

***SeeBeyond ICAN Suite***

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

*Release 5.0.5 for Schema Run-time Environment (SRE)*

*Java Version*



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

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

© 2005 SeeBeyond Technology Corporation. All Rights Reserved. This work is protected as an unpublished work under the copyright laws.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 20050405201721.

# Contents

---

## Chapter 1

<b>Introduction</b>	<b>10</b>
<b>Overview</b>	<b>10</b>
Intended Reader	10
Components	10
<b>Operational Overview</b>	<b>11</b>
<b>Supported Operating Systems</b>	<b>11</b>
<b>System Requirements</b>	<b>11</b>
GUI Host Requirements	11
Participating Host Requirements	12
External System Requirements	12

---

## Chapter 2

<b>Installation</b>	<b>13</b>
<b>Installing the Sybase e*Way on Windows</b>	<b>13</b>
Pre-installation	13
Installation Procedure	13
<b>Installing the Sybase e*Way on UNIX</b>	<b>14</b>
Pre-installation	14
Installation Procedure	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
Connection Method	18
ServerName	18
PortNumber	18
DatabaseName	18
user name	18
password	19

SelectMethod	19
timeout	19
Connector Settings	19
type	19
class	19
transaction mode	20
connection establishment mode	20
connection inactivity timeout	20
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

## Chapter 4

<b>Implementation</b>	<b>23</b>
<b>Implementing Java-enabled Components</b>	<b>23</b>
The Java Collaboration Service	24
Java-enabled Components	24
<b>The Java ETD Builder</b>	<b>24</b>
The Parts of the ETD	24
Using the DBWizard ETD Builder	25
The Generated ETDs	33
Editing an Existing .XSC Using the Database Wizard	34
<b>Using ETDs with Tables, Views, Stored Procedures, and Prepared Statements</b>	<b>34</b>
<b>Tables</b>	<b>34</b>
The query Operation	35
The insert Operation	36
The update Operation	38
The delete Operation	39
<b>The View</b>	<b>40</b>
<b>The Stored Procedure</b>	<b>40</b>
Executing Stored Procedures	40
Manipulating the ResultSet and Update Count Returned by Stored Procedure	41
<b>Prepared Statement</b>	<b>44</b>
<b>Batch Operations</b>	<b>44</b>
<b>Database Configuration Node</b>	<b>44</b>
<b>Sample Scenario—Polling from a Sybase Database</b>	<b>45</b>
Create the Schema	47
Add the Event Types and Event Type Definitions	47
Create the Collaboration Rules and the Java Collaboration	50
Add and Configure the e*Ways	54
Add and Configure the e*Way Connections	56
Add the IQs	57
Add and Configure the Collaborations	57
Run the Schema	59

## Chapter 5

<b>Sybase e*Way Methods</b>	<b>60</b>
<b>com.stc.eways.jdbcx.StatementAgent Class</b>	<b>60</b>
resultSetTypeToString	61
resultSetDirToString	61
resultSetConcurToString	62
isClosed	62
queryName	62
queryDescription	62
sessionOpen	63
sessionClosed	63
resetRequested	63
getResultSetType	63
getResultSetConcurrency	64
setEscapeProcessing	64
setCursorName	64
setQueryTimeout	65
setQueryTimeout	65
getFetchDirection	65
setFetchDirection	65
getFetchSize	66
getMaxRows	66
setMaxRows	66
getMaxFieldSize	67
setMaxFieldSize	67
getUpdateCount	67
getResultSet	67
getMoreResults	68
clearBatch	68
executeBatch	68
cancel	68
getWarnings	69
clearWarnings	69
stmtInvoke	69
<b>com.stc.eways.jdbcx.PreparedStatementAgent Class</b>	<b>70</b>
sessionOpen	70
setNull	71
setNull	71
setObject	71
setObject	72
setObject	72
setBoolean	73
setByte	73
setShort	73
setInt	74
setLong	74
setFloat	74
setDouble	75
setBigDecimal	75
setDate	75
setDate	76
setTime	76
setTime	76
setTimestamp	77
setTimestamp	77
setString	77
setBytes	78
setAsciiStream	78
setBinaryStream	78

setCharacterStream	79
setArray	79
setBlob	80
setClob	80
setRef	80
clearParameters	81
addBatch	81
execute	81
executeQuery	81
executeUpdate	81
<b>com.stc.eways.jdbcx.PreparedStatementResultSet Class</b>	<b>82</b>
Constructor PreparedStatementResultSet	84
getMetaData	84
getConcurrency	84
getFetchDirection	84
setFetchDirection	85
getFetchSize	85
setFetchSize	85
getCursorName	86
close	86
next	86
previous	86
absolute	86
relative	87
first	87
isFirst	87
last	87
isLast	88
beforeFirst	88
isBeforeFirst	88
afterLast	88
isAfterLast	89
getType	89
findColumn	89
getObject	89
getObject	90
getObject	90
getObject	91
getBoolean	91
getBoolean	91
getByte	92
getShort	92
getShort	92
getInt	93
getInt	93
getLong	93
getLong	94
getFloat	94
getFloat	95
getDouble	95
getBigDecimal	95
getBigDecimal	96
getDate	96
getDate	96
getDate	97
getTime	97
getTime	97
getTime	98
getTime	98
getTimestamp	99
getTimestamp	99
getTimestamp	99
getTimestamp	100

getString	100
getString	100
getBytes	101
getBytes	101
getAsciiStream	102
getAsciiStream	102
getBinaryStream	102
getBinaryStream	103
getCharacterStream	103
getArray	103
getBlob	104
getBlob	104
getClob	104
getClob	105
getRef	105
getRef	106
wasNull	106
getWarnings	106
clearWarnings	106
getRow	107
refreshRow	107
insertRow	107
updateRow	107
deleteRow	107
<b>com.stc.eways.jdbcx.SqlStatementAgent Class</b>	<b>108</b>
Constructor SqlStatementAgent	108
Constructor SqlStatementAgent	108
execute	109
executeQuery	109
executeUpdate	109
addBatch	110
<b>com.stc.eways.jdbcx.CallableStatementAgent Class</b>	<b>110</b>
Constructor CallableStatementAgent	111
Constructor CallableStatementAgent	111
Constructor CallableStatement Agent	112
sessionOpen	112
registerOutParameter	112
registerOutParameter	113
registerOutParameter	113
wasNull	114
getObject	114
getObject	114
getBoolean	115
getByte	115
getShort	115
getInt	116
getLong	116
getFloat	116
getDouble	117
getBigDecimal	117
getDate	117
getDate	118
getTime	118
getTime	118
getTimestamp	119
getTimestamp	119
getString	120
getBytes	120
getArray	120
getBlob	121
getClob	121
getRef	121

<b>com.stc.eways.jdbcx.TableResultSet Class</b>	<b>122</b>
select	123
next	123
previous	123
absolute	124
relative	124
first	124
isFirst	125
last	125
isLast	125
beforeFirst	125
isBeforeFirst	126
afterLast	126
isAfterLast	126
findColumn	126
getAsciiStream	127
getAsciiStream	127
getBinaryStream	127
getBinaryStream	127
getCharacterStream	127
getCharacterStream	128
refreshRow	128
insertRow	128
updateRow	128
deleteRow	128
moveToInsertRow	128
moveToCurrentRow	129
cancelRowUpdates	129
rowInserted	129
rowUpdated	129
rowDeleted	129
wasNull	130
<b>\$DB Configuration Node Methods</b>	<b>130</b>
<b>com_stc_jdbcx_sybasecfg.DataSource</b>	<b>130</b>
getClass	131
setClass	131
hasClass	131
omitClass	131
getConnectionMethod	132
setConnectionMethod	132
hasConnectionMethod	132
omitConnectionMethod	132
getServerName	133
setServerName	133
hasServerName	133
omitServerName	133
getPortNumber	133
setPortNumber	134
hasPortNumber	134
omitPortNumber	134
getDatabaseName	134
setDatabaseName	134
hasDatabaseName	134
omitDatabaseName	135
getUserName	135
setUserName	135
hasUserName	135
omitUserName	135
getPassword	136
setPassword	136
setPassword_Asls	136
hasPassword	136



## Contents

omitPassword	136
getPassword	136
setPassword	137
setPassword_Asls	137
hasPassword	137
omitPassword	137
getSelectMethod	137
setSelectMethod	138
hasSelectMethod	138
omitSelectMethod	138
getTimeout	138
setTimeout	138
hasTimeout	139
omitTimeout	139
<b>com_stc_jdbcx_sybasecfg</b>	<b>139</b>
getDataSource	139
setDataSource	139

## Index

141

# Introduction

This document describes how to install and configure the Java-enabled version of the e\*Way Intelligent Adapter for Sybase.

This Chapter Includes:

- [“Overview” on page 10](#)
- [“Operational Overview” on page 11](#)
- [“System Requirements” on page 11](#)

---

## 1.1 Overview

The Sybase e\*Way enables the e\*Gate system to exchange data with external Sybase databases. This document describes how to install and configure the Java-enabled version of the Sybase e\*Way.

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

### 1.1.1 Intended Reader

The reader of this guide is presumed to be a developer or system administrator with responsibility for maintaining the e\*Gate system; to have expert-level knowledge of Windows and UNIX operations and administration; to be thoroughly familiar with Sybase and SQL functions; and to be thoroughly familiar with Windows-style GUI operations.

### 1.1.2 Components

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

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

A complete list of installed files appears in [Table 1 on page 15](#).

---

## 1.2 Operational Overview

The Java-enabled version of the Sybase e\*Way uses Java Collaborations to interact with one or more external databases. By using the Java Collaboration Service it is possible for e\*Gate components—such as e\*Way Intelligent Adapters (e\*Ways) and Business Object Brokers (BOBs)—to connect to external databases and execute business rules written entirely in Java.

An e\*Gate component is defined as *Java-enabled* based on the selection of the Java Collaboration Service in the Collaboration Rule setup. For more information on the Java Collaboration Service, see [“The Java Collaboration Service” on page 24](#).

---

## 1.3 Supported Operating Systems

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

- Windows 2000, Windows XP, and Windows Server 2003
- HP Tru64 V5.1A
- HP-UX 11.0 and 11i (PA-RISC)
- IBM AIX 5.1L and 5.2
- Sun Solaris 8 and 9
- Japanese Windows 2000, Windows XP, and Windows Server 2003
- Japanese HP-UX 11.0 and 11i (PA-RISC)

---

## 1.4 System Requirements

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

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

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

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

### GUI Host Requirements

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

- DataDirect ODBC 4.1 wire protocol driver included on the driver installation cd.

If the GUI host machine will also be executing the Sybase e\*Way, the host machine must also meet the **“Participating Host Requirements” on page 12.**

## Participating Host Requirements

- DataDirect JDBC 3.2 included on the installation disk.

However, for implementations using protocols other than Type 4, the appropriate client drivers must be installed.

A complete list of drivers from third party vendors can be found at:

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

### 1.4.1 External 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.

- Sybase Server 11.9 or 12.5.

Before running in XA mode you must enter the following configuration on the Sybase server.

- The Sybase license manager must be running
- dtm\_tm\_role needs to be granted to each user
- You must enable DTM:

```
sp_configure 'enable DTM', 1
```

- You must enable transaction coordination:

```
sp_configure 'enable xact coordination', 1
```

# Installation

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

**This Chapter Includes:**

- [“Installing the Sybase e\\*Way on Windows” on page 13](#)
- [“Installing the Sybase e\\*Way on UNIX” on page 14](#)
- [“Files/Directories Created by the Installation” on page 14](#)

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

---

## 2.1 Installing the Sybase e\*Way on Windows

### 2.1.1 Pre-installation

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

### 2.1.2 Installation Procedure

To install the Sybase e\*Way on a Windows operating system:

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

**Note:** *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 technical support personnel, do not change the suggested “installation directory” setting.*

---

## 2.2 Installing the Sybase 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 privilege to create files in the e\*Gate directory tree.

### 2.2.2 Installation Procedure

To install the Sybase 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.

**Note:** *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 technical support personnel, do not change the suggested “installation directory” setting.*

---

## 2.3 Files/Directories Created by the Installation

The Sybase 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

<b>e*Gate Directory</b>	<b>File(s)</b>
client\configs\sybase\	sybase.def
client\etd\	dbwizard.ctl
server\registry\repository\configs\sybase	sybase.def
server\registry\repository\default\etd\	dbwizard.ctl
ThirdParty\merant\classes	DGbase.jar
ThirdParty\merant\classes	DGsybase.jar
ThirdParty\merant\classes	DGutil.jar
ThirdParty\merant\classes	spy.jar

# e\*Way Connection Configuration

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

This Chapter Includes:

- “Create e\*Way Connections” on page 16
- “DataSource Settings” on page 17
- “Connector Settings” on page 19

## 3.1 Create e\*Way Connections


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

To create and configure the e\*Way Connections:

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

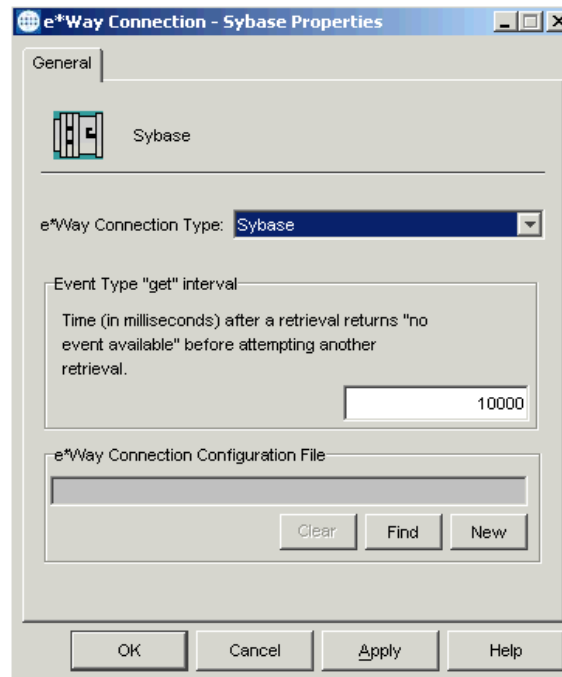
**Figure 1** The e\*Way Connections Folder



- 2 On the Palette, click the **New e\*Way Connection** button .
- 3 The **New e\*Way Connection Configuration** dialog box opens. Enter a name for the e\*Way Connection and click **OK**.
- 4 Double-click the new e\*Way Connection to open the e\*Way Connection Properties dialog box. See.



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



- 5 In the **e\*Way Connection Type** list, click **Sybase**.
- 6 In the **Event Type “get” interval** box, enter an interval.
- 7 Click **New** to create a new e\*Way Connection Configuration File.

The e\*Way Connection Configuration File Editor will appear.

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.sybase.SybaseDataSource`.

## Connection Method

Specifies which method is used to connect to the database server:

**Pooled Data Source** - A `ConnectionPoolDataSource` object for creating `PooledConnection` objects. A `PooledConnection` object represents a physical connection and is cached in memory for reuse, which saves the overhead of establishing a new connection. This is implemented by the driver.

**XA Data Source** - An `XADataSource` object for creating `XAConnection` objects, connections that can be used for distributed transaction.

You should make sure that the class specified in the "class" parameter supports the connection method that is used here.

### Required Values

The default is "Pooled Data Source"

## ServerName

### Description

Specifies the host name of the external database server.

### Required Values

Any valid string.

## PortNumber

### Description

Specifies the port on which the server is listening for connection requests. Default is 5000

### 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

Determines whether the database cursors are used for Select statements. Performance of the DataDirect driver when performing queries is affected by the choice of direct or cursor for SeletMethod.

For XA operations, this parameter needs to be set to cursor mode.

### Required Values

Cursor.

## timeout

### Description

This is the login time out in seconds.

### Required Values

The default value 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

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 the enhanced e\*Gate 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

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

## Controlling When a Connection is Made

As a user, you can control when a connection is made. Using Connector Settings, you 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.

# Implementation

This chapter discusses how to implement the Java-enabled Sybase e\*Way in a production environment. Also included is a sample configuration. This chapter assumes that the Sybase e\*Way has been successfully installed.

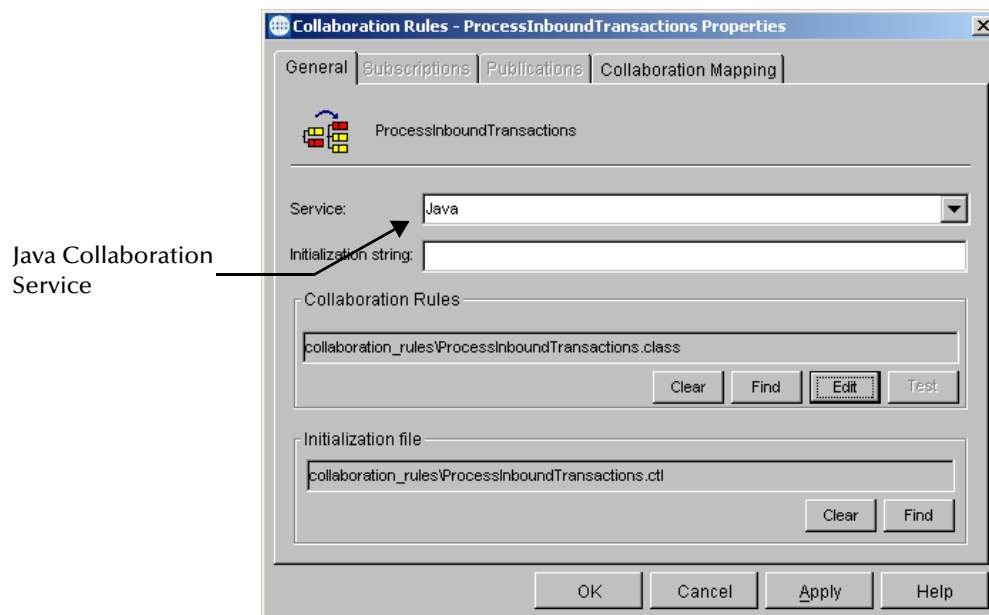
This Chapter Includes:

- “Implementing Java-enabled Components” on page 23
- “The Java ETD Builder” on page 24
- “Sample Scenario—Polling from a Sybase Database” on page 45

## 4.1 Implementing Java-enabled Components

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

**Figure 4** The Java Collaboration Service



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

### 4.1.2 Java-enabled Components

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

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

---

## 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.

**Note:** *Database ETD's are not messagable. For more information on non-messagable ETDs, please see the e\*Gate Integrators Guide.*

### 4.2.1 The Parts of the ETD

There are four possible parts to the Java-enabled Event Type Definition as shown in Figure 5 below.

**Figure 5** The Java-enabled ETD



- **Element** – This is the highest level in the ETD tree. The element is the basic container that holds the other parts of the ETD. The element can contain fields and methods.



- **Field** – Fields are used to represent data. A field can contain data in any of the following formats: string, boolean, int, double, or float.
- **Method** – Method nodes represent actual Java methods.
- **Parameter** – Parameter nodes represent the Java method’s parameters.

## 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, 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.

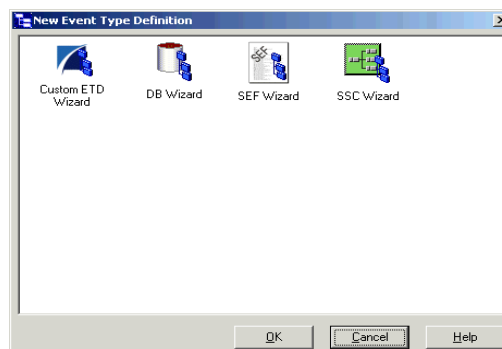
Please note that the DBWizard is using ODBC standard data types. Specific data types that are not ODBC standard will not be supported.

Please note that the DBWizard is using ODBC standard data types. Specific data types that are not ODBC standard will not be supported.

To create a new ETD using the DBWizard

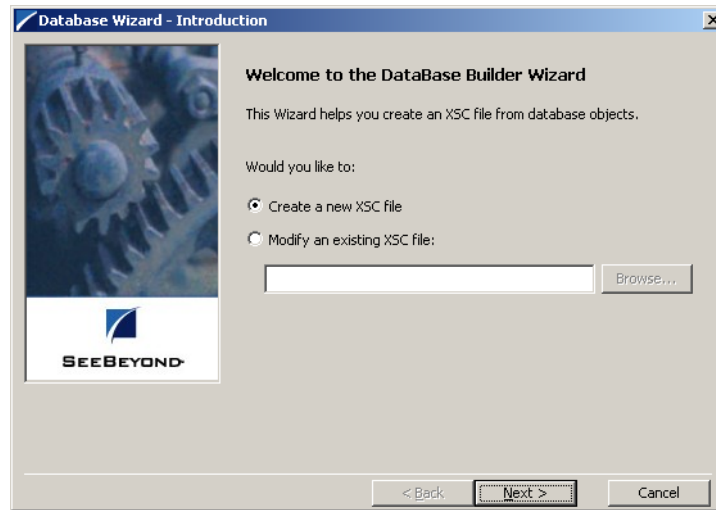
- 1 From the **Options** menu of the Schema Designer, choose **Default Editor...**
- 2 Verify that **Java** is selected, then click **OK**.
- 3 Click the **ETD Editor** button to launch the Java ETD Editor.
- 4 In the **Java ETD Editor**, click the **New** button to launch the New Event Type Definition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

**Figure 6** New Event Type Definition



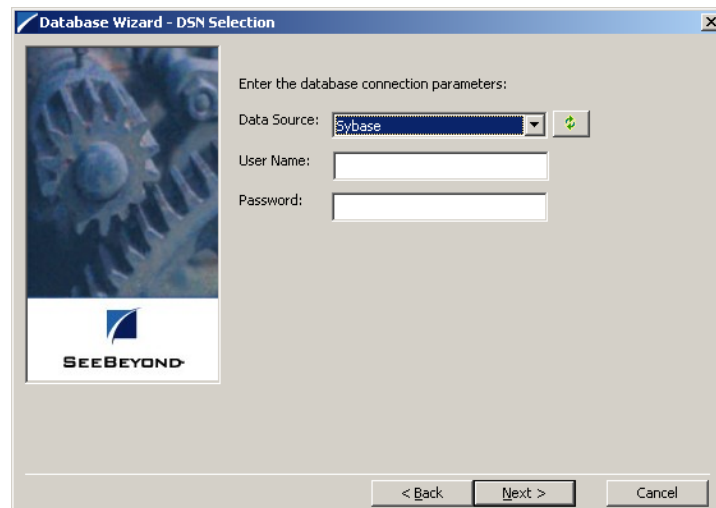
- 6 Enter the name of the new .xsc file you want to create or enter the name of the .xsc file you want to edit by browsing to its location.

**Figure 7** Database Wizard - Introduction



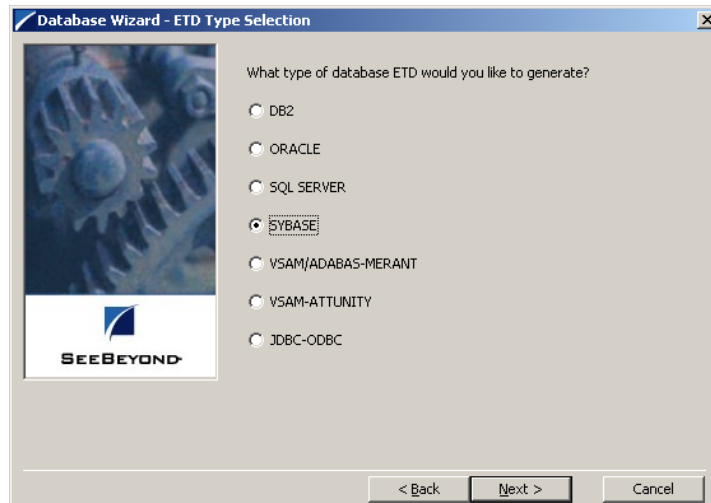
- 7 Select your **Data Source:** from the drop down list and enter your **User Name:** and **Password:**.

**Figure 8** 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 9** Database Wizard - ETD Type Selection



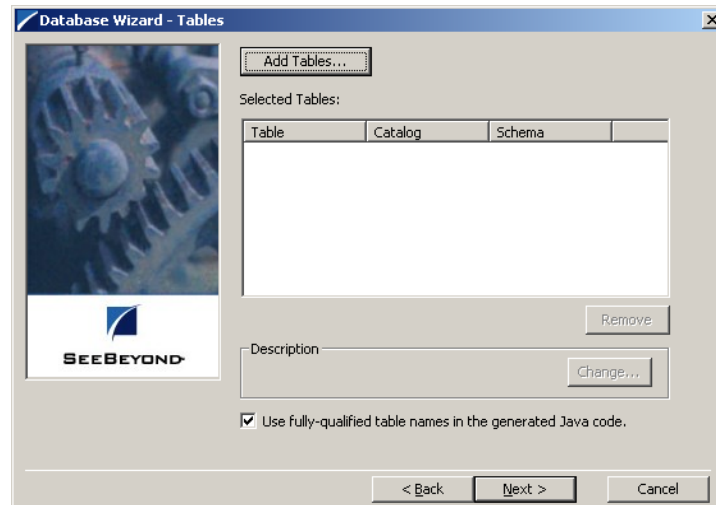
- 9 In the **Database Wizard - Object Selection** window, select any combination of **Tables**, **Views**, **Procedures**, or **Prepared Statements** you would like to include in your .xsc file. Click **Next** to continue.

**Figure 10** Database Wizard - Object Selection



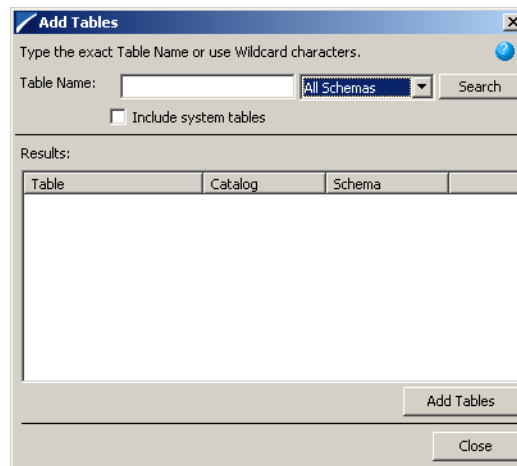
- 10 In the **Database Wizard - Tables** window, click **Add Tables**.

**Figure 11** 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 12** Add Tables



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

**Figure 13** Wildcards

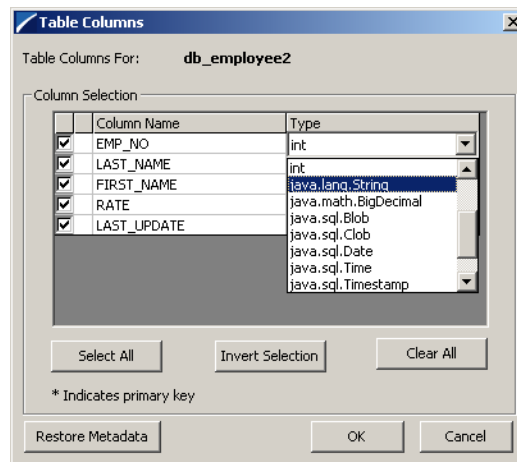
Operator	Description	Example
*	any number of characters	DB*
?	any single character	DB?

- 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**.

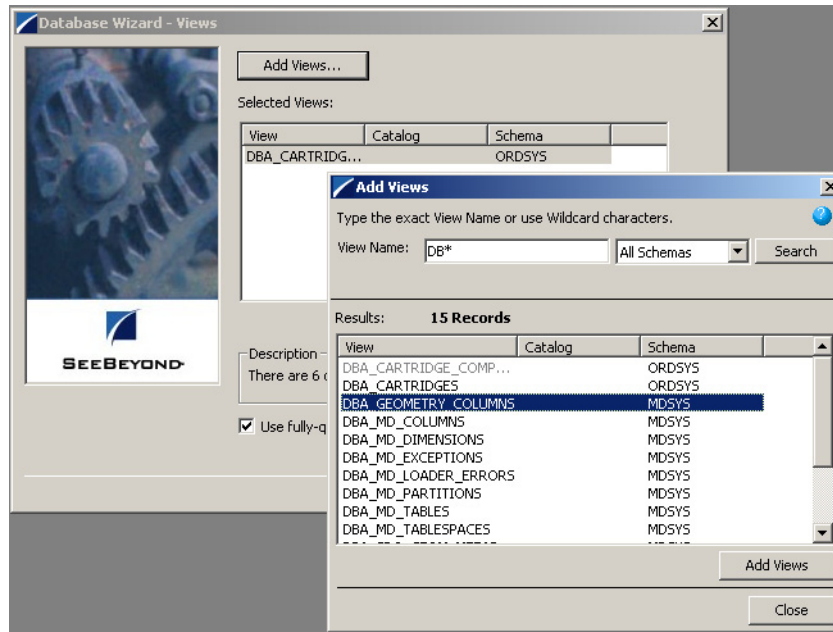
- 14 In the **Database Wizard - Tables** window, review the tables you have selected. If you would like to change any of the tables you have selected, click **Change**.
- 15 In the **Columns Selection** window, you can select or deselect your table choices. You can also change the data type for each 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 14** 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 15 Database Wizard - Views



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

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

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

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

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

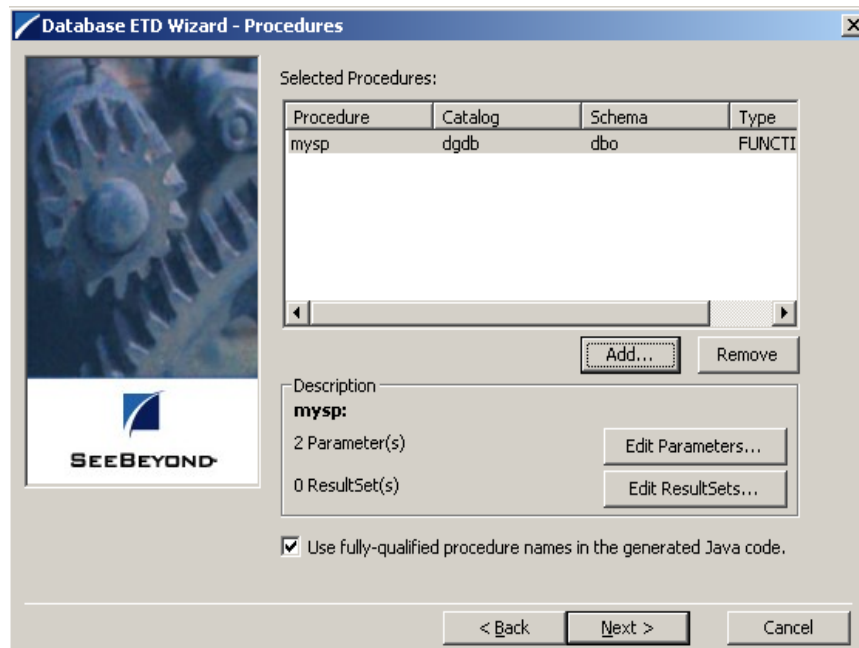
data types. This is not always possible. For example, the column name of 3\*C in this query.

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

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

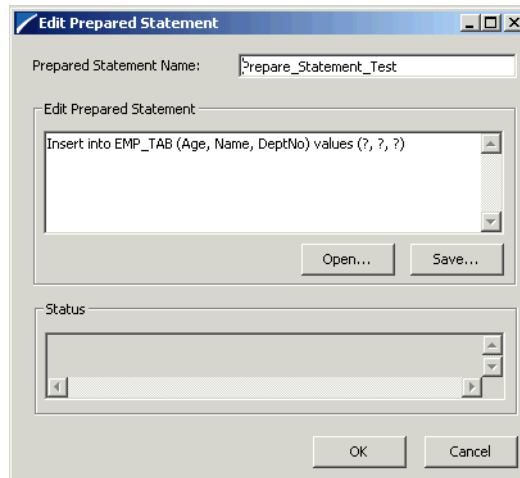
**Note:** *If you modify the ResultSet generated by the 'Execute' mode of the Database Wizard you will need to make sure the indexes match the Stored Procedure. This will assure your ResultSet indexes are preserved.*

**Figure 16** Database Wizard - Procedures



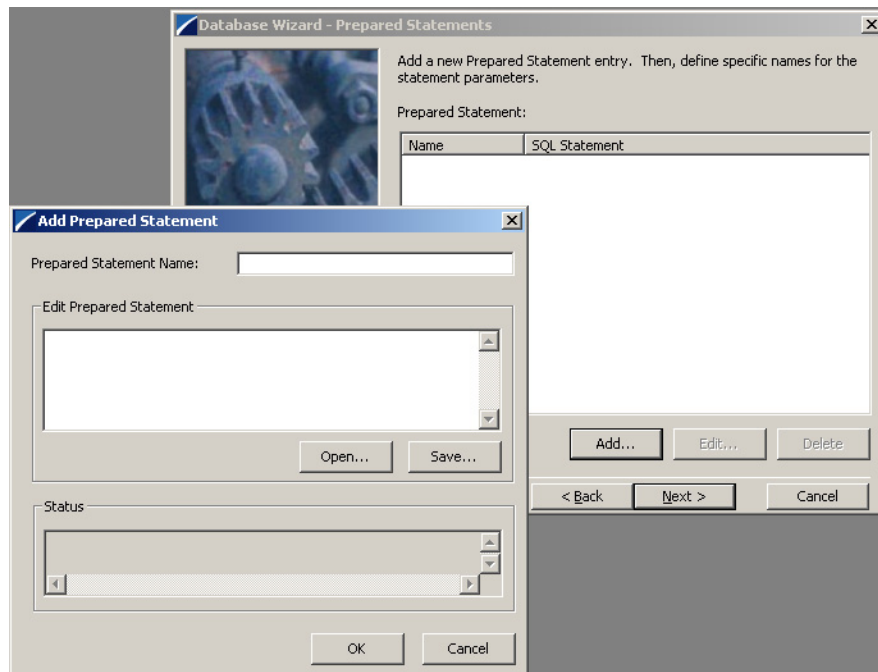
- 19 If you selected **Prepared Statements** on the **Database Wizard - Object Selection** window, you are presented with the **Database Wizard - Prepared Statement** window. To add **Prepared Statements** to your .xsc. complete the following steps:
  - A Click **Add** to add a new prepared statement
  - B Enter a prepared SQL statement.
  - C Enter the **Prepared Statement Name** to be used by the statement.
  - D Use the **Open...** or **Save...** buttons to open pre-existing statements or save the current one. See Figure 17.

**Figure 17** Add Prepared Statement



- 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.

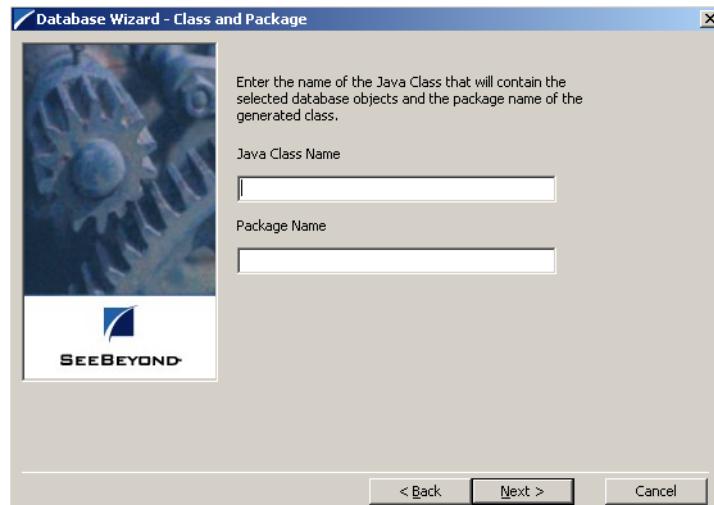
**Figure 18** Database Wizard - Prepared Statements



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

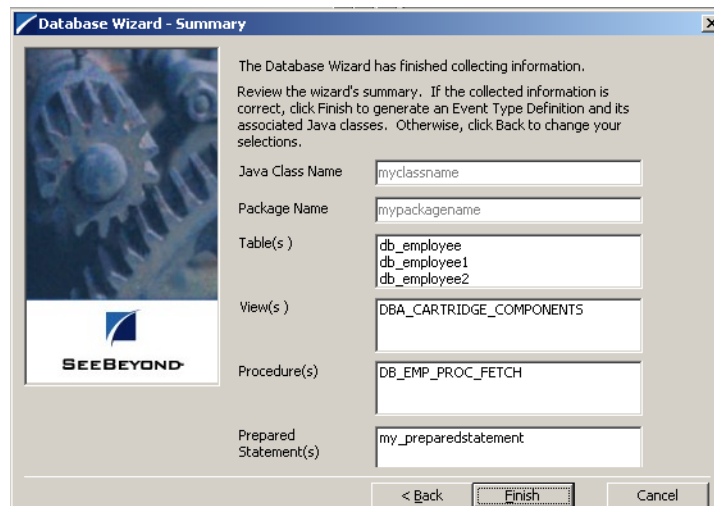


**Figure 19** Database Wizard - Class and Package



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

**Figure 20** 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 Schema Designer, choose **Default Editor....**
- 2 Verify that **Java** is selected, then click **OK**.
- 3 From the **Tools** menu, click **ETD Editor...**
- 4 From the ETD Tool menu click **File** and click **New**.
- 5 From the **New Event Type Definition** window, select **DBWizard** and click **OK**.
- 6 On the Database Wizard - Introduction window, select **Modify an existing XSC file:** and browse to the appropriate .xbs file that you would like to edit.

You are now able to edit your .xsc file.

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

*If you attempt to edit an .xsc whose elements no longer exist in the database, 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

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

### 4.3.1 Tables

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

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

- TYPE\_FORWARD\_ONLY
- TYPE\_SCROLL\_INSENSITIVE

You can also specify ResultSets with a type of Concurrency:

- CONCUR\_READ\_ONLY
- CONCUR\_UPDATABLE

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

- SetConcurrencyToUpdateable
- SetConcurrentlytoRead Only
- SetScrollTypeToForwardOnly

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

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

**Note:** *DataDirect Drivers do not support TYPE\_SCROLL\_SENSITIVE and will implicitly downgrade TYPE\_SCROLL\_SENSITIVE to TYPE\_SCROLL\_INSENSITIVE if TYPE\_SCROLL\_SENSITIVE is used.*

## The query Operation

To perform a query operation on a table:

- 1 Execute the select() method with the “where” clause specified if necessary.
- 2 Loop through the ResultSet using the “next” method.
- 3 For each loop, process the return record.

For example:

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

If you want to check if the last value read was SQL NULL or not, you can use the wasNull() method. It is most useful for native datatypes like “int”. Note that a getxxx method should be called before wasNull() is called.

For example:

```
int empNo = getDBEmp().getDB_EMPLOYEE().getEMP_NO();
if (getDBEMP().getDB_EMLOYEE().wasNULL())
{ //Check to see if empNo is SQL NULL
    //Do something if empNo is SQL NULL
}
else
{ //Do something if empNo is not SQL NULL
```

```
}

```

## The insert Operation

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

- 1 Execute the select() method. You can specify the following types of ResultSets:
  - TYPE\_FORWARD\_ONLY
  - TYPE\_SCROLL\_INSENSITIVE

You must specify ResultSets with:

- CONCUR\_UPDATABLE
- 2 Move to the insert row by the moveToInsertRow method.
  - 3 Set the fields of the table ETD
  - 4 Insert the row by calling insertRow

This example inserts an employee record.

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

### Table ResultSet Behavior

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

Before the schema runs, we have

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

Where:

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

After the schema runs we have:

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

Becomes:

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

Buffer the value of the selected column by :

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

Call moveToInsertRow()

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

Set all the columns the first time

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

Call insertRow()

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

Set all the columns except the unchanged column.

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

Call insertRow()

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

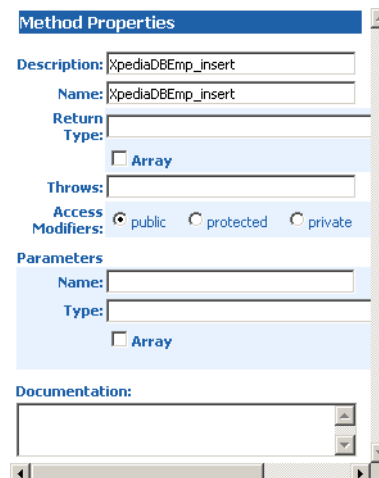
In the above example, column C3 will always have the same value (buf3).

**Figure 21** 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 : public boolean 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 22** Insert Method Properties



## The update Operation

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

- 1 Execute the select() method. You can specify the following types of ResultSets:
  - TYPE\_FORWARD\_ONLY
  - TYPE\_SCROLL\_INSENSITIVE

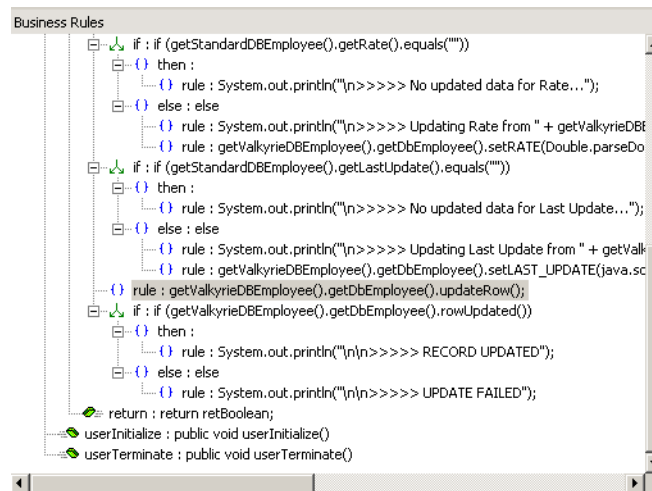
You must specify ResultSets with:

- CONCUR\_UPDATABLE
- 2 Move to the row that you want to update.
  - 3 Set the fields of the table ETD
  - 4 Update the row by calling **updateRow**.

In this example, we move to the third record and update the EMP\_NO and RATE fields.

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

**Figure 23** Update() Method Business Rule



## The delete Operation

To perform a delete operation on a table do the following:

- 1 Execute the select() method. You can specify the following types of ResultSets:
  - TYPE\_FORWARD\_ONLY
  - TYPE\_SCROLL\_INSENSITIVE

You must specify ResultSets with:

- CONCUR\_UPDATABLE
- 2 Move to the row that you want to delete.
  - 3 Set the fields of the table ETD
  - 4 Delete the row by calling **deleteRow**.

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

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

## 4.3.2 The View

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

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

## 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. Please note that the Sybase e\*Way does not support user defined data types.

Prior to using Stored Procedures, it is recommended that you run **`sp_procxmode <procedure name>, anymode`** from the ISQL command line. Not running this will produce unexpected behavior in your Collaboration. If you are not the owner of the server, you may need to use **`sp_procxmode dbo.<procedure name>, anymode`**. Stored procedures created with "chained off" mode will be able to run using DataDirect drivers.

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

Please note that if you would like to raise an error from a Sybase stored procedure use the `com.stc.eways.jdbcx.StatementAgent.getWarnings()` method.

## Executing Stored Procedures

Assuming that you have the following procedure:

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

The ETD represents the Stored Procedure "LookUpGlobal" with two parameters, an inbound parameter (INLOCALID) and an outbound parameter (OUTGLOBALPRODUCTID) can be generated by the DB Wizard as shown in Figure 24. Representing these as nodes in an ETD allows you to drag values from other ETD's to the input parameters, execute the call, and collect the output parameter data by dragging from it's node to elsewhere.

Below are the steps for executing the Stored Procedure:

- 1 Specify the input values.
- 2 Execute the Stored Procedure.

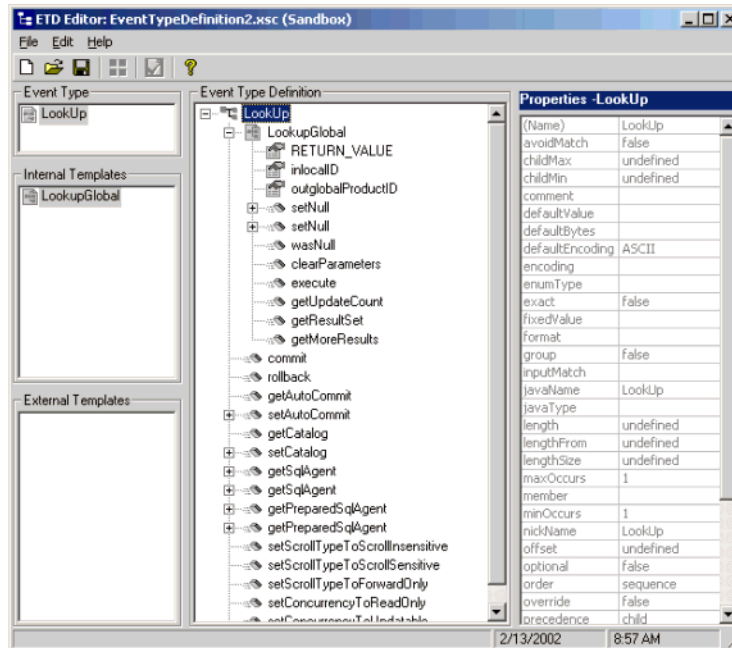


### 3 Retrieve the output parameters if any.

For example:

```
getLookup().getLookupGlobal().setIntLocalID("123");
getLookup().getLookupGlobal().execute();
String s =
getLookup().getLookupGlobal().getOutGlobalProductID;
```

**Figure 24** Stored Procedure LookUpGlobal



## Manipulating the ResultSet and Update Count Returned by Stored Procedure

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

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

The **resultsAvailable()** method, added to the PreparedStatementAgent class, simplifies the whole process of determining whether any results, be it update Counts or ResultSets, are available after a Stored Procedure has been executed. Although JDBC provides three methods (**getMoreResults()**, **getUpdateCount()**, and **getResultSet()**) to

access the results of a Stored Procedure call, the information returned from these methods can be quite confusing to the inexperienced Java JDBC programmer and they also differ between vendors. You can simply call **resultsAvailable()** and if Boolean true is returned, you can expect either a valid Update Count when **getUpdateCount()** is called and/or the next ResultSet has been retrieved and made available to one of the ResultSet nodes defined for the Stored Procedure ETD, when that node's **available()** method returns true.

Often times, Update Counts information returned from Stored Procedures is not really useful to programmers and they will likely want to only process returned Result Set information and avoid looping through all of the Update Counts. The following three methods provide you with the ability to control exactly what information should be returned from a Stored Procedure call. The **enableResultSetsOnly()** method, added to the PreparedStatement Agent class allows only ResultSets to be returned and thus every **resultsAvailable()** called will only return Boolean true if a ResultSet is available. Likewise, the **enableUpdateCountsOnly()** will cause **resultsAvailable()** to return true only if an Update Count is available. The default case of **enableResultsetsAndUpdateCount()** method allows both ResultSets and Update Counts to be returned.

### Collaboration Usability for a Stored Procedure ResultSet

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

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

    // each result set node has an available() method (similar to ETD's)
    // that tells the user
    // whether this particular result set is available. note, JDBC does
    // support access to
    // not more than one result set at a time, i.e., cannot drag from 2
    // distinct result sets
    // simultaneously
    if (getSPIIn().getSpS_multi().getNormRS().available())
    {
        while (getSPIIn().getSpS_multi().getNormRS().next())
        {
            System.err.println("Customer Id =
"+getSPIIn().getSpS_multi().getNormRS().getCustomerId());
            System.err.println("Customer Name =
"+getSPIIn().getSpS_multi().getNormRS().getCustomerName());
            System.err.println();
        }
        System.err.println("===");
    }
}
```

```
else if (getSPIn().getSpS_multi().getDbEmployee().available())
{
    while (getSPIn().getSpS_multi().getDbEmployee().next())
    {
        System.err.println("EMPNO      =
"+getSPIn().getSpS_multi().getDbEmployee().getEMPNO());
        System.err.println("ENAME      =
"+getSPIn().getSpS_multi().getDbEmployee().getENAME());
        System.err.println("JOB        =
"+getSPIn().getSpS_multi().getDbEmployee().getJOB());
        System.err.println("MGR        =
"+getSPIn().getSpS_multi().getDbEmployee().getMGR());
        System.err.println("HIREDATE   =
"+getSPIn().getSpS_multi().getDbEmployee().getHIREDATE());
        System.err.println("SAL        =
"+getSPIn().getSpS_multi().getDbEmployee().getSAL());
        System.err.println("COMM      =
"+getSPIn().getSpS_multi().getDbEmployee().getCOMM());
        System.err.println("DEPTNO    =
"+getSPIn().getSpS_multi().getDbEmployee().getDEPTNO());
        System.err.println();
    }
    System.err.println("===");
}
}
```

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

The definition of **UpdateCount** is JDBC driver dependent. Some drivers send the row count in the previous **ResultSet** back as an update count. While other drivers can only return the number of rows that actually were updated in the database as the **UpdateCount**.

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

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

- The method **resultsAvailable()** calls **getMoreResults()** when it is called more than once. You should not call both methods in your java code. Doing so may result in skipped data from one of the **ResultSets** when more than one **ResultSet** is present.
- The methods **available()** and **getResultSet()** can not be used in conjunction with multiple **ResultSets** being open at the same time. Attempting to open more the one **ResultSet** at the same time will close the previous **ResultSet**. The recommended working pattern is:
  - ♦ Open one Result Set, **ResultSet\_1** and work with the data until you have completed your modifications and updates. Open **ResultSet\_2**, (**ResultSet\_1** is now closed) and modify. When you have completed your work in **ResultSet\_2**, open any additional **ResultSets** or close **ResultSet\_2**.

- If you modify the `ResultSet` generated by the Execute mode of the Database Wizard, you will need to assure the indexes match the Stored Procedure. By doing this, your `ResultSet` indexes will be preserved.

### 4.3.4 Prepared Statement

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

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

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

To execute a Prepared Statement, set the input parameters and call `executeUpdate()` and specify the input values if any.

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

### 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. Batch operations apply only to Prepared Statements.

```
getPrepStatement().getPreparedStatementTest().setAge(23);  
getPrepStatement().getPreparedStatementTest().setName("Peter Pan");  
getPrepStatement().getPreparedStatementTest().setDeptNo(6);  
getPrepStatement().getPreparedStatementTest().addBatch();  
  
getPrepStatement().getPreparedStatementTest().setAge(45);  
getPrepStatement().getPreparedStatementTest().setName("Harrison  
Ford");  
getPrepStatement().getPreparedStatementTest().setDeptNo(7);  
getPrepStatement().getPreparedStatementTest().addBatch();  
getPrepStatement().getPreparedStatementTest().executeBatch();
```

### 4.3.6 Database Configuration Node

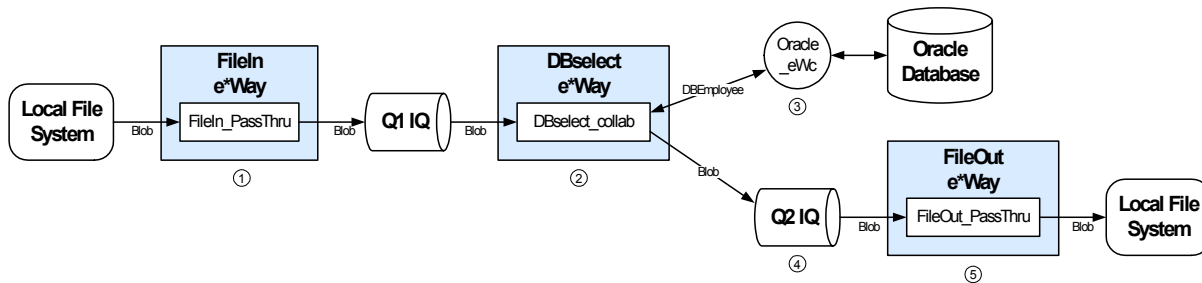
The Database Configuration node allows you to manage the "transaction mode" through the Collaboration if you have set the mode to manual in the e\*Way connection configuration. See "[Connector Settings](#)" on page 19.

## 4.4 Sample Scenario—Polling from a Sybase Database

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

Figure 25 shows a graphical overview of the sample schema.

**Figure 25** The Database Select Scenario—Overview



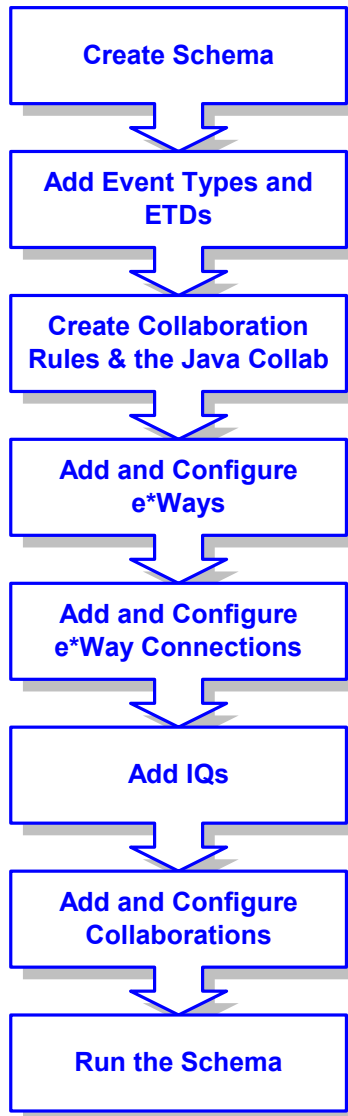
- 1 The **FileIn** e\*Way retrieves an Event (a text file) containing the database select criteria and publishes it to the **Q1 IQ**.
- 2 The **DBselect** e\*Way retrieves the Generic Event (**Blob**) from the IQ. This triggers the rest of the Collaboration which has two parts.
- 3 The information in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

### Overview of Steps

The sample implementation follows these general steps:

- [“Create the Schema” on page 47](#)
- [“Add the Event Types and Event Type Definitions” on page 47](#)
- [“Create the Collaboration Rules and the Java Collaboration” on page 50](#)
- [“Add and Configure the e\\*Ways” on page 54](#)
- [“Add and Configure the e\\*Way Connections” on page 56](#)
- [“Add the IQs” on page 57](#)
- [“Add and Configure the Collaborations” on page 57](#)
- [“Run the Schema” on page 59](#)

**Figure 26** Schema Configuration Steps



**External Database Tables**

The sample uses a simple external Sybase 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 the sample implementation is creating a new schema. After installing the Sybase e\*Way Intelligent Adapter, do the following:

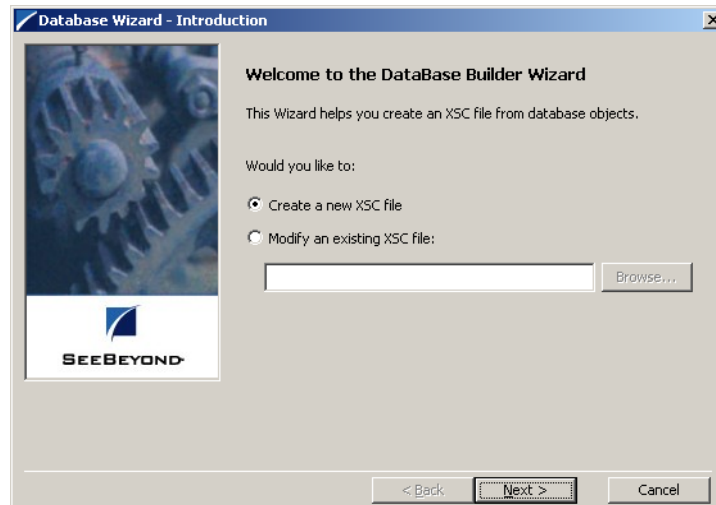
- 1 Launch the e\*Gate Schema Designer GUI.
- 2 Log into the appropriate Registry Host.
- 3 From the list of schemas, click **New** to create a new schema.
- 4 For this sample implementation, enter the name **DBSelect** and click **Open**.  
The Schema Designer will launch and display the newly created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

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

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

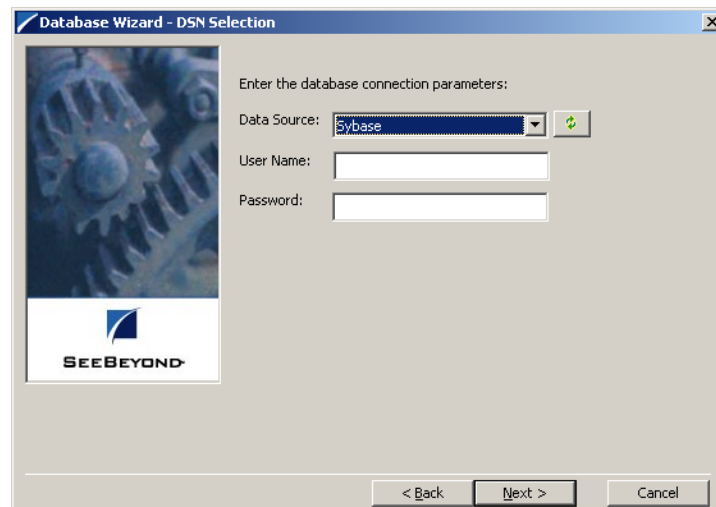
- **DBEmployee** – This Event Type represents the layout of the employee records in the **DB\_Employee** table. The Event Type uses the **DBEmployee.xsc** Event Type Definition. The ETD will be generated by using the Java ETD Editor's Database Wizard (DBWizard).
  - **GenericBlob** – This Event Type 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.
- 1 From the **Options** menu of the Schema Designer, choose **Default Editor...**
  - 2 Verify that **Java** is selected, then click **OK**.
  - 3 In the **Components** pane of the Schema Designer, select the **Event Types** folder.
  - 4 Click the **New Event Type** button to add a new Event Type.
  - 5 Enter the name **DBEmployee** and click **OK**.
  - 6 Double-click the new **DBEmployee** Event Type to display its properties.
  - 7 Click the **New** button to create a new Event Type Definition (ETD). The Java Event Type Definition Editor will appear.
  - 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 27.

**Figure 27** 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.

**Figure 28** 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 Manger, 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 appears.
- 6 From the **File** menu, choose **New**.  
The New Event Type Definition dialog box appears.
- 7 In the New Event Type Definition dialog box, select **Standard ETD** and click **OK**.
- 8 Read the Introduction screen, then click **Next** to continue. The Package Name dialog box appears.
- 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

<b>Node</b>	<b>Property</b>	<b>Value</b>
Root Node	Name	GenericBlob
	Structure	fixed
	Length	0

Node	Property	Value
Child Node	Name	Data
	Structure	fixed
	Length	0

- 13 From the **File** menu, choose **Save**.
- 14 Enter the name **GenericBlob.xsc** and click **OK**.
- 15 From the **File** menu, choose **Compile**.
- 16 From the **File** menu, choose **Promote to Run Time** and click **OK** when finished.
- 17 From the **File** menu, choose **Close** to exit the ETD Editor.
- 18 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 Schema Designer, select the **Collaboration Rules** folder.
- 2 Click the **New Collaboration Rules** button to add a new Collaboration Rule.
- 3 Name the Collaboration Rule **GenericPassThru** and click **OK**.
- 4 Click the **Properties** button to display the Collaboration Rule’s properties.
- 5 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 Schema Designer, select the **Collaboration Rules** folder.
- 2 Click the **New Collaboration Rules** button to add a new Collaboration Rule.
- 3 Name the Collaboration Rule **DBSelect** and click **OK**.

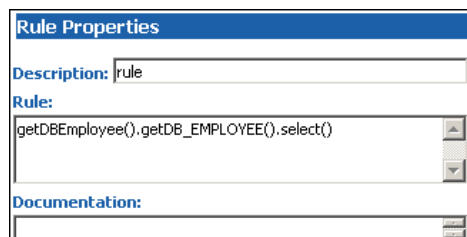
- 4 Click the **Properties** button to display the Collaboration Rule’s properties.
- 5 In the **Service** list, click **Java**.
- 6 Click the **Collaboration Mapping** tab.
- 7 Add three instances as shown in Figure 29 below:

**Figure 29** DBSelect Instances

Instance Name	Source	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 30).
- 11 From the **View** menu, choose **Display Code**.  
 This displays the Java code associated with each of the Collaboration’s rules.
- 12 In the Business Rules pane, select the **retBoolean** rule and click the **rule** button to add a new rule.
- 13 In the **Destination Events** pane, expand the **DBEmployee** Event Type until the **select** method is visible.
- 14 Drag the **select** method into the **Rule** field of the **Rule Properties** pane. Click **OK** to close the dialog box without entering any criteria. (See Figure 30).

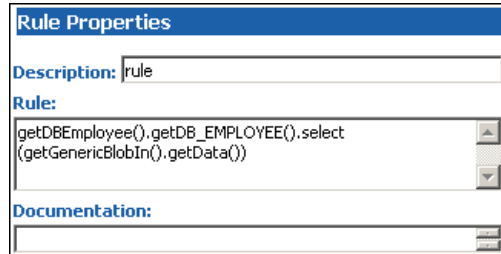
**Figure 30** Rule Properties



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

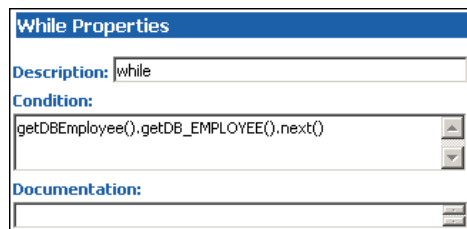
- 16 In the Rule Properties pane, position the cursor inside the parentheses of the select() method. Then drag the Data node from the Source Events pane into the select method's parentheses. (See Figure 31).

**Figure 31** Rule Properties (Continued)



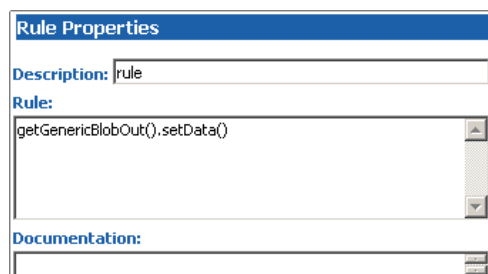
- 17 Select the newly edited rule in the **Business Rules** pane and click the **while** button to add a new while loop beneath the current rule.
- 18 Drag the **next()** method from the Destination Events pane into the Condition field of the While Properties pane. (See Figure 32).

**Figure 32** While Properties



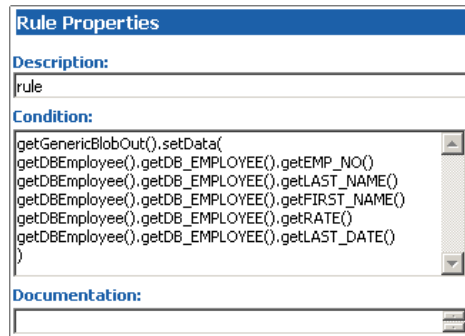
- 19 Select the newly edited while loop in the Business Rules pane and click the **rule** button to add a new rule as a **child** to the while() loop.
- 20 In the **Destination Events** pane, expand the **GenericBlobOut** Event Type until the **Data** node is visible.
- 21 Drag the Data() node into the Rule field of the Rule Properties pane. (See Figure 33).

**Figure 33** Rule Properties



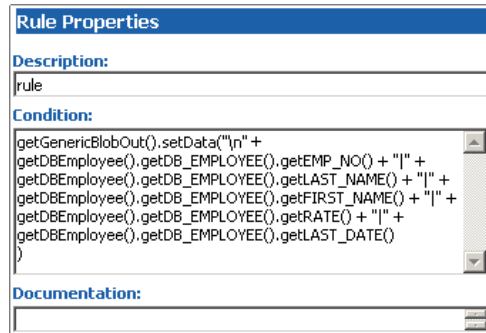
- 22 In the Rule Properties pane, position the cursor inside the parentheses of the setData() method. Then drag each of the five data nodes of DB\_EMPLOYEE from the Source Events into the parentheses of the rule. (See Figure 34).

**Figure 34** Rule Properties (Continued)



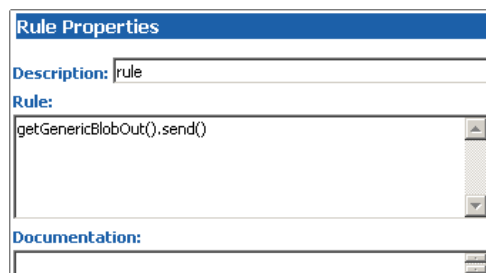
- 23 Edit the text of the condition to add a newline character and pipe (|) delimiters between each of the five data nodes. (See Figure 35).

**Figure 35** Rule Properties (Continued)



- 24 Select the newly edited rule in the **Business Rules** pane and click the **rule** button to add a new rule inside the while loop.
- 25 Drag the root node of the **GenericBlobOut** Event into the rule field in the **Rule Properties** pane.
- 26 Edit the rule; add a **send()** method as shown in Figure 36.

**Figure 36** GenericBlobOut send()



- 27 From the **File** menu, choose **Save** to save the file.
- 28 From the **File** menu, choose **Compile** to compile the Collaboration View the **Output** pane to ensure that there were no compiler errors.

- 29 From the **File** menu, choose **Close** to close the Java Collaboration Editor and return to the Collaboration Rule.

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

- 30 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 Schema Designer, select the Control Broker and click the **New e\*Way** button.
- 2 Enter **FileIn** for the component name and click **OK**.
- 3 Select the newly created e\*Way and click the **Properties** button to display the e\*Way’s properties.
- 4 Use the **Find** button to select **stcewfile.exe** as the executable file.
- 5 Click **New** to create a new configuration file.
- 6 Enter the parameters for the e\*Way as shown in Table 4.

**Table 4** FileIn e\*Way Parameters

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

- 7 Select **Save** from the **File** menu. Enter **FileIn** as the file name and click **Save**.
- 8 Select **Promote to Run Time** from the **File** menu. Click **OK** to close the e\*Way configuration file editor.
- 9 In the **Start Up** tab of the e\*Way properties, select the **Start automatically** check box.

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

To create the DBSelect e\*Way:

- 1 In the Components pane of the Schema Designer, select the Control Broker and click the **New e\*Way** button.
- 2 Enter **DBSelect** for the component name and click **OK**.
- 3 Select the newly created e\*Way and click the **Properties** button to display the e\*Way’s properties.
- 4 Use the **Find** button to select **stceway.exe** as the executable file.
- 5 Click **New** to create a new configuration file.
- 6 Enter the parameters for the e\*Way as shown in Table 5.

**Table 5** DBSelect e\*Way Parameters

Section Name	Parameter	Value
JVM Settings	All	default

- 7 Select **Save** from the **File** menu. Enter **DBSelect** as the file name and click **Save**.
- 8 Select **Promote to Run Time** from the **File** menu. Click **OK** to close the configuration file editor.
- 9 In the **Start Up** tab of the Business Object Broker properties, select the **Start automatically** check box.
- 10 Click **OK** to save the e\*Way’s properties.

To create the FileOut e\*Way:

- 1 In the Components pane of the Schema Designer, select the Control Broker and click the **New e\*Way** button.
- 2 Enter **FileOut** for the component name and click **OK**.
- 3 Select the newly created e\*Way and click the **Properties** button to display the e\*Way’s properties.
- 4 Use the **Find** button to select **stcewfile.exe** as the executable file.
- 5 Click **New** to create a new configuration file.
- 6 Enter the parameters for the e\*Way as shown in Table 6.

**Table 6** FileOut e\*Way Parameters

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

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 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.

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

The sample scenario uses one e\*Way Connection:

- **Sybase\_eWc** – This e\*Way Connection connects the **DBSelect** component to the external database and returns the requested records to be published to the Q2 IQ.

To create the e\*Way Connection:

- 1 In the Components pane of the Schema Designer, select the e\*Way Connections folder.
- 2 Click the **New e\*Way Connection** button to add a new e\*Way Connection.
- 3 Enter **Sybase\_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 **Sybase** from the e\*Way Connection Type dropdown 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** Sybase\_eWc e\*Way Connection Parameters

Section Name	Parameter	Value
DataSource	class	default
	DriverType	default
	ServerName	Use the ODBC Data Source Name.
	PortNumber	Use the port number for your data source.
	DatabaseName	Use the database name listed in your ODBC entry.
	user name	Use local settings.
	password	Use local settings.
	timeout	default



Section Name	Parameter	Value
connector	type	default
	class	default
	transaction mode	default
	connection establishment mode	default
	connection inactivity timeout	0
	connection verification interval	5 minutes

- 8 Select **Save** from the **File** menu. Enter **Sybase\_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 Schema Designer, select the IQ Manager.
- 2 Click the **New IQ** button to add a new IQ.
- 3 Enter the name **Q1** and click **Apply** to save the IQ and leave the New IQ dialog box open.
- 4 Enter the name **Q2** and click **OK** to save the second IQ.
- 5 Select the IQ Manager and click the **Properties** button.
- 6 Select the **Start automatically** check box and click **OK** to save the properties.

#### 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 Schema Designer, select the **FileIn e\*Way**.
- 2 Click the **New Collaboration** button to create a new Collaboration.
- 3 Enter the name **FileIn\_PassThru** and click **OK**.
- 4 Select the newly created Collaboration and click the **Properties** button.
- 5 Select **GenericPassThru** from the 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 Schema Designer, select the **DBSelect e\*Way**.
- 2 Click the **New Collaboration** button to create a new Collaboration.
- 3 Enter the name **DBSelect\_collab** and click **OK**.
- 4 Select the newly created Collaboration and click the **Properties** button.
- 5 Select **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 **Sybase\_CP** 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 Schema Designer, select the **FileOut e\*Way**.
- 2 Click the **New Collaboration** button to create a new Collaboration.
- 3 Enter the name **FileOut\_PassThru** and click **OK**.
- 4 Select the newly created Collaboration and click the **Properties** button.
- 5 Select **GenericPassThru** from the 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.

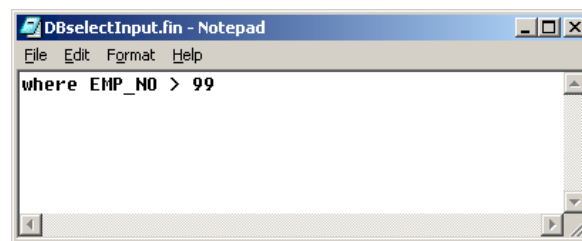
#### 4.4.8 Run the Schema

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 37.

**Figure 37** 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,

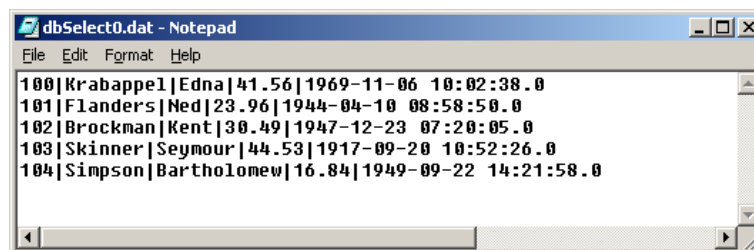
*registry* is the name of the Registry Host, and

*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 38 shows an example of the records that were returned by the sample schema.

**Figure 38** Sample Output File



## Sybase e\*Way Methods

The Sybase 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 60
- [com.stc.eways.jdbcx.PreparedStatementAgent Class](#) on page 70
- [com.stc.eways.jdbcx.PreparedStatementResultSet Class](#) on page 82
- [com.stc.eways.jdbcx.SqlStatementAgent Class](#) on page 108
- [com.stc.eways.jdbcx.CallableStatementAgent Class](#) on page 110
- [com.stc.eways.jdbcx.TableResultSet Class](#) on page 122

---

### 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 68
- [clearWarnings](#) on page 69
- [getFetchDirection](#) on page 65
- [getMaxFieldSize](#) on page 67
- [getMoreResults](#) on page 68
- [getResultSetConcurrency](#) on page 64
- [getUpdateCount](#) on page 67
- [isClosed](#) on page 62
- [queryName](#) on page 62
- [resultSetConcurToString](#) on page 62
- [resultSetTypeToString](#) on page 61
- [sessionOpen](#) on page 63
- [setEscapeProcessing](#) on page 64
- [setMaxFieldSize](#) on page 67
- [setQueryTimeout](#) on page 65
- [stmtInvoke](#) on page 69
- [clearBatch](#) on page 68
- [executeBatch](#) on page 68
- [getFetchSize](#) on page 66
- [getMaxRows](#) on page 66
- [getResultSet](#) on page 67
- [getResultSetType](#) on page 63
- [getWarnings](#) on page 69
- [queryDescription](#) on page 62
- [resetRequested](#) on page 63
- [resultSetDirToString](#) on page 61
- [sessionClosed](#) on page 63
- [setCursorName](#) on page 64
- [setFetchDirection](#) on page 65
- [setMaxRows](#) on page 66
- [setQueryTimeout](#) on page 65

---

### 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 81

[execute](#) on page 81

[executeUpdate](#) on page 81

[setArray](#) on page 79

[setBigDecimal](#) on page 75

[setBlob](#) on page 80

[setByte](#) on page 73

[setCharacterStream](#) on page 79

[setDate](#) on page 75

[setDouble](#) on page 75

[setInt](#) on page 74

[setNull](#) on page 71

[setObject](#) on page 72

[setRef](#) on page 80

[setString](#) on page 77

[setTime](#) on page 76

[setTimestamp](#) on page 77

[clearParameters](#) on page 81

[executeQuery](#) on page 81

[sessionOpen](#) on page 112

[setAsciiStream](#) on page 78

[setBinaryStream](#) on page 78

[setBoolean](#) on page 73

[setBytes](#) on page 78

[setClob](#) on page 80

[setDate](#) on page 76

[setFloat](#) on page 74

[setLong](#) on page 74

[setObject](#) on page 71

[setObject](#) on page 72

[setShort](#) on page 73

[setTime](#) on page 76

[setTimestamp](#) on page 77

---

## sessionOpen

Opens the session event handler.

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

### Overrides

sessionOpen in class StatementAgent

Name	Description
evt	Session event.

---

## setNull

Nullify value of indexed parameter.

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

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

### Throws

java.sql.SQLException

---

## setNull

Nullify value of indexed parameter.

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

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

### Throws

java.sql.SQLException

---

## setObject

Sets value of indexed parameter with an object.

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

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

**Throws**

java.sql.SQLException

---

## setObject

Sets value of indexed parameter with an object.

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

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

**Throws**

java.sql.SQLException

---

## setObject

Sets value of indexed parameter with an object.

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

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



### Throws

java.sql.SQLException

---

## setBoolean

Sets the boolean value of the indexed parameter.

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

Name	Description
index	Parameter index starting from 1.
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 86
- [beforeFirst](#) on page 88
- [close](#) on page 86
- [findColumn](#) on page 89
- [getArray](#) on page 103
- [getAsciiStream](#) on page 102
- [getBigDecimal](#) on page 95
- [getBinaryStream](#) on page 102
- [getBlob](#) on page 104
- [getBoolean](#) on page 91
- [getByte](#) on page 92
- [getBytes](#) on page 101
- [getCharacterStream](#) on page 103
- [getClob](#) on page 104
- [getConcurrency](#) on page 84
- [getDate](#) on page 96
- [getDate](#) on page 97
- [getDouble](#) on page 95
- [getFetchDirection](#) on page 84
- [getFloat](#) on page 94
- [getInt](#) on page 93
- [getLong](#) on page 93
- [getMetaData](#) on page 84
- [getObject](#) on page 90
- [getObject](#) on page 91
- [getRef](#) on page 106
- [getShort](#) on page 92
- [getString](#) on page 100
- [getTime](#) on page 97
- [getTime](#) on page 98
- [getTimestamp](#) on page 99
- [getTimestamp](#) on page 99
- [getType](#) on page 89
- [insertRow](#) on page 107
- [isBeforeFirst](#) on page 88
- [afterLast](#) on page 88
- [clearWarnings](#) on page 106
- [deleteRow](#) on page 107
- [first](#) on page 87
- [getArray](#) on page 103
- [getAsciiStream](#) on page 102
- [getBigDecimal](#) on page 96
- [getBinaryStream](#) on page 103
- [getBlob](#) on page 104
- [getBoolean](#) on page 91
- [getByte](#) on page 92
- [getBytes](#) on page 101
- [getCharacterStream](#) on page 103
- [getClob](#) on page 105
- [getCursorName](#) on page 86
- [getDate](#) on page 96
- [getDate](#) on page 97
- [getDouble](#) on page 95
- [getFetchSize](#) on page 85
- [getFloat](#) on page 95
- [getInt](#) on page 93
- [getLong](#) on page 94
- [getObject](#) on page 89
- [getObject](#) on page 90
- [getRef](#) on page 105
- [getRow](#) on page 107
- [getShort](#) on page 92
- [getString](#) on page 100
- [getTime](#) on page 97
- [getTime](#) on page 98
- [getTimestamp](#) on page 99
- [getTimestamp](#) on page 100
- [getWarnings](#) on page 106
- [isAfterLast](#) on page 89
- [isFirst](#) on page 87

[isLast](#) on page 88

[next](#) on page 86

[refreshRow](#) on page 128

[getFetchDirection](#) on page 84

[updateRow](#) on page 107

[last](#) on page 87

[previous](#) on page 86

[relative](#) on page 87

[getFetchSize](#) on page 85

[wasNull](#) on page 106

---

## 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 110

[execute](#) on page 109

[executeQuery](#) on page 109

[executeUpdate](#) on page 109

---

## 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_FORWARD_ONLY.
iConcur	Concurrency: CONCUR_READ_ONLY, CONCUR_UPDATABLE.

## execute

Executes the specified SQL statement.

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

Name	Description
sSql	SQL statement.

### Returns

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

### Throws

java.sql.SQLException

## executeQuery

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

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

Name	Description
sSql	SQL statement.

### Returns

A ResultSetAgent or null

### Throws

java.sql.SQLException

## executeUpdate

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

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

Name	Description
sSql	SQL statement.

#### Returns

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

#### Throws

java.sql.SQLException

---

### addBatch

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

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

Name	Description
sSql	SQL statement.

#### Throws

java.sql.SQLException

---

## 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 120

[getBlob](#) on page 121

[getBytes](#) on page 115

[getClob](#) on page 121

[getDate](#) on page 118

[getFloat](#) on page 116

[getLong](#) on page 116

[getObject](#) on page 114

[getShort](#) on page 115

[getTime](#) on page 118

[getTimestamp](#) on page 119

[registerOutParameter](#) on page 113

[sessionOpen](#) on page 112

[getBigDecimal](#) on page 117

[getBoolean](#) on page 115

[getBytes](#) on page 120

[getDate](#) on page 117

[getDouble](#) on page 117

[getInt](#) on page 116

[getObject](#) on page 114

[getRef](#) on page 121

[getString](#) on page 120

[getTimestamp](#) on page 119

[registerOutParameter](#) on page 112

[registerOutParameter](#) on page 113

[wasNull](#) on page 114

---

## 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,
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 java.sql.Types.

**Throws**

java.sql.SQLException

---

## registerOutParameter

Registers the indexed OUT parameter with specified type and scale.

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

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

**Throws**

java.sql.SQLException

---

## registerOutParameter

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

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

Name	Description
index	Parameter index starting from 1.
iType	A JDBC type defined by java.sql.Types.
tName	The fully-qualified name of the parameter being set. It is intended to be used by REF, STRUCT, DISTINCT, or JAVA_OBJECT.

**Throws**

java.sql.SQLException

## wasNull

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

```
public boolean wasNull()
```

### Returns

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

### Throws

java.sql.SQLException

## getObject

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

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

Name	Description
index	Parameter index starting from 1.

### Returns

The Object value

### Throws

java.sql.SQLException

## getObject

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

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

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

### Returns

An Object value

### Throws

java.sql.SQLException

---

## getBoolean

Gets the boolean value of the indexed parameter.

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

Name	Description
index	Parameter index starting from 1.

### Returns

A boolean value

### Throws

java.sql.SQLException

---

## 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 124

[beforeFirst](#) on page 125

[deleteRow](#) on page 128

[first](#) on page 124

[getAsciiStream](#) on page 127

[getBinaryStream](#) on page 127

[getCharacterStream](#) on page 128

[isAfterLast](#) on page 126

[isFirst](#) on page 125

[last](#) on page 125

[moveToInsertRow](#) on page 128

[afterLast](#) on page 126

[cancelRowUpdates](#) on page 129

[findColumn](#) on page 126

[getAsciiStream](#) on page 127

[getBinaryStream](#) on page 127

[getCharacterStream](#) on page 127

[insertRow](#) on page 128

[isBeforeFirst](#) on page 126

[isLast](#) on page 125

[moveToCurrentRow](#) on page 129

[next](#) on page 123

[previous](#) on page 123

[rowDeleted](#) on page 129

[rowUpdated](#) on page 129

[updateRow](#) on page 128

[relative](#) on page 124

[rowInserted](#) on page 129

[select](#) on page 123

[wasNull](#) on page 130

## 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\_FORWARD\_ONLY.

```
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\_FORWARD\_ONLY.

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\_FORWARD\_ONLY.

```
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\_FORWARD\_ONLY.

```
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\_FORWARD\_ONLY.

```
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\_FORWARD\_ONLY.

```
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\_FORWARD\_ONLY.

```
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\_FORWARD\_ONLY.

```
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\_FORWARD\_ONLY.

```
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\_FORWARD\_ONLY.

```
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\_FORWARD\_ONLY.

```
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

---

## 5.7 \$DB Configuration Node Methods

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

These methods are contained in the following classes:

- [com\\_stc\\_jdbcx\\_sybasecfg.DataSource](#) on page 130
- [com\\_stc\\_jdbcx\\_sybasecfg](#) on page 139

---

## 5.8 com\_stc\_jdbcx\_sybasecfg.DataSource

Java.lang.Object

```
|
+ -- com_stc_jdbcx_sybasecfg.Com_stc_jdbcx_sybasecfg.DataSource
```

### Direct Known Subclasses

```
public class Com_stc_jdbcx_sybasecfg.DataSource
extends java.lang.Object
```

### Methods of the sybase.DataSource

<a href="#">getClass</a> on page 131	<a href="#">getConnectionMethod</a> on page 132
<a href="#">getDatabaseName</a> on page 134	<a href="#">getPassword</a> on page 136
<a href="#">getPortNumber</a> on page 133	<a href="#">getSelectMethod</a> on page 137
<a href="#">getServerName</a> on page 133	<a href="#">getTimeout</a> on page 138
<a href="#">getUserName</a> on page 135	<a href="#">hasClass</a> on page 131
<a href="#">hasConnectionMethod</a> on page 132	<a href="#">hasDatabaseName</a> on page 134
<a href="#">hasPassword</a> on page 136	<a href="#">hasPortNumber</a> on page 134
<a href="#">hasSelectMethod</a> on page 138	<a href="#">hasServerName</a> on page 133
<a href="#">hasTimeout</a> on page 139	<a href="#">hasUserName</a> on page 135

<a href="#">omitClass</a> on page 131	<a href="#">omitConnectionMethod</a> on page 132
<a href="#">omitDatabaseName</a> on page 135	<a href="#">omitPassword</a> on page 136
<a href="#">omitPortNumber</a> on page 134	<a href="#">omitSelectMethod</a> on page 138
<a href="#">omitServerName</a> on page 133	<a href="#">omitTimeout</a> on page 139
<a href="#">omitUserName</a> on page 135	<a href="#">setClass</a> on page 131
<a href="#">setConnectionMethod</a> on page 132	<a href="#">setDatabaseName</a> on page 134
<a href="#">setPassword</a> on page 136	<a href="#">setPassword_AsIs</a> on page 136
<a href="#">setPortNumber</a> on page 134	<a href="#">setServerName</a> on page 133
<a href="#">setTimeout</a> on page 138	<a href="#">setUserName</a> on page 135

---

## getClass

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

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

### Returns

`java.lang.String`

---

## setClass

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

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

### Returns

None.

---

## hasClass

Returns true if the java class name has been set.

```
public boolean hasClass_()
```

### Returns

True.

---

## omitClass

Sets the java class name to null.

```
public void omitClass_()
```

## Returns

None.

---

## getConnectionMethod

Retrieves the connection method.

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

## Returns

java.lang.String

---

## setConnectionMethod

Specifies which method is used to connect to the database server.

Pooled Data Source - a `ConnectionPoolDataSource` object for creating `PooledConnection` objects. A `PooledConnection` object represents a physical connection and is cached in memory for reuse which saves the overhead of establishing a new connection. This is implemented by the driver.

XA Data Source - an `XADataSource` object for creating `XAConnection` objects, connections that can be used for distributed transactions.

One should make sure that the class specified in "class" parameter supports the connection method that is used.

The default is "Pooled Data Source".

If XA Data Source is selected, make sure to set `SelectMethod` to cursor mode.

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

## Returns

None.

---

## hasConnectionMethod

Returns true if the connection method has been set.

```
public boolean hasConnectionMethod()
```

## Returns

True.

---

## omitConnectionMethod

Sets the connection method to null.

```
public void omitConnectionMethod()
```

### Returns

None.

---

## getServerName

Retrieves the database server host name.

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

### Returns

java.lang.String.

---

## setServerName

Sets the database server host name.

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

### Returns

None.

---

## hasServerName

Returns true if the server name has been set.

```
public boolean hasServerName()
```

### Returns

True.

---

## omitServerName

Sets the server name to null.

```
public void omitServerName()
```

### Returns

None.

---

## getPortNumber

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

```
public long getPortNumber()
```

### Returns

None.

---

---

## setPortNumber

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

```
public void setPortNumber(long val)
```

### Returns

None.

---

## hasPortNumber

Returns true if the port number has been set.

```
public boolean hasPortNumber()
```

### Returns

None.

---

## omitPortNumber

Sets the port number to null.

```
public void omitPortNumber()
```

### Returns

None.

---

## getDatabaseName

Retrieves the name of the database instance.

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

### Returns

java.lang.String.

---

## setDatabaseName

Sets the name of the database instance.

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

### Returns

None.

---

## hasDatabaseName

Returns true if the database name has been set.

```
public boolean hasDatabaseName()
```

## Returns

True.

---

## omitDatabaseName

Sets the database name to null.

```
public void omitDatabaseName()
```

## Returns

None.

---

## getUserName

Retrieves the user name the e\*Way uses to connect to the database.

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

## Returns

java.lang.String

---

## setUserName

Set the user name the e\*Way uses to connect to the database.

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

## Return

None.

---

## hasUserName

Returns true if the user name has been set.

```
public boolean hasUserName()
```

## Returns

True.

---

## omitUserName

Sets the user name to null.

```
public void omitUserName()
```

## Returns

None.

---

---

## getPassword

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

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

### Returns

java.lang.String.

---

## setPassword

Sets the password (will be internally encrypted) the e\*Way uses to connect to the database.

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

### Returns

None.

---

## setPassword\_AsIs

Sets the password (will not be encrypted) the e\*Way uses to connect to the database.

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

### Returns

None.

---

## hasPassword

Returns true if the password has been set.

```
public boolean hasPassword()
```

### Returns

True.

---

## omitPassword

Sets the password to null.

```
public void omitPassword()
```

### Returns

None.

---

## getPassword

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



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

**Returns**

java.lang.String.

---

## setPassword

Sets the password (will be encrypted internally) the e\*Way uses to connect to the database.

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

**Returns**

None.

---

## setPassword\_AsIs

Sets the password (will not be encrypted internally) the e\*Way uses to connect to the database.

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

**Returns**

None.

---

## hasPassword

Returns true if password has been set.

```
public boolean hasPassword()
```

**Returns**

True.

---

## omitPassword

Sets the password to null.

```
public void omitPassword()
```

**Returns**

None.

---

## getSelectMethod

Retrieves the Select Method.

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

## Returns

`java.lang.String`.

---

## setSelectMethod

Determines whether database cursors are used for Select statements.

Performance of the DataDirect driver when performing queries is affected by the choice of direct or cursor for SelectMethod.

For XA operations, this parameter needs to be set to cursor mode.

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

## Returns

None.

---

## hasSelectMethod

Returns true if the Select Method has been set.

```
public boolean hasSelectMethod()
```

## Returns

True.

---

## omitSelectMethod

Sets the Select Method to null.

```
public void omitSelectMethod()
```

## Returns

None.

---

## getTimeout

Retrieves the login timeout in seconds.

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

## Returns

`java.lang.String`.

---

## setTimeout

Sets the login timeout in seconds.

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

### Returns

None.

---

## hasTimeout

Returns true if the login time out has been set.

```
public boolean hasTimeout()
```

### Returns

True.

---

## omitTimeout

Sets the time out to null.

```
public void omitTimeout()
```

### Returns

None.

---

## 5.9 com\_stc\_jdbcx\_sybasecfg

```
com_stc_jdbcx_sybasecfg.Com_stc_jdbcx_sybasecfg
```

### Direct Known Subclasses

```
public class Com_stc_jdbcx_sybasecfg
```

### Methods of the sybasecfg

<a href="#">getSource</a> on page 139
---------------------------------------

<a href="#">setDataSource</a> on page 139
---

---

## getSource

Returns the DataSource object.

```
public Com_stc_jdbcx_sybasecfg.DataSource getSource()
```

### Returns

None.

---

## setDataSource

Sets the DataSource object.

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

## Returns

None.

# Index

## B

Batch Operations 44

## C

class parameter

Connector settings 19

Data Source settings 17

Collaboration Service Java 23–24

component relationship 24

components, Java-enabled 24

configuration file sections

Connector settings 19

DataSource settings 17–19

configuration parameters

class 17, 19

DatabaseName 18

password 19

PortNumber 18

ServerName 18

type 19

user name 18

configuration steps, schema 46

configuring e\*Way connections 16–19

connection establishment mode 20

connection inactivity timeout 20

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 44

DatabaseName parameter 18

DataSource settings 17–19

db.ctl 15

DBWizard 25

driver class, JDBC 17

## E

e\*Way connections

configuring 16–19

creating 16

executeBusinessRules() 24

## H

host system requirements 12

## I

implementation, sample 45–59

## J

Java Collaboration Service 23–24

Java ETD Builder 24–25

Java-enabled components 24

JDBC 25

connection types 19

connector objects 19

driver class 17

## M

Mixing XA-Compliant and XA-Noncompliant

e\*Way Connections 20

## P

password parameter 19

PortNumber parameter 18

Prepared Statement 44

## R

requirements

host system 12

system 11

## S

sample schema 45–59

scenario, sample 45–59

schema configuration steps 46

Server Name parameter 18

stcjdbcx.jar 10

sybase.def 15

system requirements 11

## T

Tables 34

The insert() Method 36

The Stored Procedure 40

timeout 19

transaction mode 20

type parameter 19

## U

UNIX 14

user name parameter 18

userInitialize() 24

userTerminate() 24