

**SeeBeyond™ eBusiness Integration Suite**

# e\*Way Intelligent Adapter for SQL Server User's Guide

*Release 4.5.2*



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

e\*Gate, e\*Insight, e\*Way, e\*Xchange, e\*Xpressway, eBI, iBridge, Intelligent Bridge, IQ, SeeBeyond, and the SeeBeyond logo are trademarks and service marks of SeeBeyond Technology Corporation. All other brands or product names are trademarks of their respective companies.

© 2001-2002 by SeeBeyond Technology Corporation. All Rights Reserved. This work is protected as an unpublished work under the copyright laws.

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

Version 20020222133156.

# Contents

---

## Chapter 1

<b>Introduction</b>	<b>9</b>
<b>Overview</b>	<b>9</b>
Intended Reader	9
Components	9
<b>Operational Overview</b>	<b>10</b>
<b>System Requirements</b>	<b>10</b>
Host System Requirements	10
GUI Host Requirements	11
Participating Host Requirements	11

---

## Chapter 2

<b>Installation</b>	<b>12</b>
<b>Installing the SQL Server e*Way on Windows NT or Windows 2000</b>	<b>12</b>
Pre-installation	12
Installation Procedure	12
Registering the JDBC .jar Files	13
<b>Installing the SQL Server e*Way on UNIX</b>	<b>13</b>
Pre-installation	13
Installation Procedure	13
Registering the JDBC .jar Files	14
Configuring the DataDirect 2.2 Drivers for XA Mode	14
<b>Files/Directories Created by the Installation</b>	<b>14</b>

---

## Chapter 3

<b>e*Way Connection Configuration</b>	<b>16</b>
<b>Create e*Way Connections</b>	<b>16</b>
DataSource Settings	17
class	17
XA Compliant	18
ServerName	18
PortNumber	18
DatabaseName	18
user name	18

password	18
SelectMethod	19
timeout	19
<b>Connector Settings</b>	<b>19</b>
type	19
class	19
transaction mode	20
transaction mode	20
connection establishment mode	20
connection inactivity timeout	21
connection verification interval	21
<b>Connection Manager</b>	<b>21</b>
Controlling When a Connection is Made	22
Controlling When a Connection is Disconnected	22
Controlling the Connectivity Status	22
<b>Stceway Connection</b>	<b>23</b>
Classpath Prepend	23

## Chapter 4

<b>Implementation</b>	<b>24</b>
<b>The Java Collaboration Service</b>	<b>24</b>
Java Components	24
<b>The Java ETD Builder</b>	<b>24</b>
The Parts of the ETD	25
Using the DBWizard ETD Builder	25
The Generated ETDs	33
Editing an Existing .XSC Using the Database Wizard	33
<b>Using ETDs with Tables, Views, Stored Procedures, and Prepared Statements</b>	<b>34</b>
The Table	34
The select() Method	34
The insert() Method	34
The update() Method	35
The delete() Method	36
The View	36
The Stored Procedure	37
Collecting Data from Output Parameters	37
Returning Result Sets from Sybase, DB2, and SQL Server	38
Prepared Statement	39
Batch Operations	41
Database Configuration Node	42
<b>Sample Scenario—Polling from a Database</b>	<b>42</b>
Create the Schema	45
Add the Event Types and Event Type Definitions	45
Create the Collaboration Rules and the Java Collaboration	48
Add and Configure the e*Ways	52
Add and Configure the e*Way Connections	54
Add the IQs	55
Add and Configure the Collaborations	55

## Chapter 5

**SQL Server e\*Way Methods 58****com.stc.eways.jdbcx.StatementAgent Class 58**

resultSetTypeToString	59
resultSetDirToString	59
resultSetConcurToString	60
isClosed	60
queryName	60
queryDescription	60
sessionOpen	61
sessionClosed	61
resetRequested	61
getResultSetType	61
getResultSetConcurrency	62
setEscapeProcessing	62
setCursorName	62
setQueryTimeout	63
setQueryTimeout	63
getFetchDirection	63
setFetchDirection	63
getFetchSize	64
getMaxRows	64
setMaxRows	64
getMaxFieldSize	65
setMaxFieldSize	65
getUpdateCount	65
getResultSet	65
getMoreResults	66
clearBatch	66
executeBatch	66
cancel	66
getWarnings	67
clearWarnings	67
stmtInvoke	67

**com.stc.eways.jdbcx.PreparedStatementAgent Class 68**

sessionOpen	68
setNull	69
setNull	69
setObject	69
setObject	70
setObject	70
setBoolean	71
setByte	71
setShort	71
setInt	72
setLong	72
setFloat	72
setDouble	73
setBigDecimal	73
setDate	73
setDate	74
setTime	74
setTime	74
setTimestamp	75
setTimestamp	75
setString	75
setBytes	76

setAsciiStream	76
setBinaryStream	76
setCharacterStream	77
setArray	77
setBlob	78
setClob	78
setRef	78
clearParameters	79
addBatch	79
execute	79
executeQuery	79
executeUpdate	79
<b>com.stc.eways.jdbcx.PreparedStatementResultSet Class</b>	<b>80</b>
Constructor PreparedStatementResultSet	82
getMetaData	82
getConcurrency	82
getFetchDirection	82
setFetchDirection	83
getFetchSize	83
setFetchSize	83
getCursorName	84
close	84
next	84
previous	84
absolute	84
relative	85
first	85
isFirst	85
last	85
isLast	86
beforeFirst	86
isBeforeFirst	86
afterLast	86
isAfterLast	87
getType	87
findColumn	87
getObject	87
getObject	88
getObject	88
getObject	89
getBoolean	89
getBoolean	89
getByte	90
getShort	90
getShort	90
getInt	91
getInt	91
getLong	91
getLong	92
getFloat	92
getFloat	93
getDouble	93
getBigDecimal	93
getBigDecimal	94
getDate	94
getDate	94
getDate	95
getTime	95
getTime	95
getTime	96
getTime	96
getTimestamp	97
getTimestamp	97

getTimestamp	97
getTimestamp	98
getString	98
getString	98
getBytes	99
getBytes	99
getAsciiStream	100
getAsciiStream	100
getBinaryStream	100
getBinaryStream	101
getCharacterStream	101
getArray	101
getBlob	102
getBlob	102
getClob	102
getClob	103
getRef	103
getRef	104
wasNull	104
getWarnings	104
clearWarnings	104
getRow	105
refreshRow	105
insertRow	105
updateRow	105
deleteRow	105
<b>com.stc.eways.jdbcx.SqlStatementAgent Class</b>	<b>106</b>
Constructor SqlStatementAgent	106
Constructor SqlStatementAgent	106
execute	107
executeQuery	107
executeUpdate	108
addBatch	108
<b>com.stc.eways.jdbcx.CallableStatementAgent Class</b>	<b>108</b>
Constructor CallableStatementAgent	109
Constructor CallableStatementAgent	110
Constructor CallableStatement Agent	110
sessionOpen	110
registerOutParameter	111
registerOutParameter	111
registerOutParameter	111
wasNull	112
getObject	112
getObject	112
getBoolean	113
getByte	113
getShort	113
getInt	114
getLong	114
getFloat	114
getDouble	115
getBigDecimal	115
getDate	115
getDate	116
getTime	116
getTime	117
getTimestamp	117
getTimestamp	117
getString	118
getBytes	118
getArray	118
getBlob	119
getClob	119

getRef	120
<b>com.stc.eways.jdbcx.TableResultSet Class</b>	<b>120</b>
select	121
next	121
previous	121
absolute	122
relative	122
first	122
isFirst	123
last	123
isLast	123
beforeFirst	123
isBeforeFirst	124
afterLast	124
isAfterLast	124
findColumn	124
getAsciiStream	125
getAsciiStream	125
getBinaryStream	125
getBinaryStream	125
getCharacterStream	125
getCharacterStream	126
refreshRow	126
insertRow	126
updateRow	126
deleteRow	126
moveToInsertRow	126
moveToCurrentRow	127
cancelRowUpdates	127
rowInserted	127
rowUpdated	127
rowDeleted	127
wasNull	128

<b>Index</b>	<b>129</b>
--------------	------------



# Introduction

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

---

## 1.1 Overview

The SQL Server e\*Way enables the e\*Gate system to exchange data with external SQL Server databases. Using the java library, SQL statements are issued to interact with the SQL Server databases.

### 1.1.1 Intended Reader

The reader of this guide is presumed:

- to be a developer or system administrator with the responsibility of maintaining the e\*Gate system
- to have expert-level knowledge of Windows NT or Windows 2000 operations and administration
- to be thoroughly familiar with SQL functions
- to be thoroughly familiar with Windows-style GUI operations

### 1.1.2 Components

The e\*Way is composed of the following components:

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

A complete list of installed files appears in Table 1.

---

## 1.2 Operational Overview

The SQL Server 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.

---

## 1.3 System Requirements

The Java-enabled ETD Editor and Collaboration Editor support the following operating systems:

- Windows 2000, Windows 2000 SP1, and Windows 2000 SP2.
- Windows NT 4.0 SP6a

The SQL Server e\*Way components support the following operating systems:

- Windows 2000, Windows 2000 SP1, and Windows 2000 SP2
- Windows NT 4.0 SP6a
- Solaris 2.6, 7, and 8
- AIX 4.3.3
- HP-UX 11.0 and HP-UX 11i
- Japanese Windows 2000, Windows 2000 SP1, and Windows 2000 SP2
- Japanese Windows NT 4.0 SP6a
- Japanese Solaris 2.6, 7, and 8
- Japanese HP-UX 11.0

To use the e\*Way, you also need the following:

- An e\*Gate Participating Host, version 4.5.1 or later. For AIX operating systems, you need an e\*Gate Participating Host, version 4.5.1. or later.
- A TCP/IP network connection.

The client components for the databases that the e\*Way interfaces with have their own requirements; see that system's documentation for more details.

### 1.3.1 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 each host machine that runs the GUI editors:

- SQL Server 7 Service Pack 3 or SQL Server 2000 Service Pack 1.
- Microsoft Data Access Components (MDAC) RTM version 2.6 or greater. This component is included in the e\*Gate GUI installation.
- Merant ODBC 4.0 driver. This driver is included on the e\*Gate installation cd.

If the GUI host machine will also be executing the SQL Server e\*Way, the host machine must also meet the “Participating Host Requirements”.

## Participating Host Requirements

The SQL Server Database e\*Way requires that the following item be installed on the Participating Host:

- Merant DataDirect Connect JDBC 2.1 driver. This driver is required for use with Java JDK.
- Merant DataDirect Connect JDBC 2.2 driver if you are running in XA mode. You can request a copy of this driver by contacting SeeBeyond’s support personnel.

# Installation

This chapter describes how to install the SQL Server e\*Way.

---

## 2.1 Installing the SQL Server e\*Way on Windows NT or Windows 2000

The following information will guide you through installing the e\*Way.

### 2.1.1 Pre-installation

Before beginning the installation process, you must do the following:

- 1 Exit all Windows programs before running the setup program, including any anti-virus applications.
- 2 Assure that you have Administrator privileges before installing this e\*Way.
- 3 Install the Merant JDBC drivers (DGbase.jar, DGsqlserver.jar, DGutil.jar and spy.jar) into one of the participating host's \client\ThirdParty\merant\classes directory.

### 2.1.2 Installation Procedure

To install the SQL Server e\*Way on a Windows NT or Windows 2000 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.
- 4 The InstallShield setup application will launch. 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.**

### 2.1.3 Registering the JDBC .jar Files

- 1 Open a DOS prompt and cd to client\etd directory and verify that the sqlserver.ctl file exists in that directory.
- 2 Register the DGbase.jar, DGsqlserver.jar, DGutil.jar, and spy.jar files by executing `stcregutil.exe -v -rh <registry host> -rs default -un Administrator -up <password> -fc etd -ctl sqlserver.ctl`.

---

## 2.2 Installing the SQL Server 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.

Before beginning the installation process, you must do the following:

- 1 Install the Merant JDBC driver (DGbase.jar, DGsqlserver.jar, DGutil.jar and spy.jar) into one of the participating host's /client/ThirdParty/merant/classes directory.

### 2.2.2 Installation Procedure

To install the SQL Server 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.**

### 2.2.3 Registering the JDBC .jar Files

- 1 At the UNIX shell prompt type `cd/client/etd` directory and ensure that `sqlserver.ctl` file exists in that directory.
- 2 Register the `DGbase.jar`, `DGsqlserver.jar`, `DGutil.jar`, and `spy.jar` files by executing `stcregutil.exe -v -rh <registry host> -rs default -un Administrator -up <password> -fc etd -ctl sqlserver.ctl`.

### 2.2.4 Configuring the DataDirect 2.2 Drivers for XA Mode

If you are running in XA mode, you will need to install and run Merant's DataDirect 2.2 JDBC drivers. You may request a copy of these drivers by contacting SeeBeyond support personnel.

Once the DataDirect 2.2 drivers are installed, do the following:

- 1 Copy the `sqljdbc.dll` from your `SQLServer\JTA` directory to your `SQLServer\bin` directory.
- 2 From a command prompt, type `instjdbc.sql`. This script is provided by DataDirect.

For more information please see the *DataDirect Connect JDBC User's Guide and Reference*.

---

## 2.3 Files/Directories Created by the Installation

The SQL Server e\*Way installation process will install the following files within the e\*Gate directory tree. Files will be 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)
bin\java	jcscmp.jar stcjintegra.jar xerces.jar
classes\	stcjcs.jar stcjdbcx.jar
configs\sqlserver\	sqlserver.def
etd\	sqlserver.ctl dbwizard.ctl
etd\sqlserver\	Com_stc_jdbcx_sqlservercfg.java Com_stc_jdbcx_sqlservercfg.xsc
ThirdParty\merant\Classes\	DGsqlserver.jar DGbase.jar DGutil.jar spy.jar
ThirdParty\sun\classes	jdbc2_0-stdext.jar



# e\*Way Connection Configuration

This chapter describes how to configure the SQL Server e\*Way Connections.

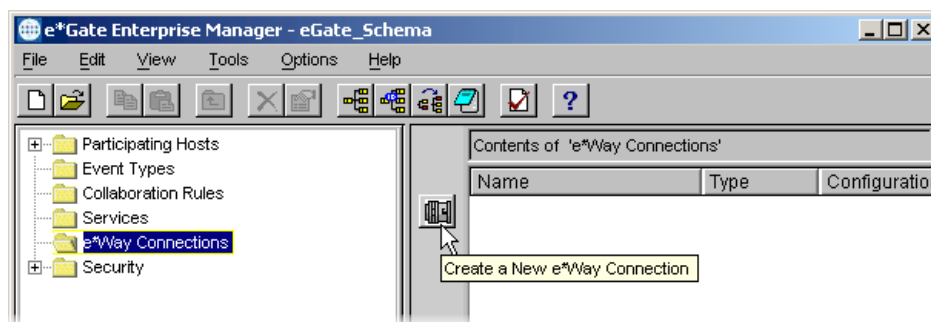
## 3.1 Create e\*Way Connections

The e\*Way Connections are created and configured in the Enterprise Manager.

To create and configure the e\*Way Connections

- 1 In the Enterprise Manager's Component editor, select the e\*Way Connections folder.

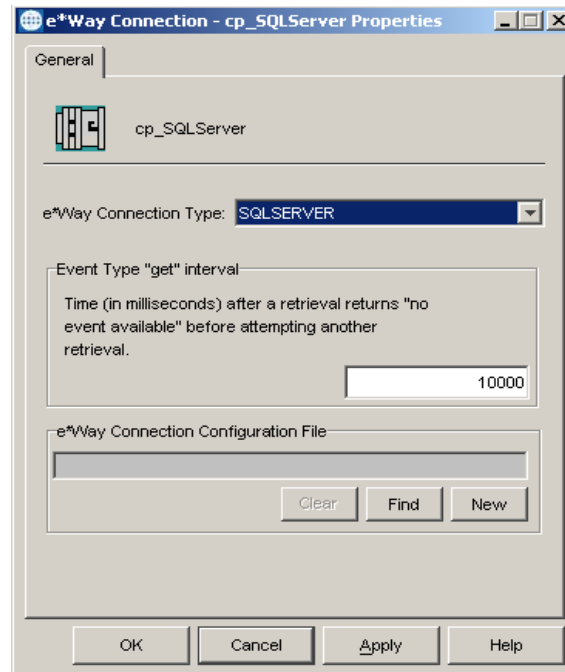
**Figure 1** The e\*Way Connections Folder



- 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.



**Figure 2** e\*Way Connection Properties Dialog Box



- 5 From the e\*Way Connection Type drop-down list, select **SQLSERVER**.
- 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 Settings** on page 17
- **Connector Settings** on page 19

### 3.1.1 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 `com.SeeBeyond.jdbcx.sqlserver.SQLServerDataSource`.

## XA Compliant

### Description

Specifies if the XA feature of the driver is compliant.

### Required Values

True or False. Specifies whether you want to turn XA on. If true, an XA connection will be used.

## ServerName

### Description

Specifies the host name of the external database server.

### Required Values

Any valid string.

## PortNumber

### Description

Specifies the port number on which the server is listening for connection requests. The default value is 1433.

### Required Values

A valid port number.

## DatabaseName

### Description

Specifies the name of the database instance.

### Required Values

Any valid string.

## user name

### Description

Specifies the user name the e\*Way will use 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.

## SelectMethod

### Description

Specifies whether database cursors are used for Select statements. Using the Direct mode is the most efficient for executing Select statements however, you may be limited to a single active statement while executing inside a transaction. Setting SelectMethod to cursor allows multiple active statements within a transaction.

### Required Values

**Cursor** or **Direct**

## timeout

### Description

This is the login timeout in seconds.

### Required Values

Any valid string. The default is **300** seconds.

## 3.1.2 Connector Settings

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

## 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

### Description

Specifies how transactions should be handled.

### Required Values

## transaction mode

This parameter specifies how a transaction should be handled.

- **Automatic** — e\*Gate will take 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 will 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 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 will be established on demand as business rules requiring a connection to the external system are performed. The connection will be closed after the methods are completed.
- **Manual** indicates that the user will explicitly call 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 will not be 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 will try 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.

---

## 3.2 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 e\*Gate 4.5.2's enhanced functionality. If you are running e\*Gate 4.5.1 or earlier, this enhanced functionality is visible but will be ignored.

The Connection Manager is controlled in the e\*Way configuration as described in [Connector Settings](#) on page 19. If you choose to manually control the e\*Way connections, you may find the following chart helpful.

**Figure 3** 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 (XA)	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 can 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 if it is still alive and you can set how often it checks. See [Connector Settings](#) on page 19.

---

## 3.3 Stceway Connection

### Classpath Prepend

#### Required Values

You will need to add the following .jar files to your classpath:

- DGbase.jar
- DGutil.jar
- DGsqlserver.jar
- spy.jar

The absolute path should be set in:

```
\client\ThirdParty\merant\classes\DGbase.jar;\client\ThirdParty  
\merant\classes\DGsqlserver.jar;\client\ThirdParty\merant\classes  
\DGutil.jar;\client\ThirdParty\merant\classes\spy.jar.
```

# Implementation

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

---

## 4.1 The Java Collaboration Service

The Java Collaboration Service makes it possible to develop external Collaboration Rules that will 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 subcollaborations, 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*.

### 4.1.1 Java 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.

---

## 4.2 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.



## 4.2.1 The Parts of the ETD

There are four possible parts to the Java-enabled Event Type Definition as shown in Figure 4.

**Figure 4** 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: boolean, char, double, float, int, long, or java.lang.String.
- **Method** – Method nodes represent actual Java methods.
- **Parameter** – Parameter nodes represent the parameters of the Java method.

## 4.2.2 Using the DBWizard ETD Builder

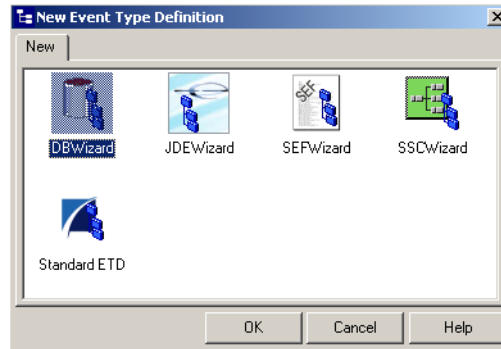
The DBWizard ETD Builder generates Java-enabled ETDs by connecting to external data sources and creating corresponding Event Type Definitions. The ETD Builder can create ETDs based on any combination of tables, views, stored procedures, or prepared SQL statements.

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.

### To create a new ETD using the DBWizard

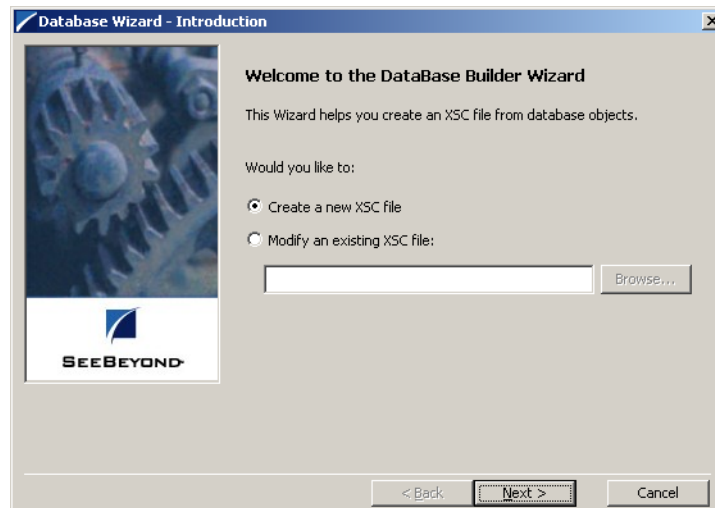
- 1 From the **Options** menu of the Enterprise Manager, choose **Default Editor....**
- 2 Verify that **Java** is selected, then click **OK**.
- 3 Click the **ETD Editor** button to launch the Java ETD Editor.
- 4 In the **Java ETD Editor**, click the **New** button to launch the New Event Type Definition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

**Figure 5** New Event Type Definition



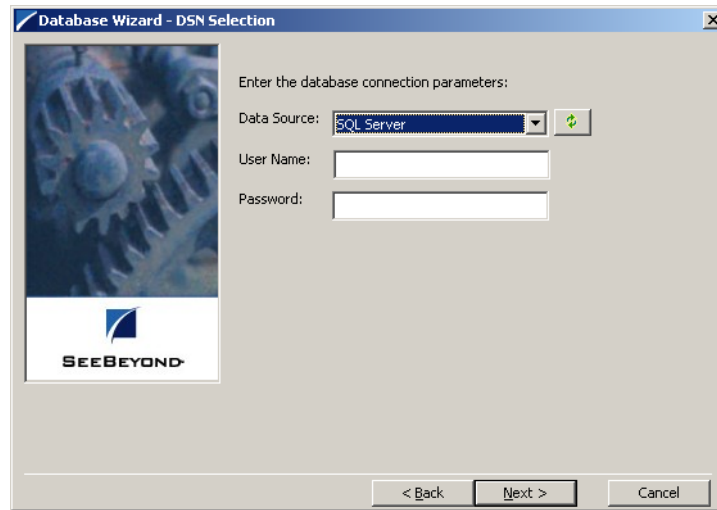
- 6 Enter the name of the new .xsc file you want to create or enter the name of the .xsc file you want to edit by browsing to its location.

**Figure 6** Database Wizard - Introduction



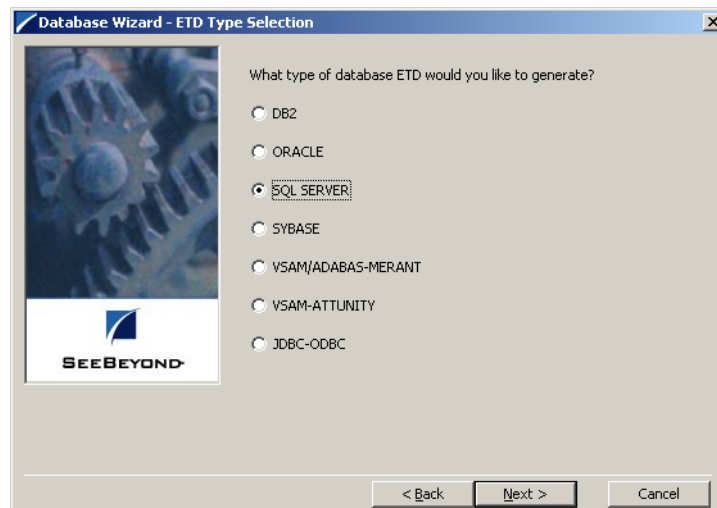
- 7 Select your **Data Source:** from the drop down list and enter your **User Name:** and **Password:**.

**Figure 7** Database Wizard - DSN Selection



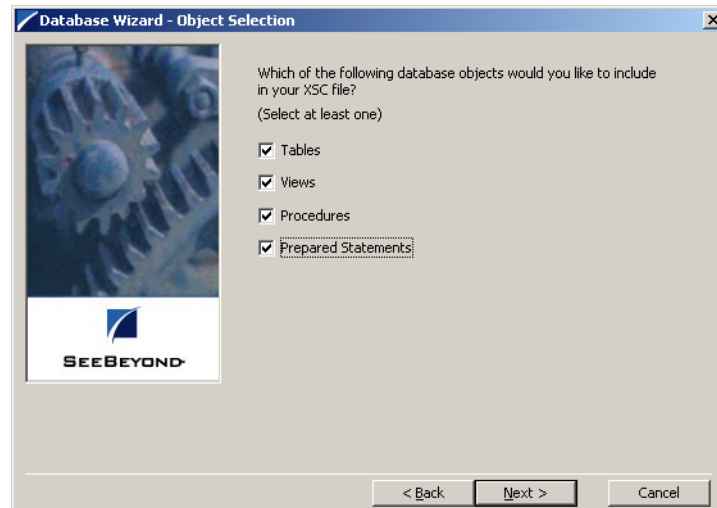
- 8 Select what type of database ETD you would like to generate. The data source you selected in the **Database Wizard - DSN Selection** window is the default. *Note: Do not change this unless instructed to do so by SeeBeyond personnel.*

**Figure 8** Database Wizard - ETD Type Selection



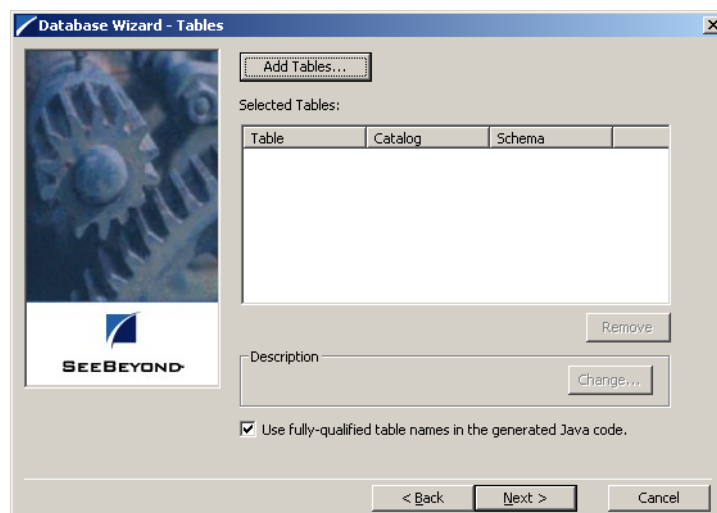
- 9 In the **Database Wizard - Object Selection** window, select any combination of **Tables, Views, Procedures, or Prepared Statements** you would like to include in your .xsc file. Click **Next** to continue.

**Figure 9** Database Wizard - Object Selection



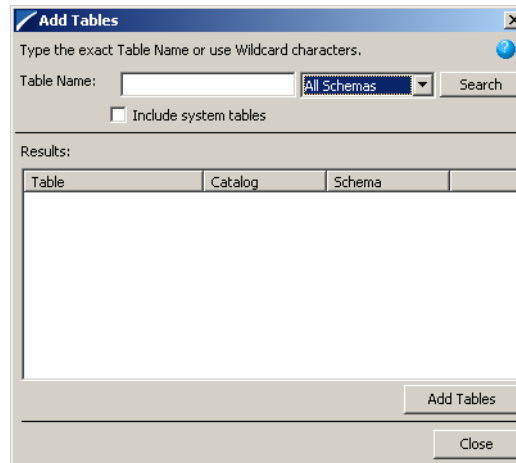
- 10 In the **Database Wizard - Tables** window, click **Add Tables**.

**Figure 10** Database Wizard - Tables



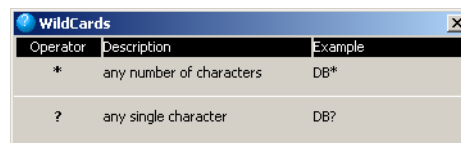
- 11 In the **Add Tables** window, type the exact name of the database table or use wildcard characters to return table names.

**Figure 11** Add Tables



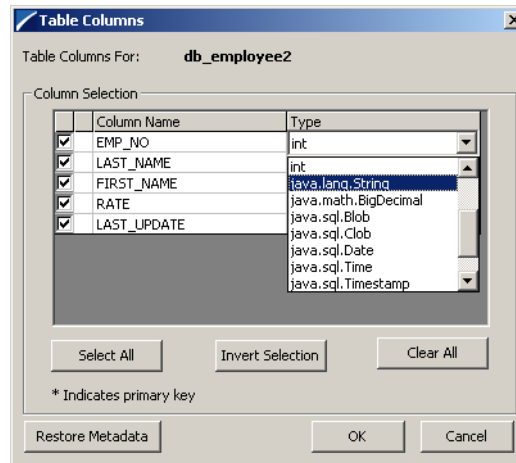
- 12 To see a list of valid wildcard characters, click the round ball with a question mark located in its center.

**Figure 12** Wildcards



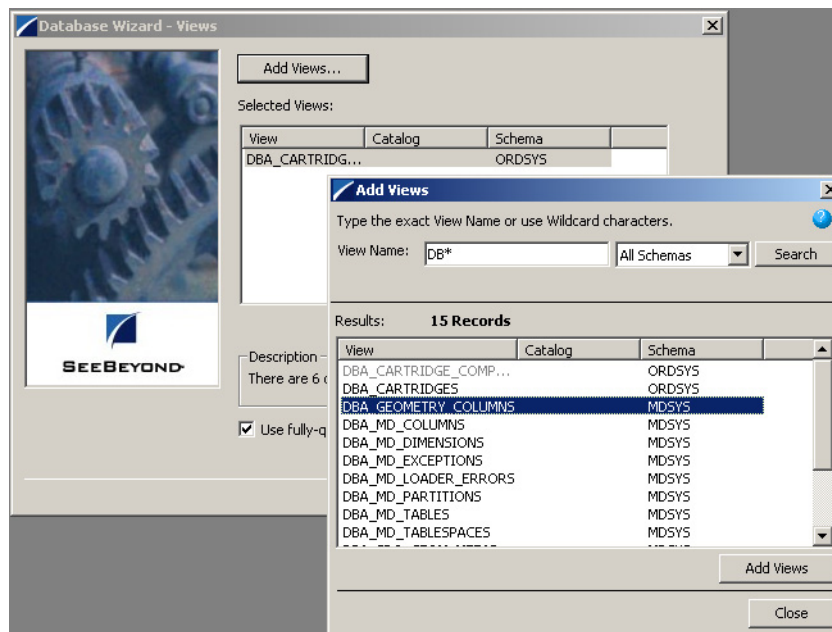
- 13 Select **Include System Tables** if you wish to include them and click **Search**. If your search was successful, you will see the results in the Results window. To select the name of the tables you wish to add to your .xsc, double click on the table name or highlight the table names and click **Add 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 you have selected, click **Change**.
- 15 In the **Columns Selection** window, you can select or deselect your table choices. 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 13 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.

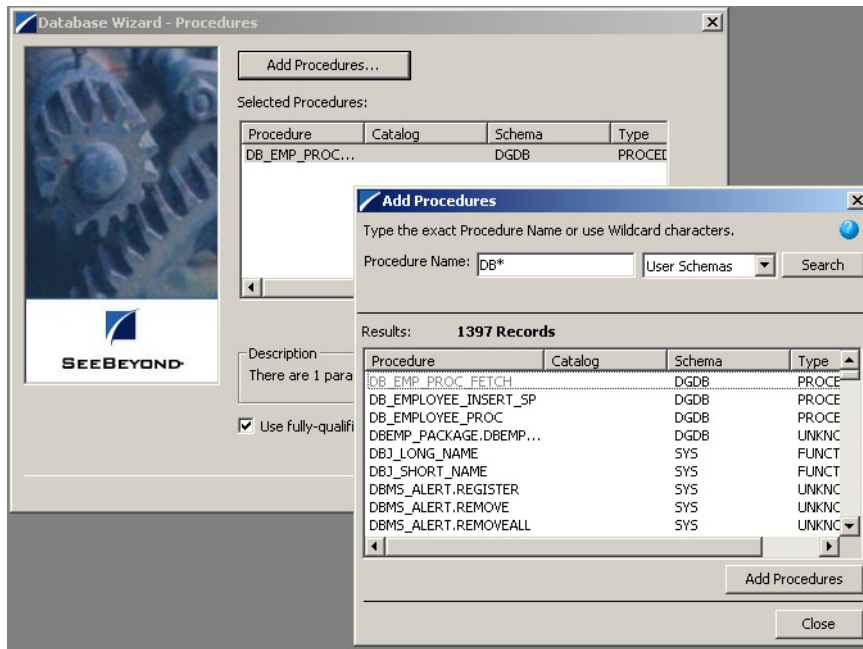
Figure 14 Database Wizard - Views



- 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

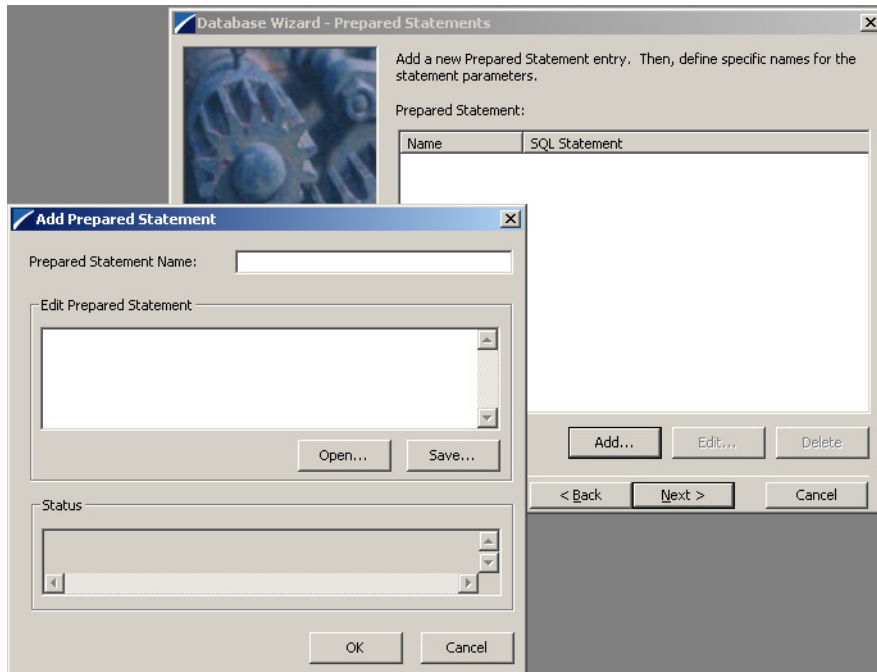
procedure names in the generated Java code, click to clear the check box and click **Next** to continue.

**Figure 15** Database Wizard - Procedures



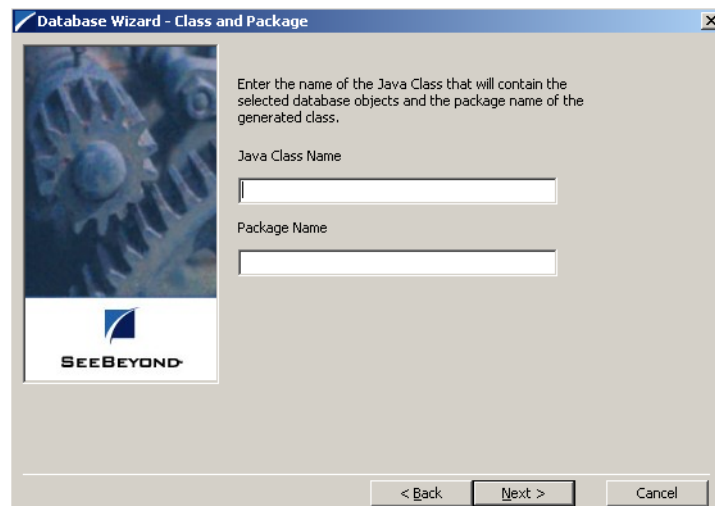
- 19 If you selected **Prepared Statements** on the **Database Wizard - Object Selection** window, you are presented with the **Database Wizard - Prepared Statement** window. To add **Prepared Statements** to your .xsc. complete the following steps:
  - A Click **Add** to add a new prepared statement
  - 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.
  - E Click **OK** to return to the **Database Wizard - Prepared Statements** screen.
- 20 Repeat steps A–E to add additional prepared statements or click **Next** to continue.

Figure 16 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.

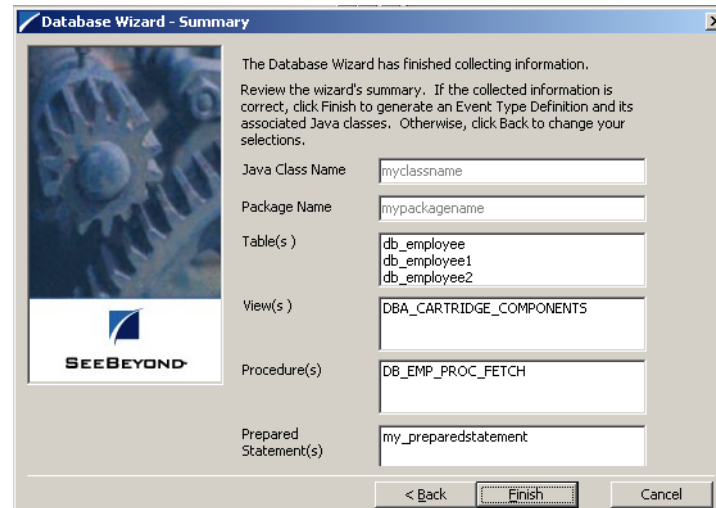
Figure 17 Database Wizard - Class and Package



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



Figure 18 Database Wizard - Summary



### 4.2.3 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 will 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 will need to open the .xsc in the DataBase Wizard.

### 4.2.4 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 Enterprise Manager, 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, you will see a warning and the element will be dropped from the ETD.*

---

## 4.3 Using ETDs with Tables, Views, Stored Procedures, and Prepared Statements

The Insert, Delete, Update, and Execute Methods are common methods used within the Collaboration Editor.

### 4.3.1 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 SELECT, UPDATE, INSERT and DELETE SQL operations in a table.

#### The select() Method

To perform a select operation on a table, add a new method in the Collaboration Editor by clicking on the **method** button. In the Method Properties box, define the method. Add a rule to the method by clicking on the **rule{}** button. The Rule Properties dialog box appears. Select the node and drag and drop it into the dialog box. This method takes an empty string or a WHERE clause. For example:

```
getDBEmp().getDB_EMPLOYEE().select("EMP_NO = 123");
```

#### The insert() Method

To perform an insert operation on a table, do the following:

- 1 From the Collaboration Editor, click the **method** button. Execute the select method with the right concurrency for example CONCUR\_UPDATABLE and scroll sensitivity for example TYPE\_SCROLL\_INSENSITIVE, TYPE\_SCROLL\_SENSITIVE.
- 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.

```

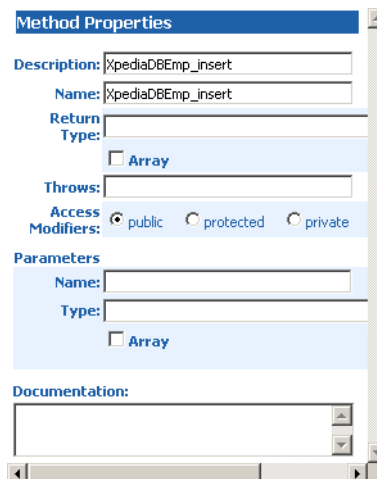
getDBEmp().getDB_EMPLOYEE(ResultSet.TYPE_SCROLL_SENSITIVE,
ResultSet.CONCUR_UPDATABLE).select("");
getDBEmp().getDB_EMPLOYEE().moveToInsertRow();
getDBEmp().getDB_EMPLOYEE().setEMP_NO(123);
.
.
.
getDBEmp().getDB_EMPLOYEE().setRATE(123.45);
getDBEmp().getDB_EMPLOYEE().insertRow();
    
```

**Figure 19** Insert Method Business Rule

```

Business Rules
  XpediaDBEmp_insert : public class XpediaDBEmp_insert extends XpediaDBEmp_insertBase implements JCollaboratorExt
  {
    XpediaDBEmp_insert : public XpediaDBEmp_insert()
    {
      rule : super();
    }
    method : void method()
    executeBusinessRules() throws Exception
    {
      retBoolean : boolean retBoolean = true;
      rule : getXpediaDBEmp().getDB_EMPLOYEE(ResultSet.TYPE_SCROLL_SENSITIVE, ResultSet.CONCUR_UPDATABLE).select("");
      rule : getXpediaDBEmp().getDB_EMPLOYEE().moveToInsertRow();
      rule : getXpediaDBEmp().getDB_EMPLOYEE().setEMP_NO(new java.math.BigDecimal(getStandardDBEmp().getEmployeeNumber()));
      rule : getXpediaDBEmp().getDB_EMPLOYEE().setLAST_NAME(getStandardDBEmp().getLastName());
      rule : getXpediaDBEmp().getDB_EMPLOYEE().setFIRST_NAME(getStandardDBEmp().getFirstName());
      rule : getXpediaDBEmp().getDB_EMPLOYEE().setRATE(Double.parseDouble(getStandardDBEmp().getRate()));
      rule : getXpediaDBEmp().getDB_EMPLOYEE().setLAST_UPDATE(java.sql.Timestamp.valueOf(getStandardDBEmp().getLastUpdate()));
      rule : getXpediaDBEmp().getDB_EMPLOYEE().insertRow();
    }
    return : return retBoolean;
    userInitialize : public void userInitialize()
    userTerminate : public void userTerminate()
  }
    
```

**Figure 20** Insert Method Properties



## The update() Method

To perform an update operation on a table, do the following:

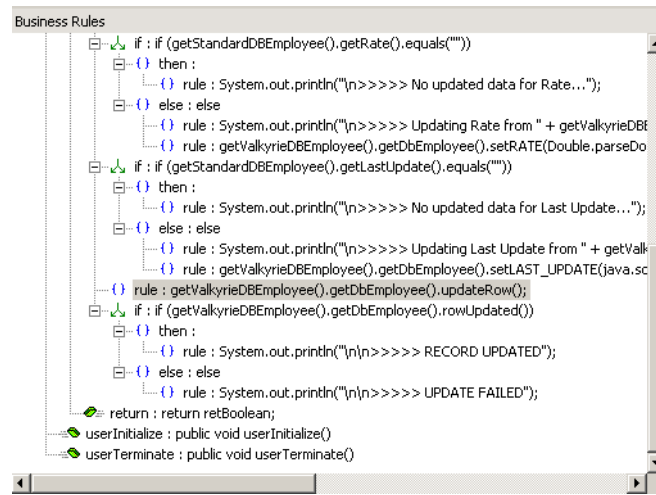
- 1 Execute the **select** method with the right concurrency for example CONCUR\_UPDATEABLE and scroll sensitivity for example TYPE\_SCROLL\_INSENSITIVE, TYPE\_SCROLL\_SENSITIVE.
- 2 Move to the row that you want to update.
- 3 Set the fields of the table ETD

#### 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 21** Update() Method Business Rule



## The delete() Method

To execute a delete operation on a table, do the following:

- 1 Execute the **select** method with the right concurrency for example CONCUR\_UPDATEABLE and scroll sensitivity for example TYPE\_SCROLL\_INSENSITIVE, TYPE\_SCROLL\_SENSITIVE.
- 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();
```

### 4.3.2 The View

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

### 4.3.3 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.

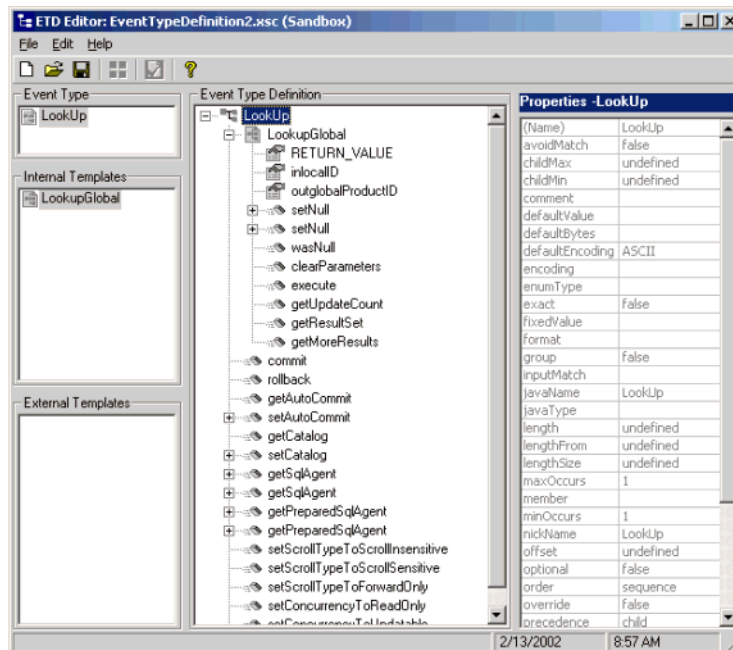
### Collecting Data from Output Parameters

You can collect data within a Stored Procedure Result Set by using output parameters. An example of how to execute a stored procedure using output parameters is:

```
create procedure LookupGlobal
( @inlocalID varchar, @outglobalProductID varchar output )
as
begin
select @outglobalProductID = globalProductID from SimpleLookup where
localID = @inlocalID
end
go
```

In the following image, the ETD represents the Stored Procedure “LookUpGlobal” with two parameters, an inbound parameter (INLOCALID) and an outbound parameter (OUTGLOBALPRODUCTID). 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.

**Figure 22** Stored Procedure LookUpGlobal



## Returning Result Sets from Sybase, DB2, and SQL Server

The following Result Set processing returns rows into the body of the Stored Procedure. Databases that allow this operation are Sybase, DB2, and SQL Server.

A stored procedure in a database can perform update and select operations. The application checks what operation has been performed. The following methods allow you to collect this information:

### getUpdateCount:

This method should be called only after a call to **execute** or **getMoreResults**, and should be called only once per result. An int greater than 0 represents the number of rows affected by an operation. A value of 0 means no rows were affected or the operation is a DDL command. -1 means the result is a result set or there are no more results.

### getResultSet:

When the **execute** method has been used to execute the stored procedure. This method returns the current result set if there is one. Otherwise, it returns a null value which indicates the result is an update count or there are no more results.

### getMoreResults:

Moves to the next result. This method returns true if it gets the next result set. It returns false if it is an update count or there are no more results.

This example prints out the update count and result sets that are returned from the stored procedures.

```
int iRow;
com.stc.eways.jdbcx.ResultSetAgent rs;
getpoller().getMrsSp().execute();

do
{
    iRow = getpoller().getMrsSp().getUpdateCount();
    System.out.println("\n*****iRow = " +
Integer.toString(iRow));

    if (iRow >= 0)
    {
        getblobout().setField1("Number of rows affected: " +
Integer.toString(iRow));
        getblobout().send();
    }
    else
    {
        rs = getpoller().getMrsSp().getResultSet();
        while (rs.next())
        {
            getblobout().setField1("Col5 = " +
Integer.toString(rs.getInt(1)) + ", Col6 = " +
Integer.toString(rs.getInt(2)) );
            getblobout().send();
        }
    }
}
while (getpoller().getMrsSp().getMoreResults());
```

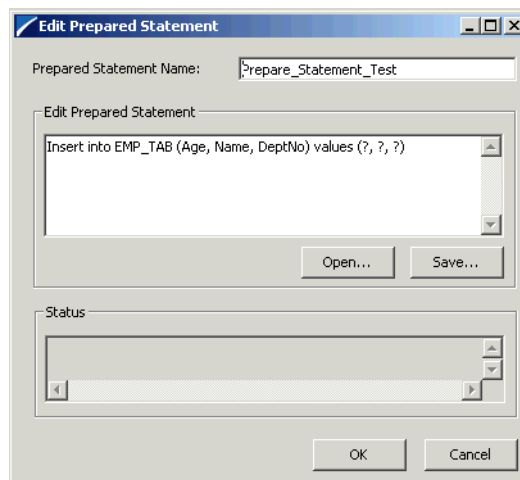
Use an outbound parameter to enrich another ETD.

## 4.3.4 Prepared Statement

To Create a prepared statement from the DataBase Wizard:

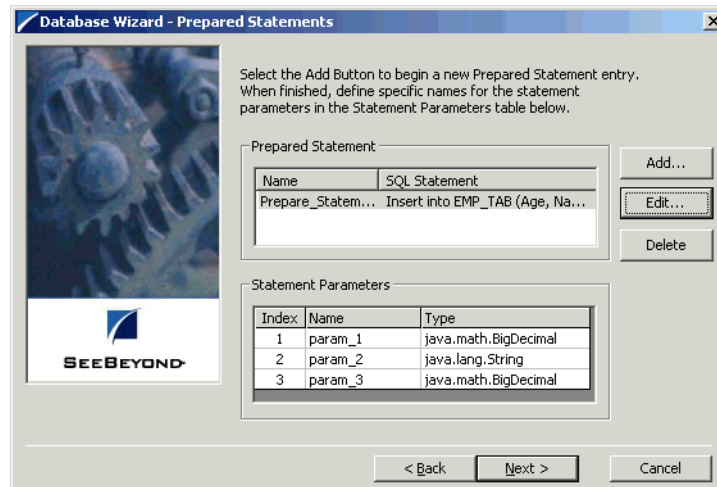
- 1 Select the **Prepared Statement** option in the DataBase Wizard and complete the following steps:
  - A Click **Add** to add a new prepared statement
  - B Enter a prepared SQL statement as shown. The question marks serve as place holders that are later defined as the parameters in [Figure 24 on page 40](#)

**Figure 23** Add Prepared Statement



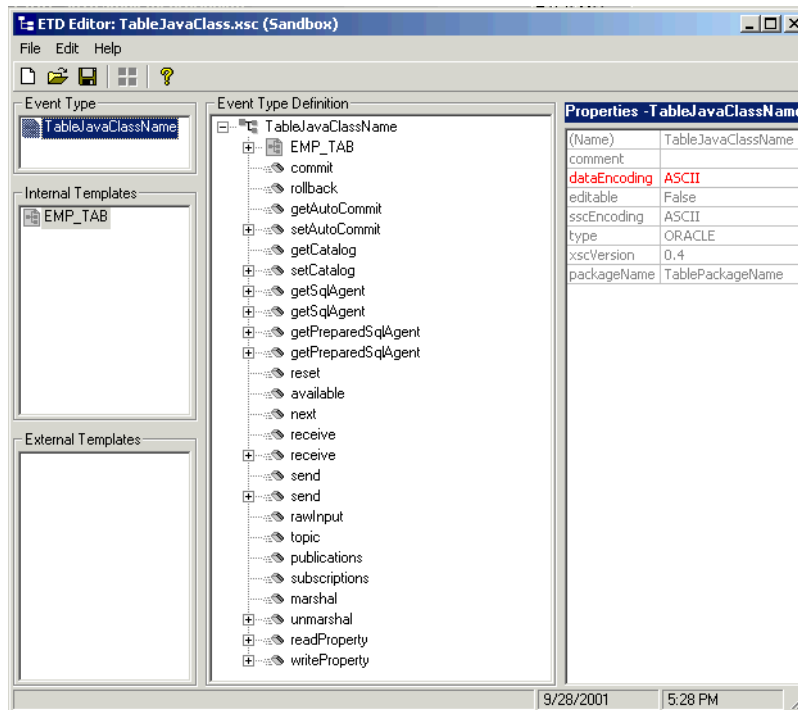
- C Enter the **Prepared Statement Name** to be used by the statement.
  - D Use the **Open...** or **Save...** buttons to open pre-existing statements or create and save the current one.
- 2 [Figure 24 on page 40](#) shows the statement along with a table of the parameters. You can give each parameter an appropriate name and data type. These parameters become nodes in the prepared statement ETD and fields within the generated Java object.

**Figure 24** Prepared Statement Summary and Parameter Modification

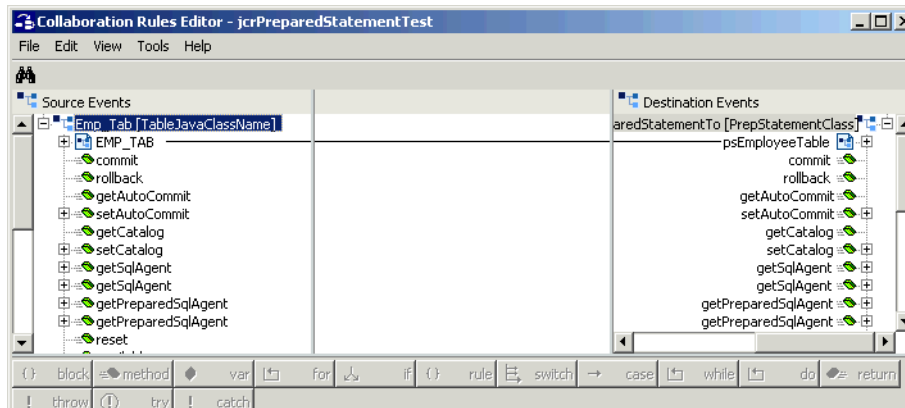


- 3 From the ETD Editor, save the Prepared Statement you just created.

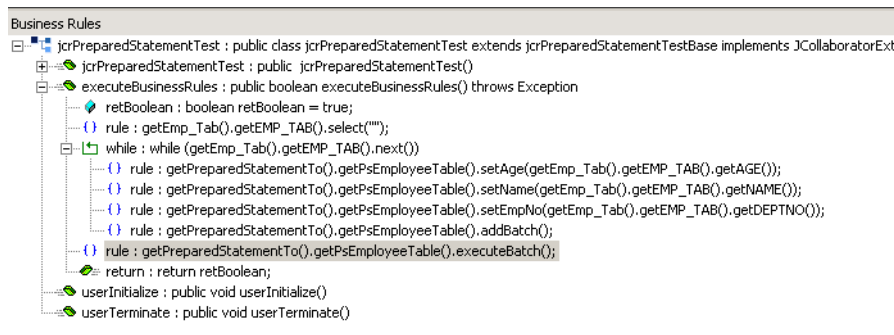
**Figure 25** TableJavaClass





**Figure 26** Collaboration Rule Editor

- 4 Map the fields you would like to populate the table with from the Source Events to the Destination Events.
- 5 Create your rules by dragging and dropping each node into the Rules window. This particular example shows the Age, Name and Department Number of three employees.

**Figure 27** Collaboration Business Rules

- 6 Run the schema and verify the new content in Employee database.

### 4.3.5 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.

```

getDBEmp().getDB_EMPLOYEE().select("");
while (getDBEmp().getDB_EMPLOYEE().next())
{

getPSDBEmp().getDbEmployeeUpdate().setEmployeeNo(getDBEmp().getDB_EMPLOYEE().getEMP_NO());
getPSDBEmp().getDbEmployeeUpdate().setNewTime(ts);
getPSDBEmp().getDbEmployeeUpdate().addBatch();
}

```

```
getPSDBEmp().getDbEmployeeUpdate().executeBatch();
```

### 4.3.6 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 20.

- 1 You will need to specify "cursor" for the connection attribute "SelectMethod" for the following reasons:
  - A This e\*Way implementation uses the same JDBC connection to create multiple JDBC statements.
  - B and the default transaction behavior is set to "false" for Autocommit.

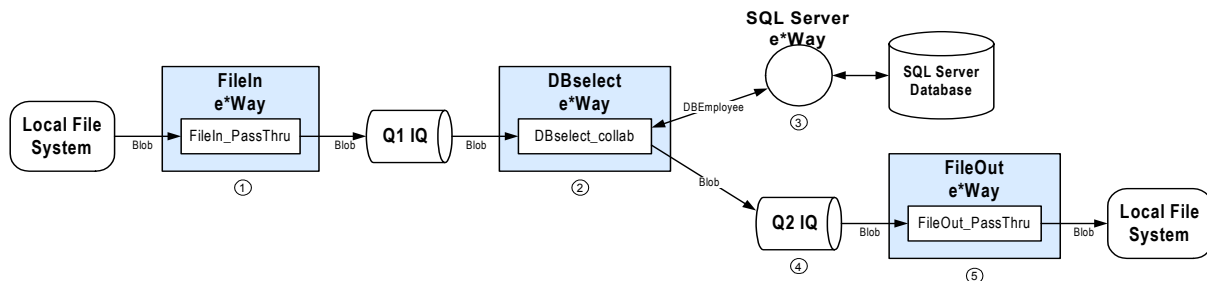
---

## 4.4 Sample Scenario—Polling from a Database

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

Figure 28 shows a graphical overview of the sample schema.

**Figure 28** The Database Select Scenario—Overview



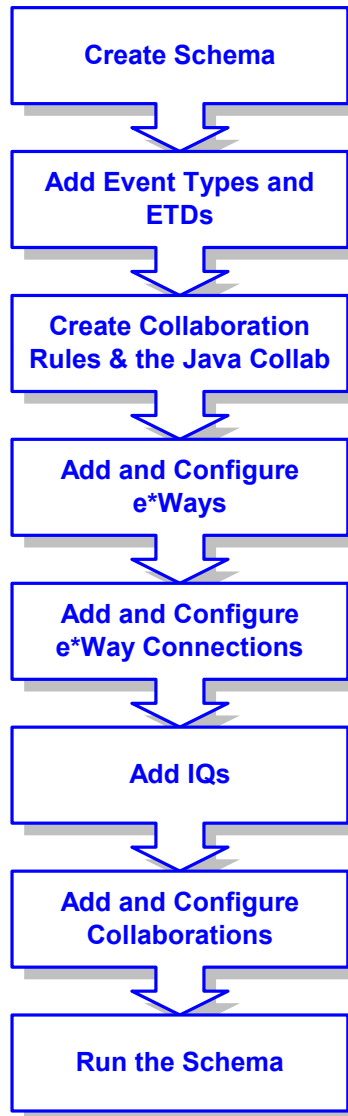
- 1 The **FileIn** e\*Way retrieves an Event (text file) containing the database select criteria and publishes it to the **Q1 IQ**.
- 2 The **DBselect** e\*Way retrieves the Generic Event (**Blob**) from the IQ. This triggers the rest of the Collaboration which has two parts.
- 3 The information in **Blob** is used to retrieve information from the database via the **SQLSERVER\_eWc** e\*Way Connection. This e\*Way Connection contains information used by the Collaboration to connect to the SQL Server 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:

- 1 Create the Schema
- 2 Add the Event Types and Event Type Definitions
- 3 Create the Collaboration Rules and the Java Collaboration
- 4 Add and Configure the e\*Ways
- 5 Add and Configure the e\*Way Connections
- 6 Add the IQs
- 7 Add and Configure the Collaborations
- 8 Run the Schema

**Figure 29** Schema Configuration Steps



**External Database Tables**

The sample uses a simple external SQL Server database with a table called **DB\_EMPLOYEE**. The table contains the following columns:

**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	DATETIME	The last transaction date for the employee

### 4.4.1 Create the Schema

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

- 1 Launch the e\*Gate Enterprise Manager 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 Enterprise Manager will launch and display the newly the created schema.

### 4.4.2 Add the Event Types and Event Type Definitions

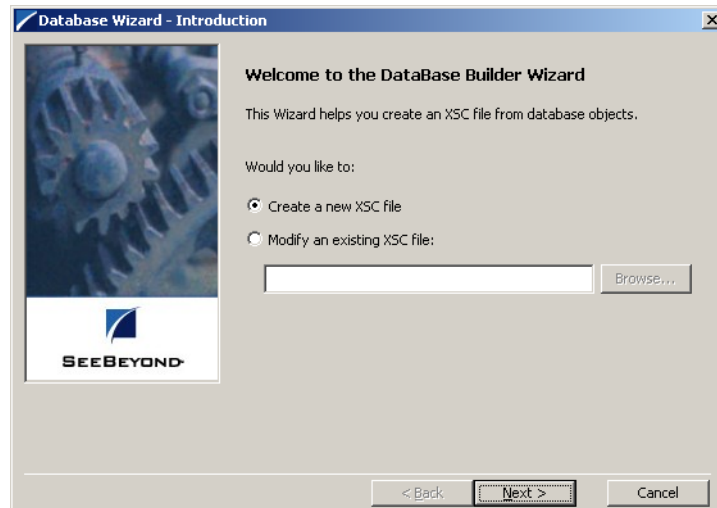
Two Event Types and Event Type Definitions are used in this sample.

- **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 will be 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 will be manually created as a fixed-length ETD.

#### To create the DBEmployee Event Type and ETD

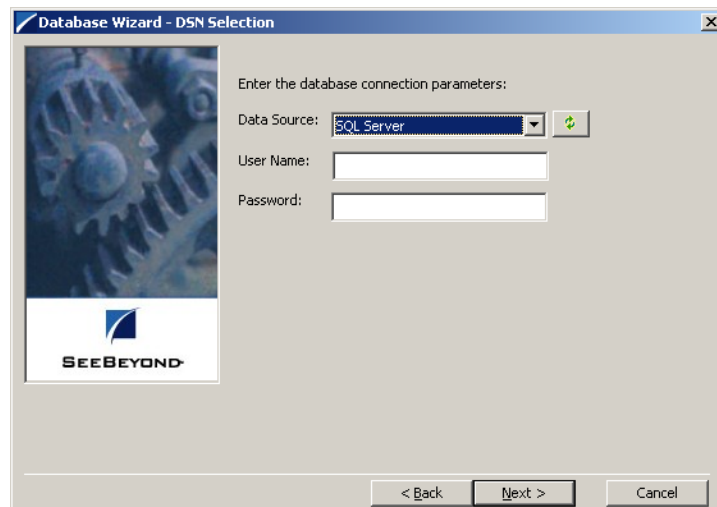
- 1 From the **Options** menu of the Enterprise Manager, choose **Default Editor...**
- 2 Verify that **Java** is selected, then click **OK**.
- 3 In the **Components** pane of the Enterprise Manager, select the **Event Types** folder.
- 4 Click the **New Event Type** button to add a new Event Type.
- 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 will appear.
- 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 30.

**Figure 30** 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 31

**Figure 31** 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 personal. 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 display 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 will appear. 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 will generate and display 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 Enterprise Manager, select the **Event Types** folder.
- 2 Click the **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 On the **File** menu, click **New**. The New Event Type Definition dialog box will appear.
- 7 In the New Event Type Definition dialog box, select **Standard ETD** and click **OK**.
- 8 Read the introductory screen and then click **Next** to continue. The Package Name dialog box will appear.
- 9 Enter **GenericBlobPackage** for the **Package Name** and click **Next** to continue.
- 10 Read the summary information and click **Finish** to generate the ETD.
- 11 In the **Event Type Definition** pane, right-click the root node, point to **Add Field** in the shortcut menu, and click **As Child Node**.
- 12 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	0

Node	Property	Value
Child Node	Name	Data
	Structure	fixed
	Length	0

- 13 On the **File** menu, click **Save**.
- 14 Enter the name **GenericBlob.xsc** and click **OK**.
- 15 On the **File** menu, click **Promote to Run Time**, and then click **OK** when finished.
- 16 On the **File** menu, click **Close** to exit the ETD Editor.
- 17 In the Event Type properties dialog box, click **OK** to save and close the Event Type.

### 4.4.3 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.

#### To create the GenericPassThru Event Type

- 1 In the components pane of the Enterprise Manager, 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 Click the **Subscriptions** tab, select the **GenericBlob** Event Type, and click the right arrow.
- 6 Click the **Publications** tab, select the **GenericBlob** Event Type, and click the right arrow.
- 7 Click **OK** to save the Collaboration Rule.

#### To create the DBSelect Event Type

- 1 In the components pane of the Enterprise Manager, 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 32:

**Figure 32** DBSelect Instances

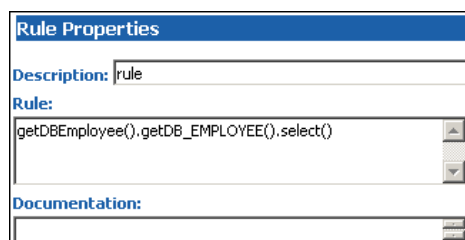
Instance Name	ETD	Find ...	Mode	Trigger	Manual Publish
GenericBlobIn	GenericBlob.xsc	Find ...	In	<input checked="" type="checkbox"/>	<input type="checkbox"/>
GenericBlobOut	GenericBlob.xsc	Find ...	Out	<input type="checkbox"/>	<input checked="" type="checkbox"/>
DBEmployee	DBEmployee.xsc	Find ...	In/Out	<input type="checkbox"/>	<input type="checkbox"/>

- 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. Note that Source and Destination Events are already supplied based on the Collaboration Rule’s Collaboration Mapping (see Figure 32).

- 11 From the **View** menu, choose **Display Code**.  
This displays the Java code associated with each of the Collaboration’s rules.
- 12 From the **Tools** menu, choose **Options**, and then click **Add File....** Select `\eGate\client\classes\stcjdbc.jar` and click **OK** to close each of the dialog boxes.
- 13 In the Business Rules pane, select the **retBoolean** rule and click the **rule** button to add a new Rule.
- 14 In the **Destination Events** pane, expand the **DBEmployee** Event Type until the **select** method is visible.
- 15 Drag the **select** method into the **Rule** field of the **Rule Properties** pane. Click **OK** to close the dialog box without entering any criteria.

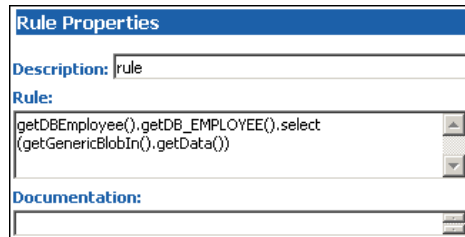
**Figure 33** Rule Properties



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

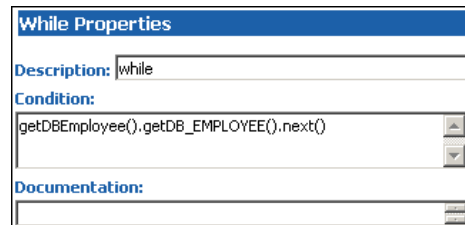
- 17 In the **Rule Properties** pane, position the cursor inside the parentheses of the **select** method. Then drag the **Data** node from the **Source Events** pane into the **select** method's parentheses. (See Figure 34).

**Figure 34** Rule Properties (Continued)



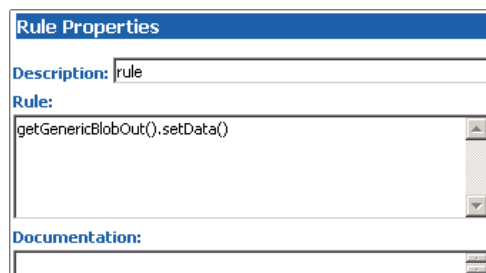
- 18 Select the newly edited rule in the **Business Rules** pane and click the **while** button to add a new while loop beneath the current rule.
- 19 Drag the **next** method from the **Destination Events** pane into the **Condition** field of the **While Properties** pane. (See Figure 35).

**Figure 35** While Properties



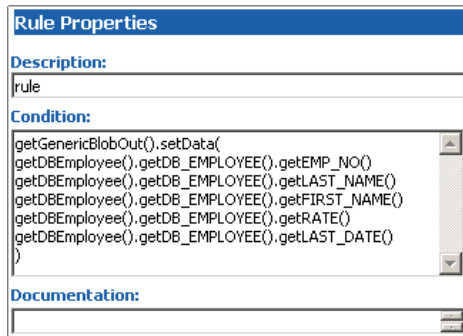
- 20 Select the newly edited while loop in the Business Rules pane and click the **rule** button to add a new rule as a **child** to the while loop.
- 21 In the **Destination Events** pane, expand the **GenericBlobOut** Event Type until the **Data** node is visible.
- 22 Drag the **Data** node into the **Rule** field of the Rule Properties pane. (See Figure 36).

**Figure 36** Rule Properties



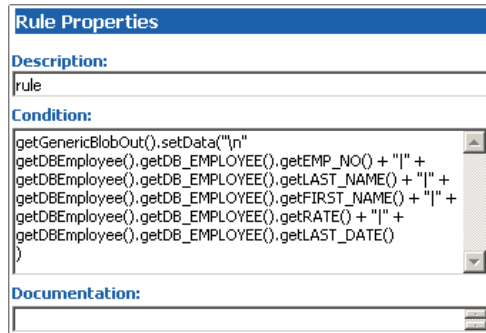
- 23 In the Rule Properties pane, position the cursor inside the parentheses of the **setData()** method. Then drag each of the five data nodes of **DB\_EMPLOYEE** from the Source Events into the parentheses of the rule. (See Figure 37).

**Figure 37** Rule Properties (Continued)



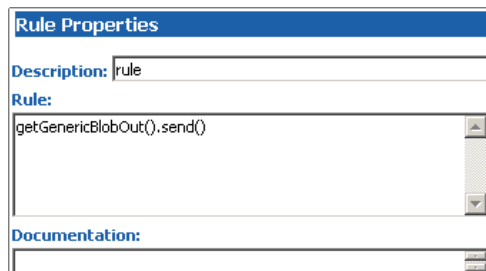
- 24 Edit the text of the condition to add a newline character and pipe (|) delimiters between each of the five data nodes. (See Figure 38).

**Figure 38** Rule Properties (Continued)



- 25 Select the newly edited rule in the **Business Rules** pane and click the **rule** button to add a new rule inside the while loop.
- 26 Drag the root node of the **GenericBlobOut** Event into the **rule** field in the **Rule Properties** pane.
- 27 Edit the rule; add a **send()** method as shown in Figure 39.

**Figure 39** GenericBlobOut send()



- 28 On the **File** menu, click **Save** to save the file.
- 29 On the **File** menu, click **Compile** to compile the Collaboration.  
View the bottom pane to ensure that there were no compiler errors.

- 30 On the **File** menu, click **Exit** to close the Java Collaboration Editor and return to the Collaboration Rule.

Note that the **Collaboration Rules** and **Initialization file** fields have been completed by closing the Java Collaboration Editor.

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

#### 4.4.4 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 Enterprise Manager, 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 Enterprise Manager, 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 Enterprise Manager, 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

Section Name	Parameter	Value
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.

#### 4.4.5 Add and Configure the e\*Way Connections

The sample scenario uses one e\*Way Connection:

- **SQLSERVER\_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 Enterprise Manager, select the **e\*Way Connections** folder.
- 2 Click the **New e\*Way Connection** button to add a new e\*Way Connection.
- 3 Enter **SQLSERVER\_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 **SQLSERVER** from the e\*Way Connection Type drop down list.
- 6 Click **New** to create a new configuration file.
- 7 Enter the parameters for the e\*Way Connection as shown in Table 7.

**Table 7** SQLSERVER\_eWc e\*Way Connection Parameters

Section Name	Parameter	Value
DataSource	class	default
	DriverType	default
	ServerName	Use the ODBC Data Source name.
	PortNumber	Use the default port number .
	DatabaseName	Use the name of the SQL Server database.
	user name	Use local settings.
	password	Use local settings.
	retry interval	default
connector	All	default

- 8 Select **Save** from the **File** menu. Enter **SQLSERVER\_eWc** as the file name and click **Save**.
- 9 Select **Promote to Run Time** from the **File** menu. Click **OK** to close the e\*Way Connection configuration file editor.
- 10 Click **OK** to save the e\*Way Connection's properties.

#### 4.4.6 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 Enterprise Manager, 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 Manger and click the **Properties** button.
- 6 Select the **Start automatically** check box and click **OK** to save the properties.

#### 4.4.7 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 Enterprise Manager, 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 dropdown 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 Enterprise Manager, 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 **GenericEventToDatabase** from the dropdown list of Collaboration Rules.
- 6 Click the upper **Add** button to add a new Subscription.
- 7 Select the **GenericEvent** 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 **SQLSERVER\_eWc** destination.
- 10 Click the lower **Add** button to add a new Publication
- 11 Select the **GenericEvent** 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 Enterprise Manager, 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 dropdown list of Collaboration Rules.
- 6 Click the upper **Add** button to add a new Subscription.
- 7 Select the **GenericEvent** Event Type and the **DBSelect\_collab** source.
- 8 Click the lower **Add** button to add a new Publication
- 9 Select the **GenericEvent** Event Type and the **<External>** destination.
- 10 Click **OK** to close the Collaboration's properties.

### 4.4.8 Run the Schema

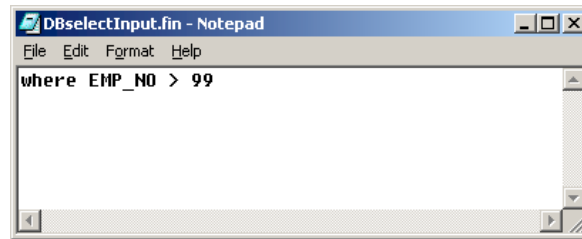
Running the sample Schema requires that a sample input file be created. Once the input file has been created, you can start the Control Broker from a command prompt to execute the Schema. After the Schema has been run, you can view the output text file to verify the results.

#### 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 40.



**Figure 40** 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

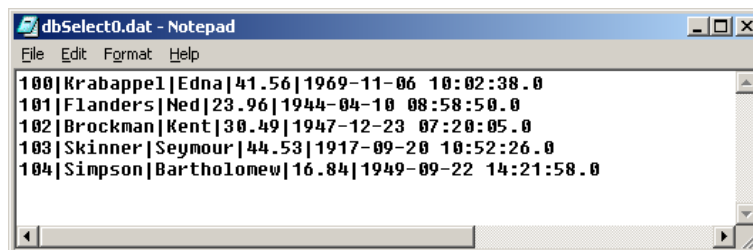
*user\_name* and *password* are a valid e\*Gate username/password combination.

**To verify the results**

Use a text editor to view the output file `C:\eGate\data\dbSelect_out\dbSelect0.dat`.

Figure 41 shows an example of the records that were returned by the sample schema.

**Figure 41** Sample Output File



# SQL Server e\*Way Methods

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

- [com.stc.eways.jdbcx.StatementAgent Class](#) on page 58
- [com.stc.eways.jdbcx.PreparedStatementAgent Class](#) on page 68
- [com.stc.eways.jdbcx.PreparedStatementResultSet Class](#) on page 80
- [com.stc.eways.jdbcx.SqlStatementAgent Class](#) on page 106
- [com.stc.eways.jdbcx.CallableStatementAgent Class](#) on page 108
- [com.stc.eways.jdbcx.TableResultSet Class](#) on page 120

---

## 5.1 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

- [cancel](#) on page 66
- [clearWarnings](#) on page 67
- [getFetchDirection](#) on page 63
- [getMaxFieldSize](#) on page 65
- [getMoreResults](#) on page 66
- [getResultSetConcurrency](#) on page 62
- [getUpdateCount](#) on page 65
- [isClosed](#) on page 60
- [queryName](#) on page 60
- [resultSetConcurToString](#) on page 60
- [resultSetTypeToString](#) on page 59
- [sessionOpen](#) on page 61
- [setEscapeProcessing](#) on page 62
- [setMaxFieldSize](#) on page 65
- [setQueryTimeout](#) on page 63
- [stmtInvoke](#) on page 67
- [clearBatch](#) on page 66
- [executeBatch](#) on page 66
- [getFetchSize](#) on page 64
- [getMaxRows](#) on page 64
- [getResultSet](#) on page 65
- [getResultSetType](#) on page 61
- [getWarnings](#) on page 67
- [queryDescription](#) on page 60
- [resetRequested](#) on page 61
- [resultSetDirToString](#) on page 59
- [sessionClosed](#) on page 61
- [setCursorName](#) on page 62
- [setFetchDirection](#) on page 63
- [setMaxRows](#) on page 64
- [setQueryTimeout](#) on page 63

### resultSetTypeToString

This method gets the symbol string corresponding to the ResultSet 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.

```
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

Returns query timeout duration.

```
public int getQueryTimeout()
```

### Returns

The number of seconds to wait before timeout.

### 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.SQLException

---

**getFetchSize**

Returns the result set prefetch record count.

```
public int getFetchSize()
```

**Returns**

The fetch size this StatementAgent object set.

**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. 0 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.SQLException

---

## 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

---

## 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.
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.

---

## 5.2 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

[addBatch](#) on page 79

[execute](#) on page 79

[executeUpdate](#) on page 79

[setArray](#) on page 77

[setBigDecimal](#) on page 73

[setBlob](#) on page 78

[setByte](#) on page 71

[setCharacterStream](#) on page 77

[setDate](#) on page 73

[setDouble](#) on page 73

[setInt](#) on page 72

[setNull](#) on page 69

[setObject](#) on page 70

[setRef](#) on page 78

[setString](#) on page 75

[setTime](#) on page 74

[setTimestamp](#) on page 75

[clearParameters](#) on page 79

[executeQuery](#) on page 79

[sessionOpen](#) on page 110

[setAsciiStream](#) on page 76

[setBinaryStream](#) on page 76

[setBoolean](#) on page 71

[setBytes](#) on page 76

[setClob](#) on page 78

[setDate](#) on page 74

[setFloat](#) on page 72

[setLong](#) on page 72

[setObject](#) on page 69

[setObject](#) on page 70

[setShort](#) on page 71

[setTime](#) on page 74

[setTimestamp](#) on page 75

---

## 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 inava.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 inava.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 inava.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 inava.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.
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 calendar 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.

Name	Description
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

---

## 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

---

## 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

---

## 5.3 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

- [absolute](#) on page 84
- [beforeFirst](#) on page 86
- [close](#) on page 84
- [findColumn](#) on page 87
- [getArray](#) on page 101
- [getAsciiStream](#) on page 100
- [getBigDecimal](#) on page 93
- [getBinaryStream](#) on page 100
- [getBlob](#) on page 102
- [getBoolean](#) on page 89
- [getByte](#) on page 90
- [getBytes](#) on page 99
- [getCharacterStream](#) on page 101
- [getClob](#) on page 102
- [getConcurrency](#) on page 82
- [getDate](#) on page 94
- [getDate](#) on page 95
- [getDouble](#) on page 93
- [getFetchDirection](#) on page 82
- [getFloat](#) on page 92
- [getInt](#) on page 91
- [getLong](#) on page 91
- [getMetaData](#) on page 82
- [getObject](#) on page 88
- [getObject](#) on page 89
- [getRef](#) on page 104
- [getShort](#) on page 90
- [getString](#) on page 98
- [getTime](#) on page 95
- [getTime](#) on page 96
- [getTimestamp](#) on page 97
- [getTimestamp](#) on page 97
- [getType](#) on page 87
- [insertRow](#) on page 105
- [isBeforeFirst](#) on page 86
- [isAfterLast](#) on page 86
- [clearWarnings](#) on page 104
- [deleteRow](#) on page 105
- [first](#) on page 85
- [getArray](#) on page 101
- [getAsciiStream](#) on page 100
- [getBigDecimal](#) on page 94
- [getBinaryStream](#) on page 101
- [getBlob](#) on page 102
- [getBoolean](#) on page 89
- [getByte](#) on page 90
- [getBytes](#) on page 99
- [getCharacterStream](#) on page 101
- [getClob](#) on page 103
- [getCursorName](#) on page 84
- [getDate](#) on page 94
- [getDate](#) on page 95
- [getDouble](#) on page 93
- [getFetchSize](#) on page 83
- [getFloat](#) on page 93
- [getInt](#) on page 91
- [getLong](#) on page 92
- [getObject](#) on page 87
- [getObject](#) on page 88
- [getRef](#) on page 103
- [getRow](#) on page 105
- [getShort](#) on page 90
- [getString](#) on page 98
- [getTime](#) on page 95
- [getTime](#) on page 96
- [getTimestamp](#) on page 97
- [getTimestamp](#) on page 98
- [getWarnings](#) on page 104
- [isAfterLast](#) on page 87
- [isFirst](#) on page 85

[isLast](#) on page 86

[next](#) on page 84

[refreshRow](#) on page 126

[getFetchDirection](#) on page 82

[updateRow](#) on page 105

[last](#) on page 85

[previous](#) on page 84

[relative](#) on page 85

[getFetchSize](#) on page 83

[wasNull](#) on page 104

---

## 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

---

## 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

---

## getBytes

Gets the byte value of the specified column.

```
public byte getBytes(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 in 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

---

---

## 5.4 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

[addBatch](#) on page 108

[execute](#) on page 107

[executeQuery](#) on page 107

[executeUpdate](#) on page 108

---

## 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(java.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

---

## 5.5 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

ResetEventListener, 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

[getArray](#) on page 118

[getBlob](#) on page 119

[getBytes](#) on page 113

[getClob](#) on page 119

[getDate](#) on page 116

[getFloat](#) on page 114

[getLong](#) on page 114

[getObject](#) on page 112

[getShort](#) on page 113

[getTime](#) on page 116

[getTimestamp](#) on page 117

[registerOutParameter](#) on page 111

[sessionOpen](#) on page 110

[getBigDecimal](#) on page 115

[getBoolean](#) on page 113

[getBytes](#) on page 118

[getDate](#) on page 115

[getDouble](#) on page 115

[getInt](#) on page 114

[getObject](#) on page 112

[getRef](#) on page 120

[getString](#) on page 118

[getTimestamp](#) on page 117

[registerOutParameter](#) on page 111

[registerOutParameter](#) on page 111

[wasNull](#) on page 112

---

## 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

---

## sessionOpen

Opens the session event handler.

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

### Overrides

sessionOpen in class PreparedStatementAgent

Name	Description
evt	Session event.

## 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 <code>inava.sql.Types</code> .

### 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 <code>inava.sql.Types</code> .
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.
iType	A JDBC type defined by <code>inava.sql.Types</code> .
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

---

## getBytes

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

## 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

---

## 5.6 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

[absolute](#) on page 122

[beforeFirst](#) on page 123

[deleteRow](#) on page 126

[first](#) on page 122

[getAsciiStream](#) on page 125

[getBinaryStream](#) on page 125

[getCharacterStream](#) on page 126

[isAfterLast](#) on page 124

[afterLast](#) on page 124

[cancelRowUpdates](#) on page 127

[findColumn](#) on page 124

[getAsciiStream](#) on page 125

[getBinaryStream](#) on page 125

[getCharacterStream](#) on page 125

[insertRow](#) on page 126

[isBeforeFirst](#) on page 124



[isFirst](#) on page 123

[last](#) on page 123

[moveToInsertRow](#) on page 126

[previous](#) on page 121

[rowDeleted](#) on page 127

[rowUpdated](#) on page 127

[updateRow](#) on page 126

[isLast](#) on page 123

[moveToCurrentRow](#) on page 127

[next](#) on page 121

[relative](#) on page 122

[rowInserted](#) on page 127

[select](#) on page 121

[wasNull](#) on page 128

---

## 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

---

## 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

---

## 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 if 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
```



# Index

## B

Batch Operations 41

## C

class 17  
 class parameter  
   Connector settings 19  
 Collaboration Service  
   Java 24  
 Collecting Data from Output Parameters 37  
 component relationship 24  
 components, Java-enabled 24  
 configuration file sections  
   Connector settings 19  
 configuration parameters  
   class 19  
   DatabaseName 18  
   password 18  
   PortNumber 18  
   ServerName 18  
   type 19  
   user name 18  
 configuration steps, schema 44  
 configuring e\*Way connections 16–19  
 connection establishment mode 20  
 connection inactivity timeout 21  
 Connection Manager 21  
 connection types, JDBC 19  
 connection verification interval 21  
 connector objects, JDBC 19  
 Connector settings 19  
 creating e\*Way connections 16

## D

Database Configuration Node 42  
 DatabaseName parameter 18  
 DataDirect 2.2 Drivers for XA Mode 14

## E

e\*Way connections  
   configuring 16–19

creating 16

## G

getMoreResults 38  
 getResultSet 38  
 getUpdateCount 38

## H

host system requirements 10

## J

Java Collaboration Service 24  
 Java ETD Builder 24–33  
 Java-enabled components 24  
 JDBC 25  
   connection types 19  
   connector objects 19

## M

Microsoft ODBC driver for SQL Server 11  
 Mixing XA-Compliant and XA-Noncompliant  
 e\*Way Connections 20

## O

ODBC drivers 11  
 or 12

## P

password parameter 18  
 PortNumber parameter 18  
 Prepared Statement 39

## R

Registering the JDBC .jar Files 14  
 requirements  
   host system 10  
   system 10  
 Returning Result Sets from Sybase, DB2, and SQL  
 Server 38

## S

schema configuration steps 44  
 SelectMethod 19  
 Server Name parameter 18  
 Stceway Connection 23

## Index

stcjdbcx.jar **9, 49**  
system requirements **10**

## T

The Stored Procedure **37**  
timeout **19**  
transaction mode **20**  
type parameter **19**

## U

UNIX **13**  
user name parameter **18**  
userInitialize() **24**  
userTerminate() **24**

## W

Windows NT or Windows 2000 **9, 10, 12**

## X

XA Compliant **18**