.file.FileTextMessage
        input, tESTCLOB.TESTCLOBOTD TESTCLOB_1 )
    throws Throwable
    {
        //@map:TEST.select("")
        TESTCLOB_1.getTEST().select( "" );

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

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

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

```
}
```

Updating a CLOB using a Stored Procedure or Prepared Statement OTD

Updating a CLOB using a Stored Procedure or Prepared Statement is similar to the procedures for Inserting a CLOB. For more information, see [Inserting a Clob using a Prepared Statement OTD](#) on page 65 and [Inserting a Clob using a Stored Procedure OTD](#) on page 65.

Note: *When using a Prepared Statement, the 'ResultsAvailable()' method will always return true. Although this method is available, you should not use it with a 'while' loop. Doing so would result in an infinite loop at runtime and will stop all of the system's CPU. If it is used, it should only be used with the 'if' statement.*

Selecting a Clob using a Table OTD

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

- 1 Select desired Rows.

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

- 2 Scroll to the desired row.

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

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

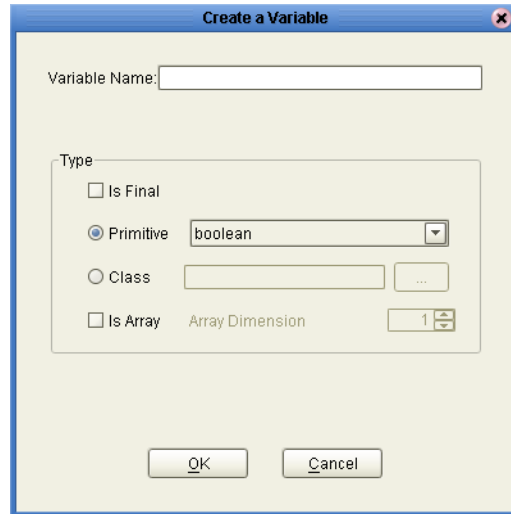
```
//@map:java.sql.Clob clobValue;  
java.sql.Clob clobValue;
```

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

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

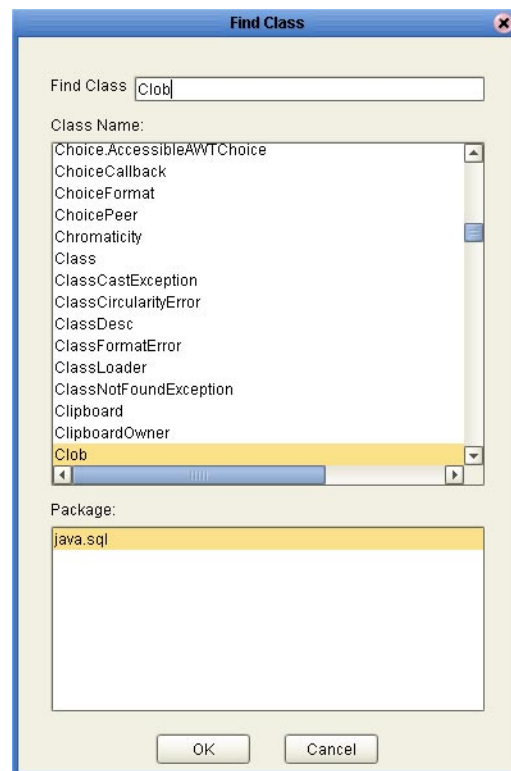
- A Open the **Create a Variable** window, as seen in Figure 43.

Figure 43 Create a Local Variable



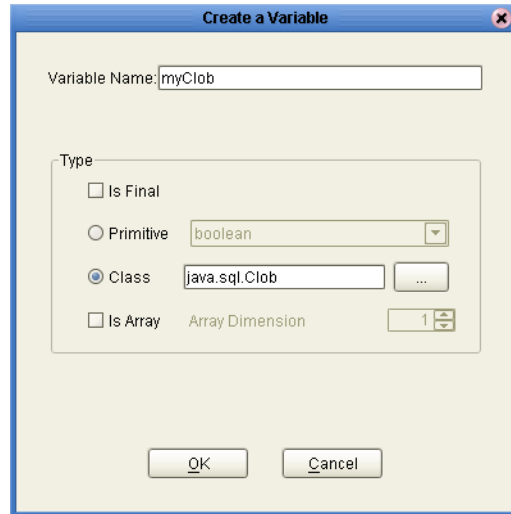
- B** Select the **Class** button and then click the ellipsis (...) button to search for a Class.
- C** In the **Find Class** window, locate the Clob class and click **OK**, as seen in Figure 44.

Figure 44 Find Class window



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

Figure 45 Create Local Variable java.sql.Clob



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

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

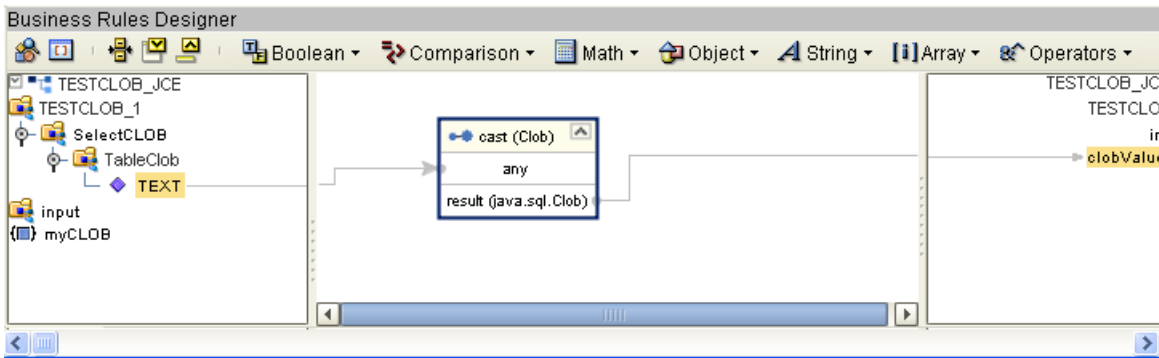
Figure 46 displays the Cast window.

Figure 46 Convert using the Cast() Method



Figure 47 displays assigning the value to the variable.

Figure 47 Assigning to the clobValue Variable



5 Declare the variable of type String.

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

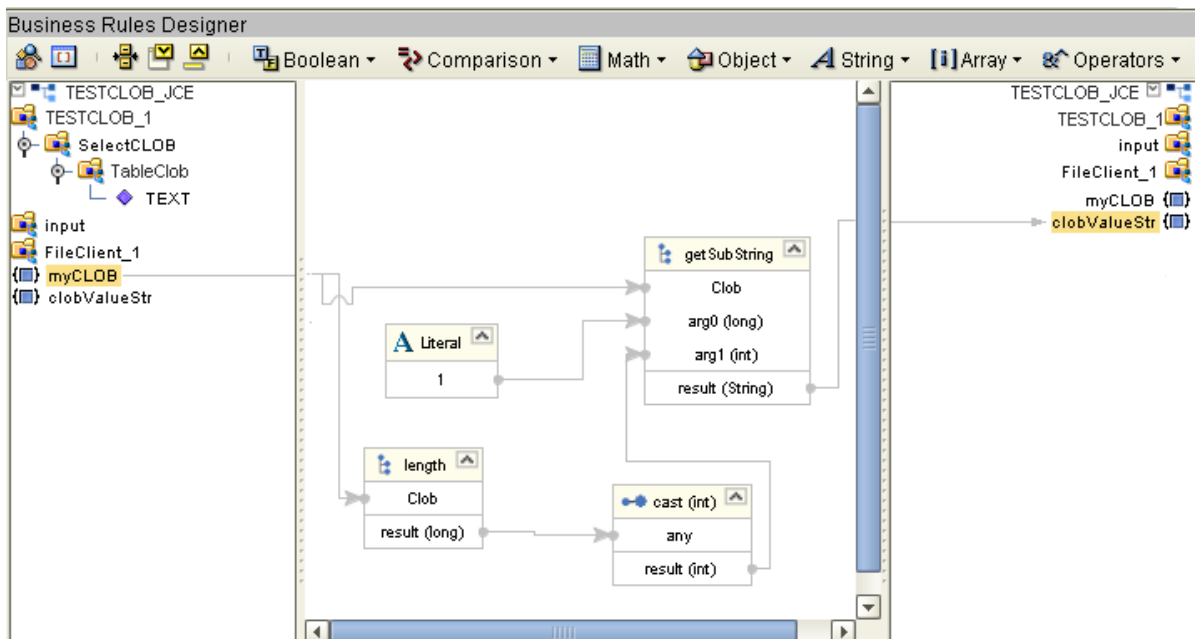
6 Fetch the Clob value to the String variable.

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

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

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

Figure 48 Copy the Clob Value to a String Variable



The Complete JCE Code Appears as:

```
public class TESTCLOB_JCE
{
    public com.stc.codegen.logger.Logger logger;

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

    public void receive(
com.stc.connector.appconn.file.FileTextMessage
        input, tTESTCLOB.TESTCLOBOTD TESTCLOB_1 )
    throws Throwable
    {
        //@map:TEST.select("")
        TESTCLOB_1.getTEST().select( "" );

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

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

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

        //@map:String clobValueStr;
        String clobValueStr;

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

Selecting a Clob using a Prepared Statement

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

- 1 Execute the query.

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

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

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

Note: The `getXXX()` method of the OTD resultSet returns the clob as `java.lang.Object`. This needs to be converted to `java.sql.Clob` before fetching the actual value from the Field. A local variable must be declared to facilitate conversion. For additional details, see the figures found in [Selecting a Clob using a Table OTD](#) on page 67.

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

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

4 Declare the variable of type String

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

5 Fetch the clob value to the String variable

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

The Complete JCE Code Appears as:

```
public class SELECTCLOB_PREPSTAT_JCE  
{  
  
    public com.stc.codegen.logger.Logger logger;  
  
    public com.stc.codegen.alerter.Alerter alerter;  
  
    public void receive(  
com.stc.connector.appconn.file.FileTextMessage  
        input,selectClobPrepStat.SelectClobPrepStatOTD  
        SelectClobPrepStat_1 )  
throws Throwable  
    {  
        //@map:SelectClobPrepStat.executeQuery  
        SelectClobPrepStat_1.getSelectClobPrepStat().executeQuery();  
  
        //while  
        while  
(SelectClobPrepStat_1.getSelectClobPrepStat().get$SelectClobPrepStatR  
esults().next()) {  
            //@map:java.sql.Clob clobValue;  
            java.sql.Clob clobValue;  
  
            //@map:Copy cast TEXT to java.sql.Clob to clobValue  
            clobValue = (java.sql.Clob)  
SelectClobPrepStat_1.getSelectClobPrepStat().get$SelectClobPrepStatR  
esults().getTEXT();  
  
            //@map:String clobValueStr;  
            String clobValueStr;  
  
            //@map:Copy SubString(1,cast clobValue.length to int) to  
            clobValueStr  
            clobValueStr = clobValue.getSubString( 1,(int)  
            clobValue.length() );  
        }  
    }  
}
```

5.5.3 The Stored Procedure

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

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

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

Executing Stored Procedures

The OTD represents the Stored Procedure “LookUpGlobal” with two parameters, an inbound parameter (INLOCALID) and an outbound parameter (OUTGLOBALPRODUCTID). These inbound and outbound parameters are generated by the Data Base 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(  
employee_db_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(  
employee_db_with_top_db_employee_1.getUpdate_date() ) );  
  
        //@map:Insert_new_employee.execute  
insert_DB_1.getInsert_new_employee().execute();  
  
        //@map:insert_DB_1.commit  
insert_DB_1.commit();  
  
        //@map:Copy "procedure executed" to Text  
FileClient_1.setText( "procedure executed" );  
  
        //@map:FileClient_1.write  
FileClient_1.write();  
    }  
}
```

Manipulating the ResultSet and Update Count Returned by Stored Procedure

The following methods are provided for using the ResultSet and Update Count, when they are returned by Stored Procedures:

- enableResultSetOnly
- enableUpdateCountsOnly
- enableResultSetandUpdateCounts
- resultsAvailable
- next
- getUpdateCount
- available

DB2 stored procedures do not return records as ResultSets, instead, the records are returned through output reference cursor parameters. Reference Cursor parameters are essentially ResultSets.

The **resultsAvailable()** method, added to the OTD, simplifies the whole process of determining whether any results, be it update Counts or ResultSets, are available after a stored procedure has been executed. Although JDBC provides three methods (**getMoreResults()**, **getUpdateCount()**, and **getResultSet()**) to access the results of a stored procedure call, the information returned from these methods can be quite confusing to the inexperienced Java JDBC programmer and they also differ between vendors. You can simply call **resultsAvailable()** and if Boolean true is returned, you can expect either a valid Update Count when **getUpdateCount()** is called and/or the next ResultSet has been retrieved and made available to one of the ResultSet nodes defined for the Stored Procedure OTD, when that node's **available()** method returns true.

Update Counts information which is returned from Stored Procedures is often insignificant. You should process returned ResultSet information and avoid looping through all of the Update Counts. The following three methods control exactly what information is returned from a stored procedure call. The **enableResultSetsOnly()** method, added to the OTD allows only ResultSets to be returned and thus every **resultsAvailable()** called only returns Boolean true if a ResultSet is available. Likewise, the **enableUpdateCountsOnly()** method causes **resultsAvailable()** to return true only if an Update Count is available. The default case of the **enableResultsetsAndUpdateCount()** method allows both ResultSets and Update Counts to be returned.

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

Collaboration usability for a Stored Procedure ResultSet

The Column data of the ResultSets can be dragged-and-dropped from their OTD nodes to the Business Rules. Below is a code snippet that can be generated by the Collaboration Editor:

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

```
        logger.info("JOB =  
"+getSPIn().getSpS_multi().getDbEmployee().getJOB());  
        logger.info("MGR =  
"+getSPIn().getSpS_multi().getDbEmployee().getMGR());  
        logger.info("HIREDATE =  
"+getSPIn().getSpS_multi().getDbEmployee().getHIREDATE());  
        logger.info("SAL =  
"+getSPIn().getSpS_multi().getDbEmployee().getSAL());  
        logger.info("COMM =  
"+getSPIn().getSpS_multi().getDbEmployee().getCOMM());  
        logger.info("DEPTNO =  
"+getSPIn().getSpS_multi().getDbEmployee().getDEPTNO());  
    }  
}
```

Note: *resultsAvailable() and available() cannot be indiscriminately called because each time they move ResultSet pointers to the appropriate locations.*

After calling "**resultsAvailable()**", the next result (if available) can be either a **ResultSet** or an **UpdateCount** if the default "**enableResultSetsAndUpdateCount()**" was used.

Because of limitations imposed by some DBMSs, it is recommended that for maximum portability, all of the results in a **ResultSet** object should be retrieved before OUT parameters are retrieved. Therefore, you should retrieve all **ResultSet(s)** and update counts first followed by retrieving the OUT type parameters and return values.

The following list includes specific **ResultSet** behavior that you may encounter:

- The method **resultsAvailable()** implicitly calls **getMoreResults()** when it is called more than once. You should not call both methods in your java code. Doing so may result in skipped data from one of the **ResultSets** when more than one **ResultSet** is present.
- The methods **available()** and **getResultSet()** cannot be used in conjunction with multiple **ResultSets** being open at the same time. Attempting to open more than one **ResultSet** at the same time closes the previous **ResultSet**. The recommended working pattern is:
 - ♦ Open one Result Set, **ResultSet_1** and work with the data until you have completed your modifications and updates. Open **ResultSet_2**, (**ResultSet_1** is now closed) and modify. When you have completed your work in **ResultSet_2**, open any additional **ResultSets** or close **ResultSet_2**.
- If you modify the **ResultSet** generated by the Execute mode of the Database Wizard, you need to assure the indexes match the stored procedure. By doing this, your **ResultSet** indexes are preserved.
- Generally, **getMoreResults** does not need to be called. It is needed if you do not want to use our enhanced methods and you want to follow the traditional JDBC calls on your own.

5.6 Editing Existing OTDs

A single OTD can consist of many Database objects. They can be a mixture of **Tables**, **Prepared Statements** and **Stored Procedures**. By using the Database OTD Wizard, the OTD Edit feature allows you to:

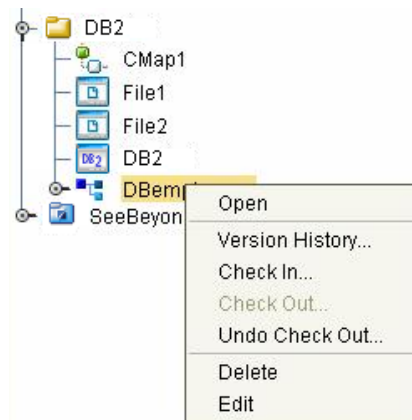
- Add or Remove **Table/Views**.
- Change data types by selecting a different one from a list.
- Add or Remove columns from a **Table** object.
- Add or Remove **Prepared Statement** objects.
- Edit **Prepared Statement** objects.
- Add or Remove **Stored Procedure** objects.
- Edit **Stored Procedure Resultsets**.

To Edit an Existing OTD

When a minor change is needed for a DB2 OTD, there is no need to rebuild it from scratch; instead, you can edit the OTD. To edit an OTD, complete the following steps:

- 1 In the Enterprise Explorer, right-click on the OTD. From the submenu, click **Edit** (see Figure 49). The Database Connection Information Wizard opens.

Figure 49 OTD Edit Menu Item



- 2 Connect to the DB2 database by entering the applicable information in the wizard. Once the connection is established, the Database Wizard opens, allowing you to make modifications to the OTD.
- 3 Once you have completed editing the OTD, click the **Finish** button to save the changes.

Caution: *Once the OTD has been edited, you must verify that the changes are reflected in the Collaboration so that no errors occur at runtime. For example, if during the edit process, you delete a database object that is included in a Collaboration, the Collaboration could fail at activation or run-time.*

When editing an OTD, you can connect to another instance of the database under the following conditions:

- The same type of DB2 database must be used. Because of incompatibility of certain features in the databases, switching between DB2 databases that run on z/OS, AS/400 and Windows/UNIX is not supported.
- The same version of the database should be used unless the newer version is compatible with the older version.
- Tables in the database must be defined with the same definition.
- The stored procedures must be identical.
- For tables/stored procedures built with 'qualified-name', the schema name for the tables/stored procedures must be identical in both database instances.

5.7 Alerting and Logging

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

Support for WebSphere Application Server

This section describes how to deploy an Enterprise Archive (EAR) file to the WebSphere™ Application Server. This includes information on installing the WebSphere Application Server interface and configuring the selected Application server to deploy the EAR file.

What's in this Appendix

- [“Uploading the Application Server Interface” on page 79](#)
- [“Creating an EAR File” on page 80](#)
- [“Deploying an EAR File” on page 81](#)
- [“Configuring the WebSphere Application Server” on page 81](#)

A.1 Uploading the Application Server Interface

To support the WebSphere Application Server, the following files must be uploaded to your system and installed:

- websphereintegserver.sar
- webspherejmsmessageserver.sar

For information on uploading and installing the selected .sar files see the *SeeBeyond ICAN Suite Installation Guide*.

A.2 Creating an EAR File

To create an EAR file from an eGate project, include the following steps during the creation of the project:

- 1 If you are using Topics or Queues in your project, make the following changes to the JMS Properties Sheet (accessed from the Connectivity Map). Set the properties to the following values:
 - ◆ Set the **JMS Client > Basic > Transaction Mode** property to **XA**
 - ◆ Set the **JMS Client > Basic > Run-as principal > Use for JMS connection** property to **false**
 - ◆ Set the **JMS Client > Advanced > Durability** property to **Nondurable**
 - ◆ Set the **JMS Client > Advanced > Security > Audit** property to **no**

Note: *The JMS Client > Basic > Run-as principal > Name property is limited to 12 characters*

- 2 During the creation of the project Environment do the following:
 - A Create the Logical Host in the Environment.
 - B From the Environment Explorer tree, right-click the logical host and select **New WebSphere Application Server** from the shortcut menu. The application server, **WebSphereSvr1**, is added to the Logical Host box and the Environment Explorer tree.
 - C If you are using JMS (Topics or Queues) you must also add the **New WebSphere JMS Server** to the Logical Host. From the Environment Explorer tree, right-click the logical host and select **New WebSphere JMS Server** from the shortcut menu. The WebSphere JMS message server, **WSMessageSvr1**, is added to the Logical Host box and the Environment Explorer tree.
- 3 During the creation of the Deployment Profile, do the following:
 - A If the service containing the Collaboration fails to be mapped to the WebSphere Application Server, drag and drop the Service to the WebSphere Application Server, **WebSphereSvr1**, in the Logical Host box.
 - B If any JMS Topics or Queues fail to be mapped to the WebSphere JMS Server, drag and drop the Topic or Queue to the WebSphere JMS Server, **WSMessageSvr1**, in the Logical Host box.
 - C Once all of the components have been mapped, click **Activate**. When prompted whether to apply the project to the logical host immediately, click **No**.
- 4 The project's EAR file is now available in the following location:

```
<ICAN>/repository/data/files/<environment name>/<logical host name>/<application server name> directory
```


where <ICAN> is the directory where the ICAN Suite is installed, <environment name> is the name of the project Environment, <logical host name> is the name of the logical host, and <application server name> is the name of the selected Application Server.

Note: For an eGate project EAR file to deploy to the WebSphere Application Server, the project must not contain any inbound eWays.

A.3 Deploying an EAR File

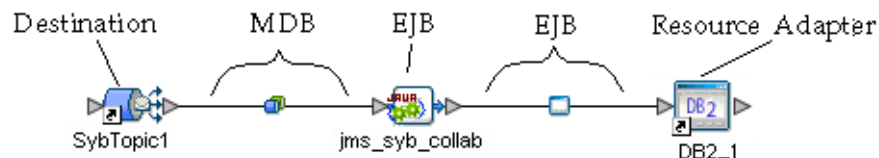
To deploy an EAR file using the WebSphere Application Server, you create the following components from the Application Server's Administrative Console:

- **Topics or Queues:** One Topic or Queue is needed for each Topic or Queue in the eGate project.
- **Connection Factories:** A Connection Factory enables JMS Clients to create JMS connections with predefined attributes. If the project contains Topics, a Topic Connection Factory is required. If it contains Queues a Queue Connection Factory is needed. If the project contains both Topics and Queues, create both a Queue and a Topic Connection Factory.

WebSphere also requires the following:

- **Listener Ports:** A listener port must be created for every MDB the project contains. The MDB is represented in the project's Connectivity Map as the connection between a **Topic or Queue** and a **Service** (see Figure 1).

Figure 1 Project/EAR File



A.4 Configuring the WebSphere Application Server

To deploy an EAR file in the WebSphere Application Server, start the Administrative Console, and do the following:

Creating the Topic or Queue

- 1 From the left pane of the WebSphere Administrative console, select **Resources** and click **WebSphere JMS Provider**.

- From the **WebSphere JMS Provider** window, under **Additional Properties** (shown in Figure 2), click **WebSphere Queue Destinations** or **WebSphere Topic Destinations**, depending on whether you are creating a topic or queue. For this example click **WebSphere Topic Destinations** to create a topic.

Figure 2 WebSphere Server Administrative Console - Additional Properties

Additional Properties	
WebSphere Queue Connection Factories	
WebSphere Topic Connection Factories	
WebSphere Queue Destinations	
WebSphere Topic Destinations	

- From the **WebSphere Topics Destinations** (or **Queue Destinations**) window click the **New** button.
- From the **Topic** or **Queue Destinations** configuration window, enter a name for the new Topic or Queue in the **Name** field. Enter the same name in the **JNDI Name** field. Enter the string value used to identify the Topic in the **Topic** field (for this example, **Topic1**) as displayed in Figure 3. Click **OK**.

Figure 3 WebSphere Server Administrative Console - Create a Topic

The screenshot shows the WebSphere Server Administrative Console interface. On the left is a navigation tree with categories like Servers, Applications, Resources, Security, Environment, System Administration, and Troubleshooting. The main area displays the configuration for a new Topic named 'Topic1' under the 'WebSphere JMS Provider > WebSphere Topic Destinations' path. A message at the top indicates that changes have been made to the local configuration and the server may need to be restarted. The configuration window includes a 'Configuration' tab and a 'General Properties' section with the following fields:

- Scope:** cells:LocalhostX260:nodes:LocalhostX260
- Name:** Topic1
- JNDI Name:** Topic1
- Description:** (empty text area)
- Category:** (empty dropdown)
- Topic:** Topic1
- Persistence:** APPLICATION DEFINED
- Priority:** APPLICATION DEFINED
- Specified Priority:** (empty text field)
- Expiry:** APPLICATION DEFINED
- Specified Expiry:** (empty text field) milliseconds

Buttons for 'Apply', 'OK', 'Reset', and 'Cancel' are located at the bottom of the configuration window.

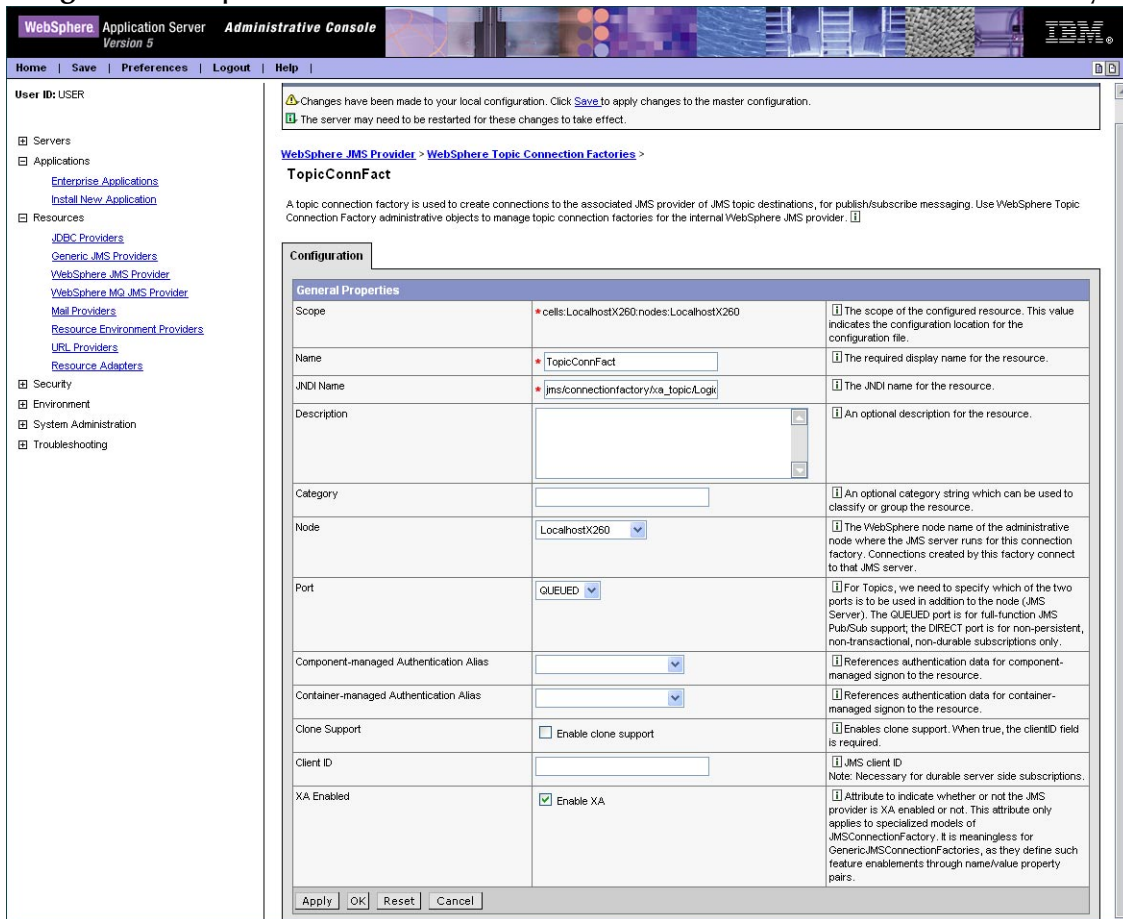
- From the **WebSphere Topic Destinations** window, click **WebSphere JMS Provider** to return to the JMS Provider window.

Creating a Connection Factory

- 1 From the **WebSphere JMS Provider** window, under **Additional Properties**, click **WebSphere Topic Connection Factories** (or Queue Connection Factories if you created a Queue).
- 2 From the **WebSphere Topic Connection Factories** window, click the **New** button.
- 3 From the **WebSphere Topic Connection Factories** configuration window, enter the Topic Connection Factory name in the **Name** field (for this example, **TopicConnFact**).
- 4 Enter the Connection Factory JNDI Name in the **JNDI Name** field. The pattern for the JNDI Name is **jms/connectionfactory/<xa_topic or xa_queue (just topic or queue if xa is not selected as the Transaction Mode)>/<Logical Host name>_<JMS Message Server name (from the Environment)>**.

For this example, the Connection Factory JNDI name is **jms/connectionfactory/xa_topic/LogicalHost1_WLMessageSvr1**
- 5 Make sure that **XA Enabled** is selected (see Figure 4). Click **OK**.

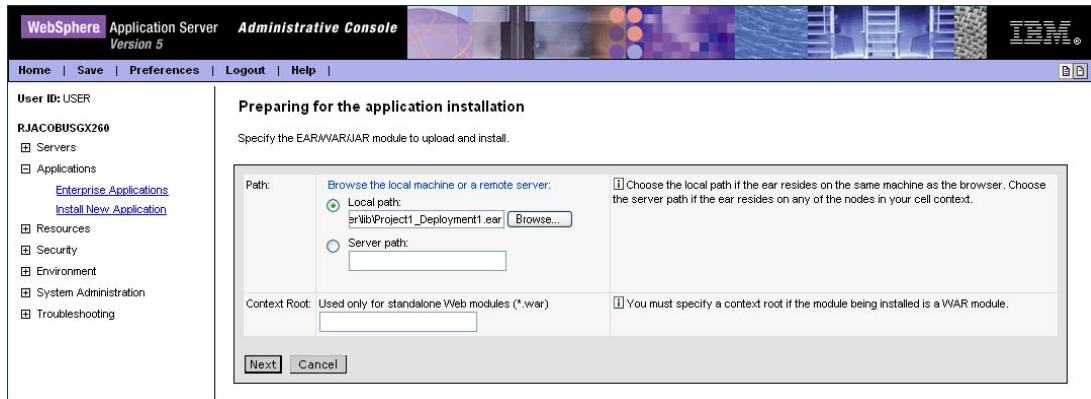
Figure 4 WebSphere Server Administrative Console - Create Connection Factory



Installing the Application

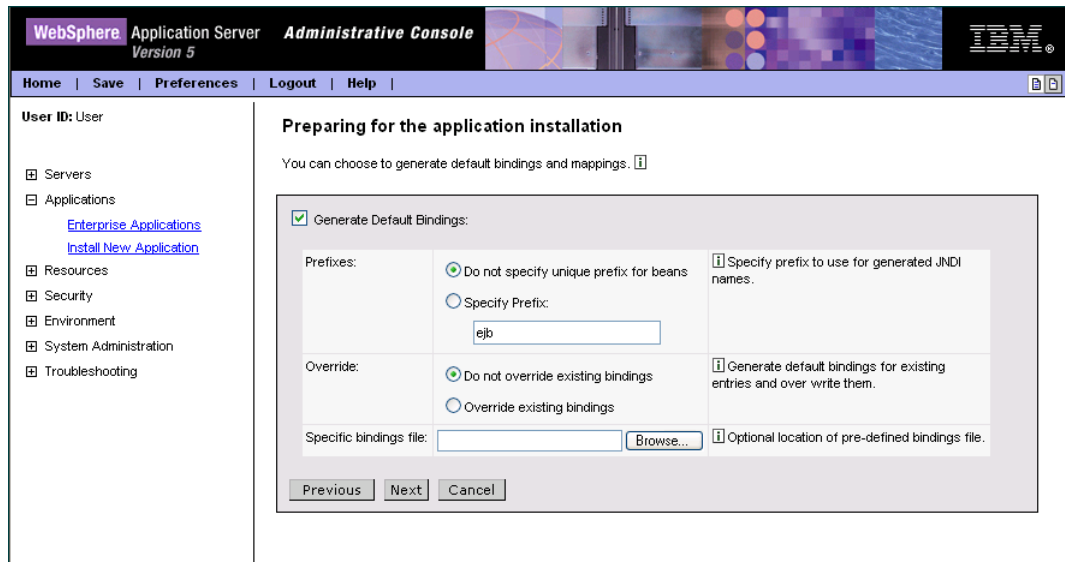
- 1 From the left pane of the WebSphere Administrative console, select **Applications**, and click **Install New Application**.
- 2 From the **Preparing for the application installation** window, select **Local path** or **Server path** and click **Browse**. Locate and select the appropriate EAR file (see Figure 5). Click **Next**.

Figure 5 WebSphere Server Administrative Console - Install New Application



- 3 From the next window select **Generate Default Bindings** (see Figure 6). Click **Next**.

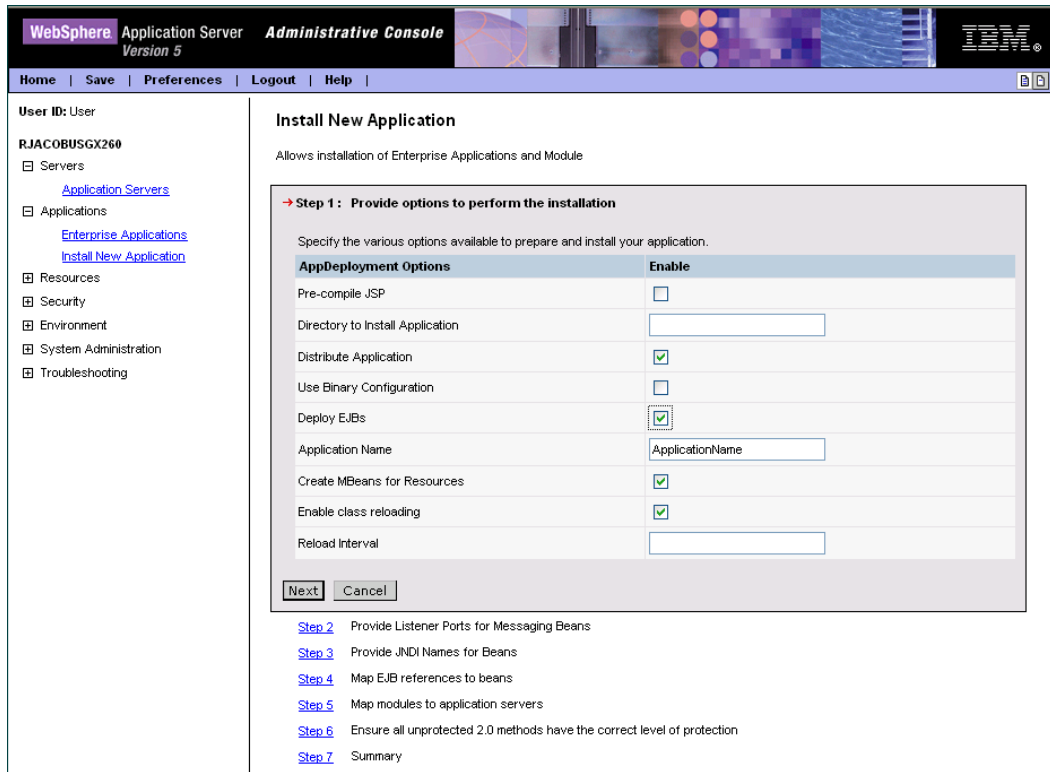
Figure 6 WebSphere Server Administrative Console - Generate Default Bindings



Note: The following application installation steps may differ depending on the nature of the EAR file.

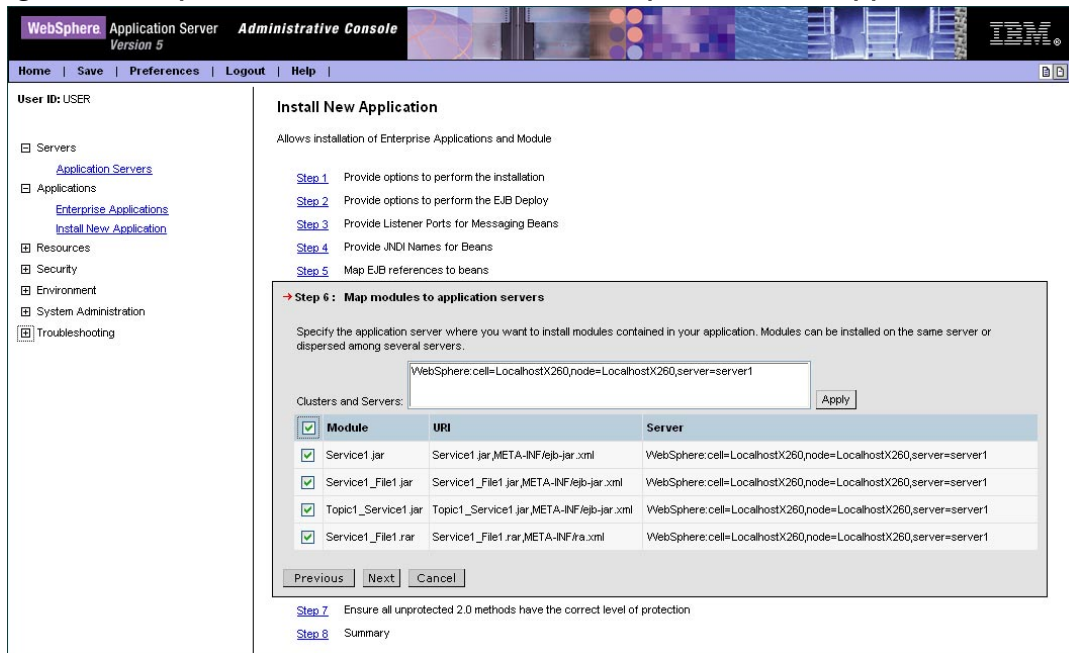
- 4 From the **Step 1: Provide options to perform the installation** window, enter a name for the application and select **Deploy EJBs** and **Enable Class Reloading** (see [Figure 7 on page 85](#)). Click **Next**.

Figure 7 WebSphere Administrative Console - Provide options to perform the installation



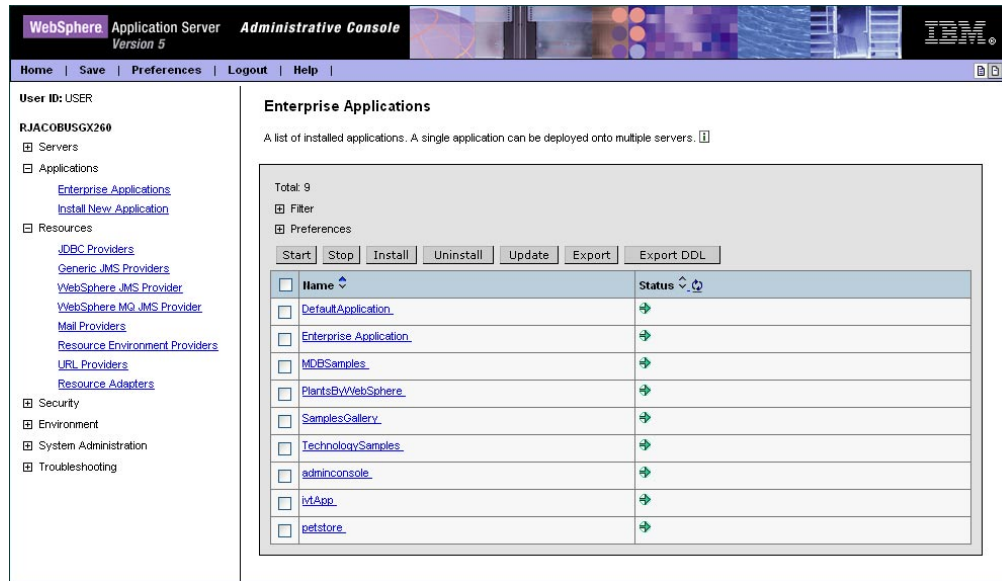
- From the **Step 3: Provide Listener Ports for Messaging Beans** window, make a note of the name that you used for the Listener Port. This Listener Port name will be used later in step 12. Proceed to the **Step 6: Map modules to application servers** window.
- From the **Step 6: Map modules to application servers** window, make a note of the Reference Binding field value (save this name as it appears). This name will be used in step 12-G.
- From the **Step 6: Map modules to application servers** window, click the Module checkbox to select all of the modules. (see [Figure 8 on page 86](#)). Click **Next**.

Figure 8 WebSphere Administrative Console - Map Modules to Application Servers



- 8 From the **Step 7: Ensure all unprotected 2.0 methods have the correct level of protection** window, click the Module checkbox to select all of the modules. Click **Next**.
- 9 From the **Step 8: Summary** window, review the selected values and click **Finish**.
- 10 WebSphere begins installing the application. This can take several minutes. When the application installs successfully, click **Save to Master Configuration**.
- 11 To review the application's status or start or stop the application, from the right pane of the Administrative Console, click **Applications**, and click **Enterprise Applications** (see [Figure 9](#) on page 87).

Figure 9 WebSphere Administrative Console - Enterprise Application



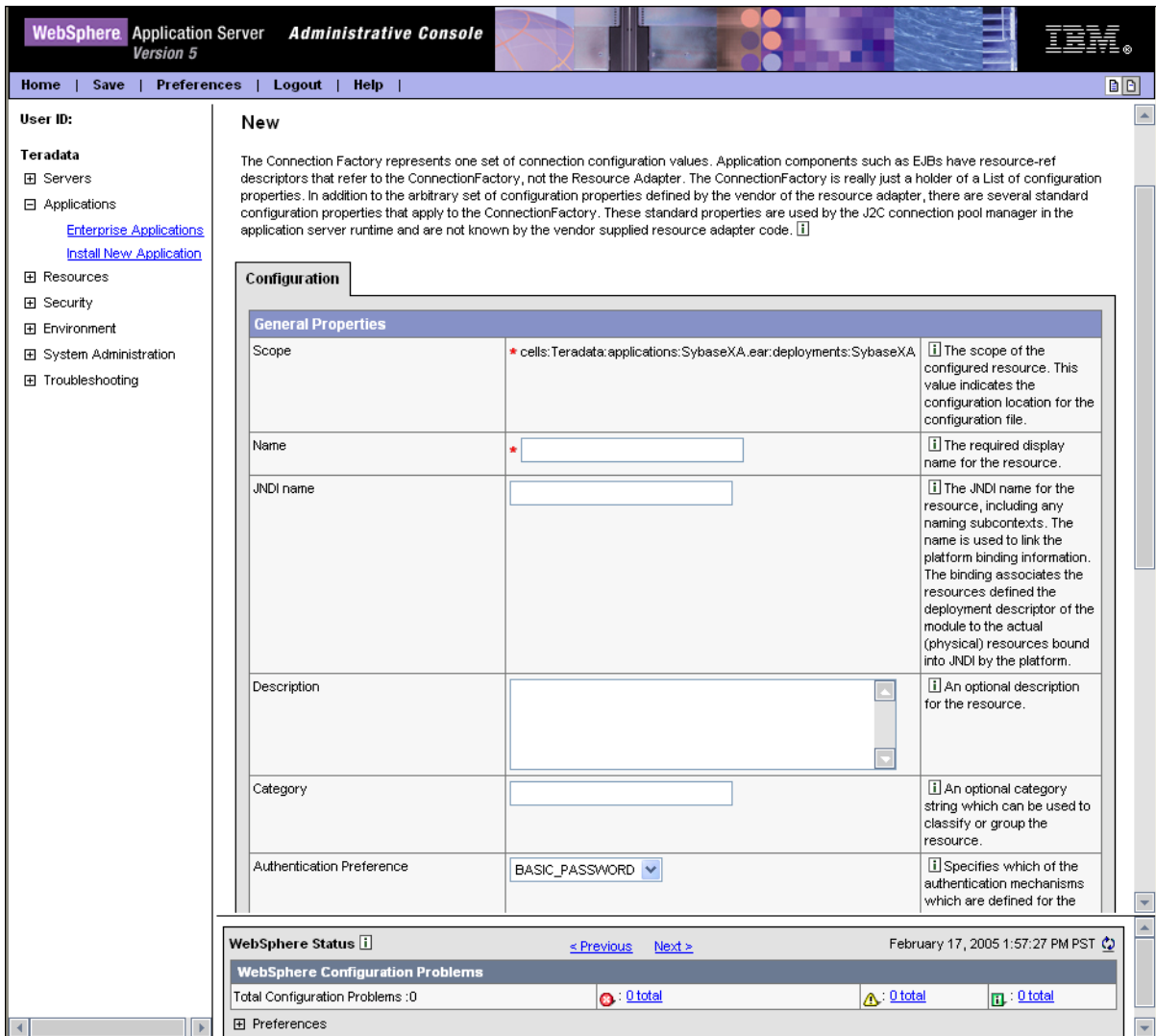
- 12 Complete the following steps to configure the J2C Connection Factories:
 - A From the Enterprise Applications window, select your application.
 - B From the Related Items box, click **Connector Modules**.
 - C From the Connector Modules window, click the deployed RAR file.
 - D From the deployed RAR file window's **Additional Properties** box, click **Resource Adapter** (see Figure 9).

Figure 10 WebSphere Administrative Console - Resource Adapter

Additional Properties	
Resource Adapter	A resource adapter is created for every connector module in the application as a part of application installation. This resource adapter can be used to further create Connection Factories that can be used to bind resources defined in the application.
View Deployment Descriptor	View the Deployment Descriptor

- E From the Resource Adapter window, Additional Properties box, click **J2C Connection Factories**. The J2C Connection Factories window appears.
- F Click the **New** button to create a new **J2C Connection Factory** (see [Figure 11 on page 88](#)).

Figure 11 WebSphere Administrative Console - J2C Con




- G From the **J2C Connection Factory's New** window, do the following:
 - ◆ Enter the **Name** for the new J2C Connction Factory.
 - ◆ Enter the **JNDI Name** for the new J2C Connction Factory. This is the same name that you saved in [step 6 on page 85](#) (This name can also be found as the **res-ref-name** in the **ejb-jar.xml** file)
 - ◆ Set **Mapping-Configuration Alias** to **DefaultPrincipleMapping**
- H Click **OK** to create the new J2C Connection Factory.










- 13 Create a Listener Port by completing the following steps:
 - A From the left pane of the WebSphere Application Server Administrative Console, expand Servers and click **Application Servers**.
 - B From Application Servers click **server1**.
 - C From the Additional Properties box, of the server1 window, click **Message Listener Service**.
 - D From the Additional Properties box, of the Message Listener Service window, click **Listener Ports**.
 - E From the Listener Ports window click **New** to create a new Listener Port.
 - F In the Configuration Tab, of the new Listener Port being created, fill in the following required fields (see Figure 12):
 - ♦ **Name** (Use the name of the Listener Port from Step 5)
 - ♦ **Initial State**
 - ♦ **Connection Factory JNDI Name**
 - ♦ **Destination JNDI Name**
 - G Click **OK** to create the Listener Port.

Figure 12 WebSphere Administrative Console - Creating a Listener Port

[Application Servers](#) > [server1](#) > [Message Listener Service](#) > [Listener Ports](#) >

New

Listener ports for Message Driven Beans to listen upon for messages. Each port specifies the JMS Connection Factory and JMS Destination that an MDB, deployed against that port, will listen upon. 

Runtime		Configuration
General Properties		
Name	* Topic1_jms_collabPort	 Name of the listener port
Initial State	* Started 	 The execution state requested when the server is first started.
Description		 A description of the listener port, for administrative purposes
Connection factory JNDI name	* jms/connectionfactory/xa_topicLogi	 The JNDI name for the JMS connection factory to be used by the listener port, for example, jms/connFactory1.
Destination JNDI name	* Topic1	 The JNDI name for the destination to be used by the listener port, for example, jms/destn1.
Maximum sessions	1	 The maximum number of concurrent JMS server sessions used by a listener to process messages, in the range 1 through 2147483647.
Maximum retries	5	 The maximum number of times that the listener tries to deliver a message before the listener is stopped, in the range 0 through 2147483647.
Maximum messages	1	 The maximum number of messages that the listener can process in one JMS server session, in the range 0 through 2147483647.
<input type="button" value="Apply"/> <input type="button" value="OK"/> <input type="button" value="Reset"/> <input type="button" value="Cancel"/>		

Index

A

Add Prepared Statements 41

B

book 79

Building an eWay Project 45

Business Process 46

C

ClassName 18, 22, 26

Clobs

 Inserting a Clob using a Prepared Statement
 OTD 65

 Inserting a Clob using a Stored Procedure OTD
 65

 Inserting a Clob using a Table OTD 64

 Selecting a Clob using a Prepared Statement 71

 Selecting a Clob using a Table OTD 67

 Updating a Clob using a Stored Procedure 67

 Updating a Clob using a Table OTD 66

 Using 64

Collaboration usability 75

CollectionID 29

CollectionId 24

Common DataType Conversions 58

configuring eWay connections 14

Configuring the eWay

 creating 14

Configuring the eWays 57

Connect to Database 33

Creating an External Environment 58

D

Data Conversions 58

Data Types

 Conversions 58

Database Connection 33

Database Objects 34

DatabaseName 16, 20

Delete Operation 47, 55

 Table OTD 63

Delimiter 21, 29

Deploying a Project 58

Description 18, 21, 26, 29

driver class, JDBC 18, 26

DriverProperties 21, 30

E

EAR file

 deploying an EAR file 81

eGate Sample Project 56

eInsight Engine and eGate Components 45

Environment Properties

 DatabaseName 16, 20

 Delimiter 21, 29

 Description 21, 29

 DriverProperties 21, 30

 Password 21, 30

 PortNumber 22, 30

 ServerName 22, 31

 User 22, 31

Environment Property Settings

 CollectionID 29

 DatabaseName 16, 20

 Delimiter 21, 29

 Description 21, 29

 DriverProperties 21, 30

 LocationName 30

 Password 16, 21, 30

 PortNumber 16, 22, 30

 ServerName 17, 31

 User 17, 22

G

Generating ResultSet nodes 40

I

Importing the Sample Project 46

Inbound Environment Properties

 DatabaseName 16

 Password 16

 PortNumber 16

 ServerName 17

 User 17

Inbound Properties

 CollectionId 24

 LocationName 24

 Password 24

 PollMilliseconds 15, 23

 PortNumber 25

 PreparedStatement 15, 23

Index

- ServerName 25
- User 25
- index
 - book 79
- InitialPoolSize 18, 27
- Insert Operation 47, 52
 - Performing on a table 62

J

- JDBC
 - driver class 18, 26

L

- LocationName 24, 30
- LoginTimeOut 18, 27

M

- MaxIdleTime 18, 27
- MaxPoolSize 19, 27
- MaxStatements 19, 27
- MinPoolSize 19, 28

N

- NetworkProtocol 19, 28

O

- Object Type Definition (OTD)
 - Editing an Existing OTD 77
- Operation
 - Delete 47, 55
 - Insert 47, 52
 - SelectAll 47, 48
 - SelectMultiple 47, 49
 - SelectOne 47, 51
 - Update 47, 53
- OTD Wizard
 - Add Prepared Statements 41
 - Database Connection 33
 - Select Database Objects 34
 - Select Table/Views 35
 - Specify the OTD Name 43
- OTDs
 - Tables, Views, Stored Procedures, and Prepared Statements 60
- Outbound Environment Properties
 - DatabaseName 20
 - Delimiter 21, 29
 - Description 21, 29

- DriverProperties 21, 30
- LocationName 30
- Password 21, 30
- PortNumber 22, 30
- ServerName 31
- User 22
- Outbound Properties
 - ClassName 18, 26, 27
 - Description 18, 26
 - DriverProperties 21
 - InitialPoolSize 18, 27
 - LoginTimeOut 18, 27
 - MaxIdleTime 18, 27
 - MaxPoolSize 19, 27
 - MaxStatements 19, 27
 - MinPoolSize 19, 28
 - NetworkProtocol 19, 28
 - PropertyCycle 19, 28
 - RoleName 20, 28
 - ServerName 22
 - User 31

P

- Packages 34
- Password 16, 21, 24, 30
- PollMilliseconds 15, 23
- PortNumber 16, 22, 25, 30
- Prepared Statements 15, 23, 41, 60
- Property settings, Environment
 - DatabaseName 16, 20
 - Delimiter 21, 29
 - Description 21, 29
 - DriverProperties 21, 30
 - Password 21, 30
 - PortNumber 22, 30
 - ServerName 22, 31
 - User 22, 31
- Property settings, Inbound
 - CollectionId 24
 - LocationName 24
 - Password 24
 - PollMilliseconds 15, 23
 - PortNumber 25
 - PreparedStatement 15, 23
 - ServerName 25
 - User 25
- Property settings, Outbound
 - ClassName 18, 26
 - Description 18, 26
 - DriverProperties 21
 - InitialPoolSize 18, 27
 - LoginTimeOut 18, 27
 - MaxIdleTime 18, 27

Index

- MaxPoolSize 19, 27
- MaxStatements 19, 27
- MinPoolSize 19, 28
- NetworkProtocol 19, 28
- PropertyCycle 19, 28
- RoleName 20, 28
- ServerName 22
- User 31
- PropertyCycle 19, 28

Q

- Query Operation
 - Performing on a table 61

R

- ResultSet 75
 - Behavior 76
 - Returned by Stored Procedure 74
- Resultset Nodes 40
- RoleName 20, 28
- Running the Sample 58

S

- Sample Project
 - Configuring the eWays 57
 - Creating and External Environment 58
 - Deploying a Project 58
 - eGate 56
 - Importing 46
 - Running the Sample 58
- Scope
 - ADABAS Natural eWay 9
- Select Database Objects 34
- Select Procedures 39
- Select Table/Views 35
- Select Wizard Type 32
- SelectAll Operation 47, 48
- SelectMultiple Operation 47, 49
- SelectOne Operation 47, 51
- ServerName 22, 25, 31
- Servename 17
- Specify the OTD Name 43
- Stored Procedures 60, 72
 - Collaboration usability for a ResultSet 75
 - Executing 73
 - Manipulating the ResultSet and Update Count 74

T

- Table/Views 35
- Tables 60

U

- Update 47
- Update Operation 53
 - Table OTD 63
- User 17, 22, 25, 31
- Using Clobs 64

V

- Views 60

W

- WebLogic and WebSphere Server Support 11
- WebSphere Application Server support 8, 79