

e*Way Intelligent Adapter for DB2 Universal Database User's Guide

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



Copyright © 2005, 2010, Oracle and/or its affiliates. All rights reserved.

This software and related documentation are provided under a license agreement containing restrictions on use and disclosure and are protected by intellectual property laws. Except as expressly permitted in your license agreement or allowed by law, you may not use, copy, reproduce, translate, broadcast, modify, license, transmit, distribute, exhibit, perform, publish, or display any part, in any form, or by any means. Reverse engineering, disassembly, or decompilation of this software, unless required by law for interoperability, is prohibited.

The information contained herein is subject to change without notice and is not warranted to be error-free. If you find any errors, please report them to us in writing.

If this is software or related software documentation that is delivered to the U.S. Government or anyone licensing it on behalf of the U.S. Government, the following notice is applicable:

U.S. GOVERNMENT RIGHTS Programs, software, databases, and related documentation and technical data delivered to U.S. Government customers are "commercial computer software" or "commercial technical data" pursuant to the applicable Federal Acquisition Regulation and agency-specific supplemental regulations. As such, the use, duplication, disclosure, modification, and adaptation shall be subject to the restrictions and license terms set forth in the applicable Government contract, and, to the extent applicable by the terms of the Government contract, the additional rights set forth in FAR 52.227-19, Commercial Computer Software License (December 2007). Oracle USA, Inc., 500 Oracle Parkway, Redwood City, CA 94065.

This software or hardware is developed for general use in a variety of information management applications. It is not developed or intended for use in any inherently dangerous applications, including applications which may create a risk of personal injury. If you use this software or hardware in dangerous applications, then you shall be responsible to take all appropriate fail-safe, backup, redundancy, and other measures to ensure the safe use. Oracle Corporation and its affiliates disclaim any liability for any damages caused by use of this software or hardware in dangerous applications.

Oracle and Java are registered trademarks of Oracle and/or its affiliates. Other names may be trademarks of their respective owners.

AMD, Opteron, the AMD logo, and the AMD Opteron logo are trademarks or registered trademarks of Advanced Micro Devices. Intel and Intel Xeon are trademarks or registered trademarks of Intel Corporation. All SPARC trademarks are used under license and are trademarks or registered trademarks of SPARC International, Inc. UNIX is a registered trademark licensed through X/Open Company, Ltd.

This software or hardware and documentation may provide access to or information on content, products, and services from third parties. Oracle Corporation and its affiliates are not responsible for and expressly disclaim all warranties of any kind with respect to third-party content, products, and services. Oracle Corporation and its affiliates will not be responsible for any loss, costs, or damages incurred due to your access to or use of third-party content, products, or services.

Version 20100719180607.

Contents

Chapter 1

Introduction	10
Overview	10
Components	10
Operational Overview	10
System Requirements	11
Host System Requirements	11
Using DataDirect Drivers on Windows or Unix Operating Systems	11
Using IBM Drivers on Windows or Unix Operating Systems	11
External System Requirements	12
At-A-Glance Operating System Driver Configuration	13

Chapter 2

Installation	14
Installing the DB2 Universal Database e*Way on Windows	14
Pre-installation	14
Installation Procedure	14
Installing the DB2 Universal Database e*Way on UNIX	15
Pre-installation	15
Installation Procedure	15
Files/Directories Created by the Installation	16
The db2cli.ini File for Use With the IBM DB2 Native Drivers	16
The db2java.zip File	16
Usejdbc2.bat File for Use With the DB2 Native JDBC Drivers	17

Chapter 3

Connectivity	18
Registering the DataSource	18
ODBC Test	20
JDBC Test	22

Chapter 4

e*Way Connection Configuration	27
Create e*Way Connections	27
DataSource Settings	28
class	28
Connection Method	29
jdbc url	29
CollectionID	29
DatabaseName	30
LocationName	30
PackageName	30
user name	30
password	30
PortNumber	31
ServerName	31
timeout	31
data source attribute value pair separator	31
data source attributes	32
Connector Settings	32
type	32
class	32
transaction mode	32
connection establishment mode	33
connection inactivity timeout	33
connection verification interval	34
Connection Manager	34
Controlling When a Connection is Made	35
Controlling When a Connection is Disconnected	35
For Pre-Existing z/OS Connections	35
DB2 JDBC Applet Server	36
Using Large ResultSets	37

Chapter 5

Implementation	38
Implementing Java-enabled Components	38
The Java Collaboration Service	39
Java-enabled Components	39
The Java ETD Builder	39
The Parts of the ETD	39
Using the DBWizard ETD Builder	40
The Generated ETDs	49
Editing an Existing .XSC Using the Database Wizard	50
Using ETDs with Tables, Views, Stored Procedures, and Prepared Statements	50
The Table	50
The query Operation	51

The insert Operation	52
The update Operation	54
The delete Operation	55
The View	56
The Stored Procedure	56
Executing Stored Procedures	56
Manipulating the ResultSet and Update Count Returned by Stored Procedure	57
Prepared Statement	61
Batch Operations	62
Database Configuration Node	62
Sample Scenario—Polling from a DB2 Database	62
Create the Schema	65
Add the Event Types and Event Type Definitions	65
Create the Collaboration Rules and the Java Collaboration	68
Add and Configure the e*Ways	72
Add and Configure the e*Way Connections	74
Add the IQs	75
Add and Configure the Collaborations	75
Run the Schema	77
Importing the z/OS Sample Schema	78

Chapter 6

DB2 e*Way Methods 79

com.stc.eways.jdbcx.StatementAgent Class	79
resultSetTypeToString	80
resultSetDirToString	80
resultSetConcurToString	81
isClosed	81
queryName	81
queryDescription	81
sessionOpen	82
sessionClosed	82
resetRequested	82
getResultsetType	82
getResultSetConcurrency	83
setEscapeProcessing	83
setCursorName	83
setQueryTimeout	84
setQueryTimeout	84
getFetchDirection	84
setFetchDirection	84
getFetchSize	85
getMaxRows	85
setMaxRows	85
getMaxFieldSize	86
setMaxFieldSize	86
getUpdateCount	86
getResultSet	86
getMoreResults	87
clearBatch	87
executeBatch	87
cancel	87
getWarnings	88
clearWarnings	88
stmtInvoke	88

com.stc.eways.jdbcx.PreparedStatementAgent Class	89
sessionOpen	89
setNull	90
setNull	90
setObject	90
setObject	91
setObject	91
setBoolean	92
setByte	92
setShort	92
setInt	93
setLong	93
setFloat	93
setDouble	94
setBigDecimal	94
setDate	94
setDate	95
setTime	95
setTime	95
setTimestamp	96
setTimestamp	96
setString	96
setBytes	97
setAsciiStream	97
setBinaryStream	97
setCharacterStream	98
setArray	98
setBlob	99
setClob	99
setRef	99
clearParameters	100
addBatch	100
execute	100
executeQuery	100
executeUpdate	100
com.stc.eways.jdbcx.PreparedStatementResultSet Class	101
Constructor PreparedStatementResultSet	103
getMetaData	103
getConcurrency	103
getFetchDirection	103
setFetchDirection	104
getFetchSize	104
setFetchSize	104
getCursorName	105
close	105
next	105
previous	105
absolute	105
relative	106
first	106
isFirst	106
last	106
isLast	107
beforeFirst	107
isBeforeFirst	107
afterLast	107
isAfterLast	108
getType	108
findColumn	108
getObject	108
getObject	109
getObject	109
getObject	110

getBoolean	110
getBoolean	110
getByte	111
getShort	111
getShort	111
getInt	112
getInt	112
getLong	112
getLong	113
getFloat	113
getFloat	114
getDouble	114
getBigDecimal	114
getBigDecimal	115
getDate	115
getDate	115
getDate	116
getTime	116
getTime	116
getTime	117
getTime	117
getTimeStamp	118
getTimeStamp	118
getTimeStamp	118
getTimeStamp	119
getString	119
getString	119
getBytes	120
getBytes	120
getAsciiStream	121
getAsciiStream	121
getBinaryStream	121
getBinaryStream	122
getCharacterStream	122
getArray	122
getBlob	123
getBlob	123
getClob	123
getClob	124
getRef	124
getRef	125
wasNull	125
getWarnings	125
clearWarnings	125
getRow	126
refreshRow	126
insertRow	126
updateRow	126
deleteRow	126
com.stc.eways.jdbcx.SqlStatementAgent Class	127
Constructor SqlStatementAgent	127
Constructor SqlStatementAgent	127
execute	128
executeQuery	128
executeUpdate	129
addBatch	129
com.stc.eways.jdbcx.CallableStatementAgent Class	129
Constructor CallableStatementAgent	130
Constructor CallableStatementAgent	131
Constructor CallableStatement Agent	131
sessionOpen	131
registerOutParameter	132
registerOutParameter	132

registerOutParameter	132
wasNull	133
getObject	133
getObject	133
getBoolean	134
getByte	134
getShort	135
getInt	135
getLong	135
getFloat	136
getDouble	136
getBigDecimal	136
getDate	137
getDate	137
getTime	137
getTime	138
getTimestamp	138
getTimestamp	139
getString	139
getBytes	139
getArray	140
getBlob	140
getClob	140
getRef	141
com.stc.eways.jdbcx.TableResultSet Class	141
select	142
next	142
previous	143
absolute	143
relative	143
first	144
isFirst	144
last	144
isLast	144
beforeFirst	145
isBeforeFirst	145
afterLast	145
isAfterLast	145
findColumn	146
getAsciiStream	146
getAsciiStream	146
getBinaryStream	146
getBinaryStream	146
getCharacterStream	147
getCharacterStream	147
refreshRow	147
insertRow	147
updateRow	147
deleteRow	148
moveToInsertRow	148
moveToCurrentRow	148
cancelRowUpdates	148
rowInserted	148
rowUpdated	148
rowDeleted	149
wasNull	149
\$DB Configuration Node Methods	149
com_stc_jdbcx_db2cfg.DataSource	149
getClass	150
setClass	150
hasClass	151
omitClass	151

Contents

getConnectionMethod	151
setConnectionMethod	151
hasConnectionMethod	152
omitConnectionMethod	152
getCollectionID	152
setCollectionID	152
hasCollectionID	152
omitCollectionID	153
getDatabaseName	153
setDatabaseName	153
hasDatabaseName	153
omitDatabaseName	153
getLocationName	154
setLocationName	154
hasLocationName	154
omitLocationName	154
getPackageName	154
setPackageName	155
hasPackageName	155
omitPackageName	155
getUserName	155
setUserName	155
hasUserName	155
omitUserName	156
getPassword	156
setPassword	156
setPassword_AsIs	156
hasPassword	156
omitPassword	157
getPortNumber	157
setPortNumber	157
hasPortNumber	157
omitPortNumber	157
getServerName	158
setServerName	158
hasServerName	158
omitServerName	158
getTimeout	158
setTimeout	158
hasTimeout	159
omitTimeout	159
com_stc_jdbcx_db2cfg	159
getDataSource	159
setDataSource	159

Appendix A

Appendix	161
ADO/ODBC Calls	161
ADO Calls	161
ODBC Calls	161
Index	162

Introduction

This document describes how to install and configure the Java-enabled e*Way Intelligent Adapter for DB2 Universal Database.

This Chapter Includes:

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

1.1 Overview

The DB2 Universal Database e*Way enables the e*Gate system to exchange data with external DB2 databases. This document describes how to install and configure the Java-enabled version of the DB2 Universal Database e*Way.

The DB2 Universal Database e*Way uses the Java library to issue SQL statements to interact with DB2 databases.

1.1.1 Components

The following components comprise the Java-enabled version of the DB2 Universal Database e*Way:

- **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.

1.2 Operational Overview

The Java-enabled DB2 Universal Database e*Way uses Java Collaborations to interact with one or more external databases. By using the Java Collaboration Service it is possible for e*Gate components—such as e*Way Intelligent Adapters (e*Ways) and Business Object Brokers (BOBs)—to connect to external databases and execute business rules written entirely in Java.

1.3 System Requirements

To use the DB2 Universal Database 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.

1.3.1 Host System Requirements

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

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

Using DataDirect Drivers on Windows or Unix Operating Systems

GUI Host Requirements Using DataDirect Drivers

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 DB2 Universal Database e*Way, the host machine must also meet the [“Participating Host Requirements Using DataDirect Drivers” on page 11](#).

Participating Host Requirements Using DataDirect Drivers

The DB2 Universal Database e*Way requires that the following items be installed on the Participating Host:

- DataDirect JDBC 3.2 drivers. These drivers are installed with e*Gate.

Note: When using DataDirect drivers, the number of ResultSets returned is one.

Using IBM Drivers on Windows or Unix Operating Systems

GUI Host Requirements Using IBM Drivers for DB2

If you have chosen not to use the DataDirect ODBC wire protocol drivers because of pre-existing DB2 4.5.2 collaborations, you will need to assure you have the following components installed and configured appropriately. Please note that if you do select to use the IBM Drivers, you will not be able to take advantage of the new enhanced features of the DB2 4.5.3 e*Way.

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:

- DB2 Universal Database (UDB) version 7.2 (with FixPak 4, Service Level WR21270), 9.5, or 9.7 client library.
- The DB2 UDB client library must be installed on Windows to utilize the build tool.
- Java 2 SDK. The JDK can be installed during the e*Gate GUI installation process if it has not already been installed.
- Microsoft Data Access Components (MDAC) RTM version 2.6 or greater. This component is included in the e*Gate installation routine. If the GUI host machine will also be executing the DB2 UDB e*Way, the host machine must also meet the **“Participating Host Requirements Using DataDirect Drivers” on page 11.**

Participating Host Requirements Using IBM Drivers for DB2

If you have chosen not to use the DataDirect JDBC wire protocol drivers because of pre-existing DB2 4.5.2 collaborations or if you want to access DB2 on z/OS, you will need to assure you have the following components installed and configured appropriately on the Participating Host.

The DB2 Universal Database e*Way Installation program installs the Type 4 DataDirect JDBC drivers but if you are continuing to use the IBM drivers (available from IBM), a Type 4 driver will be appropriate for most DB2 UDB implementations. 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>

- The DB2 Universal Database client library version 7.2 with FixPak4, Service Level WR21270, version 9.5, or version 9.7. The client must be configured to communicate with the DB2 database.
- The most recent IBM DB2 software updates must be installed. These updates provide support for the latest Java functionality.

Note: *If you have chosen to use the IBM drivers for DB2, please be aware that these drivers do not support updatable ResultSets.*

Note: *If you have chosen to use the IBM drivers for DB2, please be aware that these drivers do not support scrollable ResultSets. It is recommended that you use a prepared statement to update and insert data.*

1.4 External System Requirements

The DB2 e*Way supports the following external systems:

- DB2 Database versions 9.5 and 9.7
- DB2 Connect version 8.1.

- DB2 Universal Database (UDB) version 7.2 with FixPak 4, Service Level WR21270. FixPak 4 is available directly from IBM. For more information on FixPak 4, please review IBM's "FixPak for IBM DB2 Universal Database* Version 7.2" Readme.txt.
- DB2 Universal Database (UDB) version 7.1 when connecting to DB2 running on a z/OS operating system.

At-A-Glance Operating System Driver Configuration

The following table should help you in determining your specific driver configuration. For more detail information regarding specific driver versions as well as additional system requirements, please see [System Requirements on page 11](#) of this document.

Table 1 At-a-Glance Operating System Driver Configuration

Operating System	DataDirect ODBC	DataDirect JDBC	IBM's DBConnect ODBC (available from IBM)	IBM's DBConnect JDBC (available from IBM)	IBM's SyncPoint Manager (available from IBM)	DB2 Native Drivers
Windows	X	X				
UNIX	X	X				
Connecting to z/OS			X	X		
Connecting to AS/400			X	X		

Note: Please note that XA running on z/OS or connecting to AS/400 is not supported.

Table 2 At-a-Glance XA Operating System Driver Configuration

Operating System	DataDirect ODBC	DataDirect JDBC	IBM's DBConnect ODBC (available from IBM)	IBM's DBConnect JDBC (available from IBM)	IBM's SyncPoint Manager (available from IBM)
Windows			X	X	
UNIX			X	X	
Connecting to z/OS			X	X	X
Connecting to AS/400	Not Supported	Not Supported	Not Supported	Not Supported	Not Supported

Installation

This chapter describes how to install the DB2 Universal Database e*Way.

This Chapter Includes:

- [“Installing the DB2 Universal Database e*Way on Windows” on page 14](#)
- [“Installing the DB2 Universal Database e*Way on UNIX” on page 15](#)

Important: Open and review the *Readme.txt* for the DB2 e*Way for any additional information or requirements, prior to installation. The *Readme.txt* is located on the Installation CD_ROM at `setup\addons\ewdb2`.

2.1 Installing the DB2 Universal Database e*Way on Windows

2.1.1 Pre-installation

- 1 Quit all Windows-based programs before running the setup program, including any anti-virus applications.
- 2 You must have Administrator privileges to install this e*Way.
- 3 To use the DB2 e*Way, you must also setup your DB2 environment. This is required, because the e*Way connection configuration will look for the database name in the database directory of the DB2 client configuration on the system.

2.1.2 Installation Procedure

To install the DB2 Universal Database 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 Oracle support personnel, do not change the suggested “installation directory” setting.*

2.2 Installing the DB2 Universal Database 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.

To use the DB2 e*Way, you must also setup your DB2 environment. This is required, because the e*Way connection configuration will look for the database name in the database directory of the DB2 client configuration on the system.

2.2.2 Installation Procedure

To install the DB2 Universal Database 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 Oracle support personnel, do not change the suggested “installation directory” setting.*

2.3 Files/Directories Created by the Installation

The DB2 Universal Database 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 3 Files created by the installation

e*Gate Directory	File(s)
	stcewdb2.ctl
etd\db2	com_stc_jdbcx_db2cfg.java
etd\db2	com_stc_jdbcx_db2cfg.xsc
ThirdParty\merant\classes	DGbase.jar
ThirdParty\merant\classes	DGdb2.jar
ThirdParty\merant\classes	DGutil.jar
ThirdParty\merant\classes	spy.jar
server\registry\repository\configs\db2	DB2.def
server\registry\repository\default\etd\	db2.ctl
server\registry\repository\default\etd	dbwizard.ctl

2.4 The db2cli.ini File for Use With the IBM DB2 Native Drivers

To use a Clob/Blob while using the IBM DB2 native drivers, you will need to set the LONGDATACOMPAT = 0 in the file db2cli.ini under the DB2 root directory. This will disable the LongVarChar, LongVarBinary, and LongVarGraphic. Refer to the IBM Installation and Configuration Supplement for more information.

2.5 The db2java.zip File

The DB2 Universal Database e*Way uses the drivers contained in the **db2java.zip** file. To add the drivers, do the following:

Assure that the **db2java.zip** file from your DB2 Client software is installed into / Program Files/SQLLib/Java

From the e*Gate Schema Designer, click **File** on the tool bar. Click **Commit to Sandbox...** from the drop down menu and enter a file location. Browse to the /Program Files/SQLLib/Java folder and select the **db2java.zip** file. Click **Open**

From the e*Gate Collaboration, click **Tools** on the tool bar. Click **Options** from the drop down menu. **Add** the **db2java.zip** file.

2.6 Usejdbc2.bat File for Use With the DB2 Native JDBC Drivers

If you have chosen to use the DB2 native JDBC drivers, you will need to do the following to assure you have JDBC 2.0 compliance. If you have any questions regarding this file, please consult your IBM DB2 Universal Database documentation:

- 1 From a command line, change your working directory to `sqllib\java12` and enter **usejdbc2.bat**.

This command performs the following tasks:

- ♦ Creates a `sqllib\java11` directory for the 1.22 driver files
- ♦ Backs up the JDBC 1.22 driver files into the `sqllib\java11` directory
- ♦ Copies the JDBC 2.0 driver files from the `sqllib\java12` directory into the appropriate directories

To switch back to the JDBC 1.22 driver, execute the `usejdbc1` batch file from the `sqllib\java12` directory.

Connectivity

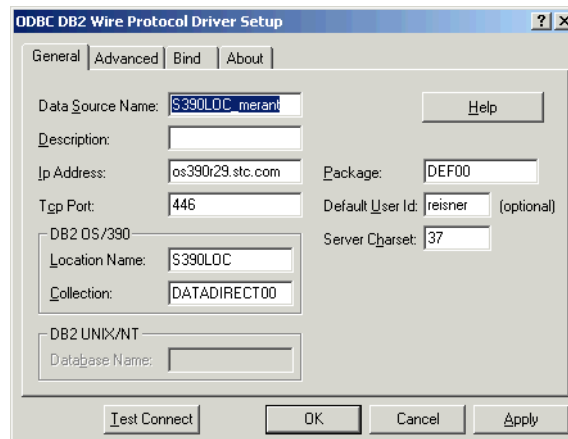
Before you can configure your DB2 e*Way, you will need to test the connectivity. Follow the instructions provided to assure connectivity.

3.1 Registering the DataSource

To register your datasource, do the following:

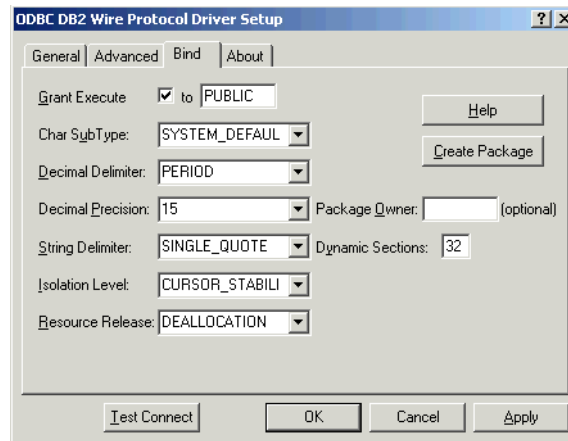
- 1 Install the DataDirect 4.1 ODBC drivers. To do this, follow the installation instructions provided with the drivers.
- 2 On the task bar, click the **Start** button, and then click **Control Panel**.
- 3 Click **Administrative Tools**.
- 4 Click **Data Sources (ODBC)**.
- 5 In the ODBC Data Source Administrator window, click the **User DSN** tab.
- 6 Click **Add**.
- 7 Select **DataDirect 4.1 32BIT DB2 Wire Protocol** and click **Finish**.
- 8 In the ODBC DB2 Wire Protocol and Driver Setup window, select the **General** tab. Enter the appropriate system information for your DB2 installation. See Figure 1.

Figure 1 ODBC DB2 Wire Protocol Driver Setup General Tab



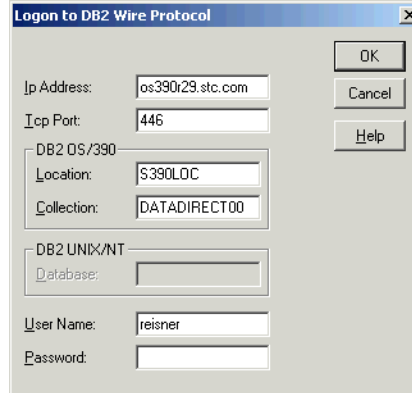
- 9 Click on the **Bind** tab. See Figure 2.

Figure 2 ODBC DB2 Wire Protocol Driver Setup Bind Tab



- 10 Enter all the information required to make a connection to your database. Once completed, test the connection by clicking the **Test Connection** button.
- 11 Verify that the information in the Logon to DB2 Wire Protocol window is correct. Click the **OK** button. See Figure 3.

Figure 3 Logon to DB2 Wire Protocol



- 12 If the connection was successful, you will see **Connection Established!** in the Test Connect window. Click **OK**. See Figure 4.

Figure 4 Test Connect



3.2 ODBC Test

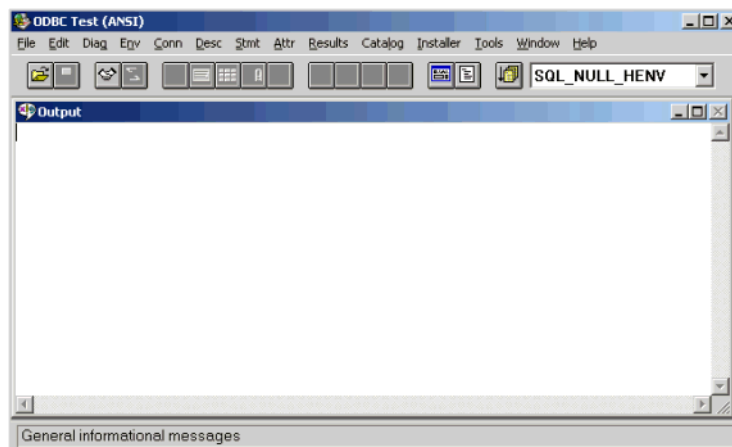
ODBC Test is a utility available from Microsoft. To use this utility, you will need to download it from www.microsoft.com.

- 1 Download ODBC Test and follow the online instructions provided by Microsoft to install this utility.

Once installed do the following:

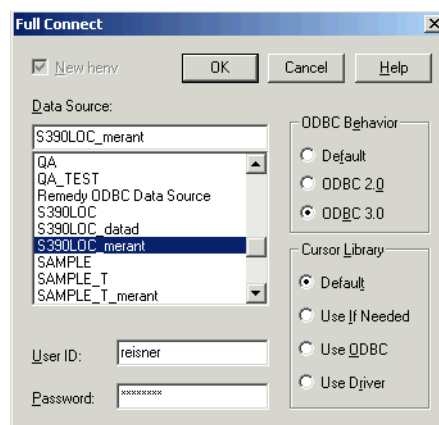
- 2 In the ODBC Test (ANSI) window, click the **Handshake** button on the toolbar to open the connection. See Figure 5.

Figure 5 ODBC Test - Handshake



- 3 In the Data Source dialog box, select the ODBC data source. Enter your **User ID** and **Password**. See Figure 6.

Figure 6 Full Connect



- 4 Click **OK**.
- 5 Once a connection has been established, the ODBC driver must be unlocked for use with ODBC Test by setting two attributes. From the **Attr** menu, select

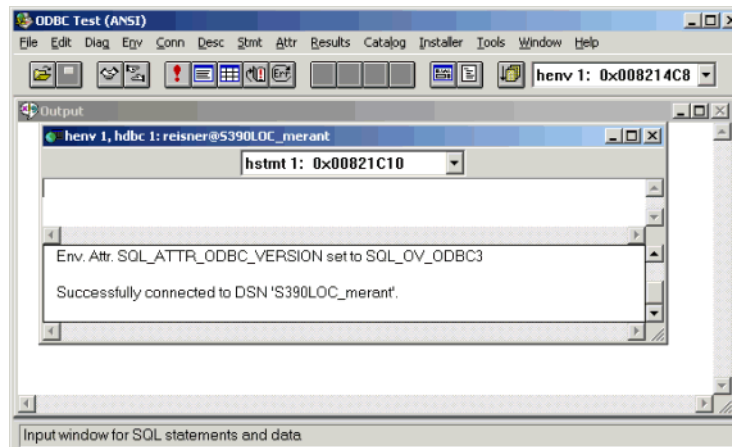
SQLSetConnectOption. Enter the following values. These values are case-sensitive and must be entered exactly as shown:

fOption: 1041, **vParam:** IVdg.LIC

fOption: 1042, **vParam:** STCDART2UI1DATAGATEQQQQQ

- 6 Click **OK**. See **Figure 7**.

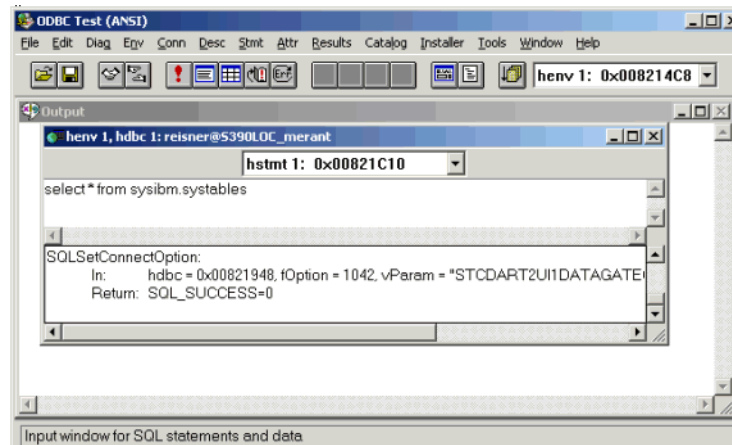
Figure 7 ODBC Test (ANSI) - Connection Established



- 7 Enter the following statement in the **Connection** dialog box. See **Figure 8**.

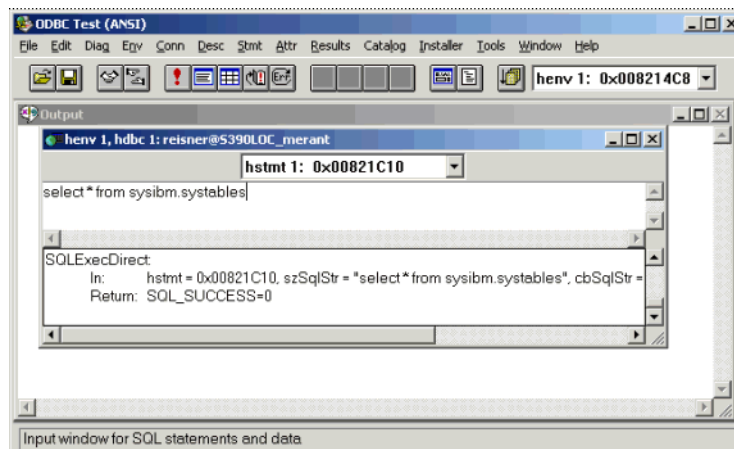
```
select * from sysibm.systables
```

Figure 8 ODBC Test (ANSI) - Select Statement



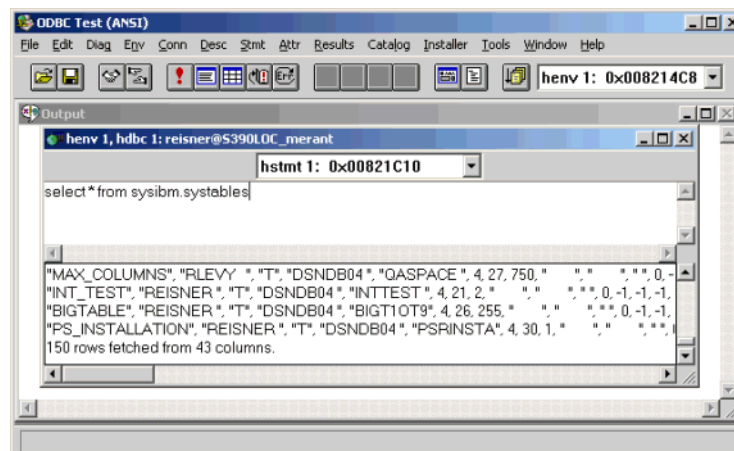
- 8 On the toolbar, click the **Execute** button.
- 9 If the connection was successful, you will see the **SQLExecDirect** return statement in the **Connection** dialog box. See **Figure 9**

Figure 9 ODBC Test (ANSI) - Connection Successful



- 10 To view the data returned, click the **Get Data** button. The data returned by the query will display in the **Connection** dialog box. See Figure 10.

Figure 10 ODBC Test (ANSI) - Get Data



3.3 JDBC Test

JDBC Test is a utility provided by DataDirect for creating the packages used by JDBC and to test connectivity.

- 1 Run JDBC Test from a command line by typing the following:

From Windows

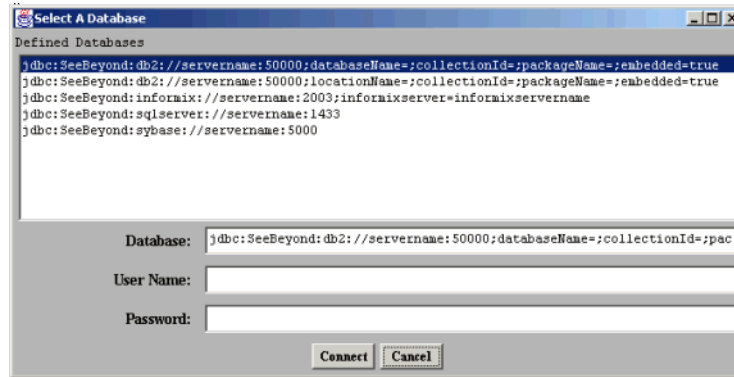
```
c:\eGate\Server\registry\repository\default\ThirdParty\merant\jdbctest\jdbctest.bat
```

From UNIX

```
c:/eGate/Server/registry/repository/default/ThirdParty/
```

- merant/jdbctest/jdbctest.sh
- 2 On the **Connection** menu, click **Connect to DB**.

Figure 11 Select a Database



- 3 For DB2 on Windows or UNIX, select the first line.
- 4 For DB2 on a mainframe system e.g. z/OS, select the second line.
- 5 Enter the information for each of the parameters that corresponds to the DB2 database you are connecting to at the **Database:** prompt.

Note: *If you are connecting to a database for the first time using JDBC, you must first create the necessary packages to assure connectivity. You must have package creation privileges on the server.*

- 6 To create the packages, add the following two parameters to the connection string:

Note: *If creating the package for the first time, `replacePackage=TRUE` should be excluded.*

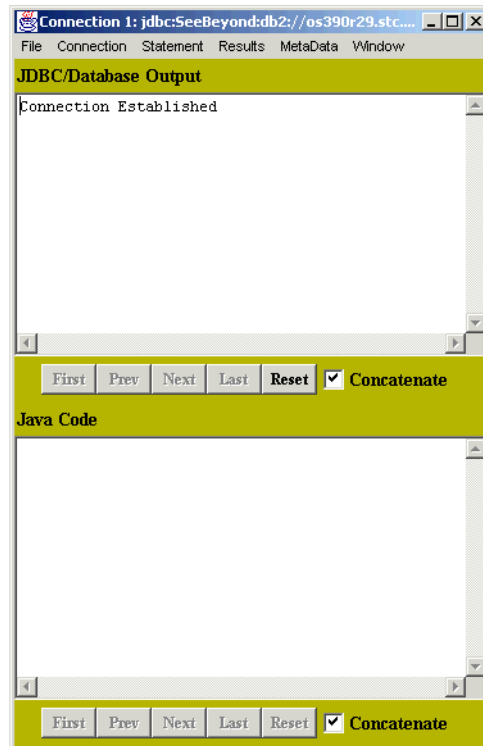
```
createDefaultPackage=TRUE;replacePackage=TRUE
```

For example:

```
jdbc:SeeBeyond:db2://os390r29.stc.com:446;locationName=S390LOC;
collectionId=JDBCPKG;packageName=JDBCPKG;embedded=true;
createDefaultPackage=TRUE;replacePackage=TRUE
```

- 7 Enter a **User Name:** and **Password:**
- 8 Click **Connect**.

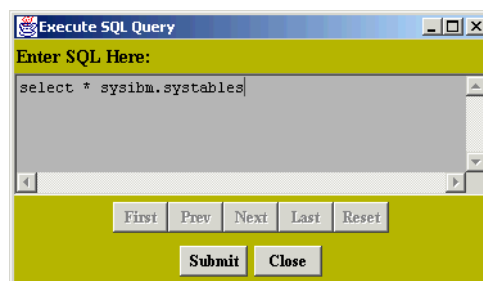
Figure 12 Connection 1 - Connection Established



- 9 On the **Connection** menu, click **Create Statement**.
- 10 On the **Statement** menu, click **Execute Stmt Query**.
- 11 Enter the following SQL statement. See Figure 13:

```
select * from sysibm.systables
```

Figure 13 Execute SQL Query



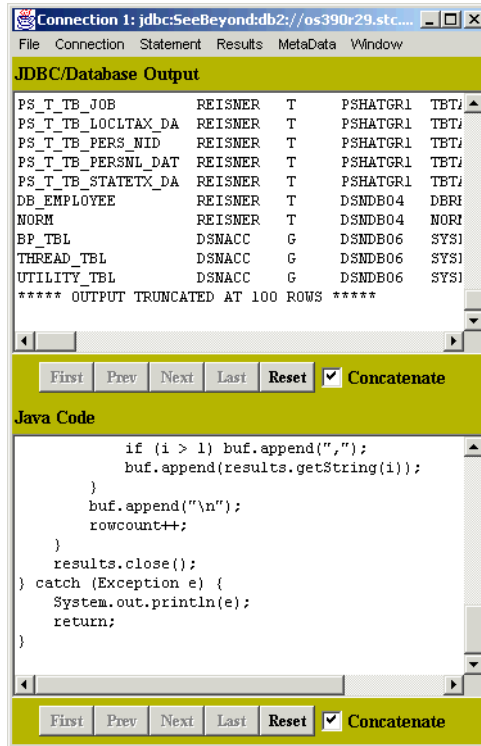
- 12 Click **Submit**.

Figure 14 Connection 1 - Connection Succeeded



- 13 On the **Results** menu, click **Show All Results** to see the results of the query. See Figure 15.

Figure 15 Connection 1 - Results



e*Way Connection Configuration

This chapter describes how to configure the DB2 Universal Database e*Way Connections.

This Chapter Includes:

- “Create e*Way Connections” on page 27
- “DataSource Settings” on page 28
- “Connector Settings” on page 32

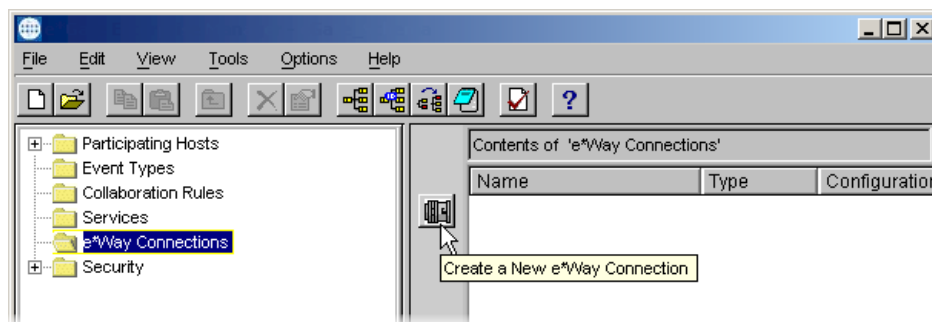
4.1 Create e*Way Connections

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

To create and configure the e*Way Connections

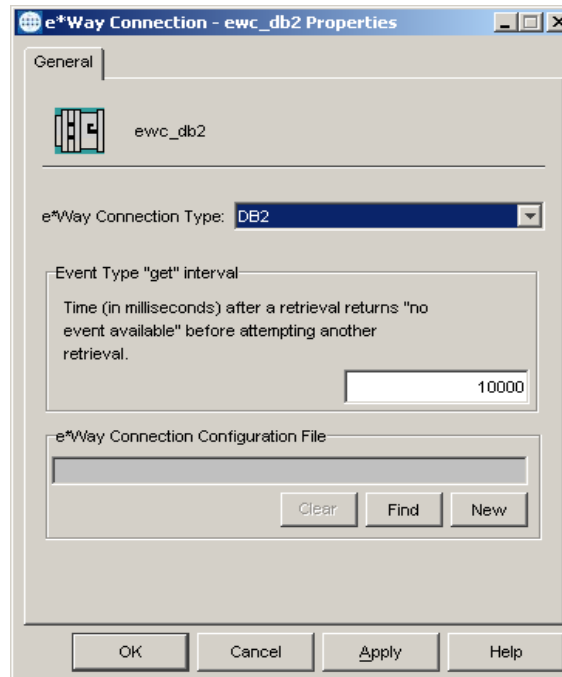
- 1 In the Schema Designer’s Component editor, select the e*Way Connections folder.

Figure 16 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 17 on page 28](#).

Figure 17 e*Way Connection Properties Dialog Box



- 5 In the **e*Way Connection Type** list, click **DB2**.
- 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
- connector

4.1.1 DataSource Settings

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

class

Description

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

Required Values

A valid class name.

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

For z/OS, select **COM.ibm.db2.jdbc.DB2ConnectionPoolDataSource**

For IBM Native z/OS drivers, select **COM.ibm.db2os390.sqlj.jdbc.DB2SQLJDriver**

For connecting to the AS/400, you will need to add to the Class by typing **COM.ibm.db2.jdbc.DB2ConnectionPoolDataSource** at the class entry line.

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.

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"

jdbc url

Description

A list of separated attribute-value pairs. This is used to identify the database and set the connection properties. The attribute name should be exactly the same as the one that is specified in the driver documentation and the value should be a valid one. The whole list will be used to specify the connection properties. To disable an attribute, simply de-select it.

This URL will not be used if the "connection method" section is specified as "URL".

The separator used in this parameter should match the one specified in the "data source attribute value pair separator" section.

The default separator used is "!".

For z/OS: `jdbc url jdbc:db2os390sqlj:<database-location>` where `database-location` is the name of the database defined in the DB2 BSD (bootstrap dataset).

For example: `jdbc:db2os390sqlj:S390LOC`

If no `database-location` is specified, the JDBC Driver will connect to the default DB2 sub-system.

Required Values

A valid URL. For example `PortNumber!8888`

CollectionID

Description

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

Required Values

Any valid string.

DatabaseName

Description

Specifies the name or alias of the database instance. The alias must match the database alias in the DB2 client's database directory. If you are using IBM's DB2 JDBC driver, you will need to enter the database alias name that is configured on the DB2 client.

Required Values

Any valid string.

LocationName

Description

The name of the DB2 location that you want to access. Used with z/OS.

Required Values

Any valid string.

PackageName

Description

Specifies the name, 7 character limit, of the package that the drive uses to process static and dynamic SQL.

Required Values

Any valid string.

user name

Description

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

Required Values

Any valid string.

password

Description

Specifies the password used to access the database.

Required Values

Any valid string.

PortNumber

Description

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

Required Values

Any valid string.

ServerName

Description

The IP address used for DataSource connections only. If you are using IBM's DB2 JDBC driver enter the name of your local machine.

If you are using the IBM DB2 Connect JDBC driver and no Hostname is specified, then the IBM DB2 Connect JDBC Type 2 driver will be invoked.

If you are using the IBM DB2 Connect JDBC driver and a Hostname is specified, then the IBM DB2 Connect JDBC Type 3 driver will be invoked.

It should be noted that XA is only supported with the IBM DB2 Connect JDBC Type 2 driver.

Required Values

Any valid string.

timeout

Description

Specifies the login timeout in seconds.

Required Values

Any valid string. The default is 300 seconds.

data source attribute value pair separator

Description

This entry specifies the character separator used to separate the attribute-value pair used in the "data source attributes" section. For example, the attribute-value pair "ServerName!myHost" has "!" as a separator.

One should select a separator that will NOT be part of the attribute-name or the attribute-value.

Required Values

The default value is "!".

data source attributes

A list of separated attribute-value pairs. This information is used to identify the database and set the connection properties. The attribute name should be exactly the same as the one that is specified in the driver documentation and the value should be a valid one. The whole list will be used to specify the connection properties. To disable an attribute, simply un check it.

For example: PortNumber! 8888

The separator used in this parameter should match the one specified in the "data source attribute value pair separator" section. By default, the separator used is "!".

Required Values

Any separated attribute-value pair.

4.1.2 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 JAVA 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.

If running e*Gate version 4.5.1 or earlier, this option is ignored.

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

Description

This value is used to specify the time-out 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 time-out specified is reached.

If running e*Gate version 4.5.1 or earlier, this option is ignored.

Required Values

Any valid string.

connection verification interval

Description

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.

If running e*Gate version 4.5.1 or earlier, this option is ignored.

Required Values

Any valid string. Default is 60000 ms.

4.2 Connection Manager

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

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

The Connection Manager is controlled in the e*Way configuration as described in [Connector Settings](#) on page 32. If you choose to manually control the e*Way connections, you may find the following chart helpful.

Figure 18 e*Way Connection Control methods

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

Controlling When a Connection is Made

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

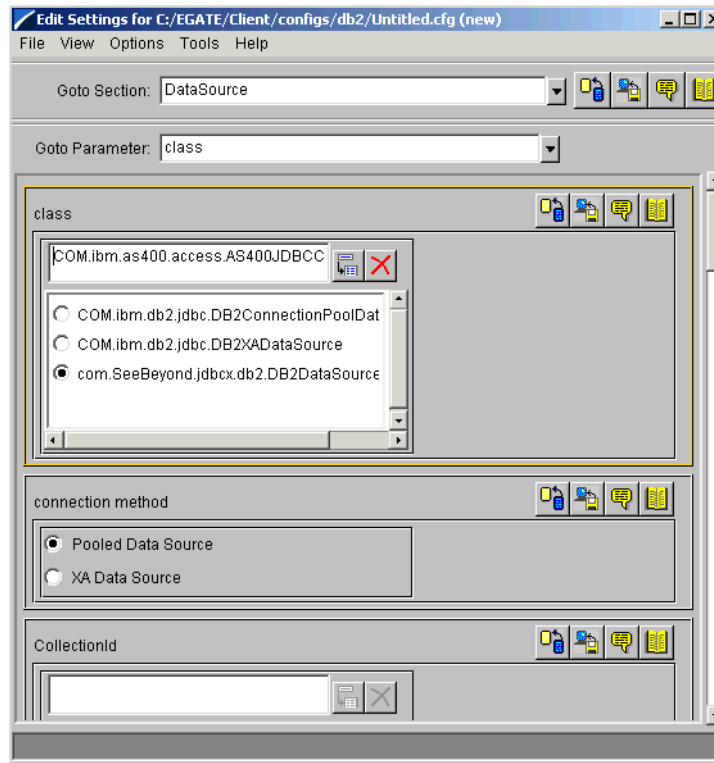
Controlling When a Connection is Disconnected

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.

4.2.1 For Pre-Existing z/OS Connections

If you have pre-existing DB2 connections to z/OS, reconfigure your connection paths to recognize this new configuration.



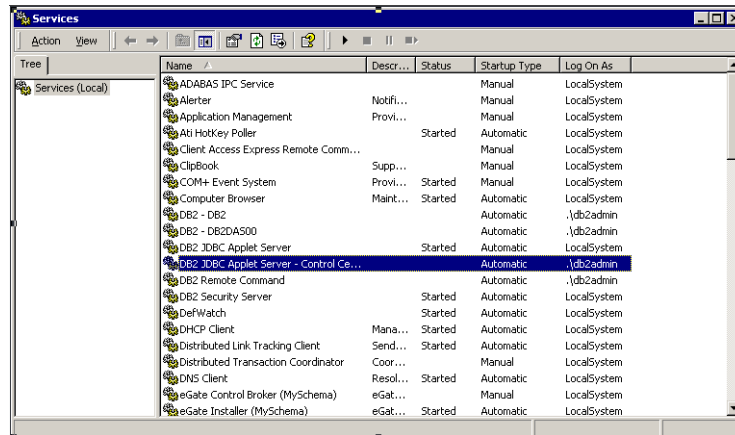
4.3 DB2 JDBC Applet Server

If you have chosen to use the native IBM JDBC drivers for DB2 or you are connecting to z/OS you must start the DB2 JDBC Applet Server.

To do this do the following:

- 1 From the Start menu, click Settings, Control Panel.
- 2 Click Administrative Tools.
- 3 Click Services.
- 4 Select and Start DB2 JDBC Applet Server

Figure 19 DB2 JDBC Applet Server



4.4 Using Large ResultSets

If you have chosen to use the DataDirect ODBC drivers, and you plan to use very large ResultSets with your DB2 e*Way you will need to also configure the data block size of the driver. To do this:

- 1 Click Start, Settings, Control Panel.
- 2 Click Administrative Tools.
- 3 Click Data Sources (ODBC).
- 4 Select the DataDirect ODBC 4.1 wire protocol from the list of drivers. Click Configure.
- 5 Click the Advanced tab on the ODBC DB2 Wire Protocol Driver Setup window. In the Query Block Size: field, enter 32.
- 6 Click OK.

Implementation

This chapter discusses how to implement the Java-enabled DB2 Universal Database e*Way in a production environment. Also included is a sample configuration. This chapter assumes that the DB2 Universal Database e*Way has been successfully installed.

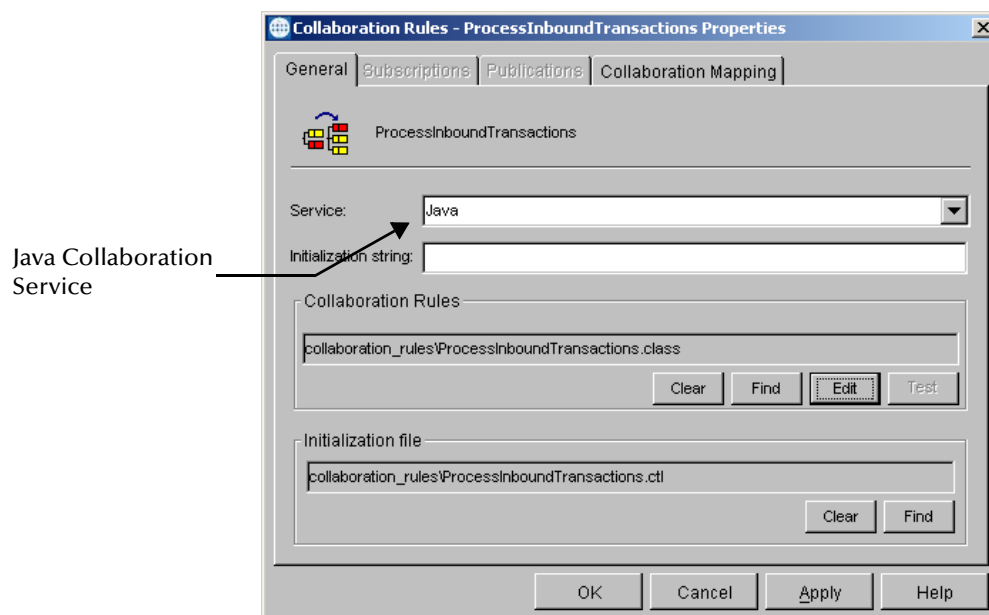
This Chapter Includes:

- “Implementing Java-enabled Components” on page 38
- “The Java ETD Builder” on page 39
- “Sample Scenario—Polling from a DB2 Database” on page 62

5.1 Implementing Java-enabled Components

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

Figure 20 The Java Collaboration Service



5.1.1 The Java Collaboration Service

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

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

5.1.2 Java-enabled Components

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

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

5.2 The Java ETD Builder

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

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

5.2.1 The Parts of the ETD

There are four possible parts to the Java-enabled Event Type Definition as shown in Figure 21.

Figure 21 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.

5.2.2 Using the DBWizard ETD Builder

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

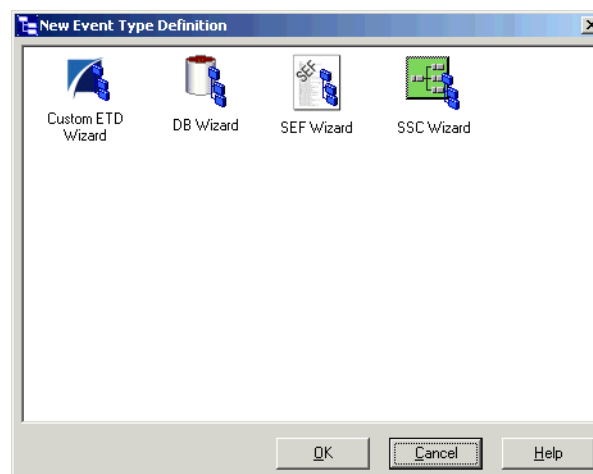
Field nodes are added to the ETD based on the tables in the external data source. Java method and parameter nodes are added to provide the appropriate JDBC functionality. For more information on the Java methods, refer to your JDBC developer's reference.

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

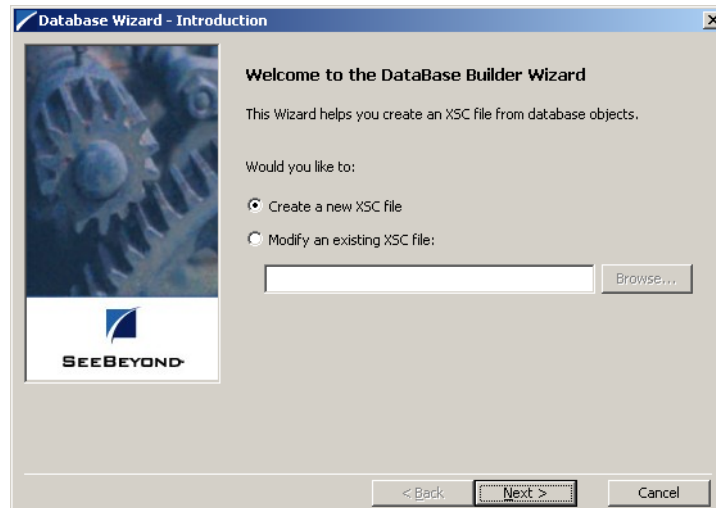
- 1 From the **Options** menu of the Schema Designer, choose **Default Editor....**
- 2 Verify that **Java** is selected, then click **OK**.
- 3 Click the **ETD Editor** button to launch the Java ETD Editor.
- 4 In the **Java ETD Editor**, click the **New** button to launch the New Event Type Definition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

Figure 22 New Event Type Definition



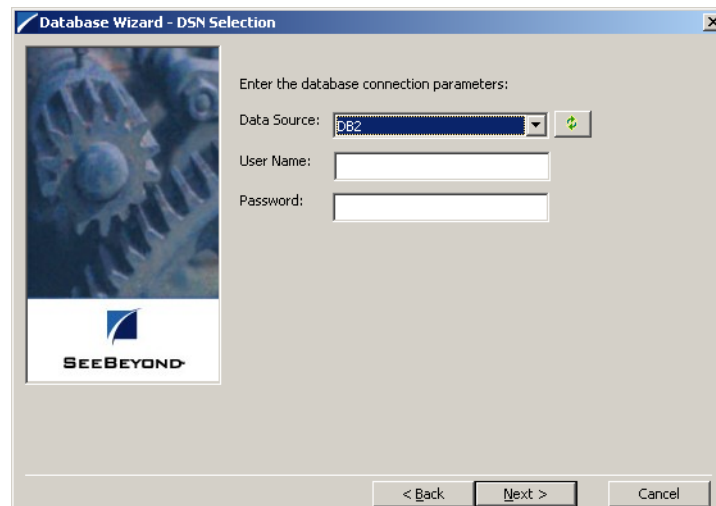
- 6 Enter the name of the new .xsc file you want to create or enter the name of the .xsc file you want to edit by browsing to its location.

Figure 23 Database Wizard - Introduction



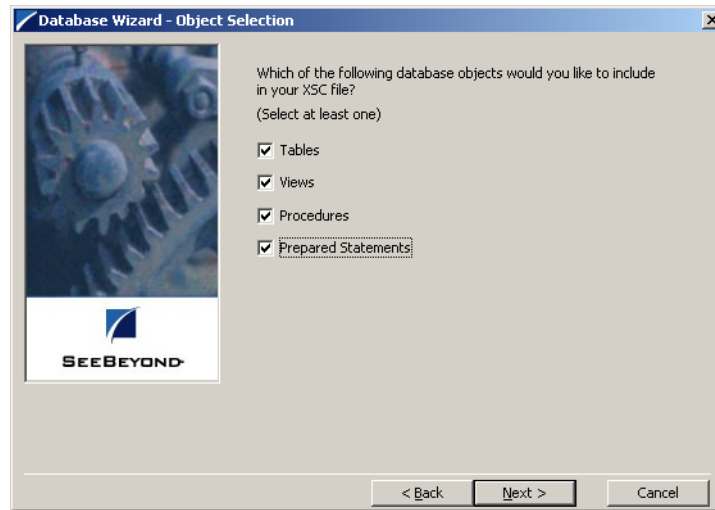
- 7 Select your **Data Source:** from the drop down list and enter your **User Name:** and **Password:**.

Figure 24 Database Wizard - DSN Selection



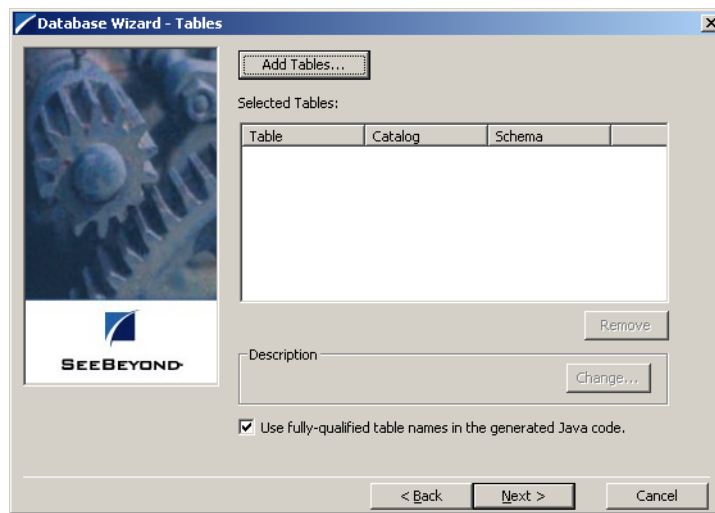
- 8 Select what type of database ETD you would like to generate. The data source you selected in the **Database Wizard - DSN Selection** window is the default. *Note: Do not change this unless instructed to do so by Oracle personnel.*
- 9 In the **Database Wizard - Object Selection** window, select any combination of **Tables, Views, Procedures, or Prepared Statements** you would like to include in your .xsc file. Click **Next** to continue.

Figure 25 Database Wizard - Object Selection



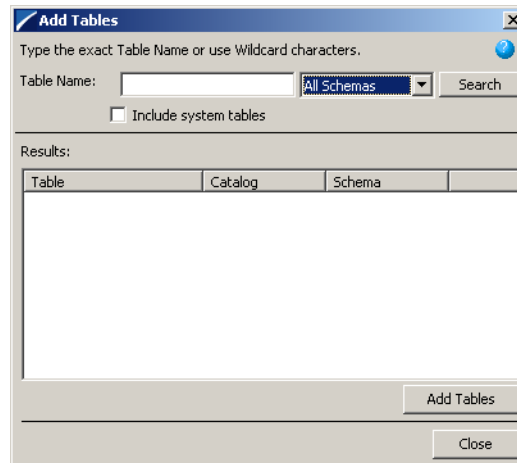
- 10 In the **Database Wizard - Tables** window, click **Add Tables**.

Figure 26 Database Wizard - Tables



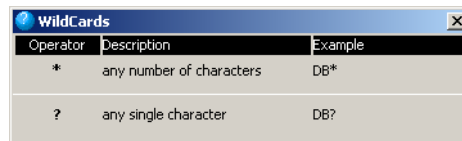
- 11 In the **Add Tables** window, type the exact name of the database table or use wildcard characters to return table names.

Figure 27 Add Tables



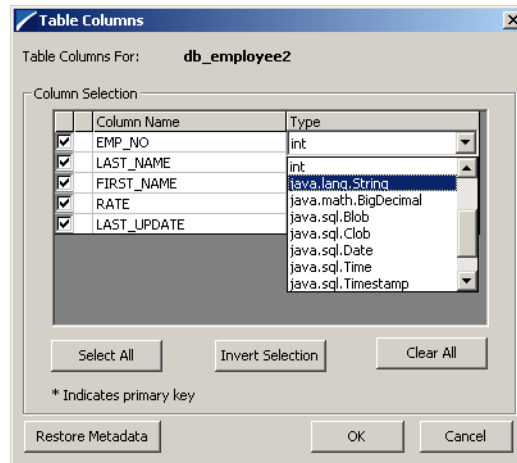
- 12 To see a list of valid wildcard characters, click the round ball with a question mark located in its center.

Figure 28 Wildcards



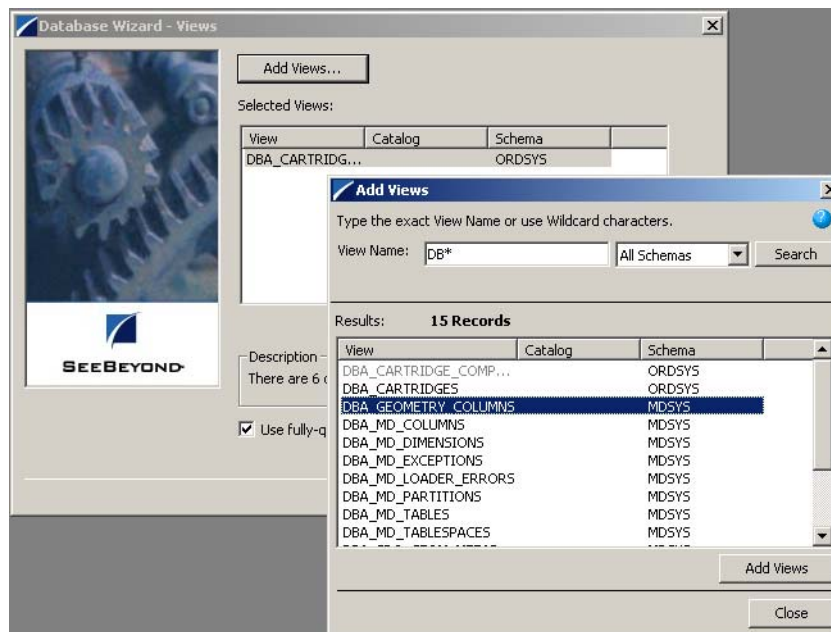
- 13 Select **Include System Tables** if you wish to include them and click **Search**. If your search was successful, you will see the results in the Results window. To select the name of the tables you wish to add to your .xsc, double click on the table name or highlight the table names and click **Add Tables**. You may also use adjacent selections or nonadjacent selections to select multiple table names. When you have finished, click **Close**.
- 14 In the **Database Wizard - Tables** window, review the tables you have selected. If you would like to change any of the tables you have selected, click **Change**.
- 15 In the **Columns Selection** window, you can select or deselect your table choices. You can also change the data type for each table by highlighting the data type and selecting a different data type from the drop down list. Once you have completed your choices, click **OK**.

Figure 29 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 30 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. **This mode is not recommend for use with the z/OS.**

“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 execute statement up to and including the table name(s) before executing the query.

When using “With Assistance” mode on z/OS you should use the unqualified table names that were specified during the bind process of the Stored Procedure:

```
SELECT
    DEPT.DEPTNO AS DEPTNO
    DEPT.DEPTNAME AS DEPTNAME,
    DEPT.MGRNO AS MGRNO,
    DEPT.ADMRDEPT AS ADMRDEPT,
    DEPT.LOCTION AS LOCATION
FROM
    SALES'
SELECT
    EMP.EMPNO AS EMPNO,
    EMP.FIRSTNME AS FIRSTNME,
    EMP.MIDINIT AS MIDINIT,
    EMP.LASTNAME AS LASTNAME,
    EMP.WORKDEPT AS WORKDEPT,
    EMP.PHONENO AS PHONENO,
    EMP.HIREDATE AS HIREDATE,
    EMP.JOB AS JOB,
    EMP.EDLEVEL AS EDLEVEL,
    EMP.SEX AS SEX,
    EMP.BIRTHDTE AS BIRTHDATE,
    EMP.SALARY AS SALARY,
    EMP.BONUS AS BONUS,
    EMP.COMM AS COMM
FROM
    EMP;
/*****
/* STEP 3: Register MULTIRS in SYSIBM.SYSROUTINES
/*          and Bind the package for MEDRS
/*****
//STEP3    EXEC PGM=IKJEFT01,DYNAMNBR=20,
//          COND=(4,LT)
```

```

//DBRMLIB DD DSN=DSN610.DBRMLIB.DATA(MEDRS),DISP=SHR
//SYSTSPRT DD SYSOUT=*
//SYSTSIN DD *
DSN SYSTEM(DSN1)
  RUN PROGRAM(DSNTIAD) PLAN(DSNTIA61) -
    LIB('DSN610.RUNLIB.LOAD')
  BIND PACKAGE(MEDRS) -
    QUALIFIER(Q) -
    MEMBER(MEDRS) ACT(REP) ISO(CS)
END
//SYSPRINT DD SYSOUT=*
//SYSUDUMP DD SYSOUT=*
//SYSIN DD DSN=&&SPDDL,DISP=(OLD,DELETE) <- From preceding step
//*

```

The schema qualifier name for the DEPT and EMP table is “DSM8610”. The SALES table does not exist.

When using “With Assistance” mode, you must insert a table qualifier of “SSM8610” to generate the ResultSet for EMP and DEPT table correctly for this Stored Procedure:

```

SELECT
  DEPTNO AS DEPTNO
  DEPTNAME AS DEPTNAME,
  MGRNO AS MGRNO,
  ADMRDEPT AS ADMRDEPT,
  LOCTION AS LOCATION
FROM
  SALES'
SELECT
  EMPNO AS EMPNO,
  FIRSTNME AS FIRSTNME,
  MIDINIT AS MIDINIT,
  LASTNAME AS LASTNAME,
  WORKDEPT AS WORKDEPT,
  PHONENO AS PHONENO,
  HIREDATE AS HIREDATE,
  JOB AS JOB,
  EDLEVEL AS EDLEVEL,
  SEX AS SEX,
  BIRTHDTE AS BIRTHDATE,
  SALARY AS SALARY,
  BONUS AS BONUS,
  COMM AS COMM
FROM
  EMP;

```

“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.

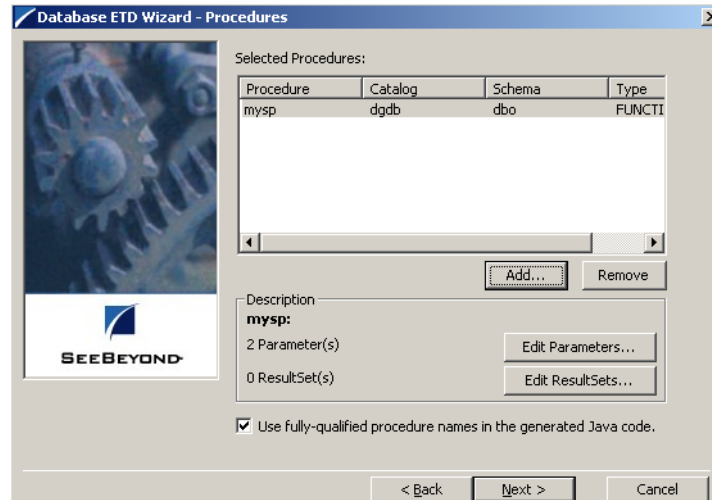
Note: *If you modify the ResultSet generated by the “Execute” mode of the Database Wizard you will need to make sure the indexes match the Stored Procedure. This will assure your ResultSet indexes are preserved,*

Please note that Stored Procedures without associated parameters will not appear in the list. For example:

Create procedure testProcNoParam – this will not appear in the list because there are no in or out parameters.

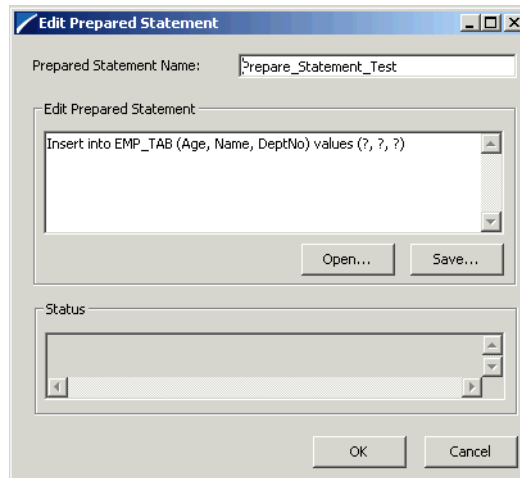
Create procedure testProcParam(IN var int) – this will appear in the list because there are in or out parameters.

Figure 31 Database Wizard - Procedures



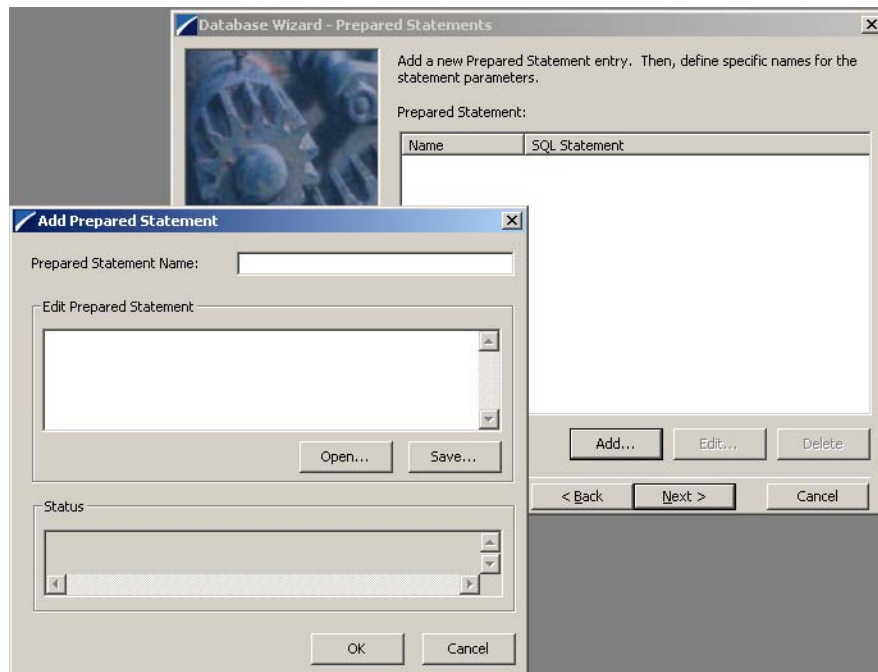
- 19 If you selected **Prepared Statements** on the **Database Wizard - Object Selection** window, you are presented with the **Database Wizard - Prepared Statement** window. To add **Prepared Statements** to your .xsc, complete the following steps:
- A Click **Add** to add a new prepared statement
 - 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 32.

Figure 32 Add Prepared Statement



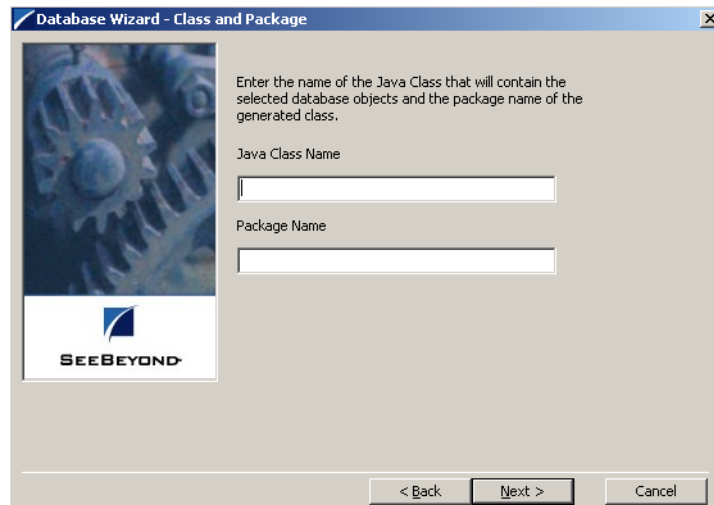
- E Click **OK** to return to the **Database Wizard - Prepared Statements** window.
- 20 Repeat steps A–E to add additional prepared statements or click **Next** to continue.

Figure 33 Database Wizard - Prepared Statements



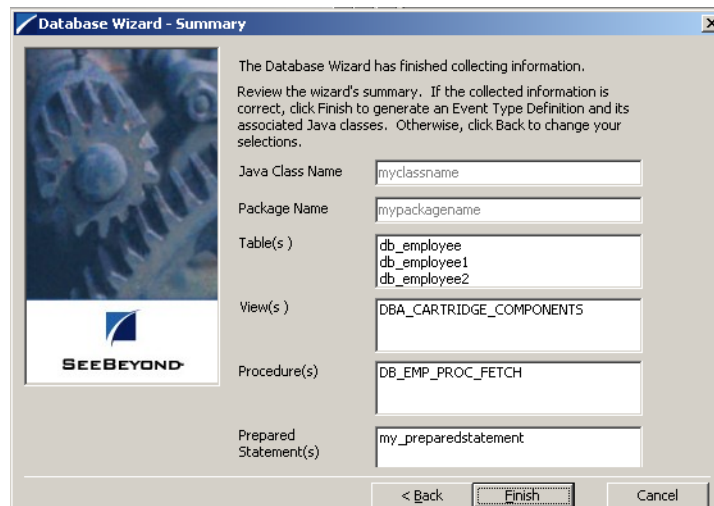
- 21 Enter the **Java Class Name** that will contain the selected tables and/or procedures and the **Package Name** of the generated classes.

Figure 34 Database Wizard - Class and Package



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

Figure 35 Database Wizard - Summary



5.2.3 The Generated ETDs

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

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

- **The View ETD** - The view ETD contains selected columns from selected tables. View ETD's are read-only.
- **The Stored Procedure ETD** – The stored procedure ETD contains fields which correspond to the input and output fields in the procedure. To edit this type of ETD, you 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

5.2.4 Editing an Existing .XSC Using the Database Wizard

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

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

You are now able to edit your .xsc file.

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

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

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

The Insert, Delete, Update, and Execute Methods are common methods used within the Collaboration Editor. Though our sample schema doesn't include these methods, we thought it would be helpful to show them to you.

5.3.1 The Table

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

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:

- `SetConcurrencytoUpdatable`
- `SetConcurrentlytoRead Only`
- `SetScrollTypetoForwardOnly`
- `SetScrollTypetoScrollSensitive`
- `SetScrollTypetoInsensitive`

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

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

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 data types 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_EMPLOYEE().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 36 Insert Method Business Rule

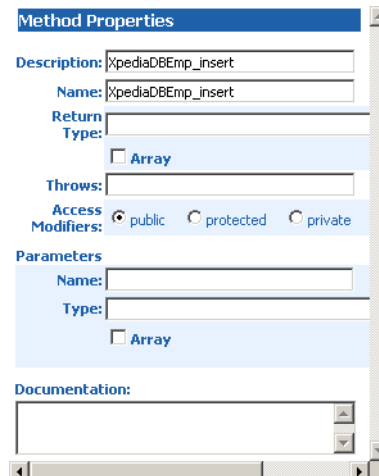
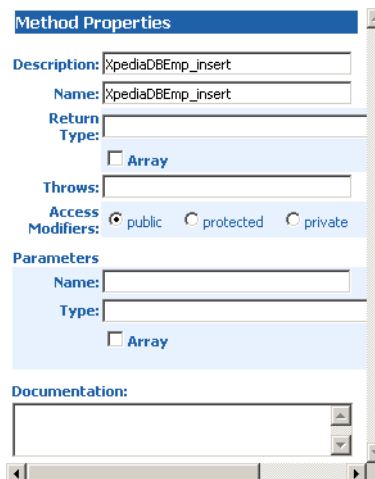


Figure 37 Insert Method Properties



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

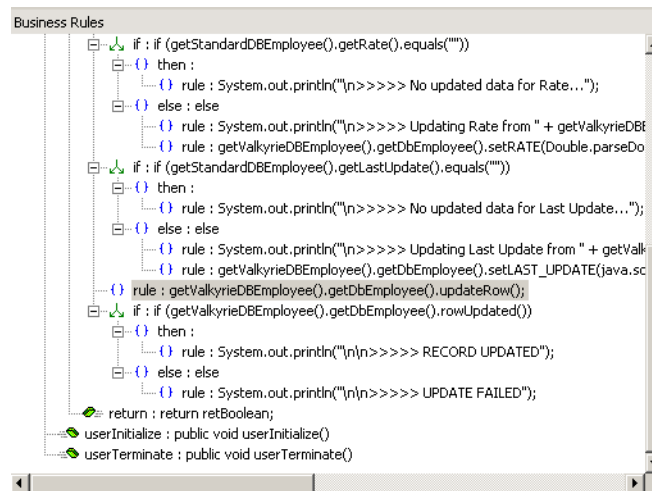
You must specify ResultSets with:

- 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 38 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();
```

5.3.2 The View

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

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

5.3.3 The Stored Procedure

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

Note: BLOB and CLOB data types are not supported when using the DataDirect drivers.

Please note that Stored Procedures without associated parameters will not appear in the list.

To retrieve a list of stored procedures, those that have no parameters will not show up in the list. For example, stored procedures like this cannot be retrieved: getEmpNo() because there is no input or output parameters.

Executing Stored Procedures

Assuming that you have the following procedure:

```
CREATE PROCEDURE DB2ADMIN.LookupGlobal ( IN inlocalID varchar(10),
                                         OUT outglobalProductID
                                         varchar(10) )
    SPECIFIC DB2ADMIN.LookupGlobal
    LANGUAGE SQL
P1: BEGIN
    DECLARE ENDTABLE INT DEFAULT 0;
    DECLARE cursor1 CURSOR FOR
        select globalProductID from SimpleLookup where localID =
inlocalID;
    SET ENDTABLE = 1;
    OPEN cursor1;
        SET ENDTABLE = 0;
        WHILE ENDTABLE = 0 DO
            FETCH cursor1 INTO outglobalProductID;
        END WHILE;
    CLOSE cursor1;
END P1
```

The ETD represents the Stored Procedure "LookUpGlobal" with two parameters, an inbound parameter (INLOCALID) and an outbound parameter (OUTGLOBALPRODUCTID) can be generated by the DBWizard as shown in Figure 39. 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:

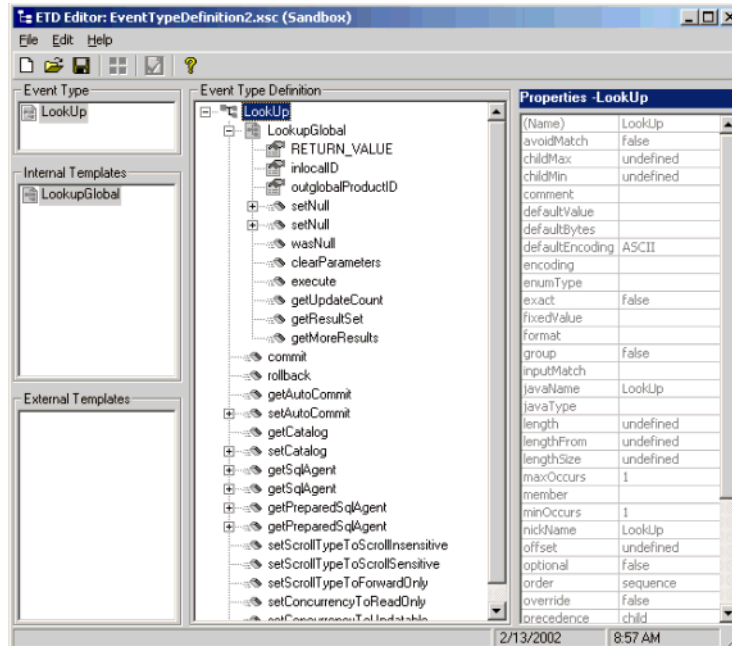
- 1 Specify the input values.

- 2 Execute the Stored Procedure.
- 3 Retrieve the output parameters if any.

For example:

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

Figure 39 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. JDBC provides

three methods (**getMoreResults()**, **getUpdateCount()**, and **getResultSet()**) to access the results of a Stored Procedure call, the information returned from these methods at times, can be confusing. You can simply call **resultsAvailable()** and if Boolean true is returned, you can expect either a valid Update Count when **getUpdateCount()** is called and/or the next ResultSet has been retrieved and made available to one of the ResultSet nodes defined for the Stored Procedure ETD, when that node's **available()** method returns true.

Frequently, Update Counts information that is returned from a Stored Procedures is insignificant. You will want to only process returned ResultSet information and avoid looping through all of the Update Counts. The following three methods control exactly what information should be returned from a Stored Procedure call. The **enableResultSetsOnly()** method, added to the PreparedStatement Agent class allows only ResultSets to be returned and thus every **resultsAvailable()** called 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 (getSPIn().getSpS_multi().resultsAvailable())
{
    // check if there's an update count
    if (getSPIn().getSpS_multi().getUpdateCount() > 0)
    {
        System.err.println("Updated
"+getSPIn().getSpS_multi().getUpdateCount()+" rows");
    }

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

```

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

```

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

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

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 e.g., the DataDirect JDBC 3.0 driver 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** object should be retrieved before OUT parameters are retrieved. Therefore, you should retrieve all **ResultSet(s)** and update counts first followed by retrieving the OUT type parameters and return values.

ResultSet Behavior

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

- The method **resultsAvailable()** implicitly calls **getMoreResults()** when it is called more than once. You should not call both methods in your java code. Doing so may result in skipped data from one of the **ResultSets** when more than one **ResultSet** is present.
- The methods **available()** and **getResultSet()** can not be used in conjunction with multiple **ResultSets** being open at the same time. Attempting to open more the one **ResultSet** at the same time 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.
- The methods **resultsAvailable()** and **available()** when called on a table that contains no records, will return an empty ResultSet.

Supported Data Types Using DB2

The following data types are supported by the ResultSet of a Stored Procedure using DB2 on Windows and Unix operating systems:

- Bigint
- Char
- Date
- Decimal
- Double
- Float
- Integer
- Long Varchar
- Numeric
- Real
- Smallint
- Time
- Timestamp
- Varchar
- Char for Bit Data
- Long Varchar for Bit Data
- Varchar for Bit Data

Data types supported by the ResultSet of a Stored Procedure using DB2 on a z/OS operating system:

- Char
- Date Decimal
- Double
- Float
- Integer
- Long Varchar
- Numeric
- Real
- Smallint

- Time
- Timestamp
- Varchar
- Char for Bit Data
- Long Varchar for Bit Data
- Varchar for Bit Data

Attempting To Get Output Values While Using DataDirect Drivers

Attempting to get output values from a Stored Procedure when a ResultSet is Not Present

When attempting to retrieve a ResultSet from a Stored Procedure that has input and output parameters but no ResultSet, try retrieving the ResultSet first. For example:

```

getTestdb2SPSDBEmp().getDB_EMPLOYEE_SELECT().execute();
    while
    (getTestdb2SPSDBEmp().getDB_EMPLOYEE_SELECT().resultsAvailable())
    {
        if
        (getTestdb2SPSDBEmp().getDB_EMPLOYEE_SELECT().getUpdateCount() > 0)
        {
            EGate.traceln(EGate.TRACE_COLLABSERVICE,EGate.TRACE_EVENT_INFORMATION
            ,">>>> Update Count is greate than 0! <<<<<");
        }
        else
        {
        }
    }

getgenout().setPayload(getTestdb2SPSDBEmp().getDB_EMPLOYEE_SELECT().g
etEMPNO_OUT() + "|" +
getTestdb2SPSDBEmp().getDB_EMPLOYEE_SELECT().getLASTNAME_OUT() + "|"
+ getTestdb2SPSDBEmp().getDB_EMPLOYEE_SELECT().getFIRSTNAME_OUT() +
"|" + getTestdb2SPSDBEmp().getDB_EMPLOYEE_SELECT().getRATE_OUT() +
"|" +
getTestdb2SPSDBEmp().getDB_EMPLOYEE_SELECT().getLASTUPDATE_OUT());

```

5.3.4 Prepared Statement

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

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

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

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

```

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

```

5.3.5 Batch Operations

While the Java API used by the e*Gate system 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.

```

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

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

5.3.6 Database Configuration Node

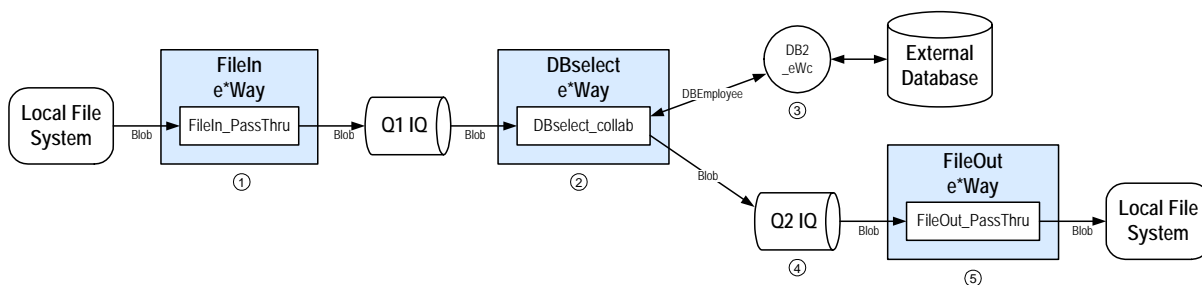
The Database Configuration node allows you to manage the “transaction mode” through the Collaboration if you have set the mode to manual in the e*Way connection configuration.

5.4 Sample Scenario—Polling from a DB2 Database

This section describes how to use the Java-enabled DB2 Universal Database e*Way in a sample implementation. This sample schema demonstrates the polling of records from a DB2 database and converting the records into e*Gate Events.

Figure 40 shows a graphical overview of the sample schema.

Figure 40 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**.

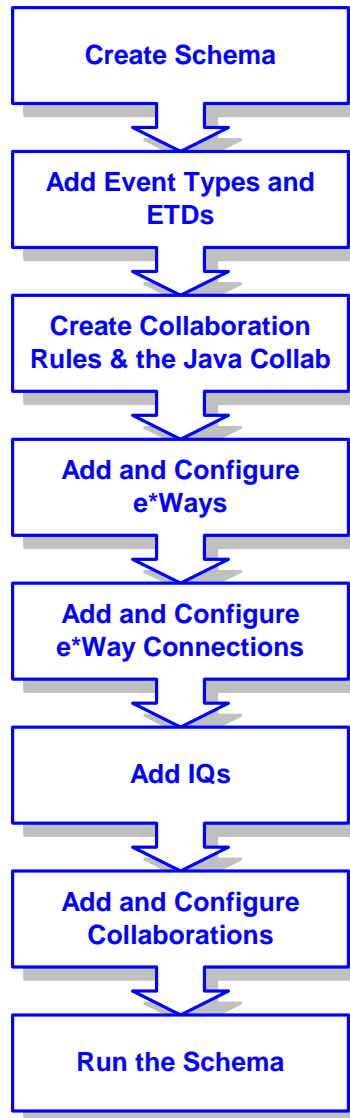
- 2 The **DBselect** e*Way retrieves the Generic Event (**Blob**) from the IQ. This triggers the rest of the Collaboration which has two parts.
- 3 The information in **Blob** is used to retrieve information from the database via the **DB2_eWc** e*Way Connection. This e*Way Connection contains information used by the Collaboration to connect to the DB2 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 65](#)
- [“Add the Event Types and Event Type Definitions” on page 65](#)
- [“Create the Collaboration Rules and the Java Collaboration” on page 68](#)
- [“Add and Configure the e*Ways” on page 72](#)
- [“Add and Configure the e*Way Connections” on page 74](#)
- [“Add the IQs” on page 75](#)
- [“Add and Configure the Collaborations” on page 75](#)
- [“Run the Schema” on page 77](#)

Figure 41 Schema Configuration Steps



External Database Tables

The sample uses a simple external DB2 database with a table called **DB_EMPLOYEE**. The table contains the following columns:

Table 4 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.

5.4.1 Create the Schema

The first step in the sample implementation is creating a new schema. After installing the DB2 Universal Database e*Way, do the following:

- 1 Start the e*Gate Schema Designer GUI.
- 2 Log in to 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 starts and the newly created schema appears.

5.4.2 Add the Event Types and Event Type Definitions

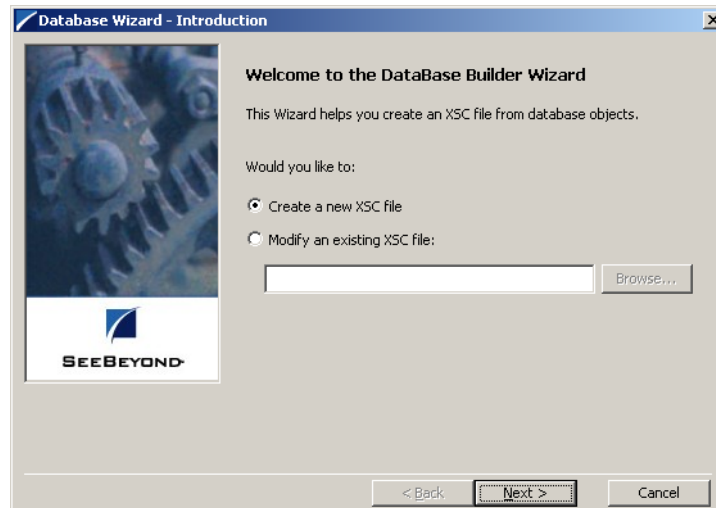
Two Event Types and Event Type Definitions are used in this sample.

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

To create the DBEmployee Event Type and ETD

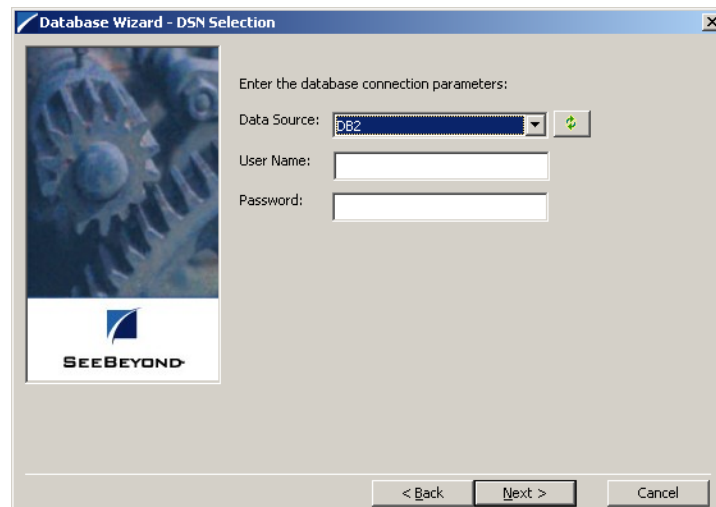
- 1 From the **Options** menu of the Schema Designer, choose **Default Editor...**
- 2 Verify that **Java** is selected, then click **OK**.
- 3 In the **Components** pane of the Schema Designer, select the **Event Types** folder.
- 4 Click the **New Event Type** button to add a new Event Type.
- 5 Enter the name **DBEmployee** and click **OK**.
- 6 Double-click the new **DBEmployee** Event Type to display its properties.
- 7 Select your **Data Source:** from the drop down list and enter your **User Name:** and **Password:**.
- 8 From the **File** menu, choose **New**. The New Event Type Definition dialog box 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 42.

Figure 42 Database Wizard Introduction



- 11 Enter the database DNS source and login information.
 - A Select the **Data Source** from the dropdown list of ODBC data sources.
 - B Enter the **User Name** and **Password** used to log into the database.
- Click **Next** to continue. See Figure 43

Figure 43 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 Oracle 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 **Custom 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 5.

Table 5 GenericBlob ETD Properties

Node	Property	Value
Root Node	Name	GenericBlob
	Structure	fixed
	Length	0

Node	Property	Value
Child Node	Name	Data
	Structure	fixed
	Length	0

Note: To use a Blob, you will need to set the `LONGDATACOMPAT = 0` in the file `db2cli.ini` under the DB2 root directory. This will disable the `LongVarChar`, `LongVarBinary`, and `LongVarGraphic`. Refer to the *IBM Installation and Configuration Supplement* for more information.

- 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.

5.4.3 Create the Collaboration Rules and the Java Collaboration

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

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

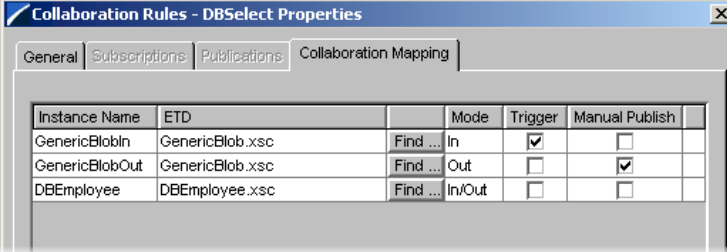
To create the **GenericPassThru** Event Type

- 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 44 below:

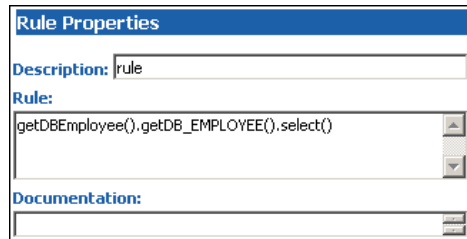
Figure 44 DBSelect Instances



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

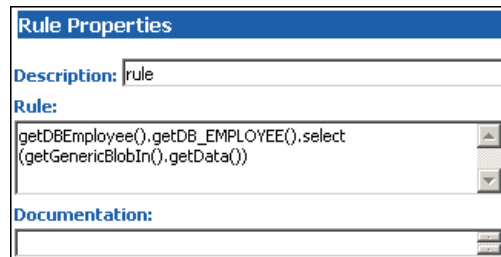
- 8 Click **Apply** to save the current changes.
- 9 Click the **General** tab.
- 10 Click **New** to create the new Collaboration file.
The Java Collaboration Editor appears. Note that Source and Destination Events are already supplied based on the Collaboration Rule's Collaboration Mapping (see Figure 44).
- 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 45).

Figure 45 Rule Properties



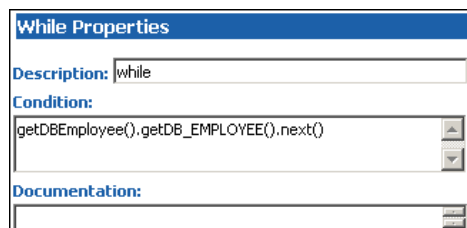
- 15 In the **Source Events** pane, expand the **GenericBlobIn** Event Type until the **Data** node is visible.
- 16 In the Rule Properties pane, position the cursor inside the parentheses of the select method. Then drag the Data node from the Source Events pane into the select method's parentheses. (See Figure 46).

Figure 46 Rule Properties (Continued)



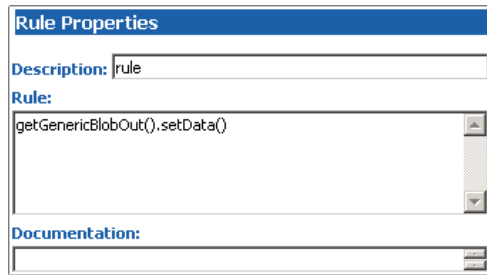
- 17 Select the newly edited rule in the **Business Rules** pane and click the **while** button to add a new while loop beneath the current rule.
- 18 Drag the next method from the Destination Events pane into the Condition field of the While Properties pane. (See Figure 47).

Figure 47 While Properties



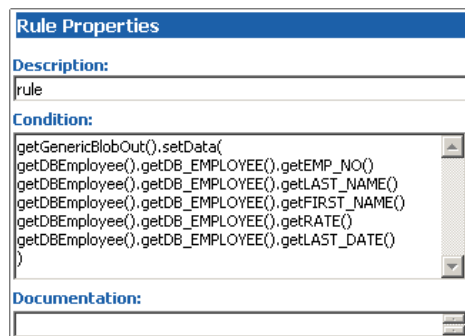
- 19 Select the newly edited while loop in the Business Rules pane and click the **rule** button to add a new rule as a **child** to the while loop.
- 20 In the **Destination Events** pane, expand the **GenericBlobOut** Event Type until the **Data** node is visible.
- 21 Drag the Data node into the Rule field of the Rule Properties pane. (See Figure 48).

Figure 48 Rule Properties



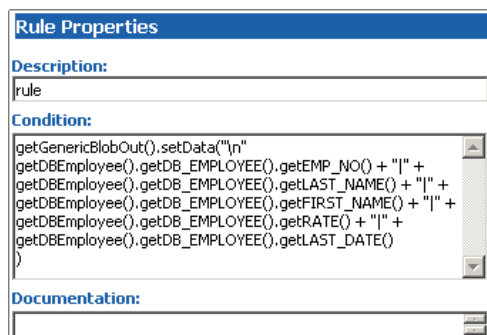
- 22 In the Rule Properties pane, position the cursor inside the parentheses of the setData() method. Then drag each of the five data nodes of DB_EMPLOYEE from the Source Events into the parentheses of the rule. (See Figure 49).

Figure 49 Rule Properties (Continued)

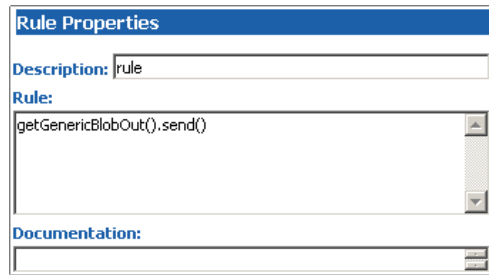


- 23 Edit the text of the condition to add a newline character and pipe (|) delimiters between each of the five data nodes. (See Figure 50).

Figure 50 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 51.

Figure 51 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.

5.4.4 Add and Configure the e*Ways

The sample scenario uses three e*Ways:

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

To create the FileIn e*Way

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

Table 6 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 configuration parameters for the e*Way. See [“Create e*Way Connections” on page 27](#).
- 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 7.

Table 7 FileOut e*Way Parameters

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

- 7 Select **Save** from the **File** menu. Enter **FileOut** as the file name and click **Save**.
- 8 Select **Promote to Run Time** from the **File** menu. Click **OK** to close the 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.

5.4.5 Add and Configure the e*Way Connections

The sample scenario uses one e*Way Connection:

- **DB2_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 **DB2_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 **DB2** 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 8.

Table 8 DB2_eWc e*Way Connection Parameters

Section Name	Parameter	Value
DataSource	class	default
	DatabaseName	Use the database name listed in your ODBC entry.
	user name	Use local settings.
	password	Use local settings.
	retry interval	default
connector	All	default

- 8 Select **Save** from the **File** menu. Enter **DB2_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.

5.4.6 Add the IQs

The sample scenario uses two IQs:

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

To add the IQs

- 1 In the components pane of the Schema Designer, select the IQ Manager.
- 2 Click the **New IQ** button to add a new IQ.
- 3 Enter the name **Q1** and click **Apply** to save the IQ and leave the New IQ dialog box open.
- 4 Enter the name **Q2** and click **OK** to save the second IQ.
- 5 Select the IQ Manager and click the **Properties** button.
- 6 Select the **Start automatically** check box and click **OK** to save the properties.

5.4.7 Add and Configure the Collaborations

The sample scenario uses three Collaborations:

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

To add the FileIn_PassThru Collaboration

- 1 In the components pane of the Schema Designer, select the **FileIn e*Way**.
- 2 Click the **New Collaboration** button to create a new Collaboration.
- 3 Enter the name **FileIn_PassThru** and click **OK**.
- 4 Select the newly created Collaboration and click the **Properties** button.
- 5 Select **GenericPassThru** from the 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 **DB2_eWc** destination.
- 10 Click the lower **Add** button to add a new Publication.
- 11 Select the **GenericEvent** Event Type and the **Q2** destination.
- 12 Click **OK** to close the Collaboration's properties.

To add the FileOut_PassThru Collaboration

- 1 In the components pane of the 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.

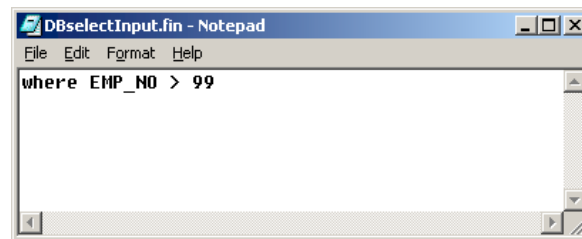
5.4.8 Run the Schema

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

The sample input file

Use a text editor to create an input file to be read by the inbound file e*Way (**FileIn**). This simple input file contains the criteria for the **dbSelect.class** Collaboration's select statement. An example of an input file is shown in Figure 52.

Figure 52 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,

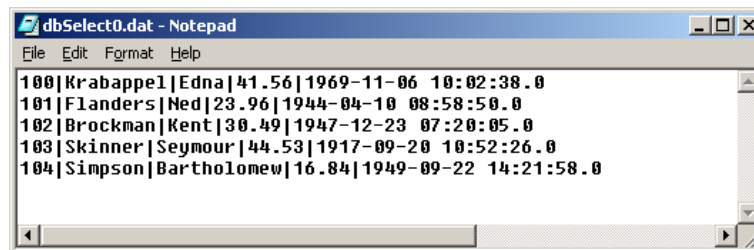
registry is the name of the Registry Host, and

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

Figure 53 Sample Output File



5.4.9 Importing the z/OS Sample Schema

To import the z/OS sample schema included on the e*Gate Installation CD-ROM, do the following:

- 1 On the **Schema Designer** toolbar, click **File, Import Definitions from File...**
- 2 Review the information provided in the **Import Wizard - Introduction** window. Click **Next**.
- 3 On the **Import Wizard - Step 1** window, select **Schema** and click **Next**.
- 4 On the **Import Wizard - Step 2** window, browse to the **ewdb2** sample folder.
- 5 Double click the **ewdb2** sample folder and select the **db2sample_OS390.zip** file.
- 6 Click **Open**.
- 7 On the **Import Wizard - Step 2** window, click **Next**.
- 8 On the **Import Wizard - Step 3** window, you can rename the host or change the port. This step is optional.
- 9 Click **Next**.
- 10 On the **Import Wizard - Finish** window, click **Finish**.
- 11 Follow the instructions provided for the Polling Sample Schema to run your schema.

DB2 e*Way Methods

The DB2 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 79
- [com.stc.eways.jdbcx.PreparedStatementAgent Class](#) on page 89
- [com.stc.eways.jdbcx.PreparedStatementResultSet Class](#) on page 101
- [com.stc.eways.jdbcx.SqlStatementAgent Class](#) on page 127
- [com.stc.eways.jdbcx.CallableStatementAgent Class](#) on page 129
- [com.stc.eways.jdbcx.TableResultSet Class](#) on page 141

6.1 com.stc.eways.jdbcx.StatementAgent Class

```

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

```

All Implemented Interfaces

ResetEventListener, SessionEventListener

Direct Known Subclasses

PreparedStatementAgent, SQLStatementAgent, TableResultSet

```

public abstract class StatementAgent
extends java.lang.Object

```

Implements SessionEventListener, ResetEventListener

Abstract class for other Statement Agent.

Methods of the StatementAgent

- [cancel](#) on page 87
- [clearWarnings](#) on page 88
- [getFetchDirection](#) on page 84
- [getMaxFieldSize](#) on page 86
- [getMoreResults](#) on page 87
- [getResultSetConcurrency](#) on page 83
- [getUpdateCount](#) on page 86
- [isClosed](#) on page 81
- [queryName](#) on page 81
- [resultSetConcurToString](#) on page 81
- [resultSetTypeToString](#) on page 80
- [sessionOpen](#) on page 82
- [setEscapeProcessing](#) on page 83
- [setMaxFieldSize](#) on page 86
- [setQueryTimeout](#) on page 84
- [stmtInvoke](#) on page 88
- [clearBatch](#) on page 87
- [executeBatch](#) on page 87
- [getFetchSize](#) on page 85
- [getMaxRows](#) on page 85
- [getResultSet](#) on page 86
- [getResultSetType](#) on page 82
- [getWarnings](#) on page 88
- [queryDescription](#) on page 81
- [resetRequested](#) on page 82
- [resultSetDirToString](#) on page 80
- [sessionClosed](#) on page 82
- [setCursorName](#) on page 83
- [setFetchDirection](#) on page 84
- [setMaxRows](#) on page 85
- [setQueryTimeout](#) on page 84

resultSetTypeToString

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

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

Name	Description
type	ResultSet type.

Returns

Enumeration symbol string.

resultSetDirToString

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

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

Name	Description
dir	ResultSet scroll directions.

Returns

Enumeration symbol string.

resultSetConcurToString

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

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

Name	Description
concur	ResultSet concurrency.

Returns

Enumeration symbol string.

isClosed

This method returns the statement agent's close status.

```
public boolean isClosed()
```

Returns

True if the statement agent is closed.

queryName

This method supplies the name of the listener.

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

Specified By

queryName in interface SessionEventListener.

Returns

The listener's class name.

queryDescription

This method gives a description of the query.

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

Returns

The description of the query.

sessionOpen

Opens the session event handler.

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

Specified by

sessionOpen in interface SessionEventListener

Name	Description
evt	Session event.

sessionClosed

Closes the session event handler.

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

Specified by

sessionClosed in interface SessionEventListener

Name	Description
evt	Session event.

resetRequested

Resets the event handler.

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

Specified by

resetRequested in interface ResetEventListener

Name	Description
evt	Requested Reset event.

Throws

java.sql.SQLException

getResultSetType

Returns the result set scroll type.

```
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.

6.2 com.stc.eways.jdbcx.PreparedStatementAgent Class

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

All Implemented Interfaces

ResetEventListener, SessionEventListener

Direct Known Subclasses

CallableStatementAgent

```
public class PreparedStatementAgent
extends StatementAgent
```

Agent hosts PreparedStatement Object

Methods of the PreparedStatementAgent

[addBatch](#) on page 100

[execute](#) on page 100

[executeUpdate](#) on page 100

[setArray](#) on page 98

[setBigDecimal](#) on page 94

[setBlob](#) on page 99

[setByte](#) on page 92

[setCharacterStream](#) on page 98

[setDate](#) on page 94

[setDouble](#) on page 94

[setInt](#) on page 93

[setNull](#) on page 90

[setObject](#) on page 91

[setRef](#) on page 99

[setString](#) on page 96

[setTime](#) on page 95

[setTimestamp](#) on page 96

[clearParameters](#) on page 100

[executeQuery](#) on page 100

[sessionOpen](#) on page 131

[setAsciiStream](#) on page 97

[setBinaryStream](#) on page 97

[setBoolean](#) on page 92

[setBytes](#) on page 97

[setClob](#) on page 99

[setDate](#) on page 95

[setFloat](#) on page 93

[setLong](#) on page 93

[setObject](#) on page 90

[setObject](#) on page 91

[setShort](#) on page 92

[setTime](#) on page 95

[setTimestamp](#) on page 96

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

6.3 com.stc.eways.jdbcx.PreparedStatementResultSet Class

java.lang.Object

|

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

```
public abstract class PreparedStatementResultSet
extends java.lang.Object
```

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

Constructors of PreparedStatementResultSet

PreparedStatementResultSet

Methods of PreparedStatementResultSet

[absolute](#) on page 105

[beforeFirst](#) on page 107

[close](#) on page 105

[findColumn](#) on page 108

[getArray](#) on page 122

[getAsciiStream](#) on page 121

[getBigDecimal](#) on page 114

[getBinaryStream](#) on page 121

[getBlob](#) on page 123

[getBoolean](#) on page 110

[getByte](#) on page 111

[getBytes](#) on page 120

[getCharacterStream](#) on page 122

[getClob](#) on page 123

[getConcurrency](#) on page 103

[getDate](#) on page 115

[getDate](#) on page 116

[getDouble](#) on page 114

[getFetchDirection](#) on page 103

[getFloat](#) on page 113

[getInt](#) on page 112

[getLong](#) on page 112

[getMetaData](#) on page 103

[getObject](#) on page 109

[getObject](#) on page 110

[getRef](#) on page 125

[getShort](#) on page 111

[getString](#) on page 119

[getTime](#) on page 116

[getTime](#) on page 117

[getTimestamp](#) on page 118

[getTimestamp](#) on page 118

[getType](#) on page 108

[insertRow](#) on page 126

[isBeforeFirst](#) on page 107

[afterLast](#) on page 107

[clearWarnings](#) on page 125

[deleteRow](#) on page 126

[first](#) on page 106

[getArray](#) on page 122

[getAsciiStream](#) on page 121

[getBigDecimal](#) on page 115

[getBinaryStream](#) on page 122

[getBlob](#) on page 123

[getBoolean](#) on page 110

[getByte](#) on page 111

[getBytes](#) on page 120

[getCharacterStream](#) on page 122

[getClob](#) on page 124

[getCursorName](#) on page 105

[getDate](#) on page 115

[getDate](#) on page 116

[getDouble](#) on page 114

[getFetchSize](#) on page 104

[getFloat](#) on page 114

[getInt](#) on page 112

[getLong](#) on page 113

[getObject](#) on page 108

[getObject](#) on page 109

[getRef](#) on page 124

[getRow](#) on page 126

[getShort](#) on page 111

[getString](#) on page 119

[getTime](#) on page 116

[getTime](#) on page 117

[getTimestamp](#) on page 118

[getTimestamp](#) on page 119

[getWarnings](#) on page 125

[isAfterLast](#) on page 108

[isFirst](#) on page 106

[isLast](#) on page 107

[next](#) on page 105

[refreshRow](#) on page 147

[getFetchDirection](#) on page 103

[updateRow](#) on page 126

[last](#) on page 106

[previous](#) on page 105

[relative](#) on page 106

[getFetchSize](#) on page 104

[wasNull](#) on page 125

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 getByte(int index)
```

Name	Description
index	Column index.

Returns

Boolean value of the column.

Throws

java.sql.SQLException

getShort

Gets the short value of the specified column.

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

Name	Description
index	Column index.

Returns

Short value of the column.

Throws

java.sql.SQLException

getShort

Gets the short value of the specified column.

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

Name	Description
index	Column name.

Returns

Short value of the column.

Throws

java.sql.SQLException

getInt

Gets the integer value of the specified column.

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

Name	Description
index	Column index.

Returns

Int value of the column.

Throws

java.sql.SQLException

getInt

Gets the integer value of the specified column.

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

Name	Description
index	Column name.

Returns

Int value of the column.

Throws

java.sql.SQLException

getLong

Gets the long value of the specified column.

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


Name	Description
index	Column index.

Returns

Long value of the column.

Throws

java.sql.SQLException

getLong

Gets the long value of the specified column.

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

Name	Description
index	Column name.

Returns

Long value of the column.

Throws

java.sql.SQLException

getFloat

Gets the float value of the specified column.

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

Name	Description
index	Column index.

Returns

Float value of the column.

Throws

java.sql.SQLException

getFloat

Gets the float value of the specified column.

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

Name	Description
index	Column name.

Returns

Float value of the column.

Throws

java.sql.SQLException

getDouble

Gets the double value of the specified column.

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

Name	Description
index	Column index.

Returns

Double value of the column.

Throws

java.sql.SQLException

getBigDecimal

Gets the decimal value of the specified column.

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

Name	Description
index	Column index.

Returns

Big decimal value of the column.

Throws

java.sql.SQLException

getBigDecimal

Gets the decimal value of the specified column.

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

Name	Description
index	Column name.

Returns

Big decimal value of the column.

Throws

java.sql.SQLException

getDate

Gets the date value of the specified column.

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

Name	Description
index	Column index.

Returns

Date value of the column.

Throws

java.sql.SQLException

getDate

Gets the date value of the specified column.

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

Name	Description
index	Column name.

Returns

Date value of the column.

Throws

java.sql.SQLException

getDate

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

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

Name	Description
index	Column name.
calendar	Calendar to use.

Returns

Date value of the column.

Throws

java.sql.SQLException

getTime

Gets the time value of the specified column.

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

Name	Description
index	Column index.

Returns

Time value of the column.

Throws

java.sql.SQLException

getTime

Gets the time value of the specified column.

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

Name	Description
index	Column name.

Returns

Time value of the column.

Throws

java.sql.SQLException

getTime

Gets the time value of the specified column.

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

Name	Description
index	Column index.
calendar	Calendar to use.

Returns

Time value of the column.

Throws

java.sql.SQLException

getTime

Gets the time value of the specified column.

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

Name	Description
index	Column name.
calendar	Calendar to use.

Returns

Time value of the column.

Throws

java.sql.SQLException

getTimestamp

Gets the timestamp value of the specified column.

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

Name	Description
index	Column index.

Returns

The timestamp value of the column.

Throws

java.sql.SQLException

getTimestamp

Gets the timestamp value of the specified column.

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

Name	Description
index	Column name.

Returns

The timestamp value of the column.

Throws

java.sql.SQLException

getTimestamp

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

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

Name	Description
index	Column index.

Returns

The timestamp value of the column.

Throws

java.sql.SQLException

getTimestamp

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

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

Name	Description
index	Column name.
calendar	Calendar to use.

Returns

The timestamp value of the column.

Throws

java.sql.SQLException

getString

Gets the string value of the specified column.

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

Name	Description
index	Column index.

Returns

Returns the String value of the column.

Throws

java.sql.SQLException

getString

Gets the string value of the specified column.

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

Name	Description
index	Column name.

Returns

Returns the String value of the column.

Throws

java.sql.SQLException

getBytes

Gets the byte array value of the specified column.

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

Name	Description
index	Column index.

Returns

Byte array value of the column.

Throws

java.sql.SQLException

getBytes

Gets the byte array value of the specified column.

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

Name	Description
index	Column name.

Returns

Byte array value of the column.

Throws

java.sql.SQLException

getAsciiStream

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

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

Name	Description
index	Column index.

Returns

ASCII output stream value of the column.

Throws

java.sql.SQLException

getAsciiStream

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

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

Name	Description
index	Column name.

Returns

ASCII output stream value of the column.

Throws

java.sql.SQLException

getBinaryStream

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

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

Name	Description
index	Column index.

Returns

Binary out steam value of the column.

Throws

java.sql.SQLException

getBinaryStream

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

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

Name	Description
index	Column name.

Returns

Binary out steam value of the column.

Throws

java.sql.SQLException

getCharacterStream

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

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

Name	Description
index	Column index.

Returns

Reader for value in the column.

Throws

java.sql.SQLException

getArray

Gets the Array value of the specified column.

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

Name	Description
index	Column index.

Returns

Array value of the column.

Throws

java.sql.SQLException

getBlob

Gets the Blob value of the specified column.

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

Name	Description
index	Column index.

Returns

Blob value of the column.

Throws

java.sql.SQLException

getBlob

Gets the Blob value of the specified column.

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

Name	Description
index	Column name.

Returns

Blob value of the column.

Throws

java.sql.SQLException

getClob

Gets the Clob value of the specified column.

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

Name	Description
index	Column index.

Returns

Clob value of the column.

Throws

java.sql.SQLException

getClob

Gets the Clob value of the specified column.

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

Name	Description
index	Column name.

Returns

Clob value of the column.

Throws

java.sql.SQLException

getRef

Gets the Ref value of the specified column.

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

Name	Description
index	Column index.

Returns

Ref value of the column.

Throws

java.sql.SQLException

getRef

Gets the Ref value of the specified column.

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

Name	Description
index	Column name.

Returns

Ref value of the column.

Throws

java.sql.SQLException

wasNull

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

```
public boolean wasNull()
```

Returns

true if SQL NULL.

Throws

java.sql.SQLException

getWarnings

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

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

Returns

SQL warning.

Throws

java.sql.SQLException

clearWarnings

Clears any warnings reported on this object.

```
public void clearWarnings()
```

Throws

java.sql.SQLException

getRow

Retrieves the current row number in the result set.

```
public int getRow()
```

Returns

Current row number

Throws

java.sql.SQLException

refreshRow

Replaces the values 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

6.4 com.stc.eways.jdbcx.SqlStatementAgent Class

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

All Implemented Interfaces

ResetEventListener, SessionEventListener

```
public class SqlStatementAgent
extends StatementAgent
```

SQLStatement Agent that hosts a managed Statement object.

Constructors of the SqlStatementAgent

```
SqlStatementAgent
```

```
SqlStatementAgent
```

Methods of the SqlStatementAgent

[addBatch](#) on page 129

[execute](#) on page 128

[executeQuery](#) on page 128

[executeUpdate](#) on page 129

Constructor SqlStatementAgent

Creates new SQLStatementAgent with scroll direction TYPE_FORWARD_ONLY and concurrency CONCUR_READ_ONLY.

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

Name	Description
session	Connection session.

Constructor SqlStatementAgent

Creates a new SQLStatementAgent.

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

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

execute

Executes the specified SQL statement.

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

Name	Description
sSql	SQL statement.

Returns

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

Throws

java.sql.SQLException

executeQuery

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

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

Name	Description
sSql	SQL statement.

Returns

A ResultSetAgent or null

Throws

java.sql.SQLException

executeUpdate

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

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

Name	Description
sSql	SQL statement.

Returns

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

Throws

java.sql.SQLException

addBatch

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

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

Name	Description
sSql	SQL statement.

Throws

java.sql.SQLException

6.5 com.stc.eways.jdbcx.CallableStatementAgent Class

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

All Implemented Interfaces

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 140

[getBlob](#) on page 140

[getBytes](#) on page 134

[getClob](#) on page 140

[getDate](#) on page 137

[getFloat](#) on page 136

[getLong](#) on page 135

[getObject](#) on page 133

[getShort](#) on page 135

[getTime](#) on page 137

[getTimestamp](#) on page 139

[registerOutParameter](#) on page 132

[sessionOpen](#) on page 131

[getBigDecimal](#) on page 136

[getBoolean](#) on page 134

[getBytes](#) on page 139

[getDate](#) on page 137

[getDouble](#) on page 136

[getInt](#) on page 135

[getObject](#) on page 133

[getRef](#) on page 141

[getString](#) on page 139

[getTimestamp](#) on page 138

[registerOutParameter](#) on page 132

[registerOutParameter](#) on page 132

[wasNull](#) on page 133

Constructor CallableStatementAgent

Creates new CallableStatementAgent with scroll direction TYPE_FORWARD_ONLY and concurrency CONCUR_READ_ONLY.

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

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

Constructor CallableStatementAgent

Creates a new CallableStatementAgent.

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

Name	Description
session	Connection session.
iScroll	Ignored.
iConcur	Ignored

Constructor CallableStatement Agent

Creates a new CallableStatementAgent.

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

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

sessionOpen

Opens the session event handler.

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

Overrides

sessionOpen in class PreparedStatementAgent

Name	Description
evt	Session event.

registerOutParameter

Registers the indexed OUT parameter with specified type.

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

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

Name	Description
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.

Name	Description
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

6.6 com.stc.eways.jdbcx.TableResultSet Class

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

All Implemented Interfaces

ResetEventListener, SessionEventListener

```
public abstract class TableResultSet  
extends StatementAgent
```

ResultSet to map selected records of table in the database

Methods of the TableResultSet

[absolute](#) on page 143

[beforeFirst](#) on page 145

[deleteRow](#) on page 148

[first](#) on page 144

[getAsciiStream](#) on page 146

[getBinaryStream](#) on page 146

[getCharacterStream](#) on page 147

[isAfterLast](#) on page 145

[isFirst](#) on page 144

[last](#) on page 144

[moveToInsertRow](#) on page 148

[previous](#) on page 143

[rowDeleted](#) on page 149

[rowUpdated](#) on page 148

[updateRow](#) on page 147

[afterLast](#) on page 145

[cancelRowUpdates](#) on page 148

[findColumn](#) on page 146

[getAsciiStream](#) on page 146

[getBinaryStream](#) on page 146

[getCharacterStream](#) on page 147

[insertRow](#) on page 147

[isBeforeFirst](#) on page 145

[isLast](#) on page 144

[moveToCurrentRow](#) on page 148

[next](#) on page 142

[relative](#) on page 143

[rowInserted](#) on page 148

[select](#) on page 142

[wasNull](#) on page 149

select

Select table records.

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

Name	Description
sWhere	Where condition for the query.

Throws

java.sql.SQLException

next

Navigate one row forward.

```
public boolean next()
```

Returns

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

Throws

java.sql.SQLException

previous

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

```
public boolean previous()
```

Returns

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

Throws

java.sql.SQLException

absolute

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

Name	Description
row	An integer other than 0.

Returns

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

Throws

java.sql.SQLException

relative

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

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

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

Returns

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

Throws

java.sql.SQLException

first

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

```
public boolean first()
```

Returns

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

Throws

java.sql.SQLException

isFirst

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

```
public boolean isFirst()
```

Returns

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

Throws

java.sql.SQLException

last

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

```
public boolean last()
```

Returns

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

Throws

java.sql.SQLException

isLast

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


```
public boolean isLast()
```

Returns

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

Throws

java.sql.SQLException

beforeFirst

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

```
public void beforeFirst()
```

Throws

java.sql.SQLException

isBeforeFirst

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

```
public boolean isBeforeFirst()
```

Returns

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

Throws

java.sql.SQLException

afterLast

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

```
public void afterLast()
```

Throws

java.sql.SQLException

isAfterLast

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

```
public boolean isAfterLast()
```

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

Throws

java.sql.SQLException

findColumn

Finds the index of the named column.

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

Throws

java.sql.SQLException

getAsciiStream

Returns the column data as an AsciiStream.

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

Throws

java.sql.SQLException

getAsciiStream

Returns the column data as an AsciiStream.

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

Throws

java.sql.SQLException

getBinaryStream

Returns the column data as BinaryStream.

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

Throws

java.sql.SQLException

getBinaryStream

Returns the column data as BinaryStream.

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

Throws

java.sql.SQLException

getCharacterStream

Returns the column data as CharacterStream.

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

Throws

java.sql.SQLException

getCharacterStream

Returns the column data as CharacterStream.

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

Throws

java.sql.SQLException

refreshRow

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

```
public void refreshRow()
```

Throws

java.sql.SQLException

insertRow

Inserts the contents of the current row into the database.

```
public void insertRow()
```

Throws

java.sql.SQLException

updateRow

Updates the contents of the current row into the database.

```
public void updateRow()
```

Throws

java.sql.SQLException

deleteRow

Deletes the contents of the current row from the database.

```
public void deleteRow()
```

Throws

```
java.sql.SQLException
```

moveToInsertRow

Moves the current position to a new insert row.

```
public void moveToInsertRow()
```

Throws

```
java.sql.SQLException
```

moveToCurrentRow

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

```
public void moveToCurrentRow()
```

Throws

```
java.sql.SQLException
```

cancelRowUpdates

Cancels any updates made to this row.

```
public void cancelRowUpdates()
```

Throws

```
java.sql.SQLException
```

rowInserted

Returns true if the current row has been inserted.

```
public boolean rowInserted()
```

Throws

```
java.sql.SQLException
```

rowUpdated

Returns true if the current row has been updated.

```
public boolean rowUpdated()
```

Throws

java.sql.SQLException

rowDeleted

Returns true if the current row has been deleted.

```
public boolean rowDeleted()
```

Throws

java.sql.SQLException

wasNull

Returns true if the last data retrieved is NULL.

```
public boolean wasNull()
```

Throws

java.sql.SQLException

6.7 \$DB Configuration Node Methods

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

These methods are contained in the following classes:

- [com_stc_jdbcx_db2cfg.DataSource](#) on page 149
- [com_stc_jdbcx_db2cfg](#) on page 159

6.8 com_stc_jdbcx_db2cfg.DataSource

Java.lang.Object

|

+ - - com_stc_jdbcx_db2cfg.Com_stc_jdbcx_db2cfg.DataSource

Direct Known Subclasses

```
public class Com_stc_jdbcx_db2cfg.DataSource
extends java.lang.Object
```

Methods of the db2cfg.DataSource

getClass on page 150	getCollectionID on page 152
getConnectionMethod on page 151	getDatabaseName on page 153
getLocationName on page 154	getPackageName on page 154
getPackageName on page 154	getPassword on page 156
getPortNumber on page 157	getServerName on page 158
getTimeout on page 158	getUserName on page 155
hasClass on page 151	hasCollectionID on page 152
hasConnectionMethod on page 152	hasDatabaseName on page 153
hasLocationName on page 154	hasPackageName on page 155
hasPassword on page 156	hasPortNumber on page 157
hasServerName on page 158	hasTimeout on page 159
hasUserName on page 155	omitClass on page 151
omitCollectionID on page 153	omitConnectionMethod on page 152
omitDatabaseName on page 153	omitLocationName on page 154
omitPackageName on page 155	omitPassword on page 157
omitPortNumber on page 157	omitServerName on page 158
omitTimeout on page 159	omitUserName on page 156
setClass on page 150	setCollectionID on page 152
setConnectionMethod on page 151	setDatabaseName on page 153
setLocationName on page 154	setPackageName on page 155
setPassword_AsIs on page 156	setPassword on page 156
setPortNumber on page 157	setServerName on page 158
setTimeout on page 158	setUserName on page 155

getClass

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

```
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".

```
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.

getCollectionID

Retrieves the connection ID.

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

Returns

java.lang.String.

setCollectionID

Sets the collection (group of packages) to which the package is bound.

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

Returns

None.

hasCollectionID

Returns true if the collection ID has been set.

```
public boolean hasCollecitonID()
```

Returns

True.

omitCollectionID

Sets the collection ID to null.

```
public void omitCollectionID()
```

Returns

None.

getDatabaseName

Retrieves the database name.

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

Returns

java.lang.String

setDatabaseName

The alias name of the database which you want to connect to. Normally, this would be used with UDB.

```
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.

getLocationName

Retrieves the name of the DB2 location that you want to access. Used with a z/OS operating system.

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

Returns

java.lang.String.

setLocationName

Sets the name of the DB2 location that you want to access. Used with a z/OS operating system.

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

Returns

None.

hasLocationName

Returns true if the location name has been set.

```
public boolean hasLocationName()
```

Returns

True.

omitLocationName

Sets the location name to null.

```
public void omitLocationName()
```

Returns

None.

getPackageName

Retrieves the name of the package that the driver uses to process static and dynamic SQL.

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

Returns

java.lang.String

setPackageName

Set the name (7-character limit) of the package that the driver uses to process static and dynamic SQL.

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

Return

None.

hasPackageName

Returns true if the Package Name has been set.

```
public boolean hasPackageName()
```

Returns

True.

omitPackageName

Sets the Package Name to null.

```
public void omitPackageName()
```

Returns

None.

getUserName

Retrieves the case-insensitive user name used to connect to your DB2 database.

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

Returns

java.lang.String.

setUserName

Sets the case-insensitive user name used to connect to your DB2 database.

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

Returns

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 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.

getPortNumber

Retrieves the TCP port (use for DataSource connections only).

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

Returns

java.lang.String.

setPortNumber

Sets the TCP port (use for DataSource connections only).

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

Returns

None.

hasPortNumber

Returns true if the port number has been set.

```
public boolean hasPortNumber()
```

Returns

True.

omitPortNumber

Sets the port number to null.

```
public void omitPortNumber()
```

Returns

None.

getServerName

Retrieves the IP address (use for DataSource connections only).

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

Returns

java.lang.String.

setServerName

Sets the IP address (use for DataSource connections only).

```
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.

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.

6.9 com_stc_jdbcx_db2cfg

```
com_stc_jdbcx_db2cfg.Com_stc_jdbcx_db2cfg
```

Direct Known Subclasses

```
public class Com_stc_jdbcx_db2cfg
```

Methods of the db2cfg

getSource on page 159

setDataSource on page 159

getSource

Retrieves the DataSource object.

```
public Com stc jdbcx db2cfg.DataSource getSource()
```

Returns

None.

setDataSource

Sets the DataSource object.

```
public void setDataSource (Com stc jdbcx db2cfg.DataSource val)
```

Returns

None.

Appendix

A.1 ADO/ODBC Calls

These calls are made by the DBWizard for metadata discovery.

A.1.1 ADO Calls

OpenSchema(adSchemaTables)	Retrieves the list of tables.
OpenSchema(adSchemaColumns)	Retrieves the column information for a table.
OpenSchema(adSchemaProcedures)	Retrieves the list of stored procedures.
OpenSchema(adSchemaProcedureParameters)	Retrieves the parameter information of a stored procedure.

A.1.2 ODBC Calls

SQLDataSources	Retrieves the list of DSNs.
SQLPrepared	Used to prepare a SQL prepared statement string.
SQLNumParams	Retrieves the number of parameters in a prepared statement.
SQLDescribeParam	Retrieves information about a prepared statement parameter.
SQLNumResultCols	Retrieves the number of resultset columns in a prepared statement.
SQLDescribeCol	Retrieves information about a resultset column in a prepared statement.

Index

B

Batch Operations 62

C

class parameter
 Connector settings 32
 Data Source settings 28
 Collaboration Service Java 38–39
 component relationship 39
 components, Java-enabled 39
 configuration file sections
 Connector settings 32
 configuration parameters
 class 28, 32
 DatabaseName 30
 password 30
 user name 30
 configuration steps, schema 64
 configuring e*Way connections 27–32
 connection establishment mode 33
 connection inactivity timeout 33
 Connection Manager 34
 connection types, JDBC 32
 connection verification interval 34
 connector objects, JDBC 32
 Connector settings 32
 creating e*Way connections 27

D

Database Configuration Node 62
 DatabaseName parameter 30
 DataDirect Drivers
 GUI Host 11
 Participating Host 11
 DataSource Settings 28
 db.ctl 16
 DB2.def 16
 driver class, JDBC 28

E

e*Way connections

 configuring 27–32
 creating 27
 executeBusinessRules() 39

H

host system requirements 11

I

implementation, sample 62–77

J

Java Collaboration Service 38–39
 Java ETD Builder 39–40
 Java JDK 12
 Java requirements 12
 Java-enabled components 39
 JDBC 40
 connection types 32
 connector objects 32
 driver class 28

M

Mixing XA-Compliant and XA-Noncompliant e*Way Connections 33

P

password parameter 30
 Prepared Statement 61

R

requirements
 host system 11
 Java version 12

S

sample schema 62–77
 scenario, sample 62–77
 schema configuration steps 64
 stcewjdbcx.ctl 16
 stcjdbcx.jar 10
 System Requirements 11

T

The Stored Procedure 56
 transaction mode 32

Index

type 32

U

UNIX 15

user name parameter 30

userInitialize() 39

userTerminate() 39