SeeBeyond ICAN Suite

e*Way Intelligent Adapter for Oracle User's Guide

Release 5.0.5 for Schema Run-time Environment (SRE)

Java Version

Version 20051011104532.

Contents

Chapter 1

Introduction

This document describes how to install and configure the e*Way Intelligent Adapter for Oracle.

This chapter includes:

1.1 Overview

The Oracle e*Way enables the e*Gate system to exchange data with external Oracle databases. This document describes how to install and configure the Oracle e*Way.

The Oracle e*Way uses the Java library to issue SQL statements to interact with Oracle databases.

1.1.1 Components

The following components comprise the Java-enabled version of the Oracle e*Way:

- **e*Way Connections:** The database e*Way Connections provide access to the information necessary for connecting to a specified external system.
- **stcjdbcx.jar**: Contains the logic required by the e*Way to interact with the external databases.

A complete list of installed files appears in **Files created by the installation** on page 14.

1.2 Operational Overview

The Java-enabled version of the Oracle e*Way uses Java Collaborations to interact with one or more external databases. By using the Java Collaboration Service it is possible for e*Gate components—such as e*Way Intelligent Adapters (e*Ways) and Business Object

Brokers (BOBs)—to connect to external databases and execute business rules written entirely in Java.

An e*Gate component is defined as *Java-enabled* based on the selection of the Java Collaboration Service in the Collaboration Rule setup. For more information on the Java Collaboration Service, see **"The Java Collaboration Service" on page 25**.

1.3 Supported Operating Systems

The Oracle e*Way is available on the following operating systems:

- Windows 2000, Windows XP, and Windows Server 2003
- HP Tru64 V5.1A and V5.1B
- HP-UX 11.0, 11i (PA-RISC), and 11i v2.0 (11.23)
- IBM AIX 5L versions 5.1, 5.2, and 5.3
- Red Hat Enterprise Linux AS 2.1 (Intel x86)
- Red Hat Linux 8 (Intel x86)
- Sun Solaris 8 and 9
- SuSE Linux Enterprise Server 8 (Intel x86)
- Japanese Windows 2000, Windows XP, and Windows Server 2003
- Japanese HP-UX 11.0, 11i (PA-RISC), and 11i v2.0 (11.23)
- Japanese IBM AIX 5L versions 5.1, 5.2, and 5.3
- Japanese Sun Solaris 8 and 9
- Korean Windows 2000, Windows XP, and Windows Server 2003
- Korean HP-UX 11.0, 11i (PA-RISC), and 11i v2.0 (11.23)
- Korean IBM AIX 5L versions 5.1 and 5.2
- Korean Sun Solaris 8 and 9

Note: *XA support is provided for Oracle 9i.*

1.4 System Requirements

To use the Oracle e*Way, you need the following:

- An e*Gate Participating Host
- A TCP/IP network connection.

1.4.1 Host System Requirements

The external system requirements are different for a GUI host machine—specifically a machine running the ETD Editor and the Java Collaboration Editor GUIs—versus a participating host which is used solely to run the e*Gate schema.

GUI Host Requirements

To enable the GUI editors to communicate with the external system, the following items must be installed on any host machines running the GUI editors:

- The Oracle client library: Oracle8i with patch 8.1.7.6 or 9i release 9.2.0 installed. The Oracle client library must be installed on a Windows operating system to utilize the build tool.
- Microsoft Data Access Components (MDAC) RTM version 2.6 or greater. This component is included in the e*Gate installation.
- You will need to download the Oracle OLE DB Version 8.1.7.2.1 or later. This driver is needed by the ETD Editor GUI for accurate discovery of stored procedures during the construction of database ETD's. You can download the driver from the Oracle OTN website at:

  http://www.oracle.com/technology/software/index.html

If the GUI host machine will also be executing the Oracle e*Way, the host machine must also meet the **"Participating Host Requirements" on page 12**.

Participating Host Requirements

The Oracle e*Way Installation program installs the type 4 Oracle JDBC 9.0.1.3.0 driver required to connect to the external Oracle database. This driver is appropriate for most Oracle implementations. However, for implementations using protocols other than a type 4 driver, the appropriate client drivers must be installed. These client drivers can be found on the Oracle client installation disc.

A complete list of drivers from third party vendors is also available at:

http://industry.java.sun.com/products/jdbc/drivers

1.5 External System Requirements

The Oracle e*Way supports the following external systems:

- Oracle version 8i with patch 8.1.7.6 or 9i release 9.2.0

Chapter 2

Installation

This chapter describes how to install the Oracle e*Way.

This Chapter Includes:

Note: *Open and review the Readme.txt for any additional requirements prior to installation. The Readme.txt is located on the Installation CD_ROM .*

2.1 Installing the Oracle e*Way on Windows

2.1.1 Pre-installation

1. Exit all Windows programs before running the setup program, including any anti-virus applications.
2. You must have Administrator privileges to install this e*Way.

2.1.2 Installation Procedure

To install the Oracle e*Way on a Windows operating system

1. Log in as an Administrator on the workstation on which you want to install the e*Way.
2. Insert the e*Gate installation CD-ROM into the CD-ROM drive.
3. If the CD-ROM drive's "Autorun" feature is enabled, the setup application should launch automatically; skip ahead to step 4. Otherwise, use Windows Explorer or the Control Panel's **Add/Remove Applications** feature to launch the file **setup.exe** on the CD-ROM drive.

The InstallShield setup application launchs. Follow the on-screen instructions to install the e*Way.

Be sure to install the e*Way files in the suggested "client" installation directory. The installation utility detects and suggests the appropriate installation directory. **Unless**

you are directed to do so by SeeBeyond support personnel, do not change the suggested "installation directory" setting.

2.2 Installing the Oracle e*Way on UNIX

2.2.1 Pre-installation

You do not require root privileges to install this e*Way. Log in under the user name that you wish to own the e*Way files. Be sure that this user has sufficient privileges to create files in the e*Gate directory tree.

2.2.2 Installation Procedure

To install the Oracle e*Way on a UNIX system

1. Log in on the workstation containing the CD-ROM drive, and insert the CD-ROM into the drive.
2. If necessary, mount the CD-ROM drive.
3. At the shell prompt, type

   cd /cdrom
4. Start the installation script by typing

   setup.sh
5. A menu of options will appear. Select the "install e*Way" option. Then follow any additional on-screen directions.

Be sure to install the e*Way files in the suggested "client" installation directory. The installation utility detects and suggests the appropriate installation directory. **Unless you are directed to do so by SeeBeyond support personnel, do not change the suggested "installation directory" setting.**

2.3 Files/Directories Created by the Installation

The Oracle e*Way installation process installs the following files within the e*Gate directory tree. Files are installed within the "egate\client" tree on the Participating Host and committed to the "default" schema on the Registry Host.

Table 1 Files created by the installation

e*Gate Directory	File(s)
Classes\	stcjdbcx.jar
configs\oracle	oracle.def

Table 1 Files created by the installation

e*Gate Directory	File(s)
etd\	oracle.ctl dbwizard.ctl
etd/oracle/	Com_stc_jdbcx_oraclecfg.java Com_stc_jdbcx_oraclecfg.xsc
ThirdParty\sun\classes	jdbc2_0-stdext.jar
ThirdParty\oracle\classes\	classes12.zip
ThirdParty\merant\classes	DGbase.jar

Chapter 3

Connectivity

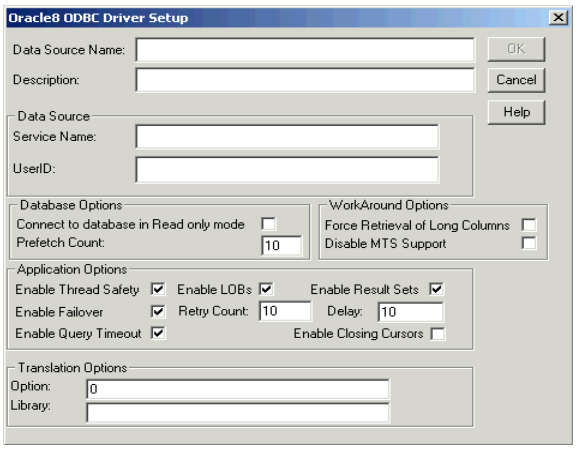
Before you can configure your Oracle e*Way, you need to test the connectivity. Follow the instructions provided to assure connectivity.

3.1 Registering the DataSource

To register your datasource, do the following:

1. Install the Oracle ODBC drivers. To do this, follow the installation instructions provided with the drivers.
2. On the task bar, click the **Start** button, and then click **Control Panel**.
3. Click **Administrative Tools**.
4. Click **Data Sources (ODBC)**.
5. In the ODBC Data Source Administrator window, click the **User DSN** tab.
6. Click **Add**.
7. Select **Oracle ODBC** and click **Finish**.
8. In the **OracleX ODBC Driver Setup** window enter the **Data Source Name**, **Description**, **Service Name**, **UserID**, and any additional system information for your Oracle installation. See Figure 1.

Figure 1 OracleX ODBC Driver Setup



9 Click on the **OK**.

Chapter 4

e*Way Connection Configuration

This chapter describes how to configure the Oracle e*Way Connections.

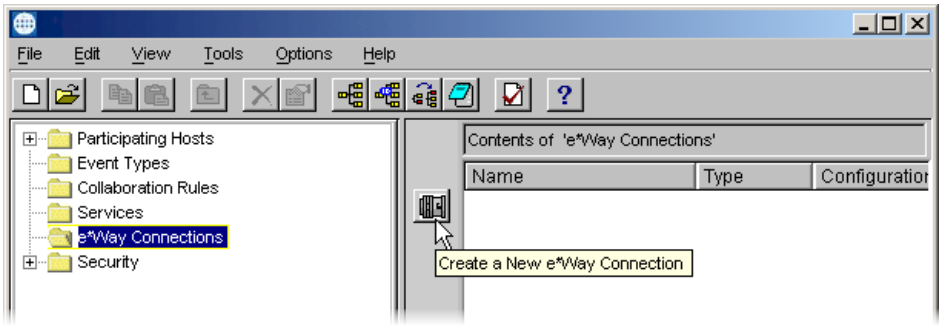
4.1 Creating e*Way Connections

The e*Way Connections are created and configured in the Schema Designer.

To create and configure the e*Way Connections

1 In the Schema Designer's Component editor, select the e*Way Connections folder.

Figure 2 The e*Way Connections Folder

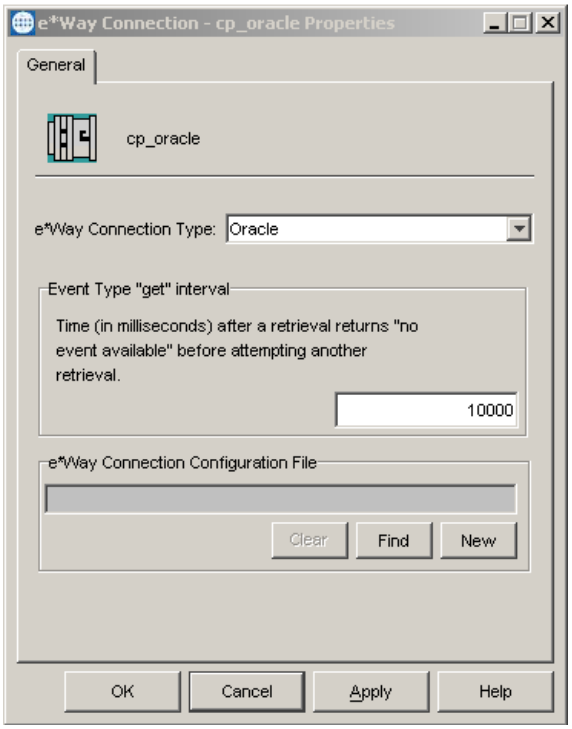


2 On the Palette, click the **New e*Way Connection** icon.

3 The **New e*Way Connection Component** dialog box opens. Enter a name for the e*Way Connection and click **OK**.

4 Double-click the new e*Way Connection to open the e*Way Connection Properties dialog box. See Figure 3.

Figure 3 e*Way Connection Properties Dialog Box



5 From the e*Way Connection Type drop-down list, select **Oracle**.

6 Enter the **Event Type "get" interval** in the dialog box provided.

7 Click **New** to create a new e*Way Connection Configuration File.

The e*Way Connection Configuration File Editor appears.

The e*Way Connection configuration file parameters are organized into the following sections:

- DataSource
- Connector

4.1.1 DataSource Settings

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

class

Description

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

Required Values

A valid class name.

The default is **oracle.jdbc.pool.OracleConnectionPoolDataSource**.

Additional Information

Use the **oracle.jdbc.xa.client.OracleXADataSource** class for XA compliant implementations.

To use XA in Oracle9i, your Database Administrator needs to GRANT SELECT ON DBA_PENDING_TRANSACTIONS TO PUBLIC.

Note: *XA functionality is not supported on Oracle 8i databases. For more information on implementing the Oracle e*Way in an XA compliant environment, see the e*Gate Integrator User's Guide.*

DriverType

Description

Specifies the JDBC driver type for Oracle. All other JDBC drivers are ignored.

Oracle implicitly issue a commit statement even if auto commit is set to false and no explicit commit or rollback is executed. See the Oracle JDBC Developer's Guide and Reference.

Any DDL operation, such as CREATE or ALTER, always includes an implicit COMMIT. If auto-commit mode is disabled, this implicit COMMIT does not only commit the DDL statement, but also any pending DML operations that had not yet been explicitly committed or rolled back.

Required Values

A valid driver type name.

The default is "**thin"**.

ServerName

Description

Specifies the host name of the external database server.

Required Values

Any valid string.

PortNumber

Description

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

Required Values

A valid port number. The default is 1521.

DatabaseName

Description

Specifies the name of the database instance.

Required Values

Any valid string.

user name

Description

Specifies the user name the e*Way uses to connect to the database.

Required Values

Any valid string.

password

Description

Specifies the password used to access the database.

Required Values

Any valid string.

timeout

Description

Specifies the login timeout in seconds.

Required Values

Any valid string. The default is 300 seconds.

4.1.2 Connector Settings

The Connector settings define the high level characteristics of the e*Way Connection.

Connector type

Description

Specifies the type of e*Way Connection. The current available type for JDBC connections is **DB**.

Required Values

The default is **DB**.

class

Description

Specifies the class name of the JDBC connector object.

Required Values

The default is **com.stc.eways.jdbcx.DbConnector**.

transaction mode

This parameter specifies how a transaction should be handled.

- **Automatic** — e*Gate takes care of transaction control and users should not issue a commit or rollback. If you are using XA, you must set your **connection establishment mode** and your **transaction mode** both to **Automatic**.
- **Manual** — You must manually take care of transaction control by issuing a commit or rollback.

Required Values

The required values are **Automatic** or **Manual**. The default is set to **Automatic**.

Mixing XA-Compliant and XA-Noncompliant e*Way Connections

A Collaboration can be XA-enabled if and only if all its sources and destinations are XA-compliant e*Way Connections. However, XA-related advantages can accrue to a Collaboration that uses one (and only one) e*Way Connection that is transactional but not XA-compliant—in other words, it connects to exactly one external system that supports commit/rollback (and is thus transactional) but does not support two-phase commit (and is thus not XA-compliant). Please see the *e*Gate Integrators User's Guide* for usage and restrictions.

connection establishment mode

This parameter specifies how a connection with the database server is established and closed.

- **Automatic** indicates that the connection is automatically established when the collaboration is started and keeps the connection alive as needed. If you are using XA, you must set your **connection establishment mode** and your **transaction mode** both to **Automatic**.
- **OnDemand** indicates that the connection is established on demand as business rules requiring a connection to the external system are performed. The connection is closed after the methods are completed.
- **Manual** indicates that the user explicitly calls the connection connect and disconnect methods in their collaboration as business rules.

Required Values

The required values are **Automatic**, **OnDemand** or **Manual**. The default is set to **Automatic**.

Note: *If you are using* ***Manual connection establishment mode****, you must also use* ***Manual transaction mode****.*

connection inactivity timeout

This value is used to specify the timeout for the Automatic connection establishment mode. If this is set to 0, the connection is not brought down due to inactivity. The connection is always kept alive; if it goes down, re-establishing the connection will automatically be attempted. If a non-zero value is specified, the connection manager tries to monitor for inactivity so that the connection is brought down if the timeout specified is reached.

Required Values

Any valid string.

connection verification interval

This value is used to specify the minimum period of time between checks for connection status to the database server. If the connection to the server is detected to be down during verification, your collaboration's **onDown** method is called. If the connection comes up from a previous connection error, your collaboration's **onUp** method is called.

Required Values

Any valid string.

4.2 Connection Manager

The Connection Manager allows you to define the connection functionality of your e*Way. You choose:

- When an e*Way connection is made.
- When to close the e*Way connection and disconnect.
- What the status of your e*Way connection is.
- When the connection fails, an OnConnectionDown method is called by the Collaboration

The Connection Manager was specifically designed to take full advantage of the higher e*Gate functionality. If you are running e*Gate 4.5.1 or earlier, this enhanced functionality is visible but is ignored.

The Connection Manager is controlled in the e*Way configuration as described in **Connector Settings** on page 21. If you choose to manually control the e*Way connections, you may find the following chart helpful.

Figure 4 e*Way Connection Control methods

	Automatic	On-Demand	Manual
onConnectionUp	yes	no	no
onConnectionDown	yes	yes only if the connection attempt fails	no
Automatic Transaction (XA)	yes	no	no
Manual Transaction	yes	no	no
connect	no	no	yes
isConnect	no	no	yes
disconnect	no	no	yes
timeout or connect	yes	yes	no
verify connection interval	yes	no	no

Controlling When a Connection is Made

As a user, you can control when a connection is made. Using Connector Settings, you may choose to have e*Way connections controlled manually — through the Collaboration, or automatically — through the e*Way Connection Configuration. If you choose to control the connection you can specify the following:

- To connect when the Collaboration is loaded
- To connect when the Collaboration is executed
- To connect by using an additional connection method in the ETD
- To connect by overriding any custom values you have assigned in the Collaboration
- To connect by using the isConnected() method. The isConnected() method is called per connection if your ETD has multiple connections.

Controlling When a Connection is Disconnected

In addition to controlling when a connection is made, you can also manually or automatically control when an e*Way connection is terminated or disconnected. To control the disconnect you can specify:

- To disconnect at the end of a Collaboration
- To disconnect at the end of the execution of the Collaborations Business Rules
- To disconnect during a timeout
- To disconnect after a method call

Controlling the Connectivity Status

You can control how often the e*Way connection checks to verify is still live and you can set how often it checks. See **Connector Settings** on page 21.

Chapter 5

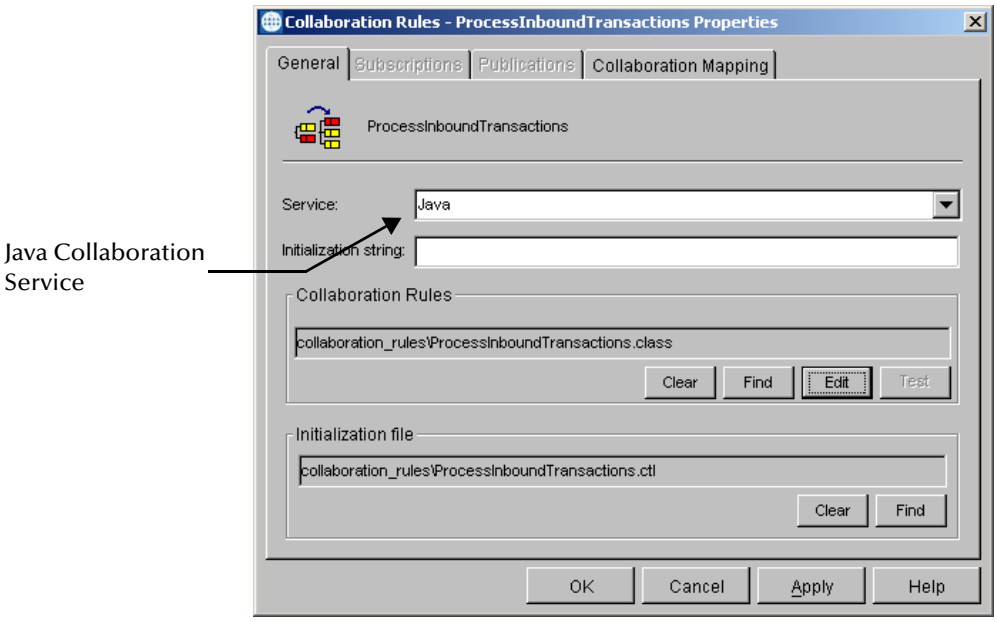
Implementation

This chapter discusses how to implement the Oracle e*Way in a production environment. Also included is a sample configuration.

5.1 Implementing Java-enabled Components

An e*Way or a BOB can be Java-enabled by selecting the Java Collaboration Service in the Collaboration Rules Properties. Either of these components can use e*Way Connections to exchange data with external systems.

Figure 5 The Java Collaboration Service



5.1.1 The Java Collaboration Service

The Java Collaboration Service makes it possible to develop external Collaboration Rules that are execute e*Gate business logic using Java code. Using the Java Collaboration Editor, you create Java classes that utilize the **executeBusinessRules()**, **userTerminate()**, and **userInitialize()** methods.

For more information on the Java Collaboration Service and sub collaborations, see the *e*Gate Integrator Collaboration Services Reference Guide*. For more information on the Java ETD Editor and the Java Collaboration Editor, see the *e*Gate Integrator User's Guide*.

5.1.2 Java-enabled Components

To make an e*Gate component Java-enabled, the component's Collaboration Rule must use the Java Collaboration Service. This requires all the intermediate components to also be configured correctly, since there is not a direct relationship between the e*Way/BOB and the Collaboration Service.

The e*Way/BOB requires one or more Collaborations. The Collaboration uses a Collaboration Rule. The Collaboration Rule uses a Collaboration Service. In order for the e*Way or BOB to be Java-enabled, the component's Collaboration Rule must use the Java Collaboration Service.

5.2 The Java ETD Builder

The Java ETD Builder is used to generate a Java-enabled ETD. The ETD Builder connects to the external database and generates the ETD corresponding to the external tables and procedures.

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

5.2.1 The Parts of the ETD

There are four possible parts to the Java-enabled Event Type Definition as shown in Figure 6.

Figure 6 The Java-enabled ETD

- **Element** – This is the highest level in the ETD tree. The element is the basic container that holds the other parts of the ETD. The element can contain fields and methods.
- **Field** – Fields are used to represent data. A field can contain data in any of the following formats: string, boolean, int, double, or float.
- **Method** – Method nodes represent actual Java methods.
- **Parameter** – Parameter nodes represent the Java methods' parameters.

5.2.2 Using the DBWizard ETD Builder

The DBWizard ETD Builder generates Java-enabled ETDs by connecting to external data sources and creating corresponding Event Type Definitions. The ETD Builder can create ETDs based on any combination of tables, stored procedures, or prepared SQL statements.

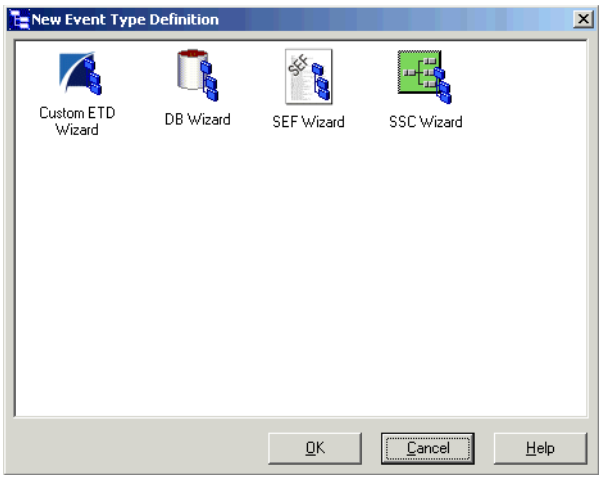
Field nodes are added to the ETD based on the tables in the external data source. Java method and parameter nodes are added to provide the appropriate JDBC functionality. For more information on the Java methods, refer to your JDBC developer's reference.

Note: *For Japanese, Korean and Traditional Chinese operating systems, you must use English table names, and English column names within the tables (DBCS is not supported).*

To create a new ETD using the DBWizard

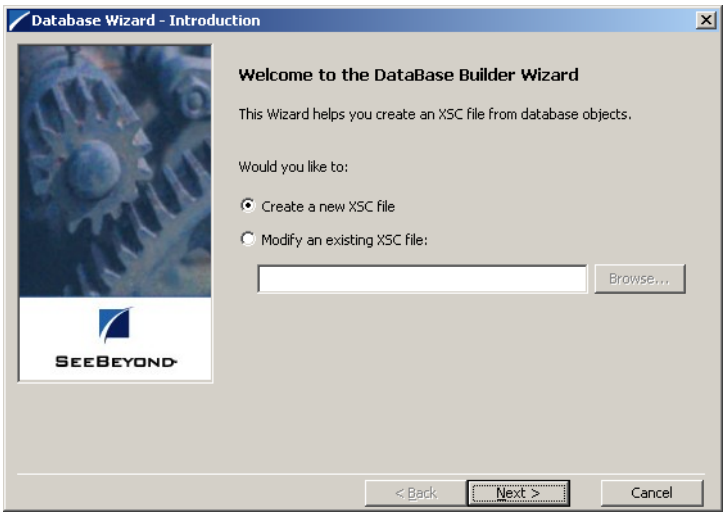
1. From the **Options** menu of the Schema Designer, choose **Default Editor…**.
2. Verify that **Java** is selected, then click **OK**.
3. Click the **ETD Editor** button to launch the Java ETD Editor.
4. In the **Java ETD Editor**, click the **New** button to launch the New Event Type Definition Wizard.
5. In the **New Event Type Definition Wizard**, select the **DBWizard** and click **OK** to continue. See Figure 7.

Figure 7 New Event Type Definition



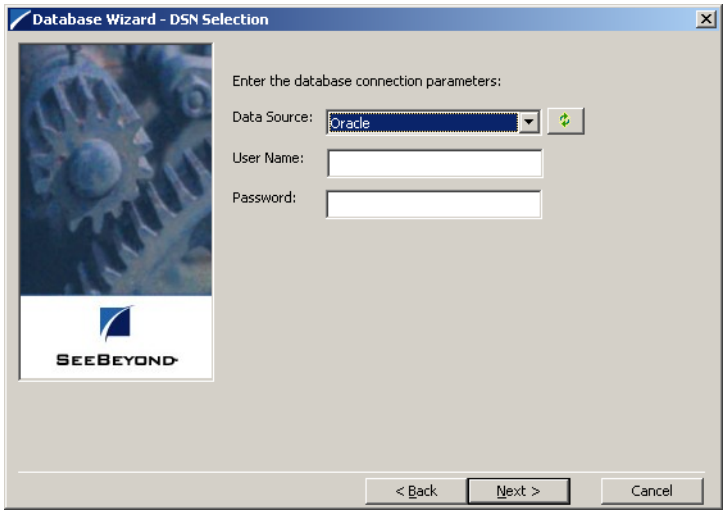
6. Enter the name of the new .xsc file you want to create or enter the name of the .xsc file you want to edit by browsing to its location. See Figure 8.

Figure 8 Database Wizard - Introduction



7 Select your **Data Source:** from the drop down list and enter your **User Name** and **Password**. See Figure 9.

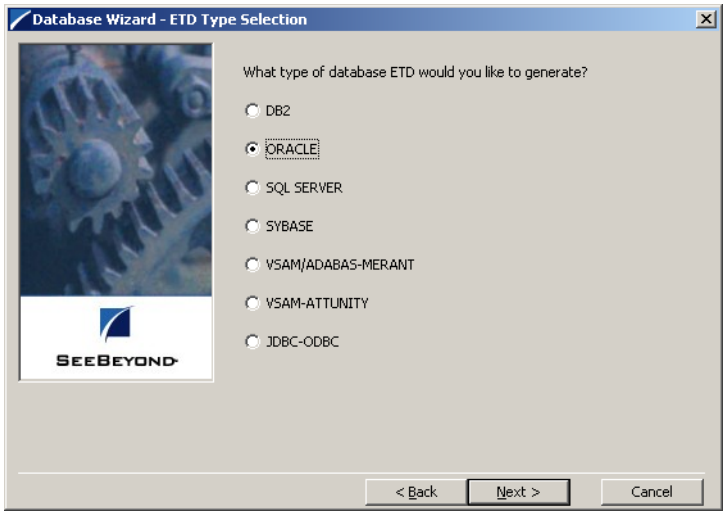
Figure 9 Database Wizard - DSN Selection



8 Select what type of database ETD you would like to generate. The data source you selected in the **Database Wizard - DSN Selection** window is the default. See Figure 10.

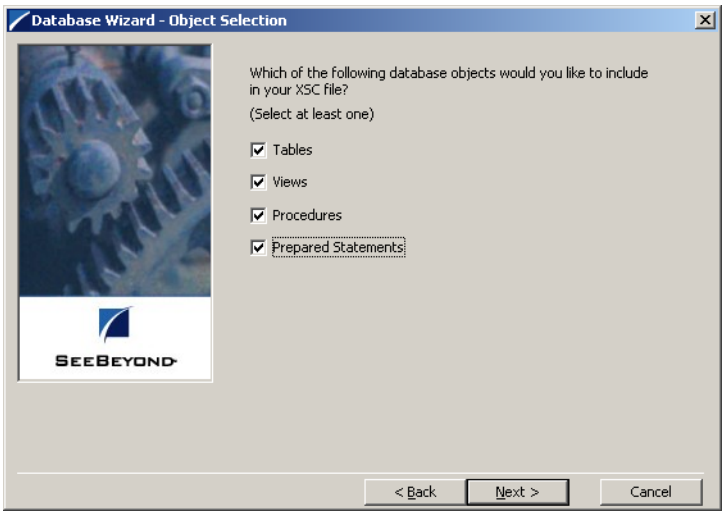
Note: *Do not change this unless instructed to do so by SeeBeyond personnel.*

Figure 10 Database Wizard - ETD Type Selection



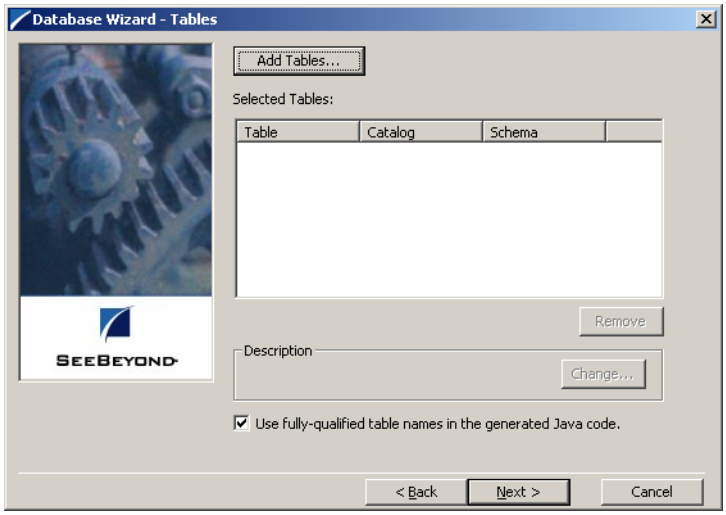
9 In the **Database Wizard - Object Selection** window, see Figure 11, select any combination of **Tables**, **Views**, **Procedures**, or **Prepared Statements** you would like to include in your .xsc file. Click **Next** to continue.

Figure 11 Database Wizard - Object Selection



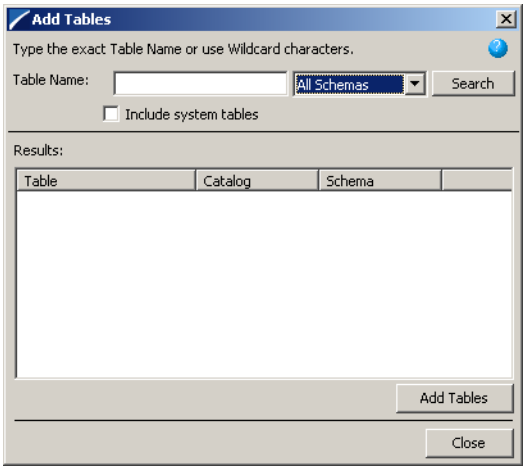
10 In the **Database Wizard - Tables** window, click **Add Tables**. See Figure 12.

Figure 12 Database Wizard - Tables



11 In the **Add Tables** window, type the exact name of the database table or use wildcard characters to return table names. See Figure 13.

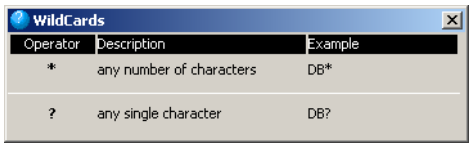
Figure 13 Add Tables



12 To see a list of valid wildcard characters, click the round ball with a question mark located in its center.

When using the Oracle native ODBC driver with the DB Wizard to search for a table, view, or procedure, do not use '_' or '%'. Doing so results in nothing being returned. Use the e*Way wildcard characters of '?', and '*'. For example, searching for tables "AB_CD", returns nothing. Use "AB?CD" or "AB*CD". See Figure 14.

Figure 14 Wildcards

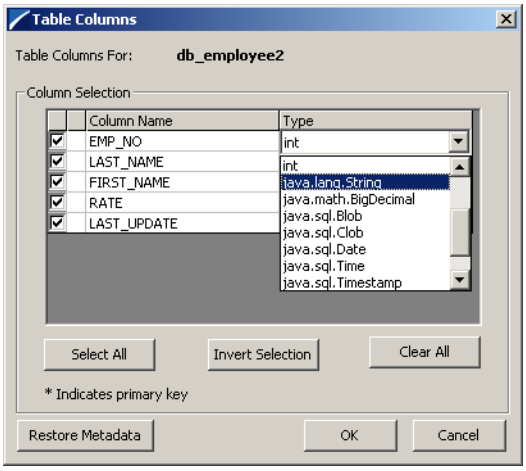


13 Select **Include System Tables** if you wish to include them and click **Search**. If your search was successful, you see the results in the Results window. To select the name of the tables you wish to add to your .xsc, double click on the table name or highlight the table names and click **Add Tables**. You may also use adjacent selections or nonadjacent selections to select multiple table names. When you have finished, click **Close**.

14 In the **Database Wizard - Tables** window, review the tables you have selected. If you would like to change any of the tables columns you have selected, click **Change**.

15 In the **Columns Selection** window, you can select or deselect your table columns. You can also change the data type for each table by highlighting the data type and selecting a different data type from the drop down list. Once you have completed your choices, click **OK**.

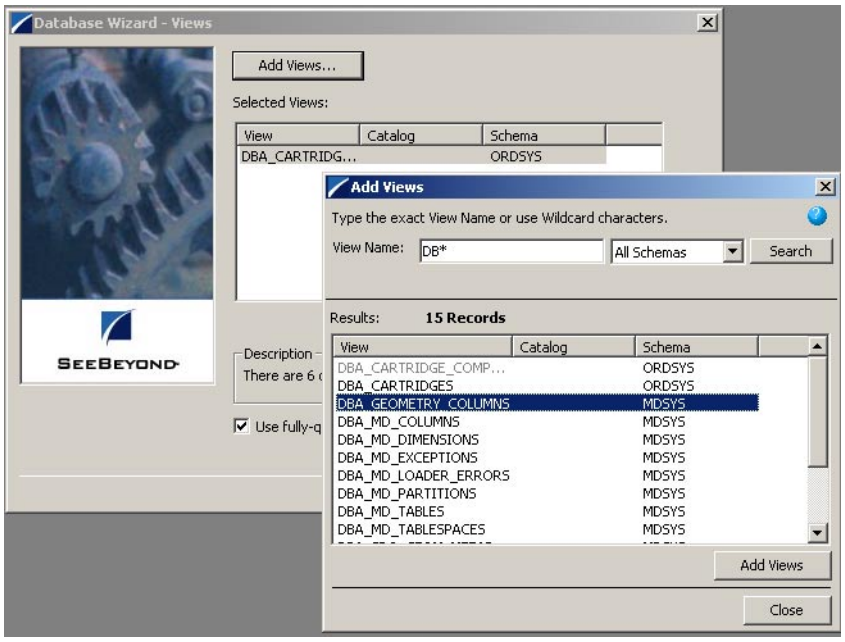
Figure 15 Columns Selection



16 In the **Database Wizard - Tables** window, review the tables you have selected. If you do not want to use fully-qualified table names in the generated Java code, click to clear the check box and click **Next** to continue.

17 If you selected **Views** on the **Database Wizard - Object Selection** window, you are presented with the **Database Wizard - Views** window. Follow steps 9 - 15 to select and add views to your .xsc. Views are read-only.

Figure 16 Database Wizard - Views



18 If you selected **Procedures** on the **Database Wizard - Object Selection** window, you are presented with the **Database Wizard - Procedures** window. Follow steps 9 - 15 to select and add **Procedures** to your .xsc. If you do not want to use fully-qualified procedure names in the generated Java code, click to clear the check box and click **Next** to continue.

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

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

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

“Manually” mode is the most flexible way to generate the result set nodes. It allows users to specify the node name, original column name and data type manually. One drawback of this method is that users need to know the original column names and

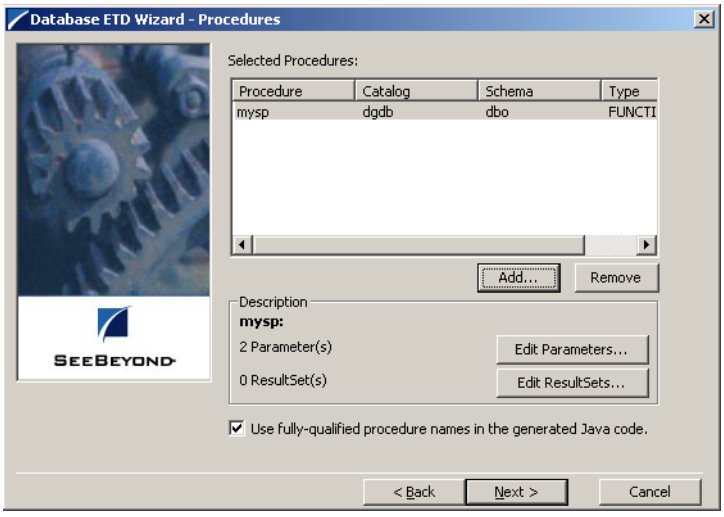
data types. This is not always possible. For example, the column name of 3*C in this query.

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

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

Note: *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.*

Figure 17 Database Wizard - Procedures



19 If you selected **Prepared Statements** on the **Database Wizard - Object Selection** window, you are presented with the **Database Wizard - Prepared Statement** window. To add **Prepared Statements** to your .xsc. complete the following steps:

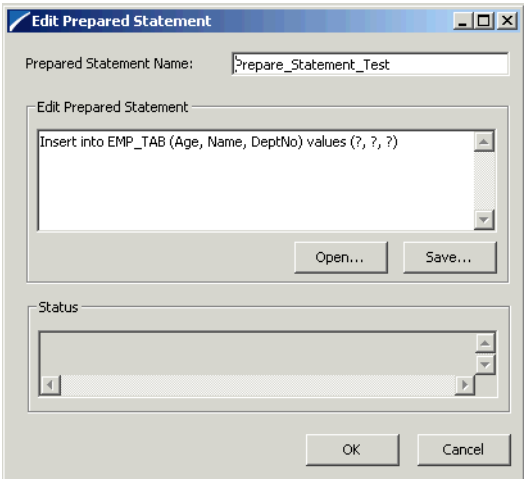
A Click **Add** to add a new prepared statement

B Enter a prepared SQL statement.

C Enter the **Prepared Statement Name** to be used by the statement.

D Use the **Open…** or **Save…** buttons to open pre-existing statements or save the current one. See Figure 18.

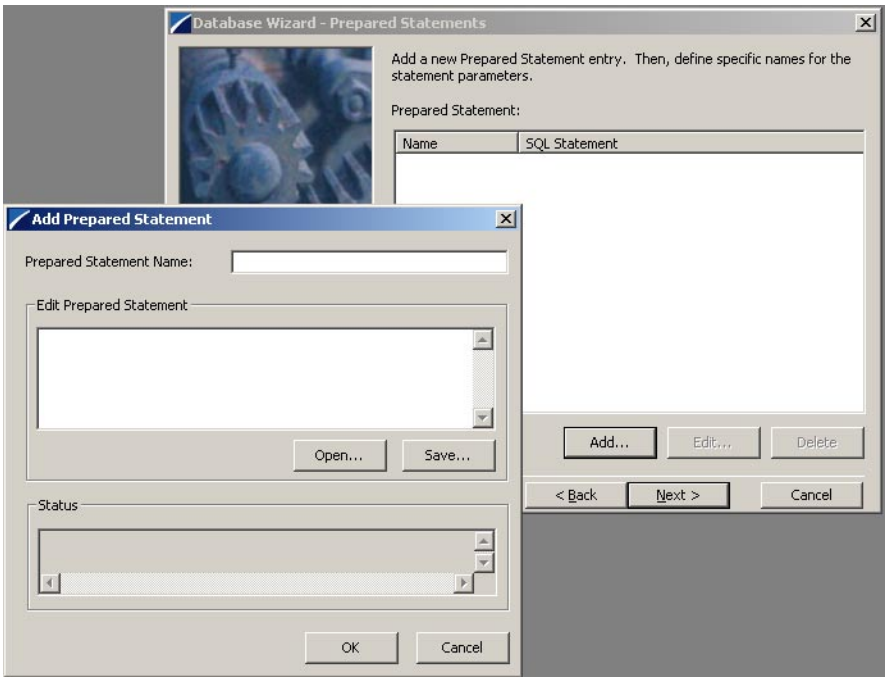
Figure 18 Add Prepared Statement



E Click **OK** to return to the **Database Wizard - Prepared Statements** window.

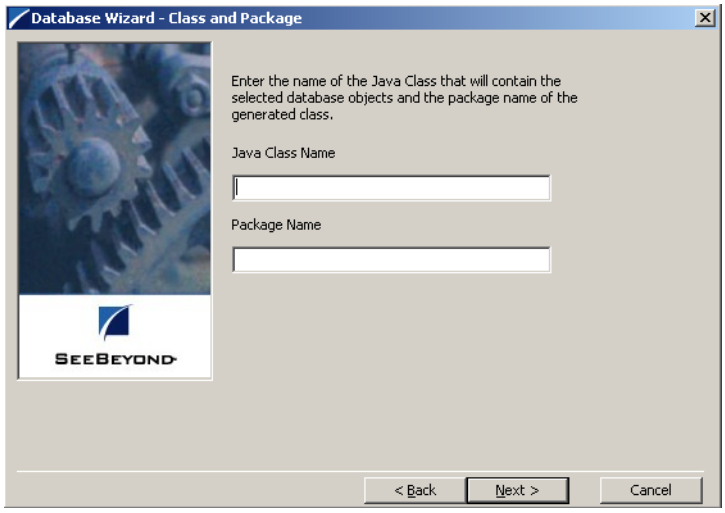
20 Repeat step 19 to add additional prepared statements or click **Next** to continue. See Figure 19.

Figure 19 Database Wizard - Prepared Statements



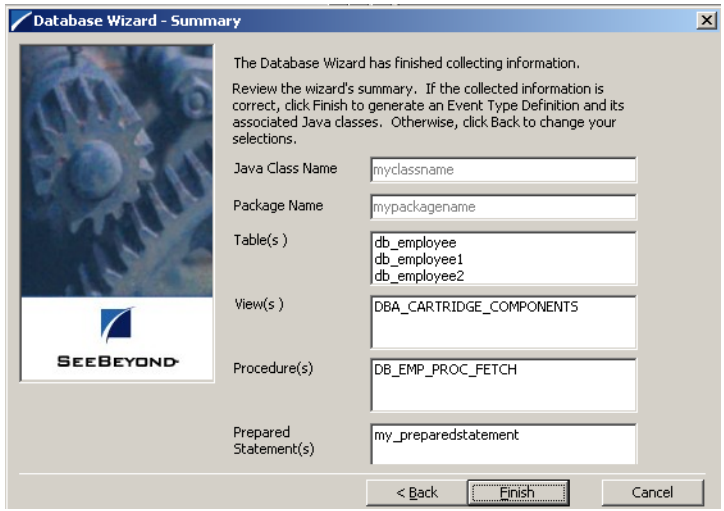
21 Enter the **Java Class Name** that will contain the selected tables and/or procedures and the **Package Name** of the generated classes. See Figure 20.

Figure 20 Database Wizard - Class and Package



22 View the summary of the database wizard information and click **Finish** to begin generating the ETD. See Figure 21.

Figure 21 Database Wizard - Summary



5.2.3 The Generated ETDs

The Database Wizard ETD builder can create three editable Event Type Definitions (ETDs) and one non-editable Event Type Definition (ETD). These types of ETDs can also be combined with each other. The four types of ETDs are:

- **The Table ETD** – The table ETD contains fields for each of the columns in the selected table as well as the methods required to exchange data with the external data source. To edit this type of ETD, you will need to open the .xsc in the Database Wizard.

- **The View ETD** - The view ETD contains selected columns from selected tables. View ETD's are read-only.
- **The Stored Procedure ETD** – The stored procedure ETD contains fields which correspond to the input and output fields in the procedure. To edit this type of ETD, you need to open the .xsc in the Database Wizard
- **The Prepared Statement ETD** – The prepared statement ETD contains a result set for the prepared statement. To edit this type of ETD, you need to open the .xsc in the Database Wizard.

5.2.4 Editing an Existing .XSC Using the Database Wizard

If you choose to edit an existing .xsc that you have created using the Database Wizard, do the following:

1. From the **Options** menu of the Schema Designer, choose **Default Editor…**.
2. Verify that **Java** is selected, then click **OK**.
3. From the **Tools** menu, click **ETD Editor...**
4. From the ETD Tool menu click **File** and click **New**.
5. From the **New Event Type Definition** window, select **DBWizard** and click **OK**.
6. On the Database Wizard - Introduction window, select **Modify an existing XSC file:** and browse to the appropriate .xbs file that you would like to edit.

You are now able to edit your .xsc file.

Note: *When you add a new element type to your existing .xsc, you must reselect any pre-existing elements or you will lose them when the new .xsc is created.*

If you attempt to edit an .xsc whose elements no longer exist in the database, a warning is displayed and the element is dropped from the ETD.

5.3 Using ETDs with Tables, Views, Stored Procedures, and Prepared Statements

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

5.3.1 The Table

A table ETD 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 ETD. This allows you to perform query, update, insert, and delete SQL operations in a table.

Using the select() method, you can specify the following types of ResultSets:

- TYPE_FORWARD_ONLY
- TYPE_SCROLL_INSENSITIVE
- TYPE_SCROLL_SENSITIVE

You can also specify ResultSets with a type of Concurrency:

- CONCUR_READ_ONLY
- CONCUR_UPDATABLE

To perform the update, insert or delete operation, the type of the ResultSet returned by the select() method must be CONCUR_UPDATABLE. Instead of specifying the type of ResultSet and concurrency in the select() method, you can also use the following methods:

- SetConcurrencytoUpdatabale
- SetConcurrentlytoRead Only
- SetScrollTypetoForwardOnly
- SetScrollTypetoScrollSensitive
- SetScrollTypetoInsensitive

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

```
getDBEmp().setConcurToUpdatable();
getDBEmp().setScroll_TypeToForwardOnly();
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. For each loop, process the return record.

For example:

```
getDBEmp().getDB_EMPLOYEE().select("");
While(getDBEmp().getDB_EMPLOYEE().next());
{ //Process the returning record
    getGenericOut.SetPayLoad(getDBEmp().getDB_Employee().
    getDBEmp().getFirstName());
}
```

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 datatypes like "int".

Note: *A getxxx method should be called before wasNull() is called.*

For example:

```
int empNo = getDBEmp().getDB_EMPLOYEE().getEMP_NO();
if (getDBEMP().getDB_EMLOYEE().wasNULL())
{ //Check to see if empNo is SQL NULL
 //Do something if empNo is SQL NULL
```

```
}
else
{ //Do something if empNo is not SQL NULL
}
```

The Insert Operation

To perform an insert operation on a table

1. Execute the select() method. You can specify the following types of ResultSets:
   - TYPE_FORWARD_ONLY
   - TYPE_SCROLL_INSENSITIVE
   - TYPE_SCROLL_SENSITIVE

   You must specify ResultSets with:
   - CONCUR_UPDATABLE
2. Move to the insert row by the moveToInsertRow method.
3. Set the fields of the table ETD
4. Insert the row by calling insertRow

   This example inserts an employee record.

```
getDBEmp().getDB_EMPLOYEE(ResultSet.TYPE_SCROLL_SENSITIVE,
ResultSet.CONCUR_UPDATABLE).select("");
getDBEmp().getDB_EMPLOYEE().moveToInsertRow();
getDBEmp().getDB_EMPLOYEE().setEMP_NO(123);
. . .
getDBEmp().getDB_EMPLOYEE().setRATE(123.45);
getDBEmp().getDB_EMPLOYEE().insertRow();
```

Table ResultSet Behavior

To make repeated insertions using a “select” into the table ResultSet without having to re-populate all the column values:

Before the schema runs, we have

```
SQL> select * from MARKET_TEMP;
```

Where:

```
C1 C2       C3
-- -------- --------
1  A1       B1
```

After the schema runs we have:

```
SQL> select * from MARKET_TEMP;
```

Becomes:

```
C1 C2       C3
--  -------- --------
1  A1       B1
2  A2       B1
3  A3       B1
```

Buffer the value of the selected column by :

```
String buf3 = getTempTbl().getMARKET_TEMP().getC3();
```

Call moveToInsertRow()

```
getTempTbl().getMARKET_TEMP().moveToInsertRow();
```

Set all the columns the first time

```
getTempTbl().getMARKET_TEMP().setC1("2");;
getTempTbl().getMARKET_TEMP().setC2("A2");
getTempTbl().getMARKET_TEMP().setC3(buf3);
```

Call insertRow()

```
getTempTbl().getMARKET_TEMP().insertRow();
```

Set all the columns except the unchanged column.

```
getTempTbl().getMARKET_TEMP().setC1("3");
getTempTbl().getMARKET_TEMP().setC2("A3");
```

Call insertRow()

```
getTempTbl().getMARKET_TEMP().insertRow();
```

In the above example, column C3 always has the same value (buf3).

Figure 22 Insert Method Business Rule

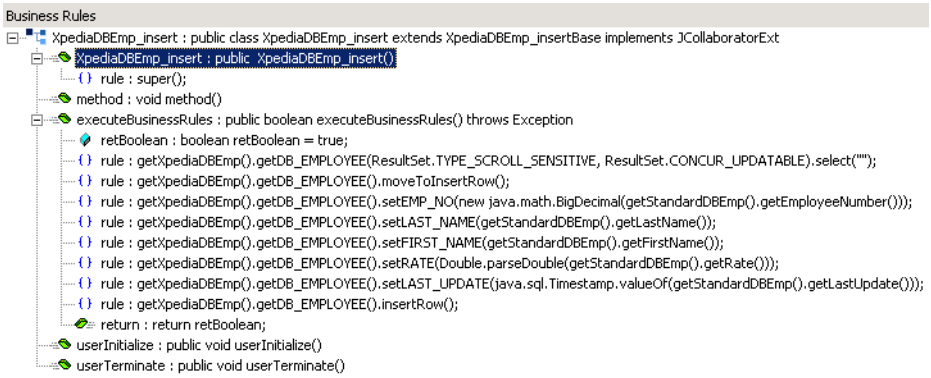
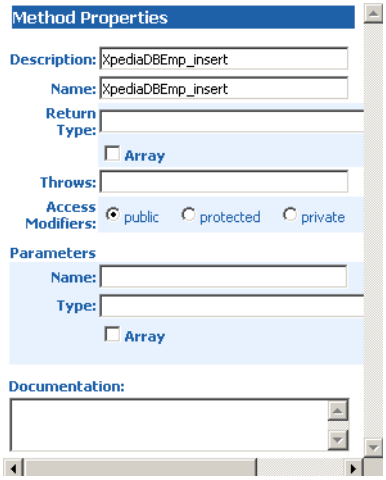


Figure 23 Insert Method Properties



The Update Operation

To perform an update operation on a table

1. Execute the select() method. You can specify the following types of ResultSets:
   - TYPE_FORWARD_ONLY
   - TYPE_SCROLL_INSENSITIVE
   - TYPE_SCROLL_SENSITIVE

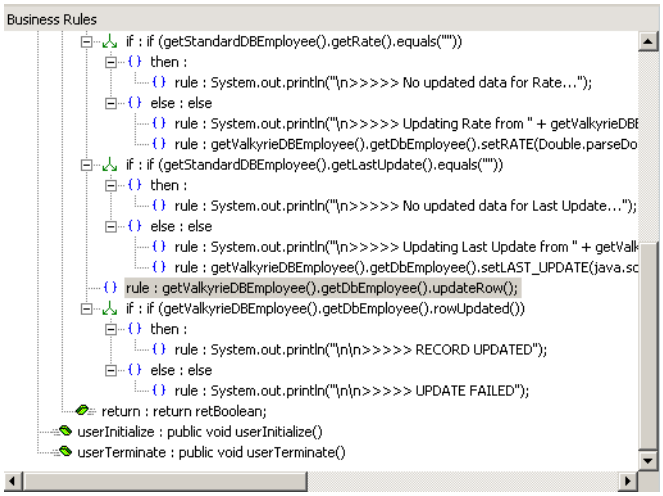
   You must specify ResultSets with:
   - CONCUR_UPDATABLE
2. Move to the row that you want to update.
3. Set the fields of the table ETD

4 Update the row by calling **updateRow**.

In this example, we move to the third record and update the EMP_NO and RATE fields.

```
getDBEmp().getDB_EMPLOYEE(ResultSet.TYPE_SCROLL_SENSITIVE,
ResultSet.CONCUR_UPDATABLE).select("");
getDBEmp().getDB_EMPLOYEE().absolute(3);
getDBEmp().getDB_EMPLOYEE().setEMP_NO(123);
getDBEmp().getDB_EMPLOYEE().setRATE(123.45);
getDBEmp().getDB_EMPLOYEE().updateRow();
```

Figure 24 Update() Method Business Rule



The Delete Operation

To perform a delete operation on a table

1 Execute the select() method. You can specify the following types of ResultSets:

- TYPE_FORWARD_ONLY
- TYPE_SCROLL_INSENSITIVE
- TYPE_SCROLL_SENSITIVE

You must specify ResultSets with:

- CONCUR_UPDATABLE

2 Move to the row that you want to delete.

3 Set the fields of the table ETD

4 Delete the row by calling **deleteRow**.

In this example DELETE the first record of the result set.

```
getDBEmp().getDB_EMPLOYEE(ResultSet.TYPE_SCROLL_SENSITIVE,
ResultSet.CONCUR_UPDATABLE).select("");
getDBEmp().getDB_EMPLOYEE().first();
getDBEmp().getDB_EMPLOYEE().deleteRow();
```

5.3.2 The View

Views are used to look at data from selected columns within selected tables. Views are read-only.

For query operations, please refer to "Tables" sub section.

5.3.3 The Stored Procedure

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

Executing Stored Procedures

Assuming that you have the following procedure:

```
create procedure LookupGlobal
( inlocalID in varchar, outglobalProductID out varchar )
as
begin
select outglobalProductID into globalProductID from SimpleLookup
where localID = inlocalID
end
```

The ETD represents the Stored Procedure "LookUpGlobal" with two parameters, an inbound parameter (INLOCALID) and an outbound parameter (OUTGLOBALPRODUCTID) can be generated by the DB Wizard as shown in Figure 25. Representing these as nodes in an ETD allows you to drag values from other ETD's to the input parameters, execute the call, and collect the output parameter data by dragging from it's node to elsewhere.

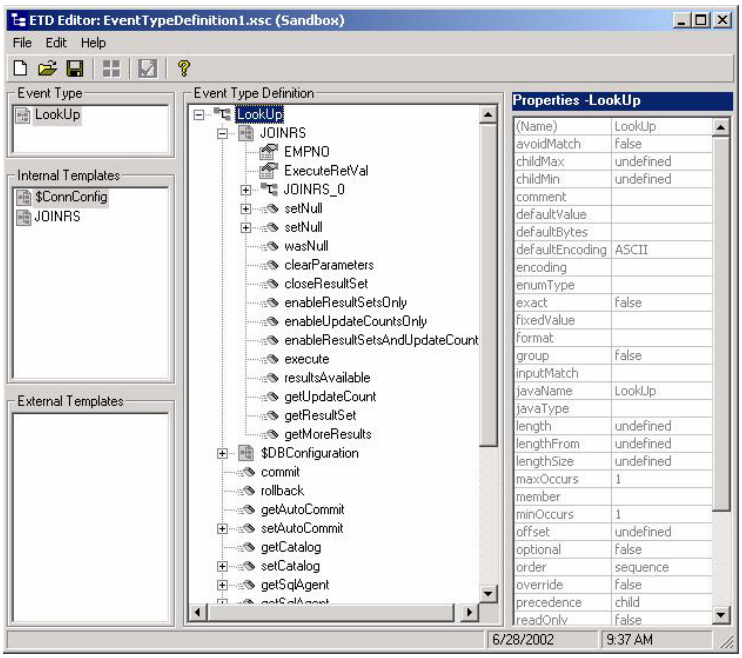
Below are the steps for executing the Stored Procedure:

1. Specify the input values.
2. Execute the Stored Procedure.
3. Retrieve the output parameters if any.

For example:

```
getLookUp().getLookUpGlobal().setIntlocalID("123");
getLookUp().getLookUPGlobal().execute();
String s =
getLookUp().getLookUpGlobal.getOutGlobelProductID;
```

Figure 25 Stored Procedure LookUpGlobal



Manipulating the ResultSet and Update Count Returned by Stored Procedure

For Stored Procedures that return ResultSets and Update Count, the following methods are provided to manipulate the ResultSet:

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

Oracle 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 PreparedStatementAgent class, 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 ETD, when that node's **available()** method returns true.

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

Collaboration usability for a stored procedure ResultSet

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

```
// resultsAvailable() will be true if there's an update count and/or a
result set available.
// note, it should not be called indiscriminantly because each time
the results pointer is
// advanced via getMoreResults() call.
while (getSPIn().getSpS_multi().resultsAvailable())
{
  // check if there's an update count
  if (getSPIn().getSpS_multi().getUpdateCount() > 0)
  {
    System.err.println("Updated
"+getSPIn().getSpS_multi().getUpdateCount()+" rows");
  }

  // each result set node has an available() method (similar to ETD'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())
    {
      System.err.println("Customer Id   =
"+getSPIn().getSpS_multi().getNormRS().getCustomerId());
      System.err.println("Customer Name =
"+getSPIn().getSpS_multi().getNormRS().getCustomerName());
      System.err.println();
    }
    System.err.println("===");
  }
  else if (getSPIn().getSpS_multi().getDbEmployee().available())
  {
    while (getSPIn().getSpS_multi().getDbEmployee().next())
    {
      System.err.println("EMPNO    =
"+getSPIn().getSpS_multi().getDbEmployee().getEMPNO());
```

```
      System.err.println("ENAME    =
"+getSPIn().getSpS_multi().getDbEmployee().getENAME());
      System.err.println("JOB      =
"+getSPIn().getSpS_multi().getDbEmployee().getJOB());
      System.err.println("MGR      =
"+getSPIn().getSpS_multi().getDbEmployee().getMGR());
      System.err.println("HIREDATE =
"+getSPIn().getSpS_multi().getDbEmployee().getHIREDATE());
      System.err.println("SAL      =
"+getSPIn().getSpS_multi().getDbEmployee().getSAL());
      System.err.println("COMM     =
"+getSPIn().getSpS_multi().getDbEmployee().getCOMM());
      System.err.println("DEPTNO   =
"+getSPIn().getSpS_multi().getDbEmployee().getDEPTNO());
      System.err.println();
    }
    System.err.println("===");
  }
}
```

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

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

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

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

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

- The DBWizard Assistant when creating a ResultSet expects the Oracle 8i or 9i column names to be in English.

5.3.4 Prepared Statement

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

Prepared statements can be used to perform insert, update, delete and query operations. A prepared statement uses a question mark (?) as a place holder for input. For example:

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

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

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

5.3.5 Batch Operations

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

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

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

5.3.6 Database Configuration Node

The Database Configuration node allows you to manage the "transaction mode" through the Collaboration if you have set the mode to manual in the e*Way connection configuration. See **"Connector Settings" on page 21**.

5.3.7 Using Clobs

Because the Clob data type is handled differently within Oracle databases, you should not try to insert a string directly into a Clob column in the database, you need to do the following to insert or update a Clob with a String:

Inserting or Updating a Clob in a Stored Procedure

Clob parameters of a stored procedure appear in your ETD as java.sql.Clob. To pass a String into the Clob parameter, drag and drop the setXXX method into the collaboration rule and drag the input data of String type. The String is automatically converted to a temporary Clob type before being passed into the Stored Procedure. Refer to the *Oracle 9i Application Developer's Guide* for more information regarding temporary Clobs.

Although the string is converted to a temporary Clob when it is passed into a Clob parameter of a Stored Procedure, the temporary Clob does not persist in the database. The temporary space for the data of the temporary Clob is released after the stored procedure returns. Therefore, you need to make sure your stored procedure copies the temporary Clob to a permanent lob locator.

For example, if you have a table that contains a Clob column:

```
create table clob_test
(customer_id NUMBER not null,
processed_text CLOB);
```

The following sample stored procedure updates the same table with a temporary Clob:

```
create or replace procedure clob_update(id IN NUMBER, ob IN CLOB)
  as
   c clob;
  begin
   select processed_text into c from clob_test where customer_id=id;
   dbms_lob.copy(c, ob, dbms_lob.getLength(ob));
end;
```

The sample stored procedure that inserts a temporary Clob into the above table is this:

```
create or replace procedure clob_import(id IN NUMBER, ob IN CLOB)
as
c clob;
begin
insert into clob_test values(id, empty_clob)returning processed_text
into c;
dbms_lob.copy(c, ob, dbms_lob.getLength(ob));
end;
```

Then in your collaboration, you can drag a String into the Clob parameter of the stored procedure like the following:

```
getCallStoredProc().getCLOB_IMPORT().setID(new Double(987));
getCallStoredProc().getCLOB_IMPORT().setOB("This is the String passed
into Clob");
getCallStoredProc().getCLOB_IMPORT().execute();
```

where "getCallStoredProc()" gets the ETD group and "getCLOB_IMPORT()" gets the stored procedure "clob_import".

The sample schema for the above has been included in the samples directory on the e*Gate installation disk **\samples\eworacle\ClobSupport\StoredProcedure.zip**.

Inserting or Updating a Clob using a Table ETD

Oracle's JDBC driver does not allow direct insertion of a string with more than 4000 characters into a Clob column. To insert into a Clob column, an empty Clob needs to be inserted first. Then, a lob locator can be retrieved to populate the Clob with data.

For example, if you have a table that contains a Clob column:

```
create table clob_test
  (customer_id NUMBER not null, processed_text CLOB);
```

The sample collaboration rule that uses a table to insert a new row into the above table with the value 111 for the customer_id and a string into the processed_text is:

```
int CustID = 111;
getMyEtdGroup().setScrollTypeToScrollSensitive();
getMyEtdGroup().setConcurrencyToUpdatable();
getMyEtdGroup().getCLOB_TEST().select("");
getMyEtdGroup().getCLOB_TEST().moveToInsertRow();
getMyEtdGroup().getCLOB_TEST().setCUSTOMER_ID(CustID);
getMyEtdGroup().getCLOB_TEST().setPROCESSED_TEXT(oracle.sql.CLOB.empt
y_lob());
getMyEtdGroup().getCLOB_TEST().select("CUSTOMER_ID" + CustID + "for
update");
while(getMyEtdGroup().getCLOB_TEST().next())
{
oracle.sql.CLOB
myClob=(oracle.sql.CLOB)(getMyEtdGroup().getCLOB_TEST().getPROCESSED_
TEXT());
myClob.putString(1, "mystring that goes into Clob");
getMyEtdGroup().getCLOB_TEST().updateRow();
}
```

where "getMyEtdGroup()" gets the ETD group and "getCLOB_TEST()" gets the table ETD for the table "clob_test".

Note: *Use "oracle.sql.CLOB.empty_lob()" for the empty Clob to be inserted.*

Note: *Do not forget the "for update" clause when selecting the update row.*

The sample schema for the above has been included in the samples directory on the e*Gate installation disk **\samples\eworacle\ClobSupport\TableEtd.zip**.

Inserting or Updating a Clob Using a Prepared Statement

You can also use a prepared statement to insert or update Clob's in an Oracle database. Oracle's JDBC driver does not allow direct insertion of a String with more than 4000 characters into a Clob column. To insert a Clob, an empty Clob needs to be inserted first. Then a lob locator can be retrieved to populate the Clob with data.

For example, if you have a table that contains a Clob column:

```
create table clob_test
(customer_id NUMBER not null,
processed_text CLOB);
```

To insert a new row into the above table with a value for the customer_id and a String into the processed_text, two Prepared Statements need to be used. The first Prepared Statement is for inserting an empty Clob with the value for customer_id:

Insert into clob_test values(?, empty_clob())

The second prepared statement is for updating the empty Clob with a String:

select processed_text from clob_test where customer_id = ? for update

The collaboration rule that uses the above second Prepared Statements to insert a new row into the above table with value 999 for the customer_id and a Sting into the processed_text is:

```
int id = 999;
getMyEtdGroup().getInsertEmptyClob().setParam1(id);
getMyEtdGrouop90.getSelectClobById().setParam1(id);
com.stc.eways.jdbcx.ResultSetAgent rsResults =
getMyEtdGroup().getSelectClobById().executeQuary();
if(rsResults.next())
{
   oracle.sql.CLOB myClob = (oracle.sql.CLOB)(rsResults.getClob(1));
   myclob.putString(1, "The string goes into Clob");
}
```

where "getMyEtdGroup()" gets the ETD group; "getInsertEmptyClob()" gets the first prepared statement; and the "getSelectClobById()" gets the second prepared statement.

The sample schema for the above has been included in the samples directory on the e*Gate installation disk.

\samples\eworacle\ClobSupport\PreparedStatement.zip.

To update an existing Clob in the database, omit the use of the first prepared statement in the above, and the collaboration is:

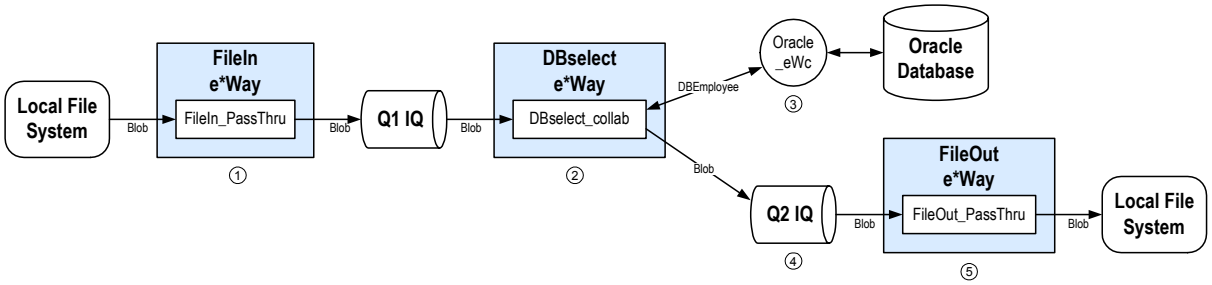
```
int id = 999;
getMyEtdGroup().getSelectClobById().setParam1(id);
com.stc.eways.jdbcx.ResultSetAgent rsResults=
getMyEtdGroup().getSelectClobById().executeQuery();
if (rsResults.next())
{
   oracle.sql.CLOB myClob = (oracle.sql.CLOB)(rsReuslts.getClob(1));
   myClob.putString(1, "The String goes into Clob");
}
```

5.4 Sample Scenario—Polling from an Oracle Database

This section describes how to use the Java-enabled Oracle e*Way in a sample implementation. This sample schema demonstrates the polling of records from an Oracle database and converting the records into e*Gate Events.

Figure 26 shows a graphical overview of the sample schema.

Figure 26 The Database Select Scenario—Overview



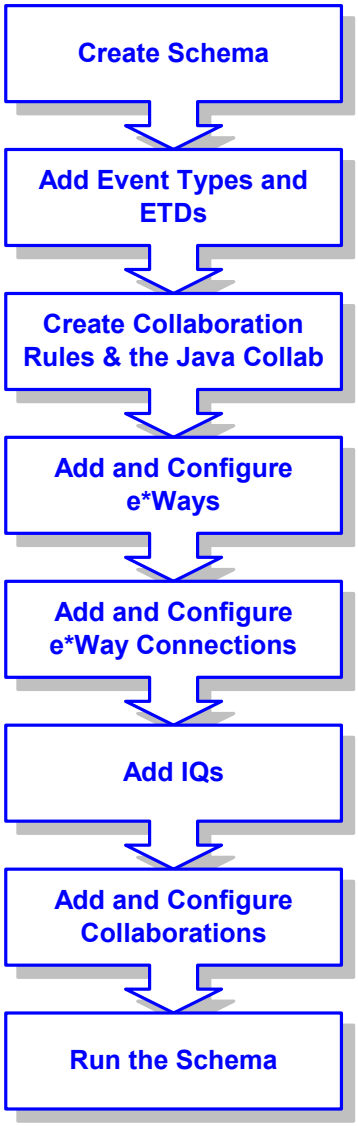
1. The **FileIn** e*Way retrieves an Event (text file) containing the database select criteria and publishes it to the **Q1** IQ.
2. The **DBselect** e*Way retrieves the Generic Event (**Blob**) from the IQ. This triggers the rest of the Collaboration which has two parts.
3. The information in **Blob** is used to retrieve information from the database via the **Oracle_eWc** e*Way Connection. This e*Way Connection contains information used by the Collaboration to connect to the Oracle database.
4. The information retrieved from the database is copied to the Generic Event (**Blob**) and published to the **Q2** IQ.
5. The **FileOut** e*Way retrieves the Generic Event (**Blob**) from the **Q2** IQ then writes it out to a text file on the local file system.

Overview of Steps

The sample implementation follows these general steps:

- **"Create the Schema" on page 52**
- **"Add the Event Types and Event Type Definitions" on page 52**
- **"Create the Collaboration Rules and the Java Collaboration" on page 55**
- **"Add and Configure the e*Ways" on page 59**
- **"Add and Configure the e*Way Connections" on page 61**
- **"Add the IQs" on page 62**
- **"Add and Configure the Collaborations" on page 63**
- **"Run the Schema" on page 64**

Figure 27 Schema Configuration Steps



External Database Tables

The sample uses a simple external Oracle database with a table called **DB_EMPLOYEE**. The table contains the following columns as shown in Table 2.

Table 2 The DB_EMPLOYEE Table

Column	Format	Description
EMP_NO	INTEGER	The employee number.
LAST_NAME	VARCHAR2	The employee's last name.
FIRST_NAME	VARCHAR2	The employee's first name.
RATE	FLOAT	The employee's pay rate.
LAST_DATE	DATE	The last transaction date for the employee

5.4.1 Create the Schema

The first step in deploying the sample implementation is to create a new schema. After installing the Oracle e*Way Intelligent Adapter, do the following:

1. Launch the e*Gate Schema Designer GUI.
2. Log into the appropriate Registry Host.
3. From the list of schemas, click **New** to create a new schema.
4. For this sample implementation, enter the name **DBSelect** and click **Open**.

   The Schema Designer launchs and displays the newly created schema.

5.4.2 Add the Event Types and Event Type Definitions

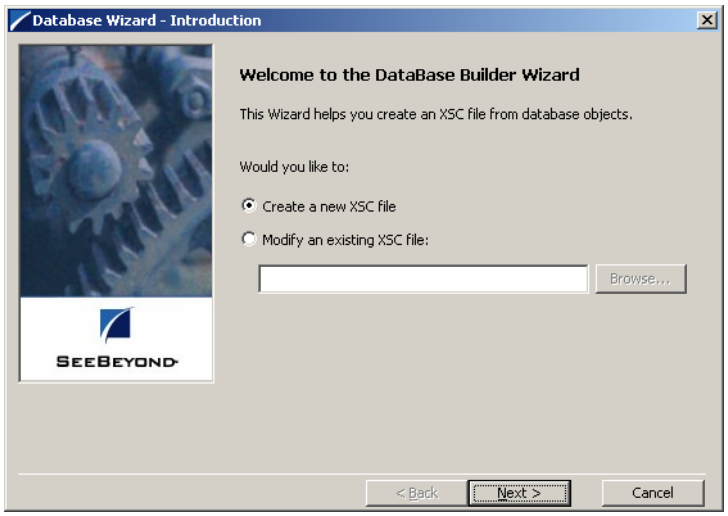
Two Event Types and Event Type Definitions are used in this sample.

- **DBEmployee** – This Event Type represents the layout of the employee records in the **DB_Employee** table. The Event Type uses the **DBEmployee.xsc** Event Type Definition. The ETD is generated by using the Java ETD Editor's Database Wizard (DBWizard).
- **GenericBlob** – This Event Type is used to pass records with no specific format (blob). The Event Type uses the **GenericBlob.xsc** ETD. The ETD is manually created as a fixed-length ETD.

To create the DBEmployee Event Type and ETD

1. From the **Options** menu of the Schema Designer, choose **Default Editor…**.
2. Verify that **Java** is selected, then click **OK**.
3. In the **Components** pane of the Schema Designer, select the **Event Types** folder.
4. Click the **New Event Type** button to add a new Event Type.
5. Enter the name **DBEmployee** and click **OK**.
6. Double-click the new **DBEmployee** Event Type to display its properties.
7. Select your **Data Source** from the drop down list and enter your **User Name** and **Password**.
8. From the **File** menu, choose **New**. The **New Event Type Definition** dialog box appears.
9. In the **New Event Type Definition** dialog box, select **DBWizard** and click **OK**.
10. Select **Create a new .XSC file**. Click **Next** to continue. See Figure 28.

Figure 28 Database Wizard Introduction



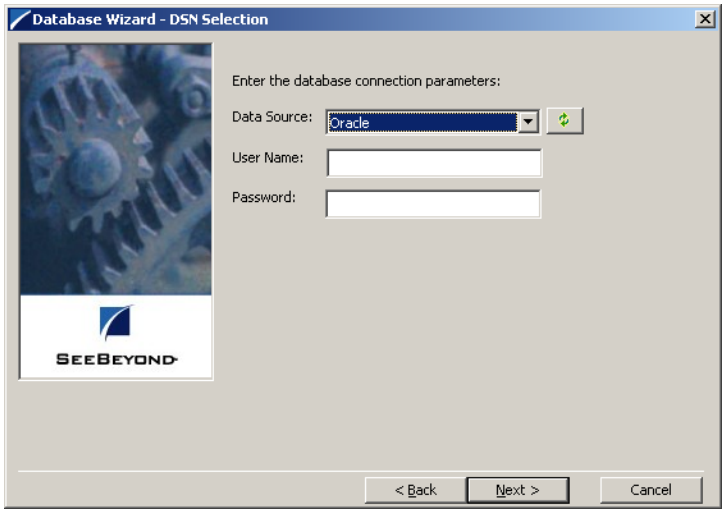
11 Enter the database DNS source and login information.

A Select the **Data Source** from the dropdown list of ODBC data sources.

B Enter the **User Name** and **Password** used to log into the database.

Click **Next** to continue. See Figure 29.

Figure 29 Database Wizard - DSN Selection



12 The **Database Wizard - ETD Type Selection** window appears. The DNS source you selected on the previous window is the default selection for this window. Do not change this selection type unless instructed to do so by SeeBeyond support personnel. Click **Next** to continue.

13 This scenario uses a table rather than a procedure. Select **Table** and click **Next** to continue.

14 From the **Database Wizard - Tables** window, click **Add Tables...** Enter the exact **Table Name** or enter any valid wildcards. From the drop-down list select the appropriate database schema and click **Search**. The wizard connects to the data source and displays a list of tables.

15 Select the table to be included in the ETD and click **Next**.

16 The Java Class Name/Package Name dialog box appears. Enter the **Group** and **Package** information.

A Enter your database name as the **Java Class Name**.

B Enter **DBEmployee** for the Package Name and click **Next** to continue.

17 Click **Finish** to complete the Wizard. The Wizard generates and displays the ETD.

18 From the **File** menu, choose **Save**.

19 Name the ETD **DBEmployee.xsc** and click **OK**.

20 From the **File** menu, choose **Promote to Run Time** and click **OK** when finished.

21 From the **File** menu, choose **Close** to exit the ETD Editor.

To create the GenericBlob Event Type and ETD

1 In the **Components** pane of the Schema Designer, select the **Event Types** folder.

2 Click the **Create a New Event Type** button to add a new Event Type.

3 Enter the name **GenericBlob** and click **OK**.

4 Double-click the new **GenericBlob** Event Type to display its properties.

5 Click the **New** button to create a new Event Type Definition.

The Java Event Type Definition Editor will appear.

6 From the **File** menu, choose **New**.

The New Event Type Definition dialog box will appear.

7 In the New Event Type Definition dialog box, select **Custom ETD** and click **OK**.

8 Read the introductory screen, then click **Next** to continue.

The Root Node Name / The Package Name dialog box will appear.

9 Enter a Root Node Name for the GenericBlob

10 Enter **GenericBlobPackage** for the **Package Name** and click **Next** to continue.

11 Read the summary information and click **Finish** to generate the ETD.

12 In the **Event Type Definition** pane, right-click the root node, point to **Add Field** in the shortcut menu, and click **As Child Node**.

13 Enter the properties for the two nodes as shown in Table 3.

Table 3 GenericBlob ETD Properties

Node	Property	Value
Root Node	Name	GenericBlob
	Structure	fixed
	Length	undefined
Child Node	Name	Data
	Structure	fixed
	Length	undefined

14 From the **File** menu, choose **Save**.

15 Enter the name **GenericBlob.xsc** and click **OK**.

16 From the **File** menu, choose **Compile**.

17 From the **File** menu, choose **Promote to Run Time** and click **OK** when finished.

18 From the **File** menu, choose **Close** to exit the ETD Editor.

19 In the Event Type properties dialog box, click **OK** to save and close the Event Type.

5.4.3 Create the Collaboration Rules and the Java Collaboration

The sample scenario uses two Collaboration Rules and one Java Collaboration:

- **GenericPassThru** – This Collaboration Rule is used to pass the GenericBlob Event Type through the schema without modifying the Event.
- **DBSelect** – This Collaboration Rule is used to convert the inbound Event's selection criteria into a SQL statement, poll the external database, and return the matching records as an outbound Event.
- **DBSelectCollab** – This Java Collaboration contains the logic required to communicate with the external database.

Before creating the Collaboration Rules, assure your default Collaboration Editor is set to Java. To do this do the following:

1 From the e*Gate Schema Designer toolbar, click **Options**.

2 Click **Default Editor...**

3 Select **Java**.

4 Click **OK**.

To create the GenericPassThru Collaboration Rule

1 In the components pane of the Schema Designer, select the **Collaboration Rules** folder.

2 Click the **New Collaboration Rules** button to add a new Collaboration Rule.

3 Name the Collaboration Rule **GenericPassThru** and click **OK**.

4 Click the **Properties** button to display the Collaboration Rule's properties.

5 On the **General tab**, select **Pass Through** from the Services drop-down list.

6 Click the **Subscriptions** tab, select the **GenericBlob** Event Type, and click the right arrow.

7 Click the **Publications** tab, select the **GenericBlob** Event Type, and click the right arrow.

8 Click **OK** to save the Collaboration Rule.

To create the DBSelect Event Type

1 In the components pane of the Schema Designer, select the **Collaboration Rules** folder.

2 Click the **New Collaboration Rules** button to add a new Collaboration Rule.

3 Name the Collaboration Rule **DBSelect** and click **OK**.

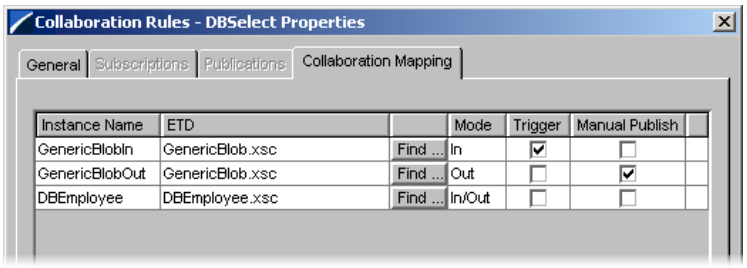
4 Click the **Properties** button to display the Collaboration Rule's properties.

5 In the **Service** list, click **Java**.

6 Click the **Collaboration Mapping** tab.

7 Add three instances as shown in Figure 30.

Figure 30 DBSelect Instances



8 Click **Apply** to save the current changes.

9 Click the **General** tab.

10 Click **New** to create the new Collaboration file.

The Java Collaboration Editor appears. The Source and Destination Events are already supplied based on the Collaboration Rule's Collaboration Mapping. See Figure 30.

11 From the **View** menu, choose **Display Code**.

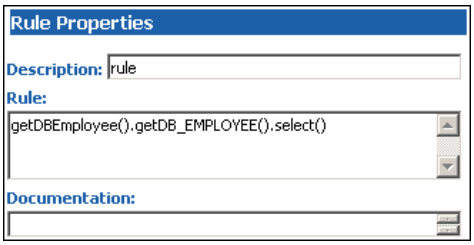
This displays the Java code associated with each of the Collaboration's rules.

12 In the Business Rules pane, select the **retBoolean** rule and click the **rule** button to add a new Rule.

13 In the **Destination Events** pane, expand the **DBEmployee** Event Type until the **select** method is visible.

14 Drag the **select** method into the **Rule** field of the **Rule Properties** pane. Click **OK** to close the dialog box without entering any criteria. See Figure 31.

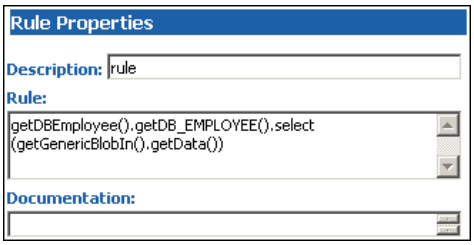
Figure 31 Rule Properties



15 In the **Source Events** pane, expand the **GenericBlobIn** Event Type until the **Data** node is visible.

16 In the **Rule Properties** pane, position the cursor inside the parentheses of the **select** method. Then drag the **Data** node from the **Source Events** pane into the **select** method's parentheses. See Figure 32.

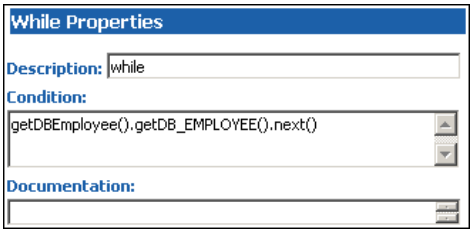
Figure 32 Rule Properties (Continued)



17 Select the newly edited rule in the **Business Rules** pane and click the **while** button to add a new while loop beneath the current rule.

18 Drag the **next** method from the **Destination Events** pane into the **Condition** field of the **While Properties** pane. See Figure 33.

Figure 33 While Properties

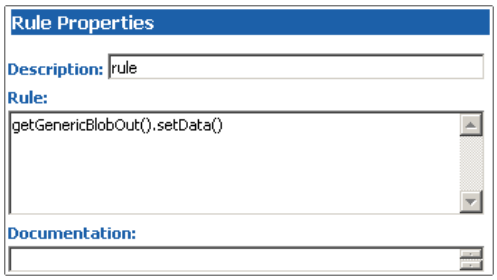


19 Select the newly edited while loop in the Business Rules pane and click the **rule** button to add a new rule as a **child** to the while loop.

20 In the **Destination Events** pane, expand the **GenericBlobOut** Event Type until the **Data** node is visible.

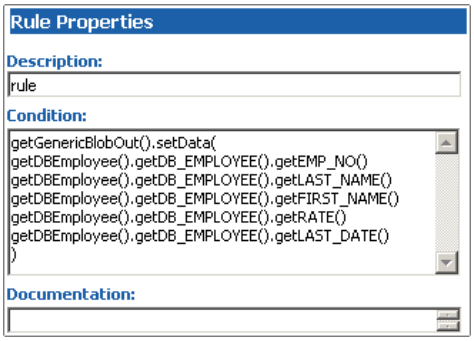
21 Drag the **Data** node into the **Rule** field of the Rule Properties pane. See Figure 34.

Figure 34 Rule Properties



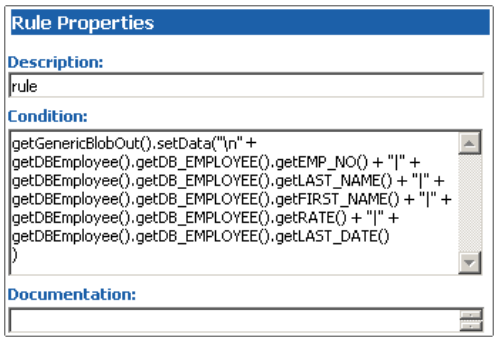
22 In the Rule Properties pane, position the cursor inside the parentheses of the **setData()** method. Then drag each of the five data nodes of **DB_EMPLOYEE** from the Source Events into the parentheses of the rule. See Figure 35.

Figure 35 Rule Properties (Continued)



23 Edit the text of the condition to add a newline character and pipe (|) delimiters between each of the five data nodes. See Figure 36.

Figure 36 Rule Properties (Continued)

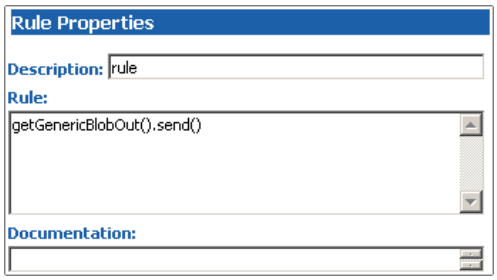


24 Select the newly edited rule in the **Business Rules** pane and click the **rule** button to add a new rule inside the while loop.

25 Drag the root node of the **GenericBlobOut** Event into the **rule** field in the **Rule Properties** pane.

26 Edit the rule; add a **send()** method as shown in Figure 37.

Figure 37 GenericBlobOut iqPut()



27 From the **File** menu, choose **Save** to save the file.

28 From the **File** menu, choose **Compile** to compile the Collaboration.

View the bottom pane to ensure that there were no compiler errors.

29 From the **File** menu, choose **Close** to close the Java Collaboration Editor and return to the Collaboration Rule. The **Collaboration Rules** and **Initialization file** fields have been completed by closing the Java Collaboration Editor.

30 Click **OK** to save and close the **DBSelect** Collaboration Rule.

5.4.4 Add and Configure the e*Ways

The sample scenario uses three e*Ways:

- **FileIn** – This e*Way retrieves an Event (text file) containing the database select criteria and publishes it to the **Q1** IQ.
- **DBSelect** – This e*Way retrieves the Generic Event (**BLOB**) from the **Q1** IQ. This triggers the e*Way to request information from the external database (via the e*Way Connection) and publishes the results to the **Q2** IQ.
- **FileOut** – This e*Way retrieves the Generic Event (**BLOB**) from the **Q2** IQ then writes it out to a text file on the local file system.

To create the FileIn e*Way

1 In the Components pane of the Schema Designer, select the Control Broker and click the **New e*Way** button.

2 Enter **FileIn** for the component name and click **OK**.

3 Select the newly created e*Way and click the **Properties** button to display the e*Way's properties.

4 Use the **Find** button to select **stcewfile.exe** as the executable file.

5 Click **New** to create a new configuration file.

6 Enter the parameters for the e*Way as shown in Table 4.

Table 4 FileIn e*Way Parameters

Section Name	Parameter	Value
General Settings	AllowIncoming	YES
	AllowOutgoing	NO
	PerformanceTesting	default
Outbound (send) settings	All	default
Poller (inbound) settings	PollDirectory	c:\egate\data\dbSelect_In
	All others	default
Performance Testing	All	default

7 Select **Save** from the **File** menu. Enter **FileIn** as the file name and click **Save**.

8 Select **Promote to Run Time** from the **File** menu. Click **OK** to close the e*Way configuration file editor.

9 In the **Start Up** tab of the e*Way properties, select the **Start automatically** check box.

10 Click **OK** to save the e*Way properties.

To create the DBSelect e*Way

1 In the Components pane of the Schema Designer, select the Control Broker and click the **New e*Way** button.

2 Enter **DBselect** for the component name and click **OK**.

3 Select the newly created e*Way and click the **Properties** button to display the e*Way's properties.

4 Use the **Find** button to select **stceway.exe** as the executable file.

5 Click **New** to create a new configuration file.

6 Enter the parameters for the e*Way as shown in Table 5.

Table 5 DBSelect e*Way Parameters

Section Name	Parameter	Value
JVM Settings	All	default

7 Select **Save** from the **File** menu. Enter **DBSelect** as the file name and click **Save**.

8 Select **Promote to Run Time** from the **File** menu. Click **OK** to close the configuration file editor.

9 In the **Start Up** tab of the Business Object Broker properties, select the **Start automatically** check box.

10 Click **OK** to save the e*Way's properties.

To create the FileOut e*Way

1 In the Components pane of the Schema Designer, select the Control Broker and click the **New e*Way** button.

2 Enter **FileOut** for the component name and click **OK**.

3 Select the newly created e*Way and click the **Properties** button to display the e*Way's properties.

4 Use the **Find** button to select **stcewfile.exe** as the executable file.

5 Click **New** to create a new configuration file.

6 Enter the parameters for the e*Way as shown in Table 6.

Table 6 FileOut e*Way Parameters

Section Name	Parameter	Value
General Settings	AllowIncoming	NO
	AllowOutgoing	YES
	PerformanceTesting	default
Outbound (send) settings	OutputDirectory	c:\egate\data\dbSelect_Out
	OutputFileName	dbSelect%d.dat
Poller (inbound) settings	All	default
Performance Testing	All	default

7 Select **Save** from the **File** menu. Enter **FileOut** as the file name and click **Save**.

8 Select **Promote to Run Time** from the **File** menu. Click **OK** to close the configuration file editor.

9 In the **Start Up** tab of the e*Way properties, select the **Start automatically** check box.

10 Click **OK** to save the e*Way properties.

5.4.5 Add and Configure the e*Way Connections

The sample scenario uses one e*Way Connection:

- **Oracle_eWc** – This e*Way Connection connects the **DBselect** component to the external database and returns the requested records to be published to the **Q2** IQ.

To create the e*Way Connection

1 In the Components pane of the Schema Designer, select the **e*Way Connections** folder.

2 Click the **New e*Way Connection** button to add a new e*Way Connection.

3 Enter **Oracle_eWc** for the component name and click **OK**.

4 Select the newly created e*Way Connection and click the **Properties** button to display the e*Way Connection's properties.

5 Select **Oracle** from the e*Way Connection Type drop-down list.

6 Enter the timeout parameter of the e*Way by entering a number in milliseconds in the dialog box. The default value is 10000.

7 Click **New** to create a new configuration file.

8 Enter the parameters for the e*Way Connection as shown in Table 7.

Table 7 Oracle_eWc e*Way Connection Parameters

Section Name	Parameter	Value
DataSource	class	default
	DriverType	default
	ServerName	Machine name/IP.
	PortNumber	Use the port number from the TNS service (usually 1521).
	DatabaseName	Use SID listed in your tnsnames.ora file.
	user name	Use local settings.
	password	Use local settings.
	timeout	default
connector	type	default
	class	default
	transaction mode	default
	connection establishment mode	default.
	connection inactivity timeout	0
	connection verification interval	5 minutes

9 Select **Save** from the **File** menu. Enter **Oracle_eWc** as the file name and click **Save**.

10 Select **Promote to Run Time** from the **File** menu. Click **OK** to close the e*Way Connection configuration file editor.

11 Click **OK** to save the e*Way Connection's properties.

5.4.6 Add the IQs

The sample scenario uses two IQs

- **Q1** – This IQ queues the inbound Events for the DBSelect e*Way.
- **Q2** – This IQ queues the outbound Events for the FileOut e*Way.

To add the IQs

1 In the components pane of the Schema Designer, select the IQ Manager.

2 Click the **New IQ** button to add a new IQ.

3 Enter the name **Q1** and click **Apply** to save the IQ and leave the New IQ dialog box open.

4 Enter the name **Q2** and click **OK** to save the second IQ.

5 Select the IQ Manager and click the **Properties** button.

6 Select the **Start automatically** check box and click **OK** to save the properties.

5.4.7 Add and Configure the Collaborations

The sample scenario uses three Collaborations:

- **FileIn_PassThru** – This Collaboration uses the **GenericPassThru** Collaboration Rule.
- **DBselect_collab** – This Collaboration uses the **GenericEventToDatabase** Collaboration Rule to execute the **dbCollab.class** Java Collaboration file.
- **FileOut_PassThru** – This Collaboration uses the **GenericPassThru** Collaboration Rule.

To add the FileIn_PassThru Collaboration

1 In the components pane of the Schema Designer, select the **FileIn** e*Way.

2 Click the **New Collaboration** button to create a new Collaboration.

3 Enter the name **FileIn_PassThru** and click **OK**.

4 Select the newly created Collaboration and click the **Properties** button.

5 Select **GenericPassThru** from the drop-down list of Collaboration Rules.

6 Click the upper **Add** button to add a new Subscription.

7 Select the **GenericEvent** Event Type and the **<External>** source.

8 Click the lower **Add** button to add a new Publication

9 Select the **GenericEvent** Event Type and the **Q1** destination.

10 Click **OK** to close the Collaboration's properties.

To add the DBselect_collab Collaboration

1 In the components pane of the Schema Designer, select the **DBSelect** e*Way.

2 Click the **New Collaboration** button to create a new Collaboration.

3 Enter the name **DBselect_collab** and click **OK**.

4 Select the newly created Collaboration and click the **Properties** button.

5 Select **DBSelect** from the drop-down list of Collaboration Rules.

6 Click the upper **Add** button to add a new Subscription.

7 Select the **GenericBlobIn** Instance Name, **GenericBlob** Event Type and the **FileIn_PassThru** source.

8 Click the lower **Add** button to add a new Publication

9 Select the **DBEmployee** Event Type and the **Oracle_eWc** destination.

10 Click the lower **Add** button to add a new Publication

11 Select the **GenericBlobOut** Instance Name**, GenericBlob** Event Type and the **Q2** destination.

12 Click **OK** to close the Collaboration's properties.

To add the FileOut_PassThru Collaboration

1 In the components pane of the Schema Designer, select the **FileOut** e*Way.
2 Click the **New Collaboration** button to create a new Collaboration.
3 Enter the name **FileOut_PassThru** and click **OK**.
4 Select the newly created Collaboration and click the **Properties** button.
5 Select **GenericPassThru** from the drop-down list of Collaboration Rules.
6 Click the upper **Add** button to add a new Subscription.
7 Select the **GenericBlob** Event Type and the **DBSelect_collab** source.
8 Click the lower **Add** button to add a new Publication
9 Select the **GenericBlob** Event Type and the **<External>** destination.
10 Click **OK** to close the Collaboration's properties.

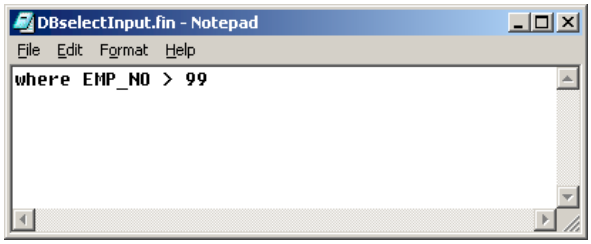
5.4.8 Run the Schema

Running the sample schema requires that a sample input file be created. Once the input file has been created, you can start the Control Broker from a command prompt to execute the schema. After the schema has been run, you can view the output text file to verify the results.

The sample input file

Use a text editor to create an input file to be read by the inbound file e*Way (**FileIn**). This simple input file contains the criteria for the **dbSelect.class** Collaboration's select statement. An example of an input file is shown in Figure 38.

Figure 38 Sample Input File



To start the Control Broker

From a command prompt, type the following command:

```
stccb -ln logical_name -rh registry -rs DBSelect -un user_name
-up password
```

where

logical_name is the logical name of the Control Broker,

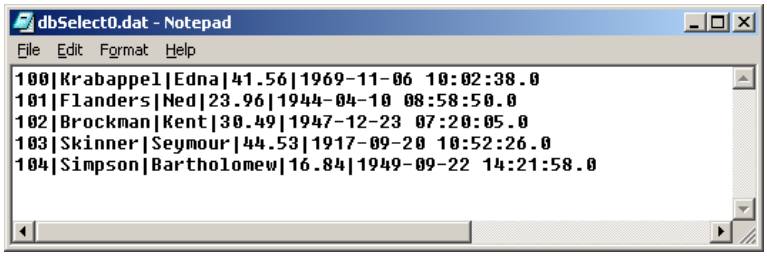
registry is the name of the Registry Host, and

user_name and ***password*** are a valid e*Gate username/password combination.

To verify the results

Use a text editor to view the output file **c:\eGate\data\dbSelect_out\dbSelect0.dat**. Figure 39 shows an example of the records that were returned by the sample schema.

Figure 39 Sample Output File



Chapter 6

Oracle e*Way Methods

The Oracle e*Way contains Java methods that are used to extend the functionality of the e*Way. These methods are contained in the following classes:

6.1 com.stc.eways.jdbcx.StatementAgent Class

```
java.lang.Object
|
+ - - com.stc.eways.jdbcx.StatementAgent
```

All Implemented Interfaces

ResetEventListener, SessionEventListener

Direct Known Subclasses

PreparedStatementAgent, SQLStatementAgent, TableResultSet

```
public abstract class StatementAgent
```

extends java.lang.Object

Implements SessionEventListener, ResetEventListener

Abstract class for other Statement Agent.

Methods of the StatementAgent

resultSetTypeToString

This method gets the symbol string corresponding to the ResultSet type enumeration.

```
public static java.lang.String resultSetTypeToString(int type)
```

Name	Description
type	ResultSet type.

Returns

Enumeration symbol string.

resultSetDirToString

This method gets the symbol string corresponding to the ResultSet direction enumeration.

```
public static java.lang.String resultSetDirToString(int dir)
```

Name	Description
dir	ResultSet scroll directions.

Returns

Enumeration symbol string.

resultSetConcurToString

This method gets the symbol string corresponding to the ResultSet concurrency enumeration.

```
public static java.lang.String resultSetConcurToString(int concur)
```

Name	Description
concur	ResultSet concurrency.

Returns

Enumeration symbol string.

isClosed

This method returns the statement agent's close status.

```
public boolean isClosed()
```

Returns

True if the statement agent is closed.

queryName

This method supplies the name of the listener.

```
public java.lang.String queryName()
```

Specified By

queryName in interface SessionEventListener.

Returns

The listener's class name.

queryDescription

This method gives a description of the query.

```
public java.lang.String queryDescription()
```

Returns

The description of the query.

sessionOpen

Opens the session event handler.

```
public void sessionOpen(SessionEvent evt)
```

Specified by

sessionOpen in interface SessionEventListener

Name	Description
evt	Session event.

sessionClosed

Closes the session event handler.

```
public void sessionClosed(SessionEvent evt)
```

Specified by

sessionClosed in interface SessionEventListener

Name	Description
evt	Session event.

resetRequested

Resets the event handler.

```
public void resetRequested(ResetEvent evt)
```

Specified by

resetRequested in interface ResetEventListener

Name	Description
evt	Requested Reset event.

Throws

java.sql.SQLException

getResultSetType

Returns the result set scroll type.l

```
public int getResultSetType()
```

Returns

ResultSet type

Throws

java.sql.SQLException

getResultSetConcurrency

Returns the result set concurrency mode.

```
public int getResultSetConcurrency()
```

Returns

ResultSet concurrency

Throws

java.sql.SQLException

setEscapeProcessing

Sets escape syntax processing

```
public void setEscapeProcessing (boolean bEscape)
```

Name	Description
bEscape	True to enable False to disable

Throws

java.sql.SQLException

setCursorName

Sets result set cursor name.

```
public void setCursorName(java.lang.String sName)
```

Name	Description
sName	Cursor name.

Throws

java.sql.SQLException

setQueryTimeout

Sets the query timeout duration

```
public void setQueryTimeout(int nInterval)
```

Name	Description
nInterval	The number of seconds before timeout.

Throws

java.sql.SQLException

getFetchDirection

Returns result set fetch direction.

```
public int getFetchDirection()
```

Returns

The fetch direction of the ResultSet: FETCH_FORWARD, FETCH_REVERSE, FETCH_UNKNOWN.

Throws

java.sql.SQLException

setFetchDirection

Sets result set fetch direction.

```
public void setFetchDirection (int iDir)
```

Name	Description
iDir	The fetch direction of the ResultSet: FETCH_FORWARD, FETCH_REVERSE, FETCH_UNKNOWN.

Throws

java.sql.SQLExeption

getFetchSize

Returns the result set prefetch record count.

```
public int getFetchSize()
```

Returns

The fetch size this StatementAgent object set.

Throws

java.sql.SQLException

setFetchSize

Sets the ResultSets prefetch record count.

```
public int getFetchSize()
```

Returns

None.

Throws

java.sql.SQLException

getMaxRows

Returns the maximum number of fetch records.

```
public int getMaxRows()
```

Returns

The maximum number of rows that a ResultSetAgent may contain.

Throws

java.sql.SQLException

setMaxRows

Sets the maximum number of fetch records.

```
public void setMaxRows (int nRow)
```

Name	Description
nRow	The maximum number of rows in the ResultSetAgent.

Throws

java.sql.SQLException

getMaxFieldSize

Returns the maximum field data size.

```
public int getMaxFieldSize()
```

Returns

The maximum number of bytes that a ResultSetAgent column may contain; 0 means no limit.

Throws

java.sql.SQLException

setMaxFieldSize

Sets the maximum field data size.

```
public void setMaxFieldSize (int nSize)
```

Name	Description
nSize	The maximum size for a column in a ResultSetAgent.

Throws

java.sql.SQLException

getUpdateCount

Returns the records count of the last executed statement.

```
public int getUpdateCount()
```

Returns

The number of rows affected by an updated operation. O if no rows were affected or the operation was a DDL command. -1 if the result is a ResultSetAgent or there are no more results.

Throws

java.sql.SQLException

getResultSet

Returns the result set of the last executed statement.

```
public ResultSetAgent getResultSet()
```

Returns

The ResultSetAgent that was produced by the call to the method execute.

Throws

java.sql.SQLExcetpion

getMoreResults

Returns if there are more result sets.

```
public boolean getMoreResults()
```

Returns

True if the next result is a ResultSetAgent; False if it is an integer indicating an update count or there are no more results).

Throws

java.sql.SQLException

getQueryTimeout

Returns query timeout duration.

```
public int getQueryTimeout()
```

Returns

Returns the timeout duration of a query as an integer.

Throws

java.sql.SQLException

clearBatch

Clears the batch operation.

```
public void clearBatch()
```

Throws

java.sql.SQLException

executeBatch

Executes batch statements.

```
public int[] executeBatch ()
```

Returns

An array containing update counts that correspond to the commands that executed successfully. An update count of -2 means the command was successful but that the number of rows affected is unknown.

Throws

java.sql.SQLException

cancel

Cancels a statement that is being executed.

```
public void cancel()
```

Throws

java.sql.SQLException

getWarnings

Returns SQL warning object.

```
public java.sql.SQLWarning getWarnings()
```

Returns

The first SQL warning or null if there are no warnings.

Throws

java.sql.SQLException

clearWarnings

Clear all SQL Warning objects.

```
public void clearWarnings()
```

Throws

java.sql.SQLException

stmtInvoke

Invokes a method of the database Statement object of this ETD.

```
public java.lang.Object stmtInvoke (java.lang.String methodName,
java.lang.Class[] argsCls, java.lang.Object[] args)
```

Name	Description
methodName	The name of the method.
argsCls	Class array for types of formal arguments for method, in the declared order. Can be null if there are no formal arguments. However, cannot invoke constructor here.

Name	Description
args	Object array of formal arguments for method in the declared order. Can be null if there are no formal arguments. However, cannot invoke constructor here.

Returns

The Object instance resulting from the method invocation. Can be null if nothing is returned (void return declaration).

Throws

java.lang.Exception. Whatever exception the invoked method throws.

6.2 com.stc.eways.jdbcx.PreparedStatementAgent Class

```
java.lang.Object
|
+ --com.stc.eways.jdbcx.StatementAgent
   |
    + -- com.stc.eways.jdbcx.PreparedStatementAgent
```

All Implemented Interfaces

ResetEventListener, SessionEventListener

Direct Known Subclasses

CallableStatementAgent

```
public class PreparedStatementAgent
```

extends StatementAgent

Agent hosts PreparedStatement Object

Methods of the PreparedStatementAgent

sessionOpen

Opens the session event handler.

```
public void sessionOpen(SessionEvent evt)
```

Overrides

sessionOpen in class StatementAgent

Name	Description
evt	Session event.

setNull

Nullify value of indexed parameter.

```
public void setNull(int index, int type)
```

Name	Description
index	Parameter index starting from 1.
type	A JDBC type defined by java.sql.Types

Throws

java.sql.SQLException

setNull

Nullify value of indexed parameter.

```
public void setNul(int index, int type, java.lang.String tname)
```

Name	Description
index	Parameter index starting from 1.
type	A JDBC type defined by java.sql.Types
tname	The fully-qualified name of the parameter being set. If type is not REF, STRUCT, DISTINCT, or JAVA_OBJECT, this parameter will be ignored.

Throws

java.sql.SQLException

setObject

Sets value of indexed parameter with an object.

```
public void setObject(int index, java.lang.Object ob)
```

Name	Description
index	Parameter index starting from 1.
ob	An instance of a Java Object containing the input parameter value.

Throws

java.sql.SQLException

setObject

Sets value of indexed parameter with an object.

```
public void setObject(int index, java.lang.Object ob, int iType)
```

Name	Description
index	Parameter index starting from 1.
ob	An instance of a Java Object containing the input parameter value.
iType	A JDBC type defined by java.sql.Types

Throws

java.sql.SQLException

setObject

Sets value of indexed parameter with an object.

```
public void setObject(int index, java.lang.Object ob, int iType, int
iScale)
```

Name	Description
index	Parameter index starting from 1.
ob	An instance of a Java Object containing the input parameter value.
iType	A JDBC type defined by java.sql.Types
iScale	The number of digits to the right of the decimal point. Only applied to DECIMAL and NUMERIC types

Throws

java.sql.SQLException

setBoolean

Sets the boolean value of the indexed parameter.

```
public void setBoolean(int index, boolean b)
```

Name	Description
index	Parameter index starting from 1.

Name	Description
b	true or false.

Throws

java.sql.SQLException

setByte

Sets the byte value of the indexed parameter.

```
public void setByte(int index, byte byt)
```

Name	Description
index	Parameter index starting from 1.
byt	The byte parameter value to be set.

Throws

java.sql.SQLException

setShort

Sets the short value of the indexed parameter.

```
public void setShort(int index, short si)
```

Name	Description
index	Parameter index starting from 1.
si	The short parameter value to be set.

Throws

java.sql.SQLException

setInt

Sets the integer value of the indexed parameter.

```
public void setInt(int index, int i)
```

Name	Description
index	Parameter index starting from 1.
i	The integer parameter value to be set.

Throws

java.sql.SQLException

setLong

Sets the long value of the indexed parameter.

```
public void setLong(int index, long l)
```

Name	Description
index	Parameter index starting from 1.
l	The long parameter value to be set.

Throws

java.sql.SQLException

setFloat

Sets the float value of the indexed parameter.

```
public void setFloat(int index, float f)
```

Name	Description
index	Parameter index starting from 1.
f	The float parameter value to be set.

Throws

java.sql.SQLException

setDouble

Sets the double value of the indexed parameter.

```
public void setDouble(int index, double d)
```

Name	Description
index	Parameter index starting from 1.
d	The double parameter value to be set.

Throws

java.sql.SQLException

setBigDecimal

Sets the decimal value of the indexed parameter.

```
public void setBigDecimal(int index, java.math.BigDecimal dec)
```

Name	Description
index	Parameter index starting from 1.
dec	The BigDecimal parameter value to be set.

Throws

java.sql.SQLException

setDate

Sets the date value of the indexed parameter.

```
public void setDate(int index, java.sql.Date date)
```

Name	Description
index	Parameter index starting from 1.
date	The Date parameter value to be set.

Throws

java.sql.SQLException

setDate

Sets the date value of indexed parameter with time zone from calendar.

```
public void setDate(int index, java.sql.Date date, java.util.Calendar
cal)
```

Name	Description
index	Parameter index starting from 1.
date	The Date parameter value to be set.
cal	The calender object used to construct the date.

Throws

java.sql.SQLException

setTime

Sets the time value of the indexed parameter.

```
public void setTime(int index, java.sql.Time t)
```

Name	Description
index	Parameter index starting from 1.
t	The Time parameter value to be set.

Throws

java.sql.SQLException

setTime

Sets the time value of the indexed parameter.

```
public void setTime(int index, java.sql.Time t, java.util.Calendar
cal)
```

Name	Description
index	Parameter index starting from 1.
t	The Time parameter value to be set.
cal	The Calendar object used to construct the time.

Throws

java.sql.SQLException

setTimestamp

Sets the timestamp value of the indexed parameter.

```
public void setTimestamp(int index, java.sql.Timestamp ts)
```

Name	Description
index	Parameter index starting from 1.
ts	The Timestamp parameter value to be set.

Throws

java.sql.SQLException

setTimestamp

Sets the timestamp value of the indexed parameter with the time zone from the calendar.

```
public void setTimestamp(int index, java.sql.timestamp ts,
java.util.Calendar cal)
```

Name	Description
index	Parameter index starting from 1.
ts	The Timestamp parameter value to be set.
cal	The Calendar object used to construct the timestamp.

Throws

java.sql.SQLException

setString

Sets the string value of the indexed parameter.

```
public void setString(int index, java.lang.String s)
```

Name	Description
index	Parameter index starting from 1.
s	The String parameter value to be set.

Throws

java.sql.SQLException

setBytes

Sets the byte array value of the indexed parameter.

```
public void setBytes(int index, byte[] bytes)
```

Name	Description
index	Parameter index starting from 1.
bytes	The byte array parameter value to be set.

Throws

java.sql.SQLException

setAsciiStream

Sets the character value of the indexed parameter with an input stream and specified length.

```
public void setAsciiStream(int index, java.io.InputStream is, int
length)
```

Name	Description
index	Parameter index starting from 1.
is	The InputStream that contains the Ascii parameter value to be set.
length	The number of bytes to be read from the stream and sent to the database.

Throws

java.sql.SQLException

setBinaryStream

Sets the binary value of the indexed parameter with an input stream and specified length.

```
public void setBinaryStream(int index, java.io.InputStream is, int
length)
```

Name	Description
index	Parameter index starting from 1.
is	The InputStream that contains the binary parameter value to be set.
length	The number of bytes to be read from the stream and sent to the database.

Throws

java.sql.SQLException

setCharacterStream

Sets the character value of the indexed parameter with a reader stream and specified length.

```
public void setCharacterStream(int index, java.io.Reader rd, int
length)
```

Name	Description
index	Parameter index starting from 1.
rd	The Reader that contains the Unicode parameter value to be set.
length	The number of characters to be read from the stream and sent to the database.

Throws

java.sql.SQLException

setArray

Sets the Array value of the indexed parameter.

```
public void setArray(int index, java.sql.Array a)
```

Name	Description
index	Parameter index starting from 1.
a	The Array value to be set.

Throws

java.sql.SQLException

setBlob

Sets the Blob value of the indexed parameter.

```
public void setBlob(int index, java.sql.Blob blob)
```

Name	Description
index	Parameter index starting from 1.
blob	The Blob value to be set.

Throws

java.sql.SQLException

setClob

Sets the Clob value of the indexed parameter.

```
public void setClob(int index, java.sql.Clob clob)
```

Name	Description
index	Parameter index starting from 1.
clob	The Clob value to be set.

Throws

java.sql.SQLException

setRef

Sets the Ref value of the indexed parameter.

```
public void setRef(int index, java.sql.Ref ref)
```

Name	Description
index	Parameter index starting from 1.
ref	The Ref parameter value to be set.

Throws

java.sql.SQLException

closeResultSet

Immediately releases ResultSet object's resources, close the cursor.

```
public void closeResultSet()
```

Throws

java.sql.SQLException

clearParameters

Clears the parameters of all values.

```
public void clearParameters()
```

Throws

java.sql.SQLException

addBatch

Adds a set of parameters to the list of commands to be sent as a batch.

```
public void addBatch()
```

Throws

java.sql.SQLException

enableResultSetsAndUpdateCounts

Enables ResultSets and UpdateCounts to be received after an execute call.

```
public void enableResultSetsAndUpdateCounts()
```

Returns

ResultSets and UpdateCounts.

Throws

java.sql.SQLException

enableResultSetsOnly

Enables ResultSets to be received after an execute call.

```
public void enableResultSetsOnly()
```

Returns

ResultSets.

Throws

java.sql.SQLException

enableUpdateCountsOnly

Enables UpdateCountsOnly to be received after an execute call.

```
public void UpdateCountsOnly()
```

Returns

UpdateCounts.

Throws

java.sql.SQLException

execute

Executes the Prepared SQL statement.

```
public void execute()
```

Throws

java.sql.SQLException

executeQuery

Executes the prepared SQL query and returns a ResultSetAgent that contains the generated result set.

```
public ResultSetAgent executeQuery()
```

Returns

ResultSetAgent or null.

Throws

java.sql.SQLException

executeUpdate

Executes the prepared SQL statement and returns the number of rows that were affected.

```
public int executeUpdate()
```

Returns

The number of rows affected by the update operation; 0 if no rows were affected.

Throws

java.sql.SQLException

getExecuteRetValue

Returns the value of the execute method.

```
public boolean getExecuteRetValue()
```

Returns

The execute method.

Throws

java.sql.SQLException

getUpdateCount

Returns the record count from the last executed statement.

```
public int getUpdateCount()
```

Returns

The record count.

Throws

java.sql.SQLException

matchResultSetSignature

Matches a ResultSet's column signature with the current one and points the current ResultSet Agent to it.

```
public boolean matchResultSetSignature(int ord)
```

Name	Description
ord	The ordinal position of the particular ResultSet.

Returns

None.

Throws

java.sql.SQLException

resultsAvailable

Determines whether results (UpdateCounts and/or ResultSets) are available after an execute.

```
public boolean resultsAvailable()
```

Returns

True if UpdateCounts and/or ResultSets are available.

Throws

java.sql.SQLException

6.3 com.stc.eways.jdbcx.PreparedStatementResultSet Class

java.lang.Object

|

+ -- **com.stc.eways.jdbcx.PreparedStatementResultSet**

```
public abstract class PreparedStatementResultSet
```

extends java.lang.Object

Base class for Result Set returned from a Prepared Statement execution.

Constructors of PreparedStatementResultSet

PreparedStatementResultSet

Methods of PreparedStatementResultSet

isBeforeFirst on page 96
isFirst on page 95
isLast on page 96
last on page 96
next on page 94
previous on page 94
refreshRow on page 138
relative on page 95
setFetchDirection on page 71
setFetchSize on page 72
updateRow on page 115
wasNull on page 114
setResultSetAgent on page 116

Constructor PreparedStatementResultSet

Constructs a Prepared Statement Result Set object.

```
public PreparedStatementResultSet(ResultSetAgent rsAgent)
```

Name	Description
rsAgent	The ResultSetAgent underlying control.

getMetaData

Retrieves a ResultSetMetaData object that contains ResultSet properties.

```
public java.sql.ResultSetMetaData getMetaData()
```

Returns

ResultSetMetaData object

Throws

java.sql.SQLException

getConcurrency

Gets the concurrency mode for this ResultSet object.

```
public int getConcurrency()
```

Returns

Concurrency mode

Throws

java.sql.SQLException

getFetchDirection

Gets the direction suggested to the driver as the row fetch direction.

```
public int getFetchDirection()
```

Returns

Row fetch direction

Throws

java.sql.SQLException

setFetchDirection

Gives the driver a hint as to the row process direction.

```
public void setFetchDirection(int iDir)
```

Name	Description
iDir	Fetch direction to use.

Throws

java.sql.SQLException

getFetchSize

Gets the number of rows to fetch suggested to the driver.

```
public int getFetchSize()
```

Returns

Number of rows to fetch at a time.

Throws

java.sql.SQLException

setFetchSize

Gives the drivers a hint as to the number of rows that should be fetched each time.

```
public void setFetchSize(int nSize)
```

Name	Description
nSize	Number of rows to fetch at a time.

Throws

java.sql.SQLException

getCursorName

Retrieves the name for the cursor associated with this ResultSet object.

```
public java.lang.String getCursorName()
```

Returns

Name of cursor

Throws

java.sql.SQLException

close

Immediately releases a ResultSet object's resources.

```
public void close()
```

Throws

java.sql.SQLException

next

Moves the cursor to the next row of the result set.

```
public boolean next()
```

Returns

true if successful

Throws

java.sql.SQLException

previous

Moves the cursor to the previous row of the result set.

```
public boolean previous()
```

Returns

true if successful

Throws

java.sql.SQLException

absolute

Moves the cursor to the specified row of the result set.

```
public boolean absolute(int index)
```

Returns

true if successful

Throws

java.sql.SQLException

---

available

Determines whether data for this particular ResultSet is available after an execute command.

```
public boolean available()
```

Returns

true if data is available

Throws

java.sql.SQLException

---

relative

Moves the cursor to the specified row relative to the current row of the result set.

```
public boolean relative(int index)
```

Returns

true if successful

Throws

java.sql.SQLException

---

first

Moves the cursor to the first row of the result set.

```
public boolean first()
```

Returns

true if successful

Throws

java.sql.SQLException

---

isFirst

Determines whether the cursor is on the first row of the result set.

```
public boolean isFirst()
```

Returns

true if on the first row.

Throws

java.sql.SQLException

last

Moves the cursor to the last row of the result set.

```
public boolean last()
```

Returns

true if successful

Throws

java.sql.SQLException

isLast

Determines whether the cursor is on the last row of the result set.

```
public boolean isLast()
```

Returns

true if on the last row

Throws

java.sql.SQLException

beforeFirst

Moves the cursor before the first row of the result set.

```
public void beforeFirst()
```

Throws

java.sql.SQLException

isBeforeFirst

Determines whether the cursor is before the first row of the result set.

```
public boolean isBeforeFirst()
```

Returns

true if before the first row

Throws

java.sql.SQLException

afterLast

Moves the cursor after the last row of the result set.

```
public void afterLast()
```

Throws

java.sql.SQLException

isAfterLast

Determines whether the cursor is after the last row of the result set.

```
public boolean isAfterLast()
```

Returns

true if after the last row

Throws

java.sql.SQLException

getType

Retrieves the scroll type of cursor associated with the result set.

```
public int getType()
```

Returns

Scroll type of cursor.

Throws

java.sql.SQLException

findColumn

Returns the column index for the named column in the result set.

```
public int findColumn(java.lang.String index)
```

Name	Description
index	Column name.

Returns

Corresponding column index.

Throws

java.sql.SQLException

getObject

Gets the object value of the specified column.

```
public java.lang.Object getObject(int index)
```

Name	Description
index	Column index.

Returns

Object form of column value.

Throws

java.sql.SQLException

getObject

Gets the object value of the specified column.

```
public java.lang.Object getObject(java.lang.String index)
```

Name	Description
index	Column index.

Returns

Object form of column value.

Throws

java.sql.SQLException

getObject

Gets the object value of the specified column using the given type map.

```
public java.lang.Object getObject(int index, java.util.Map.map)
```

Name	Description
index	Column index.
map	Type map.

Returns

Object form of column value.

Throws

java.sql.SQLException

getObject

Gets the object value of the specified column using the given type map.

```
public java.lang.Object getObject(java.lang.String index,
java.util.Map map)
```

Name	Description
index	Column index.
map	Type map.

Returns

Object form of column value.

Throws

java.sql.SQLException

getBoolean

Gets the boolean value of the specified column.

```
public boolean getBoolean(int index)
```

Name	Description
index	Column index.

Returns

Boolean value of the column.

Throws

java.sql.SQLException

getBoolean

Gets the boolean value of the specified column.

```
public boolean getBoolean(java.lang.String index))
```

Name	Description
index	Column name.

Returns

Boolean value of the column.

Throws

java.sql.SQLException

getByte

Gets the byte value of the specified column.

```
public byte getByte(int index)
```

Name	Description
index	Column index.

Returns

Boolean value of the column.

Throws

java.sql.SQLException

getShort

Gets the short value of the specified column.

```
public short getShort(int index)
```

Name	Description
index	Column index.

Returns

Short value of the column.

Throws

java.sql.SQLException

getShort

Gets the short value of the specified column.

```
public short getShort(java.lang.String index)
```

Name	Description
index	Column name.

Returns

Short value of the column.

Throws

java.sql.SQLException

getInt

Gets the integer value of the specified column.

```
public int getInt(int index)
```

Name	Description
index	Column index.

Returns

Int value of the column.

Throws

java.sql.SQLException

getInt

Gets the integer value of the specified column.

```
public int getInt(java.lang.String index)
```

Name	Description
index	Column name.

Returns

Int value of the column.

Throws

java.sql.SQLException

getLong

Gets the long value of the specified column.

```
public long getLong(int index)
```

Name	Description
index	Column index.

Returns

Long value of the column.

Throws

java.sql.SQLException

getLong

Gets the long value of the specified column.

```
public long getLong(java.lang.String index)
```

Name	Description
index	Column name.

Returns

Long value of the column.

Throws

java.sql.SQLException

getFloat

Gets the float value of the specified column.

```
public float getFloat(int index)
```

Name	Description
index	Column index.

Returns

Float value of the column.

Throws

java.sql.SQLException

getFloat

Gets the float value of the specified column.

```
public float getFloat(java.lang.String index)
```

Name	Description
index	Column name.

Returns

Float value of the column.

Throws

java.sql.SQLException

getDouble

Gets the double value of the specified column.

```
public double getDouble(int index)
```

Name	Description
index	Column index.

Returns

Double value of the column.

Throws

java.sql.SQLException

getBigDecimal

Gets the decimal value of the specified column.

```
public java.math.BigDecimal getBigDecimal(int index)
```

Name	Description
index	Column index.

Returns

Big decimal value of the column.

Throws

java.sql.SQLException

getBigDecimal

Gets the decimal value of the specified column.

```
public java.math.BigDecimal getBigDecimal(java.lang.String index)
```

Name	Description
index	Column name.

Returns

Big decimal value of the column.

Throws

java.sql.SQLException

getDate

Gets the date value of the specified column.

```
public java.sql.Date getDate(int index)
```

Name	Description
index	Column index.

Returns

Date value of the column.

Throws

java.sql.SQLException

getDate

Gets the date value of the specified column.

```
public java.sql.Date getDate(java.lang.String index)
```

Name	Description
index	Column name.

Returns

Date value of the column.

Throws

java.sql.SQLException

getDate

Gets the date value of the specified column using the time zone from the calendar.

```
public java.sql.Date getDate(java.lang.String index,
java.util.Calendar calendar)
```

Name	Description
index	Column name.
calendar	Calendar to use.

Returns

Date value of the column.

Throws

java.sql.SQLException

getTime

Gets the time value of the specified column.

```
public java.sql.Time getTime(int index)
```

Name	Description
index	Column index.

Returns

Time value of the column.

Throws

java.sql.SQLException

getTime

Gets the time value of the specified column.

```
public java.sql.Time getTime(java.lang.String index)
```

Name	Description
index	Column name.

Returns

Time value of the column.

Throws

java.sql.SQLException

getTime

Gets the time value of the specified column.

```
public java.sql.Time getTime(int index, java.util.Calendar calendar)
```

Name	Description
index	Column index.
calendar	Calendar to use.

Returns

Time value of the column.

Throws

java.sql.SQLException

getTime

Gets the time value of the specified column.

```
public java.sql.Time getTime(java.lang.String index,
java.util.Calendar calendar)
```

Name	Description
index	Column name.
calendar	Calendar to use.

Returns

Time value of the column.

Throws

java.sql.SQLException

---

getTimestamp

Gets the timestamp value of the specified column.

```
public java.sql.Timestamp getTimestamp(int index)
```

Name	Description
index	Column index.

Returns

The timestamp value of the column.

Throws

java.sql.SQLException

---

getTimestamp

Gets the timestamp value of the specified column.

```
public java.sql.Timestamp getTimestamp(java.lang.String index)
```

Name	Description
index	Column name.

Returns

The timestamp value of the column.

Throws

java.sql.SQLException

---

getTimestamp

Gets the timestamp value of the specified column using the time zone from the calendar.

```
public java.sql.Timestamp getTimestamp(int index, java.util.Calendar
calendar)
```

Name	Description
index	Column index.

Returns

The timestamp value of the column.

Throws

java.sql.SQLException

getTimestamp

Gets the timestamp value of the specified column using the time zone from the calendar.

```
public java.sql.Timestamp getTimestamp(java.lang.String index,
java.util.Calendar calendar)
```

Name	Description
index	Column name.
calendar	Calendar to use.

Returns

The timestamp value of the column.

Throws

java.sql.SQLException

getString

Gets the string value of the specified column.

```
public java.lang.String getString(int index)
```

Name	Description
index	Column index.

Returns

Returns the String value of the column.

Throws

java.sql.SQLException

getString

Gets the string value of the specified column.

```
public java.lang.String getString(java.lang.String index)
```

Name	Description
index	Column name.

Returns

Returns the String value of the column.

Throws

java.sql.SQLException

getBytes

Gets the byte array value of the specified column.

```
public byte[] getBytes(int index)
```

Name	Description
index	Column index.

Returns

Byte array value of the column.

Throws

java.sql.SQLException

getBytes

Gets the byte array value of the specified column.

```
public byte[] getBytes(java.lang.String index)
```

Name	Description
index	Column name.

Returns

Byte array value of the column.

Throws

java.sql.SQLException

getAsciiStream

Retrieves the value of the specified column value as a stream of ASCII characters.

```
public java.io.InputStream getAsciiStream(int index)
```

Name	Description
index	Column index.

Returns

ASCII output stream value of the column.

Throws

java.sql.SQLException

getAsciiStream

Retrieves the value of the specified column value as a stream of ASCII characters.

```
public java.io.InputStream getAsciiStream(java.lang.String index)
```

Name	Description
index	Column name.

Returns

ASCII output stream value of the column.

Throws

java.sql.SQLException

getBinaryStream

Retrieves the value of the specified column as a stream of uninterpreted bytes.

```
public java.io.InputStream getBinaryStream(int index)
```

Name	Description
index	Column index.

Returns

Binary out steam value of the column.

Throws

java.sql.SQLException

getBinaryStream

Retrieves the value of the specified column as a stream of uninterpreted bytes.

```
public java.io.InputStream getBinaryStream(java.lang.String index)
```

Name	Description
index	Column name.

Returns

Binary out steam value of the column.

Throws

java.sql.SQLException

getCharacterStream

Retrieves the value of the specified column as a Reader object.

```
public java.io.Reader getCharacterStream(int index)
```

Name	Description
index	Column index.

Returns

Reader for value in the column.

Throws

java.sql.SQLException

getArray

Gets the Array value of the specified column.

```
public java.sql.Array getArray(int index)
```

Name	Description
index	Column index.

Returns

Array value of the column.

Throws

java.sql.SQLException

getBlob

Gets the Blob value of the specified column.

```
public java.sql.Blob getBlob(int index)
```

Name	Description
index	Column index.

Returns

Blob value of the column.

Throws

java.sql.SQLException

getBlob

Gets the Blob value of the specified column.

```
public java.sql.Blob getBlob(java.lang.String index)
```

Name	Description
index	Column name.

Returns

Blob value of the column.

Throws

java.sql.SQLException

getClob

Gets the Clob value of the specified column.

```
public java.sql.Clob getClob(int index)
```

Name	Description
index	Column index.

Returns

Clob value of the column.

Throws

java.sql.SQLException

getClob

Gets the Clob value of the specified column.

```
public java.sql.Clob getClob(java.lang.String index)
```

Name	Description
index	Column name.

Returns

Clob value of the column.

Throws

java.sql.SQLException

getRef

Gets the Ref value of the specified column.

```
public java.sql.Ref getRef(int index)
```

Name	Description
index	Column index.

Returns

Ref value of the column.

Throws

java.sql.SQLException

getRef

Gets the Ref value of the specified column.

```
public java.sql.Ref getRef(java.lang.String index)
```

Name	Description
index	Column name.

Returns

Ref value of the column.

Throws

java.sql.SQLException

wasNull

Checks to see if the last value read was SQL NULL or not.

```
public boolean wasNull()
```

Returns

true if SQL NULL.

Throws

java.sql.SQLException

getWarnings

Gets the first SQL Warning that has been reported for this object.

```
public java.sql.SQLWarning getWarnings()
```

Returns

SQL warning.

Throws

java.sql.SQLException

clearWarnings

Clears any warnings reported on this object.

```
public void clearWarnings()
```

Throws

java.sql.SQLException

getRow

Retrieves the current row number in the result set.

```
public int getRow()
```

Returns

Current row number

Throws

java.sql.SQLException

refreshRow

Replaces the values int the current row of the result set with their current values in the database.

```
public void refreshRow()
```

Throws

java.sql.SQLException

insertRow

Inserts the contents of the insert row into the result set and the database.

```
public void insertRow()
```

Throws

java.sql.SQLException

updateRow

Updates the underlying database with the new contents of the current row.

```
public void updateRow()
```

Throws

java.sql.SQLException

deleteRow

Deletes the current row from the result set and the underlying database.

public void deleteRow()

Throws

java.sql.SQLException

setResultSetAgent

Sets the underlying ResultSetAgent control for this ResultSet.

```
public void setResultSetAgent(ResultSetAgent rsAgent)
```

Name	Description
rsAgent	The ResultSetAgent underlying control.

Returns

None

Throws

java.sql.SQLException

updateAsciiStream

6.4 com.stc.eways.jdbcx.SqlStatementAgent Class

java.lang.Object

|

+ -- com.stc.eways.jdbcx.StatementAgent

|

+ -- **com.stc.eways.jdbcx.SqlStatementAgent**

All Implemented Interfaces

ResetEventListener, SessionEventListener

public class SqlStatementAgent

extends StatementAgent

SQLStatement Agent that hosts a managed Statement object.

Constructors of the SqlStatementAgent

SqlStatementAgent

SqlStatementAgent

Methods of the SqlStatementAgent

Constructor SqlStatementAgent

Creates new SQLStatementAgent with scroll direction TYPE_FORWARD_ONLY and concurrency CONCUR_READ_ONLY.

```
public SqlStatementAgent(Session session)
```

Name	Description
session	Connection session.

Constructor SqlStatementAgent

Creates a new SQLStatementAgent.

```
public SqlStatementAgent(Session session, int iScroll, int iConcur)
```

Name	Description
session	Connection session.
iScroll	Scroll direction: TYPE_FORWARD_ONLY, TYPE_SCROLL_INSENSITIVE, TYPE_SCROLL_SENSITIVE.
iConcur	Concurrency: CONCUR_READ_ONLY, CONCUR_UPDATABLE.

execute

Executes the specified SQL statement.

```
public boolean execute(java.lang.String sSql)
```

Name	Description
sSql	SQL statement.

Returns

true if the first result is a ResultSetAgent or false if it is an integer.

Throws

java.sql.SQLException

executeQuery

Executes the specified SQL query and returns a ResultSetAgent that contains the generated result set.

```
public ResultSetAgent executeQuery(java.lang.String sSql)
```

Name	Description
sSql	SQL statement.

Returns

A ResultSetAgent or null

Throws

java.sql.SQLException

executeUpdate

Executes the specified SQL statement and returns the number of rows that were affected.

```
public int executeUpdate(jave.lang.String sSql)
```

Name	Description
sSql	SQL statement.

Returns

The number of rows affected by the update operation; 0 if no rows were affected.

Throws

java.sql.SQLException

addBatch

Adds the specified SQL statement to the list of commands to be sent as a batch.

```
public void addBatch(java.lang.String sSql)
```

Name	Description
sSql	SQL statement.

Throws

java.sql.SQLException

6.5 com.stc.eways.jdbcx.CallableStatementAgent Class

java.lang.Object

|

+ -- com.stc.eways.jdbcx.StatementAgent

|

+ -- com.stc.eways.jdbcx.PreparedStatementAgent

|

+ -- **com.stc.eways.jdbcx.CallableStatementAgent**

All Implemented Interfaces

com.stc.eways.jdbcx.ResetEventListener

com.stc.eways.jdbcx.SessionEventListener

Direct Known Subclasses

StoredProcedureAgent

```
public abstract class CallableStatementAgent
```

extends PreparedStatementAgent

Agent hosts CallableStatement interface

Constructors of the CallableStatementAgent

CallableStatementAgent

CallableStatementAgent

CallableStatementAgent

Methods of the CallableStatementAgent

Constructor CallableStatementAgent

Creates new CallableStatementAgent with scroll direction TYPE_FORWARD_ONLY and concurrency CONCUR_READ_ONLY.

```
public CallableStatementAgent(Session session, java.lang.String
sCommand)
```

Name	Description
session	Connection session.
sCommand	The Call statement used to invoke a stored procedure.

Constructor CallableStatementAgent

Creates a new CallableStatementAgent.

```
public CallableStatementAgent(Session session, int iScroll, int
iConcur)
```

Name	Description
session	Connection session.
iScroll	Ignored.
iConcur	Ignored

Constructor CallableStatement Agent

Creates a new CallableStatementAgent.

```
public CallableStatementAgent(Session session, java.lang.String
sCommand, int iScroll, int iConcur)
```

Name	Description
session	Connection session.
sCommand	The Call statement used to invoke a stored procedure.
iScroll	Scroll direction: TYPE_FORWARD_ONLY, TYPE SCROLL_INSENSITIVE, TYPE_SCROLL_SENSITIVE
iConcur	Concurrency: CONCUR_READ_ONLY, CONCUR_UPDATEABLE

clearParameters

When using an Oracle native driver that supports clobs, temporary clobs are freed when all parameter values are cleared.

```
public void clearParameters()
```

Throws

java.sql.SQLException

sessionOpen

Opens the session event handler.

```
public void sessionOpen(SessionEvent evt)
```

Overrides

sessionOpen in class PreparedStatementAgent

Name	Description
evt	Session event.

sessionClosed

When using an Oracle native driver that supports clobs, temporary clobs are freed when all parameter values are cleared.

```
public void sessionClosed(com.stc.eways.jdbcx.SessionEvent evt)
```

Throws

java.sql.SQLException

registerOutParameter

Registers the indexed OUT parameter with specified type.

```
public void registerOutParameter(int index, int iType)
```

Name	Description
index	Parameter index starting from 1.
iType	A JDBC type defined by java.sql.Types.

Throws

java.sql.SQLException

registerOutParameter

Registers the indexed OUT parameter with specified type and scale.

```
public void registerOutParameter(int index, int iType, int iScale)
```

Name	Description
index	Parameter index starting from 1.
iType	A JDBC type defined by java.sql.Types.
iScale	The number of digits to the right of the decimal point. Only applied to DECIMAL and NUMERIC types.

Throws

java.sql.SQLException

registerOutParameter

Registers the indexed OUT parameter with specified user-named type or REF type.

```
public void registerOutParameter(int index, int iType,
java.lang.String sType)
```

Name	Description
index	Parameter index starting from 1.

Name	Description
iType	A JDBC type defined by java.sql.Types.
tName	The fully-qualified name of the parameter being set. It is intended to be used by REF, STRUCT, DISTINCT, or JAVA_OBJECT.

Throws

java.sql.SQLException

wasNull

Returns whether or not the last OUT parameter read had the SQL NULL value.

```
public boolean wasNull()
```

Returns

true if the parameter read is SQL NULL; otherwise, false

Throws

java.sql.SQLException

getObject

Gets the value of the indexed parameter as an instance of Object.

```
public java.lang.Object getObject(int index)
```

Name	Description
index	Parameter index starting from 1.

Returns

The Object value

Throws

java.sql.SQLException

getObject

Gets the value of the indexed parameter as an instance of Object and uses map for the customer mapping of the parameter value.

```
public java.lang.Object getObject(int index, java.util.Map map)
```

Name	Description
index	Parameter index starting from 1.
map	A Map object for mapping from SQL type names for user-defined types to classes in the Java programming language.

Returns

An Object value

Throws

java.sql.SQLException

getBoolean

Gets the boolean value of the indexed parameter.

```
public boolean getBoolean(int index)
```

Name	Description
index	Parameter index starting from 1.

Returns

A boolean value

Throws

java.sql.SQLException

getByte

Gets byte value of the indexed parameter.

```
public byte getByte(int index)
```

Name	Description
index	Parameter index starting from 1.

Returns

A byte value

Throws

java.sql.SQLException

getShort

Gets short value of the indexed parameter.

```
public short getShort(int index)
```

Returns

A short value

Throws

java.sql.SQLException

getInt

Gets integer value of the indexed parameter.

```
public int getInt(int index)
```

Name	Description
index	Parameter index starting from 1.

Returns

A int value

Throws

java.sql.SQLException

getLong

Gets long value of the indexed parameter.

```
public long getLong(int index)
```

Name	Description
index	Parameter index starting from 1.

Returns

A long value

Throws

java.sql.SQLException

getFloat

Gets float value of the indexed parameter.

```
public float getFloat(int index)
```

Name	Description
index	Parameter index starting from 1.

Returns

A float value

Throws

java.sql.SQLException

getDouble

Gets double value of the indexed parameter.

```
public double getDouble(int index)
```

Name	Description
index	Parameter index starting from 1.

Returns

A float value

Throws

java.sql.SQLException

getBigDecimal

Gets decimal value of the indexed parameter.

```
public java.math.BigDecimal getBigDecimal(int index)
```

Name	Description
index	Parameter index starting from 1.

Returns

A BigDecimal object

Throws

java.sql.SQLException

getDate

Gets date value of the indexed parameter.

```
public java.sql.Date getDate(int index)
```

Name	Description
index	Parameter index starting from 1.

Returns

A Date object

Throws

java.sql.SQLException

getDate

Gets date value of the indexed parameter with time zone from calendar.

```
public java.sql.Date getDate(int index, java.util.Calendar calendar)
```

Name	Description
index	Parameter index starting from 1.
cal	The Calendar object used to construct the timestamp.

Returns

A Date object

Throws

java.sql.SQLException

getTime

Gets time value of the indexed parameter.

```
public java.sql.Time getTime(int index)
```

Name	Description
index	Parameter index starting from 1.

Returns

A Time object

Throws

java.sql.SQLException

getTime

Gets time value of the indexed parameter with time zone from calendar.

```
public java.sql.Time getTime(int index, java.util.Calendar calendar)
```

Name	Description
index	Parameter index starting from 1.
cal	The Calendar object used to construct the timestamp.

Returns

A Time object

Throws

java.sql.SQLException

getTimestamp

Gets timestamp value of the indexed parameter.

```
public java.sql.timestamp getTimestamp(int index)
```

Name	Description
index	Parameter index starting from 1.

Returns

A Timestamp object

Throws

java.sql.SQLException

getTimestamp

Gets timestamp value of the indexed parameter.

```
public java.sql.timestamp getTimestamp(int index, java.util.Calendar
calendar)
```

Name	Description
index	Parameter index starting from 1.
cal	The Calendar object used to construct the timestamp.

Returns

A Timestamp object

Throws

java.sql.SQLException

getString

Gets string value of the indexed parameter.

```
public java.lang.String getString(int index)
```

Name	Description
index	Parameter index starting from 1.

Returns

A String object

Throws

java.sql.SQLException

getBytes

Gets byte array value of the indexed parameter.

```
public byte[] getBytes(int index)
```

Name	Description
index	Parameter index starting from 1.

Returns

An array of bytes

Throws

java.sql.SQLException

getArray

Gets Array value of the indexed parameter.

```
public java.sql.Array getArray(int index)
```

Name	Description
index	Parameter index starting from 1.

Returns

An Array object

Throws

java.sql.SQLException

getBlob

Gets Blob value of the indexed parameter.

```
public java.sql.Blob getBlob(int index)
```

Name	Description
index	Parameter index starting from 1.

Returns

A Blob object

Throws

java.sql.SQLException

getClob

Gets Clob value of the indexed parameter.

```
public java.sql.Clob getClob(int index)
```

Name	Description
index	Parameter index starting from 1.

Returns

A Blob object

Throws

java.sql.SQLException

setClob

Sets the Clob value of the indexed parameter.

public void setClob(int index, java.lang.Object obj)

Name	Description
index	Parameter index starting from 1.

Throws

java.sql.SQLException

getRef

Gets Ref value of the indexed parameter.

```
public java.sql.Ref getRef(int index)
```

Name	Description
index	Parameter index starting from 1.

Returns

A Ref object

Throws

java.sql.SQLException

6.6 com.stc.eways.jdbcx.TableResultSet Class

java.lang.Object

```
|
+ -- com.stc.eways.jdbcx.StatementAgent
        |
        + -- com.stc.eways.jdbcx.TableResultSet
```

All Implemented Interfaces

ResetEventListener, SessionEventListener

```
public abstract class TableResultSet
```

extends StatementAgent

ResultSet to map selected records of table in the database

Methods of the TableResultSet

select

Select table records.

```
public void select(java.lang.String sWhere)
```

Name	Description
sWhere	Where condition for the query.

Throws

java.sql.SQLException

---

next

Navigate one row forward.

```
public boolean next()
```

Returns

true if the move to the next row is successful; otherwise, false.

Throws

java.sql.SQLException

---

refreshRow

Refreshes the current row with its most recent value from the database.

```
public void refreshRow()
```

Returns

None

Throws

java.sql.SQLException

---

previous

Navigate one row backward. It should be called only on ResultSetAgent objects that are TYPE_SCROLL_SENSITIVE or TYPE_SCROLL_INSENSITIVE.

```
public boolean previous()
```

Returns

true if the cursor successfully moves to the previous row; otherwise, false.

Throws

java.sql.SQLException

---

absolute

Move cursor to specified row number. It should be called only on ResultSetAgent objects that are TYPE_SCROLL_SENSITIVE or TYPE_SCROLL_INSENSITIVE.

Name	Description
row	An integer other than 0.

Returns

true if the cursor successfully moves to the specified row; otherwise, false.

Throws

java.sql.SQLException

relative

Move the cursor forward or backward a specified number of rows. It should be called only on ResultSetAgent objects that are TYPE_SCROLL_SENSITIVE or TYPE_SCROLL_INSENSITIVE.

```
public boolean relative(int rows)
```

Name	Description
rows	The number of rows to move the cursor, starting at the current row. If the rows are positive, the cursor moves forward; if the rows are negative, the cursor moves backwards.

Returns

true if the cursor successfully moves to the number of rows specified; otherwise, false.

Throws

java.sql.SQLException

first

Move the cursor to the first row of the result set. It should be called only on ResultSetAgent objects that are TYPE_SCROLL_SENSITIVE or TYPE_SCROLL_INSENSITIVE.

```
public boolean first()
```

Returns

true if the cursor successfully moves to the first row; otherwise, false.

Throws

java.sql.SQLException

isFirst

Check if the cursor is on the first row. It should be called only on ResultSetAgent objects that are TYPE_SCROLL_SENSITIVE or TYPE_SCROLL_INSENSITIVE.

```
public boolean isFirst()
```

Returns

true if the cursor successfully moves to the first row; otherwise, false.

Throws

java.sql.SQLException

last

Move to the last row of the result set. It should be called only on ResultSetAgent objects that are TYPE_SCROLL_SENSITIVE or TYPE_SCROLL_INSENSITIVE.

```
public boolean last()
```

Returns

true if the cursor successfully moves to the last row; otherwise, false.

Throws

java.sql.SQLException

isLast

Check if the cursor is positioned on the last row. It should be called only on ResultSetAgent objects that are TYPE_SCROLL_SENSITIVE or TYPE_SCROLL_INSENSITIVE.

```
public boolean isLast()
```

Returns

true if the cursor is on the last row; otherwise, false

Throws

java.sql.SQLException

beforeFirst

Move the cursor before the first row. It should be called only on ResultSetAgent objects that are TYPE_SCROLL_SENSITIVE or TYPE_SCROLL_INSENSITIVE.

```
public void beforeFirst()
```

Throws

java.sql.SQLException

isBeforeFirst

Check if the cursor is positioned before the first row. It should be called only on ResultSetAgent objects that are TYPE_SCROLL_SENSITIVE or TYPE_SCROLL_INSENSITIVE.

```
public boolean isBeforeFirst()
```

Returns

true if the cursor successfully moves before the first row; otherwise, false

Throws

java.sql.SQLException

afterLast

Move the cursor after the last row.It should be called only on ResultSetAgent objects that are TYPE_SCROLL_SENSITIVE or TYPE_SCROLL_INSENSITIVE.

```
public void afterLast()
```

Throws

java.sql.SQLException

isAfterLast

Returns true if the cursor is positioned after the last row. It should be called only on ResultSetAgent objects that are TYPE_SCROLL_SENSITIVE or TYPE_SCROLL_INSENSITIVE.

```
public boolean isAfterLast()
```

Returns true if the cursor successfully moves after the last row; otherwise, false.

Throws

java.sql.SQLException

findColumn

Finds the index of the named column.

public int findColumn(java.lang.String index)

Throws

java.sql.SQLException

getAsciiStream

Returns the column data as an AsciiStream.

```
public java.io.InputStream getAsciiStream(int index)
```

Throws

java.sql.SQLException

getAsciiStream

Returns the column data as an AsciiStream.

```
public java.io.InputStream getAsciiStream(java.lang.String
columnName)
```

Throws

java.sql.SQLException

getBinaryStream

Returns the column data as BinaryStream.

```
public java.io.InputStream getBinaryStream(int index)
```

Throws

java.sql.SQLException

getBinaryStream

Returns the column data as BinaryStream.

```
public java.io.InputStream getBinaryStream(java.lang.String
columnName)
```

Throws

java.sql.SQLException

getCharacterStream

Returns the column data as CharacterStream.

```
public java.io.Reader getCharacterStream(int index)
```

Throws

java.sql.SQLException

getCharacterStream

Returns the column data as CharacterStream.

```
public java.io.Reader getCharacterStream(java.lang.String columnName)
```

Throws

java.sql.SQLException

refreshRow

Refreshes the current row with its most recent value from the database.

```
public void refreshRow()
```

Throws

java.sql.SQLException

insertRow

Inserts the contents of the current row into the database.

```
public void insertRow()
```

Throws

java.sql.SQLException

updateRow

Updates the contents of the current row into the database.

```
public void updateRow()
```

Throws

java.sql.SQLException

deleteRow

Deletes the contents of the current row from the database.

```
public void deleteRow()
```

Throws

java.sql.SQLException

moveToInsertRow

Moves the current position to a new insert row.

```
public void moveToInsertRow()
```

Throws

java.sql.SQLException

moveToCurrentRow

Moves the current position to the current row. It is used after you insert a row.

```
public void moveToCurrentRow()
```

Throws

java.sql.SQLException

closeResultSet

Immediately releases the ResultSet object's resources, close the cursor.

```
public void closeResultSet()
```

Throws

java.sql.SQLException

cancelRowUpdates

Cancels any updates made to this row.

```
public void cancelRowUpdates()
```

Throws

java.sql.SQLException

rowInserted

Returns true if the current row has been inserted.

```
public boolean rowInserted()
```

Throws

java.sql.SQLException

rowUpdated

Returns true i the current row has been updated.

```
public boolean rowUpdated()
```

Throws

java.sql.SQLException

rowDeleted

Returns true if the current row has been deleted.

```
public boolean rowDeleted()
```

Throws

java.sql.SQLException

wasNull

Returns true if the last data retrieved is NULL.

```
public boolean wasNull()
```

Throws

java.sql.SQLException

6.7 $DB Configuration Node Methods

The following methods are associated with the $DB configuration node in the Collaboration. These methods are driver and database specific and will vary from database to database. It is recommended that you consult your specfic databases documentation.

These methods are contained in the following classes:

- **com_stc_jdbcx_oraclecfg.DataSource** on page 140
- **com.stc_jdbcx_oraclecfg** on page 147

6.8 com_stc_jdbcx_oraclecfg.DataSource

Java.lang.Object

|

+ - - com_stc_jdbcx_oraclecfg.Com_stc_jdbcx_oraclecfg.DataSource

Direct Known Subclasses

```
public class Com_stc_jdbcx_oraclecfg.DataSource
```

extends java.lang.Object

Methods of the oraclecfg.DataSource

getClass on page 141	**getDatabaseName** on page 144
getDriverType on page 142	**getPassword** on page 145
getPortNumber on page 143	**getServerName** on page 142
getTimeout on page 146	getUserName()
hasClass on page 141	**hasDatabaseName** on page 144
hasDriverType on page 142	**hasPassword** on page 146
hasServerName on page 143	**hasTimeout** on page 146
hasUserName on page 145	**omitClass** on page 141
omitDatabaseName on page 144	**omitDriverType** on page 142

omitPassword on page 146	**omitPortNumber** on page 144
omitServerName on page 143	**omitTimeout** on page 147
omitUserName on page 145	**setClass** on page 141
hasDatabaseName on page 144	**setDriverType** on page 142
setPassword_AsIs on page 146	**setPassword** on page 145
setPortNumber on page 143	**setServerName** on page 142
setTimeout on page 146	**setUserName** on page 145

getClass

Retrieves the name of the Java class in the JDBC driver that implemtns the ConnectionPoolDataSource interface.

```
public java.lang.String getClass_()
```

Returns

The name of the Java class in the JDBC driver

setClass

Sets the name of the Java class in the JDBC dirver that implements the ConnectionPoolDataSource interface.

```
public void setClass_(java.lang.String val)
```

Returns

None.

hasClass

Returns true if the class has been set.

```
public boolean hasClass_()
```

Returns

True

omitClass

Sets the class to null.

```
public void omitClass_()
```

Returns

None.

getDriverType

Retrieves the driver type.

```
public java.lang.String getDriverType()
```

Returns

Driver type.

setDriverType

Sets the driver type

```
public void setDriverType(java.lang.String val)
```

Returns

String.

hasDriverType

Returns true if the driver type has been set

```
public boolean hasDriverType()
```

Returns

True.

omitDriverType

Set the driver type to null.

```
public void omitDriverType()
```

Returns

None.

getServerName

Retrieves the database server host name.

```
public java.lang.String getServerName()
```

Returns

Database server host name.

setServerName

Set database server host name.

```
public void setServerName(java.lang.String val)
```

Returns

None.

hasServerName

Sets the server name to null.

```
public void omitServerName()
```

Returns

None.

omitServerName

Set the server name to null.

```
public void omitServerName()
```

Returns

None.

getPortNumber

Retrieves the I/O port number of the database server.

```
public long getPortNumber
```

Returns

Long.

setPortNumber

Set I/O port number of the database server.

```
public void setPortNumber(long var)
```

Returns

None.

hasPortNumber

Returns true if the port number has been set.

```
public boolean hasPortNumber()
```

Returns

True.

omitPortNumber

Sets the port nmber to null.

```
public void omitPortNumber()
```

Returns

None.

getDatabaseName

Retrieves the name of the database instance.

```
public java.lang.String getDatabaseName()
```

Returns

Name of the database instance.

setDatabaseName

Sets the name of the database instance.

```
public void setDatabaseName(java.lang.String val)
```

Returns

None.

hasDatabaseName

Returns true if the database name has been set

```
public boolean hasDatabaseName()
```

Returns

True.

omitDatabaseName

Sets the database name to null.

```
public void omitDatabaseName()
```

Returns

None.

getUserName

Retrieves the registered user name used by the e*Way to connect to the database.

```
public java.lang.String getUserName()
```

Returns

Registered user name.

setUserName

Sets the registered user name used by the e*Way to connect to the database.

```
public void setUserName(java.lang.String val)
```

Returns

None.

hasUserName

Returns true if the registered user name has been set.

```
public boolean hasUserName()
```

Returns

True.

omitUserName

Sets the registered user name to null.

```
public void omitUserName()
```

Returns

None.

getPassword

Retrieves the password that the e*Way uses to connect to the database.

```
public java.lang.String getPassword()
```

Returns

Password.

setPassword

Sets the internally encrypted password that the e*Way uses to connect to the database.

```
public void setPassword(java.lang.String val)
```

Returns

None.

setPassword_AsIs

Sets the non-encrypted password that the e*Way uses to connect to the database.

```
public void setPassword_AsIs(java.lang.String val)
```

Returns

None.

hasPassword

Returns True if the password has been set

```
public boolean hasPassword()
```

Returns

True.

omitPassword

Sets the password to null.

```
public void omitPassword()
```

Returns

None.

getTimeout

Retrieves the login timeout in seconds.

```
public java.lang.String getTimeout()
```

Returns

Timeout.

setTimeout

Sets the login timeout in seconds.

```
public void setTimeout(java.lang.String val)
```

Returns

None.

hasTimeout

Returns true if the timeout has been set.

```
public boolean hasTimeout()
```

Returns

True.

omitTimeout

Sets the timeout to null.

```
public void omitTimeout()
```

Returns

None.

6.9 com.stc_jdbcx_oraclecfg

```
com.stc.eways.jdbcx.oraclecfg.COM_stc_jdbcx_oraclecfg
```

Methods of the oracledfg

getDataSource on page 147

setDataSource on page 147

getDataSource

Returns the Datasource object.

```
public Com_stc_jdbcx_oraclecfg.DataSource getDataSource()
```

Returns

DataSource object.

setDataSource

Sets the DataSource object.

```
public void setDataSource(Com_stc_jdbcx_oraclecfg.DataSource val)
```

Returns

None.

Index

U

X