SeeBeyond ICAN Suite

e*Way Intelligent Adapter for IAM User's Guide

Release 5.0.5 for Schema Run-time Environment (SRE)

Version 20050406092803.

Contents

Chapter 1

Introduction

This guide provides instructions for installing and configuring the SeeBeyond™ Technology Corporation's (SeeBeyond™) e*Way™ Intelligent Adapter for IAM. This chapter provides an introduction to the e*Way and this guide.

1.1 IAM e*Way: Overview

The IAM e*Way enables e*Gate Integrator to exchange data with an external Innovation Access Method (IAM) mainframe file system.

1.1.1 IAM System

IAM is a file management system used on IBM mainframe computers. IAM speeds up access to data in files by using its own index of all records, added to each file. Many legacy software systems use IAM to implement database systems, called *data sets*.

IAM is a transparent alternative to and is designed to coexist with the mainframe Virtual Storage Access Method (VSAM) system. IAM does not require you to make any modifications to your operating system and does not replace any operating system access methods.

IAM provides the following additional features:

- Processes large file sizes
- Compresses data
- Supports Entry-sequenced Data Set (ESDS) and Key-sequenced Data Set (KSDS) files
- Provides compatibility with IBM's System Managed Storage (SMS)

The IAM e*Way allows you to utilize and extend all of these features. For more information on IAM, see the following Web site:

http://www.fdr.com/iam.cfm

1.1.2 IAM e*Way

This e*Way is similar to other database e*Ways, such as VSAM, Oracle, DB2/UDB, SQL Server, and Sybase. It uses e*Way Connections and Java Collaborations to enable IAM data integration.

This e*Way enables e*Gate Integrator running on Windows and UNIX systems to communicate with IAM files on an OS/390 host system via TCP/IP. The e*Way utilizes a CrossAccess Corp. software component that allows you to view and access IAM in the same way as a standard relational database system.

Note: *See the "System Requirements" sections at the end of this chapter and* **"Complete Installation: Overview" on page 15** *for more information on the e*Way's CrossAccess software requirements.*

1.2 Intended Reader

The reader of this guide is presumed:

- To be a developer or system administrator with the responsibility of maintaining the e*Gate system
- To have expert-level knowledge of Windows or UNIX operations and administration
- To be thoroughly familiar with SQL functions and IAM operation
- To be thoroughly familiar with Windows-style operations

1.3 e*Way General Operation

The IAM e*Way uses Java Collaborations to interact with one or more external IAM file systems. By using the Java Collaboration Service, it is possible for e*Gate components to connect to external IAM data and execute e*Gate Business Rules written entirely in Java.

1.3.1 Technical Overview

The CrossAccess eXadas Data Integrator (also called the IAM Server or Legacy Server) runs as a batch job or started task on the OS/390 system and requires a TCP/IP port for it to listen on connections from the e*Ways on Windows and UNIX systems. The number of concurrent and maximum users is configurable.

In addition, the software can be configured to support multiple users in a single address space, or you can distribute the load across multiple address spaces. The IAM Server is a Type 3 driver.

The following IAM features are supported:

- KSDS and ESDS files
- Variable-length and fixed-length IAM files
- Reading sequentially
- Reading randomly, that is, keyed read on primary and alternate indexes
- IAM files under the control of CICS
- Mainframe security handled through SAF, therefore supporting RACF, ACF2, and so on
- Simple mapping from simple representations of hierarchical data structures, repeating groups of data nested within other repeating groups, and records whose layout depends on record-type identifiers to relational structures, making it simple to navigate
- Multi-user write access

1.3.2 Components

The IAM e*Way is made up of the following components:

- **e*Way Connections**: These components provide access to the information necessary for connecting to a specified collection of IAM files.
- **Java client-side drivers**: CrossAccess drivers enable the e*Way to connect out from the Windows or UNIX environment to the OS/390 system where the IAM files reside.
- **eXadas Data Integrator**: You can install and configure this CrossAccess product to access IAM files on the OS/390 host system (Type 3 driver).
- **CrossAccess ODBC driver**: The Database Builder wizard utilizes this driver (file name **caoem32.dll**) to connect to the OS/390 server and obtain IAM metadata for the creation of e*Gate Event Type Definitions (ETDs).
- **CrossAccess JDBC driver**: This file (**cacjdbcoem21.jar**) contains the logic required by the e*Way to interact with external IAM data (required to connect to the external IAM files if you are installing the IAM e*Way in a Windows environment).
- **eXadas DataMapper**: This CrossAccess product enables the mapping of target data files.

A complete list of installed files appears in **Table 1 on page 22**.

1.4 Supported Operating Systems

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

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

Note: *This e*Way connects to OS/390 systems.*

1.5 System Requirements

To use the IAM e*Way, you need:

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

Although the IAM e*Way components can run on the supported operating systems listed in this section, the e*Gate Schema Designer (including the ETD Editor and Collaboration Rules Editor) requires a Windows system.

1.5.1 Additional Windows Requirements

To enable the e*Gate Schema Designer's editors to communicate with the IAM file system, the following CrossAccess, version 2.25, items must be installed on any host machines running the editors:

- eXadas DataMapper
- ODBC driver
- ODBC data source
- JDBC driver

Note: *During the e*Way's installation, wizards for these applications appear in succession, allowing you to install them in the proper order. See* **Chapter 2** *for details including installation serial numbers.*

1.5.2 Additional UNIX Requirements

If you are installing the IAM e*Way on a UNIX system, you need:

- CrossAccess JDBC driver, version 2.25

All of these eXadas software products are also available from the CrossAccess Corp. The company's URL is:

http://www.crossaccess.com

1.6 External System Requirements

This section lists the e*Way's external system requirements.

OS/390 Requirements

- CrossAccess eXadas Data Integrator, version 2.25, installed via a 3480 cartridge tape
- The serial number for the installation, shipped with the tape

Note: *The eXadas Data Integrator is shipped with the e*Way, but you must install it separately.*

- OS/390 version 2.9 or 2.10

Chapter 2

Installation

This chapter explains how to install the e*Way Intelligent Adapter for IAM and its related components.

2.1 Complete Installation: Overview

You must install the IAM e*Way and its components in the following order:

1 CrossAccess eXadas Data Integrator

Note: *The eXadas Data Integrator is shipped with the e*Way, but you must install it separately.*

2 IAM e*Way

The following CrossAccess components are installed when you install the e*Way:

- eXadas DataMapper
- JDBC driver
- ODBC driver

During the e*Way's installation, wizards for these applications appear in succession, allowing you to install them in the proper order.

Note: *Once the installation operation is complete, you may want to pass a small amount of data to ensure you have a working connection.*

Installing eXadas Data Integrator

For complete details on how to install eXadas Data Integrator, see the CrossAccess *eXadas Data Integrator OS/390 Getting Started* manual, the chapter that details OS/390 installation. All eXadas software products, as well as additional information on each product, are available from the CrossAccess Corp. The company's URL is:

http://www.crossaccess.com

Important: *When you install eXadas Data Integrator, be sure to configure it for IAM. See the appropriate CrossAccess documentation for details.*

2.2 Installing eXadas DataMapper

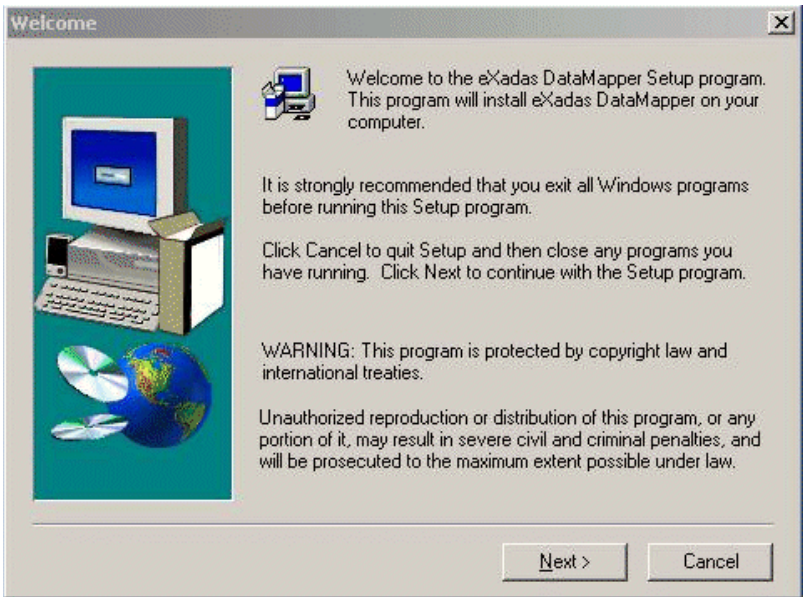
This section describes installing eXadas DataMapper for Windows systems. This application must be installed on a Windows system.

To install eXadas DataMapper

1. Install the e*Way, as explained under **"Installing the e*Way on Windows" on page 20**.
2. During the e*Way installation process, you are prompted to install eXadas DataMapper. Click **OK**.

   The **Welcome** dialog box appears (see Figure 1).

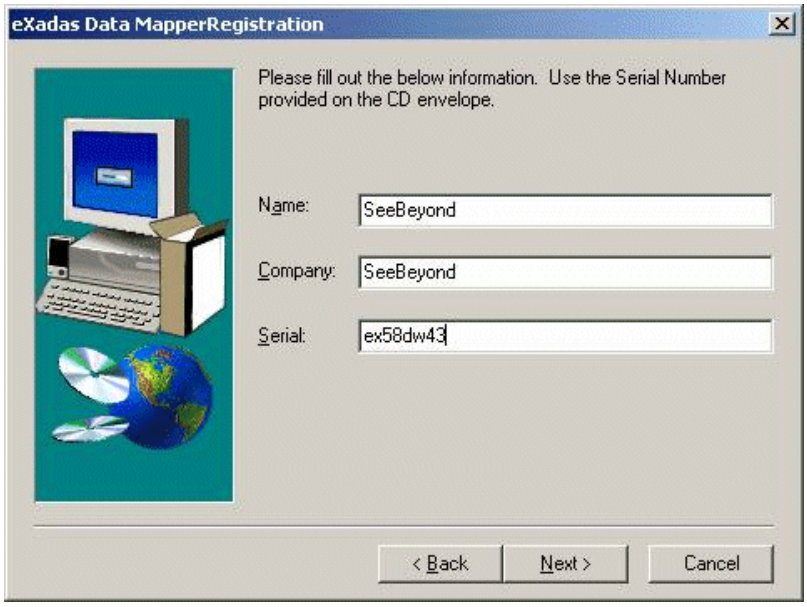
Figure 1 DataMapper Welcome Dialog Box



3. Click **Next**.

The **eXadas DataMapper Registration** dialog box appears (see Figure 2).

Figure 2 eXadas DataMapper Registration Dialog Box



4 Register the eXadas DataMapper as shown in the previous figure. You must enter the following serial number:

ex58dw43

5 Click **Next**.

6 A dialog box appears asking you to confirm the information you just entered. Click **Yes**.

7 Follow the instructions on each wizard dialog box to complete the installation. When you are done with the final wizard, click **Finish** to complete the e*Way installation.

The installation program returns you to the process explained under **"Installing the e*Way on Windows" on page 20**.

The eXadas DataMapper installation has the following optional sets of files:

- **Program Files**: The executable files and run-time system for running DataMapper.
- **Help Files**: Make up the online Help system for DataMapper.
- **Sample Files**: Including sample COBOL Copybooks and IMS/DL/I DBDs and IDMS schemas and sub-schemas for testing DataMapper's features. CrossAccess recommends installing all three sets of files. The minimum requirement is installing the program files in option 1.

When installation is complete, a program group named eXadas is created in Windows. The group contains the program items for DataMapper, the DataMapper Help system, and the DataMapper **Readme.txt** file. To start DataMapper, double-click the mouse on the DataMapper icon in the eXadas program group.

Refer to the eXadas DataMapper documentation located on the e*Gate installation CD-ROM for further instructions.

2.3 JDBC and ODBC Drivers

The standard e*Gate and IAM e*Way installations automatically installs these drivers for you.

JDBC Driver

There are no additional steps needed to install and configure the CrossAccess JDBC driver. It is installed as a part of e*Gate.

ODBC Driver

This section explains steps you must take to install and configure the CrossAccess ODBC driver.

2.3.1 ODBC Driver Installation

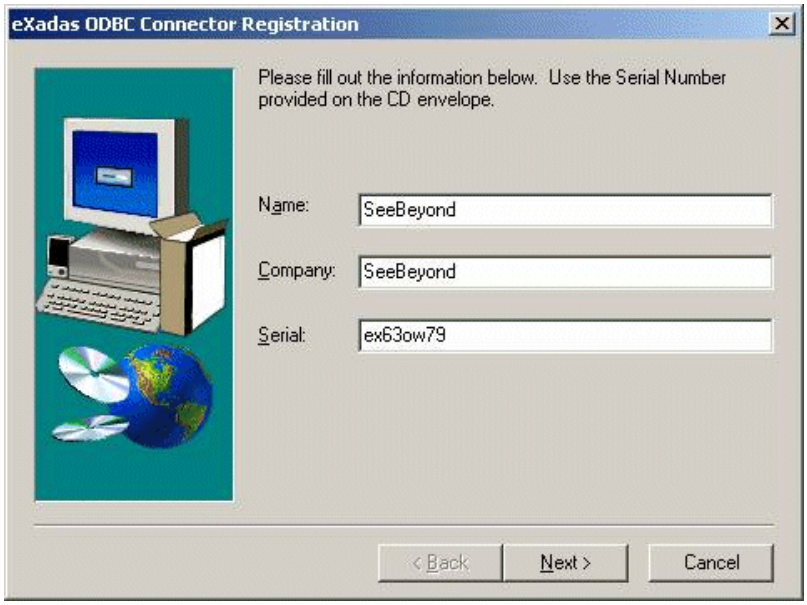
You must take extra steps while installing the e*Way, to install the ODBC driver. This driver is necessary for the correct operation of the e*Way. This section explains these extra steps.

To install the ODBC driver

1. Install the e*Way, as explained under **"Installing the e*Way on Windows" on page 20**.
2. During the e*Way installation process, you are prompted to install the ODBC driver. Click **OK**.

   The **eXadas ODBC Connector Registration** dialog box appears (see **Figure 3 on page 19**).

Figure 3 ODBC Connector Registration Dialog Box



3 Register the eXadas ODBC connector as shown in the previous figure. You must enter the following serial number:

ex63ow79

4 Click **Next**.

5 A dialog box appears asking you to confirm the information you just entered. Click **Yes**.

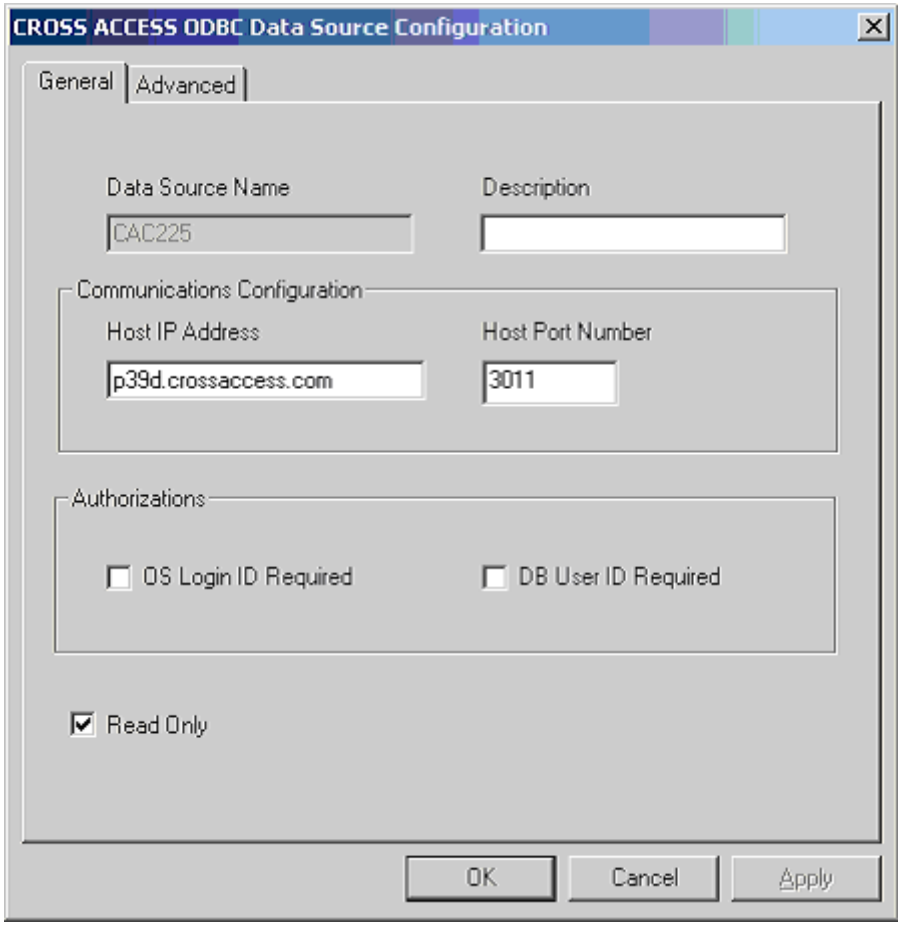
6 Follow the instructions on each wizard dialog box to complete the installation. When you are done with the final wizard, click **Finish** to complete the e*Way installation.

The installation program returns you to the process explained under **"Installing the e*Way on Windows" on page 20**.

2.3.2 ODBC Driver Configuration

You must configure the CrossAccess ODBC driver after it is installed. Set this configuration using this CrossAccess application (see the CrossAccess documentation for details). **Figure 4 on page 20** shows the values you must use to configure this driver for the e*Way.

Figure 4 CrossAccess ODBC Driver Configuration



2.4 Installing the e*Way on Windows

This section guides you through installing the IAM e*Way on a Windows system.

2.4.1 Pre-installation

Before beginning the installation process, you must:

- Exit all Windows programs before running the setup program, including any anti-virus applications.
- Ensure that you have Administrator privileges before installing this e*Way.

2.4.2 Installation Procedure

To install the e*Way on a Windows 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 the 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 launches. Follow the on-screen instructions to install the e*Way.

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

Caution: *Unless you are directed to do so by SeeBeyond support personnel, do not change the suggested installation directory setting.*

2.5 Installing the e*Way on UNIX

This section guides you through installing the IAM e*Way on a UNIX system.

2.5.1 Pre-installation

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

2.5.2 Installation Procedure

To install the 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 appears. Select the **Install e*Way** option. Then follow any additional on-screen directions.

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

Caution: *Unless you are directed to do so by SeeBeyond support personnel, do not change the suggested installation directory setting.*

2.6 Files/Directories Created by Installation

The IAM e*Way installation process installs the files shown in Table 1 within the e*Gate directory tree.

Note: *If you have any pre-existing ETDs, you must recompile them.*

The files are installed within the **eGate\client** tree on the Participating Host and committed to the "default" schema on the Registry Host.

Table 1 Files/Directories Created by Installation

Directories	Files
classes\	stcjdbcx.jar
configs\iam\	iam.def
etd\	iam.ctl
etd\iam\	Com_stc_jdbcx_legacy.java Com_stc_jdbcx_legacy.xsc
ThirdParty\cac\classes\	cacjdbcoem21.jar
ThirdParty\sun\classes\	jdbc2_0-stdext.jar

Chapter 3

e*Way Connection Configuration

This chapter explains how to configure the e*Way Intelligent Adapter for IAM e*Way Connections.

3.1 e*Way Connections: Overview

The e*Way Connection is the e*Gate Integrator component that allows an e*Way to connect to another system. Each e*Way Connection has parameters whose values you must set to define the desired connection. The e*Way Connection parameters for the IAM e*Way are specialized for that e*Way.

Note: *For a complete explanation of e*Way Connections and how they operate in the overall e*Gate system, see the* ***e*Gate Integrator User's Guide****.*

This chapter explains how to create, configure, and manage IAM e*Way Connections under the following sections:

- **"Creating e*Way Connections" on page 23**: Explains how to create e*Way Connections using the e*Gate Schema Designer.
- **"Configuring e*Way Connections" on page 26**: Explains how to configure the IAM e*Way Connection parameter settings.
- **"Connection Manager" on page 28**: Describes and explains how to use the e*Gate Connection Manager feature.

3.2 Creating e*Way Connections

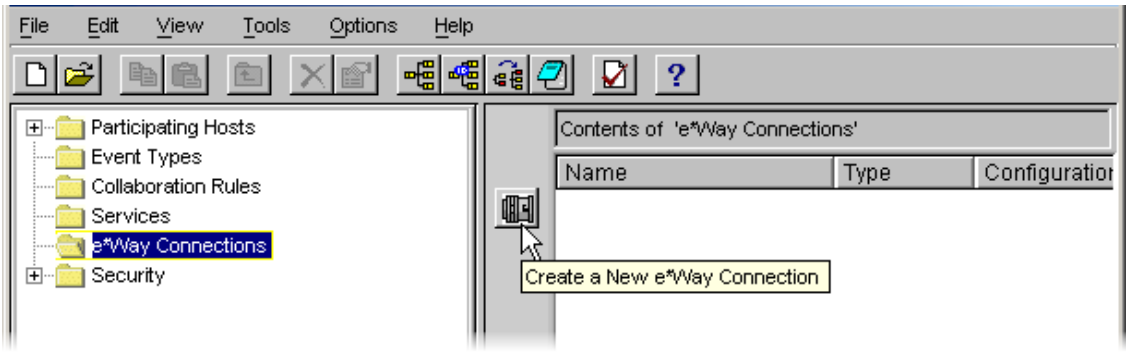
The e*Way Connections are created and configured using the e*Gate Schema Designer. This section lists the *required* steps for this process.

Note: *For complete information on how to create and configure e*Way Connections using the Schema Designer, see the* ***e*Gate Integrator User's Guide****.*

To create and configure the e*Way Connections

1 In the Schema Designer's Component mode, select the **e*Way Connections** folder (see Figure 5).

Figure 5 Schema Designer: e*Way Connections Folder

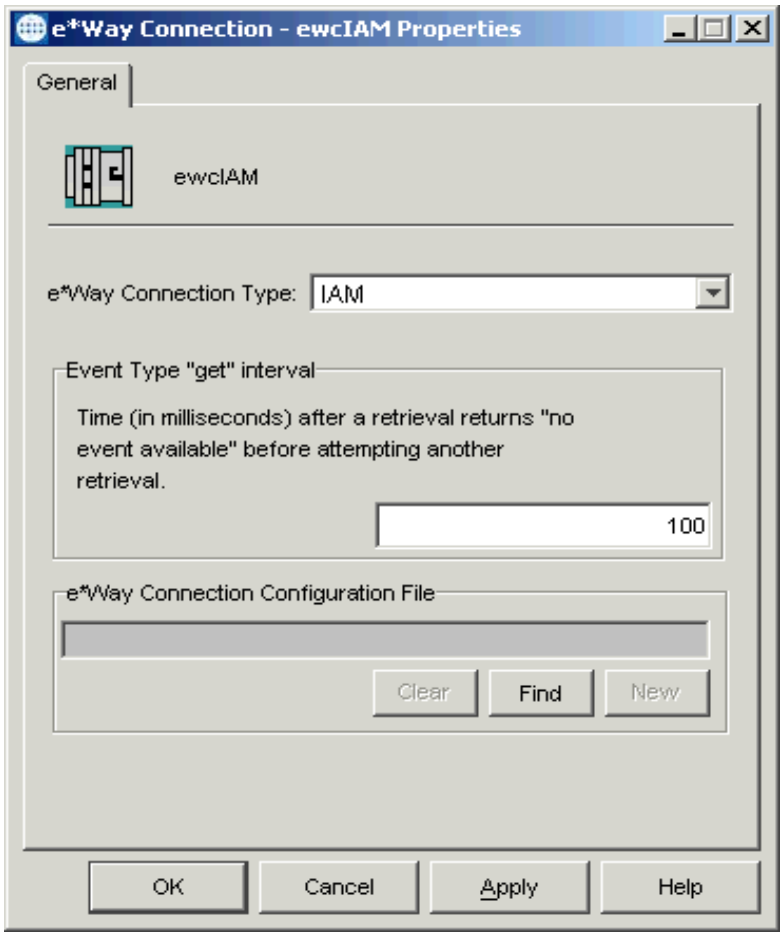


2 On the Palette, click the **Create a New e*Way Connection** icon.

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

4 Double-click the new e*Way Connection to open the **e*Way Connection Properties** dialog box (see **Figure 6 on page 25**).

Figure 6 e*Way Connection Properties Dialog Box



5 From the **e*Way Connection Type** drop-down box, select **IAM**.

Note: *You can enter the* ***Event Type "get" interval*** *in the text box provided (optional).*

6 Click **New** to create a new e*Way Connection configuration (**.cfg**) file.

The e*Way Configuration Editor interface appears.

Use this Editor tool to configure the e*Way's parameter settings. The e*Way Configuration Editor allows you to create and save a configuration (**.cfg**) file for the current e*Way Connection. This file stores your configuration settings and can be edited again at any time using this Editor tool.

3.3 Configuring e*Way Connections

In the e*Way Configuration Editor, the IAM e*Way Connection's configuration parameters are organized into the following sections for each group of settings:

- "DataSource Settings" on page 26
- "Connector Settings" on page 27

The rest of this section explains these parameters and how to configure them.

3.3.1 DataSource Settings

These settings define the parameters used to interact with the external database.

Driver

Description

Specifies the Java class in the JDBC driver.

Required Values

A valid driver.

The default is **com.cac.jdbc.driver**.

JDBC URL

Description

This is the JDBC URL necessary to gain access to the IAM database. Typically, the IAM JDBC URL looks like the following example:

jdbc:cac:<Datasource>:tcp/<host>/<port>

Where **<host>** is the IP address or node name of the eXadas Legacy server, **<port>** is the port number of the eXadas Legacy server, and **<Datasource>** is the name of the data source on the eXadas Legacy server.

Required Values

A valid JDBC URL.

User Name

Description

This is the case-insensitive user name used to connect to your IAM files.

Required Values

Any valid string.

Password

Description

This is the password the e*Way uses to connect to the database.

Required Values

A valid string encryption.

3.3.2 Connector Settings

These settings define the high-level characteristics of the e*Way Connection.

type

Description

Specifies the type of e*Way Connection. The only currently available type for connection is **DB**.

Required Values

The default is **DB**.

class

Description

Specifies the class name of the JDBC connector object.

Required Values

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

transaction mode

Specifies how a transaction is to be handled:

- **Automatic**: e*Gate takes care of transaction control and users do not issue a commit or rollback.
- **Manual**: You must manually take care of transaction control by issuing a commit or rollback command.

Required Values

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

connection inactivity timeout

Specifies the time-out for the **Automatic** connection establishment mode. If this parameter is not set or is set to 0, the connection is not brought down due to inactivity. The connection is always kept alive; if it goes down, re-establishing the connection is automatically attempted.

If a non-zero value is specified, the Connection Manager tries to monitor for inactivity so that the connection is brought down if the time-out specified is reached.

Required Values

Any valid string.

connection verification interval

Specifies the minimum period of time (in milliseconds) between checks for connection status to the database server. If the connection to the server is detected to be down during verification, the Collaboration's **onConnectionDown()** method is called.

If the connection comes up from a previous connection error, the Collaboration's **onConnectionUp()** method is called.

If no value is specified, 60000 ms is used.

Required Values

Any valid string.

3.4 Connection Manager

The e*Gate Connection Manager allows you to define external connection functionality of an e*Way. You can choose:

- When to make a connection
- When to close a connection and disconnect
- What the status of a connection is
- When a connection is established, an **onConnectionUp()** method is called by the Collaboration
- When a connection fails, an **onConnectionDown()** method is called by the Collaboration

3.4.1 Using the Connection Manager

The Connection Manager is controlled in the IAM e*Way's configuration as explained under **"Connector Settings" on page 27**.

If you choose to manually control the connections, Table 2 provides additional information on this option.

Table 2 e*Way Connection Control Settings

Method or Action	Automatic	On-demand	Manual
onConnectionUp()	Yes	No	No
onConnectionDown()	Yes	Yes only if the connection attempt fails	No
Manual Transaction	Yes	No	No
connect()	No	No	Yes
isConnected()	No	No	Yes
disconnect()	No	No	Yes
Timeout or connect()	Yes	Yes	No
Verify connection interval	Yes	No	No

3.4.2 Controlling Connection Timing and Status

This section explains how you can control when a connection is made and when it is disconnected. You can also control connectivity status.

When a Connection Is Made

You can control when a connection is made. Using the **Connector** parameters, you can choose to have an e*Way's connections controlled manually, through the Collaboration, or automatically, through the e*Way Connection's configuration.

If you choose to control a connection, you can specify:

- 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 (called per connection if your ETD has multiple connections)

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's 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 Collaboration's Business Rules
- To disconnect during a time-out
- To disconnect after a method call

Connectivity Status

You can control how often an e*Way Connection checks to verify whether its external connection is still alive. You can also set how often it checks. See the following references:

- **"Connector Settings" on page 27** for how to set connection control-related configuration parameters
- **Table 2 on page 29** for a list of connection control settings

Chapter 4

Implementation

This chapter explains how to implement the e*Way Intelligent Adapter for IAM in a production environment, including how to use the Database Builder wizard. Also included is a sample schema.

4.1 Implementation: Overview

To implement the IAM e*Way, you must first create and configure an e*Gate Integrator schema that can use the e*Way. Also, you must create Java-enabled e*Gate Collaboration and Event Type Definition (ETD) components in the schema, to transport and (if necessary) transform your data.

The e*Gate system contains tools that help you do these tasks, including a Database Builder wizard that generates Java-enabled ETDs corresponding to IAM database external tables and prepared statements.

This chapter contains the following sections:

- **"Java Collaboration Service" on page 32**: Provides a general overview of the Java-enabled operation of the e*Way.
- **"Database Builder Wizard" on page 34**: Explains how to use the e*Gate Schema Designer's Database Builder wizard to generate a IAM-based ETD.
- **"Using Database ETDs" on page 46**: Explains the types of ETDs used with the e*Way within the Collaboration Rules Editor, including the structure and make-up of each ETD.
- **"Sample Schema: Polling from a Database" on page 51**: Tells you how to create and configure the sample IAM e*Way schema on the CD-ROM installation disk.

4.2 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 e*Gate Schema Designer's Collaboration Rules Editor, you can create Java classes that utilize the **executeBusinessRules()**, **userTerminate()**, and **userInitialize()** methods.

For complete details, see the following references:

- 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 Schema Designer's ETD Editor and Collaboration Rules Editor, see the *e*Gate Integrator User's Guide*.

4.2.1 Java Components

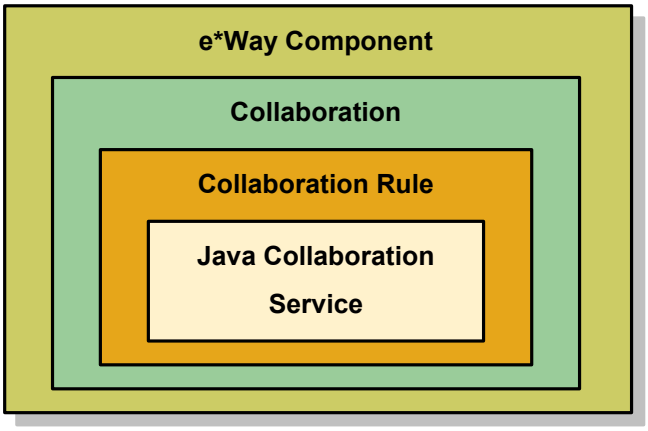
To make an e*Gate component Java-enabled, the component's Collaboration Rule must use the Java Collaboration Service. This requires all the intermediate components to also be configured correctly, since there is not a direct relationship between an e*Way and a Collaboration Service.

Before working with the Collaborations, you need to import the Java package **com.cac.***. To do this action, go to the e*Gate Schema Designer and click **Tools**, **Import**, then select the following file from the drop-down list:

ThirdParty\cac\classes\cacjdbcoem21.jar

Figure 7 illustrates the relationship between a higher-level e*Gate component and a Collaboration Service.

Figure 7 Component Relationship



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

4.2.2 ETD Overview

Each ETD serves the following important purpose and function in the e*Gate system:

- An ETD is a structural representation of an Event (package of data). It is an actual blueprint of data to be processed in e*Gate. The e*Gate run-time components use ETDs to parse, validate, and (if necessary) transform Events.
- Event Types and ETDs contain the instructions which e*Gate uses to identify Events. You build ETDs on individual Event Type specifications, and they become the foundation of e*Gate data processing.
- A major advantage of ETDs is their reusability. If there are formats that recur in many of your Events, you can create definitions for those formats and use them as templates in other ETDs.

An ETD is a graphical representation of data made up of sub-components called *nodes*. You need to determine which Events are used within e*Gate to process data. You build a specific ETD for each and all Events that pass through the e*Gate environment.

Each ETD is the skeleton or blueprint of the type of data needed to be parsed, and describes to the system how to locate data within an Event. These parsing instructions are stored in an ETD.

4.2.3 Parts of the ETD

There are four types of nodes in a Java-enabled ETD as shown in Figure 8.

Figure 8 Java-enabled ETD

These node types are:

- **Elements**: The highest level in the ETD tree (also called the *root node*). The element is the basic container that holds the other parts of the ETD and can contain fields and methods.
- **Fields**: Used to represent data. A field can contain data in any of the following Java formats: string, Boolean, integer, double, or float.
- **Methods**: Represent actual Java methods.
- **Parameters**: Represent a Java method's parameters.

Each Java-based ETD is stored in a file with the extension .**xsc**.

4.3 Database Builder Wizard

You can use the Database Builder wizard to generate a Java-enabled ETD. This ETD builder connects to the external IAM file system as if it were connecting to a database. The builder then generates the ETD corresponding to the external tables and prepared statements.

4.3.1 Using the Database Builder Wizard

The Database Builder wizard generates Java-enabled ETDs by connecting to external data sources and creating corresponding ETDs. The ETD builder can then create ETDs based on any combination of tables or prepared SQL statements.

Note: *ETDs created by the Database Builder wizard are read-only. The contents of the generated ETD cannot be edited.*

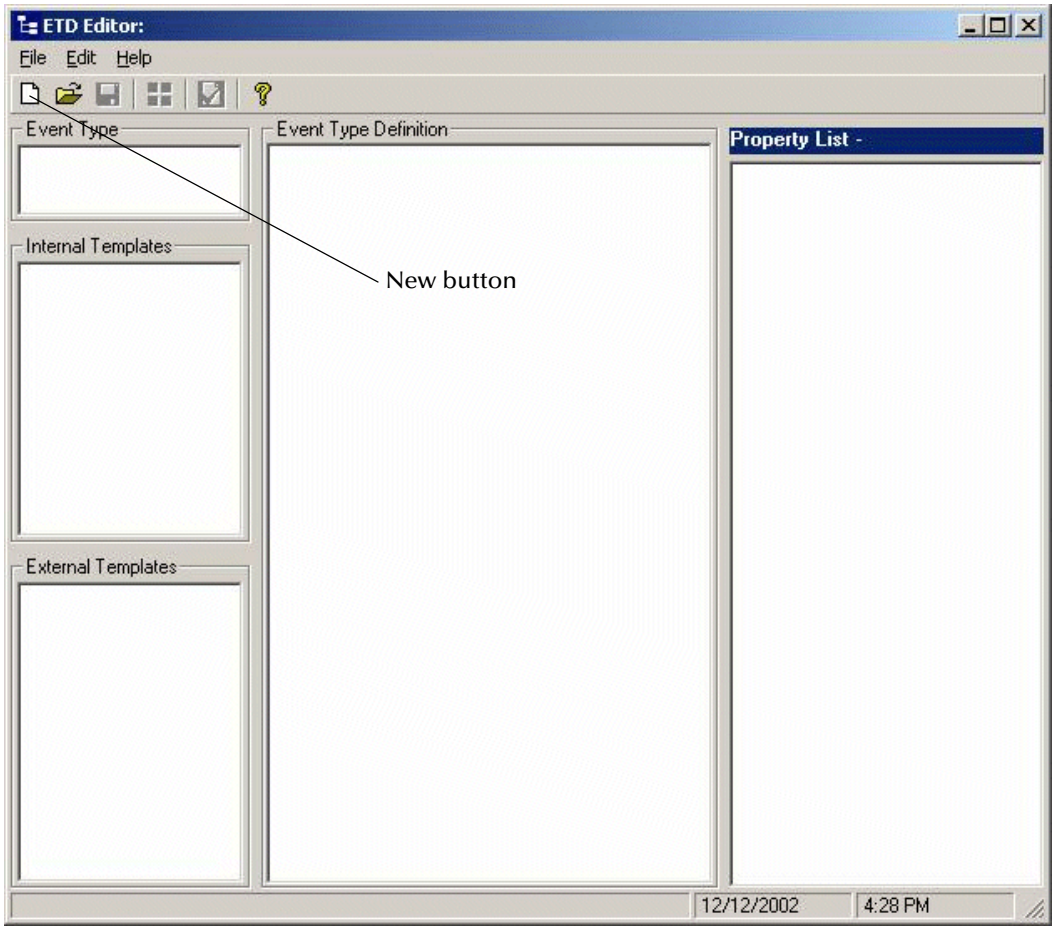
Field nodes are added to the ETD based on the tables in the external data source. Java method and parameter nodes are added to provide the appropriate JDBC functionality. For more information on these Java methods, refer to your JDBC developer's reference.

To create a new ETD using the Database Builder wizard

1. From the **Options** menu of the e*Gate 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 (see Figure 9).

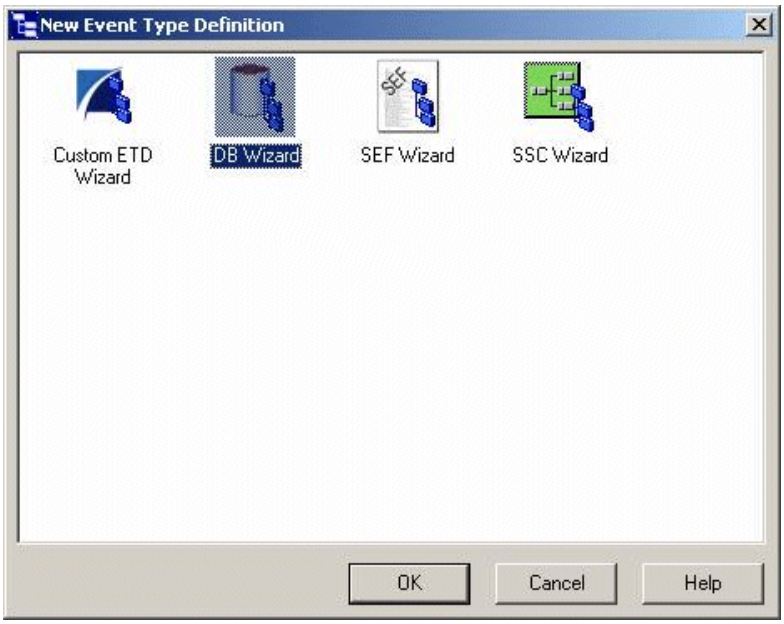
Figure 9 ETD Editor Main Window



4 In the ETD Editor, click the **New** button to open the **New Event Type Definition** dialog box.

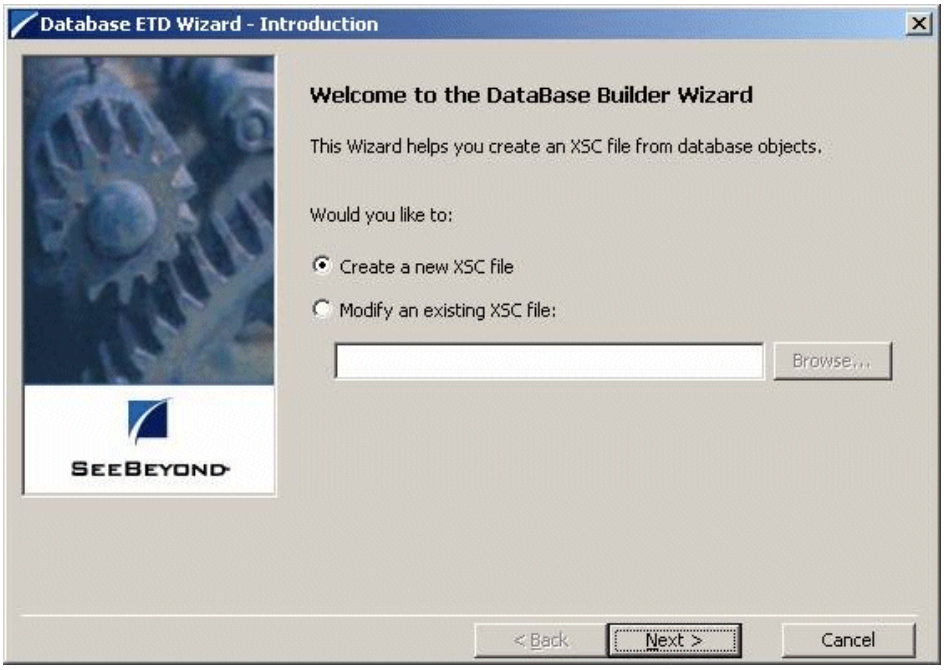
5 In the this dialog box, select the **DB Wizard** icon and click **OK** to continue (see Figure 10).

Figure 10 New Event Type Definition Dialog Box



The wizard's **Introduction** dialog box appears (see Figure 11).

Figure 11 Database Builder Wizard: Introduction

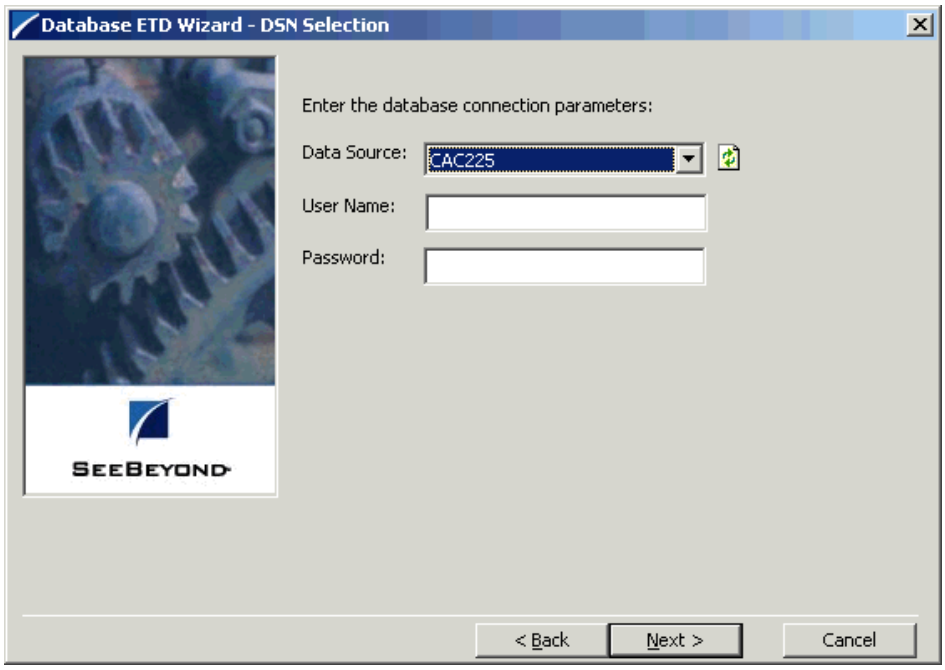


6 Do one of the following actions by checking the desired option button:

- Create a new .**xsc** file.
- Enter the name of an .**xsc** file you want to edit by entering its name or browsing to its location.

7 Click **Next** to continue. The **DSN Selection** dialog box appears (see Figure 12).

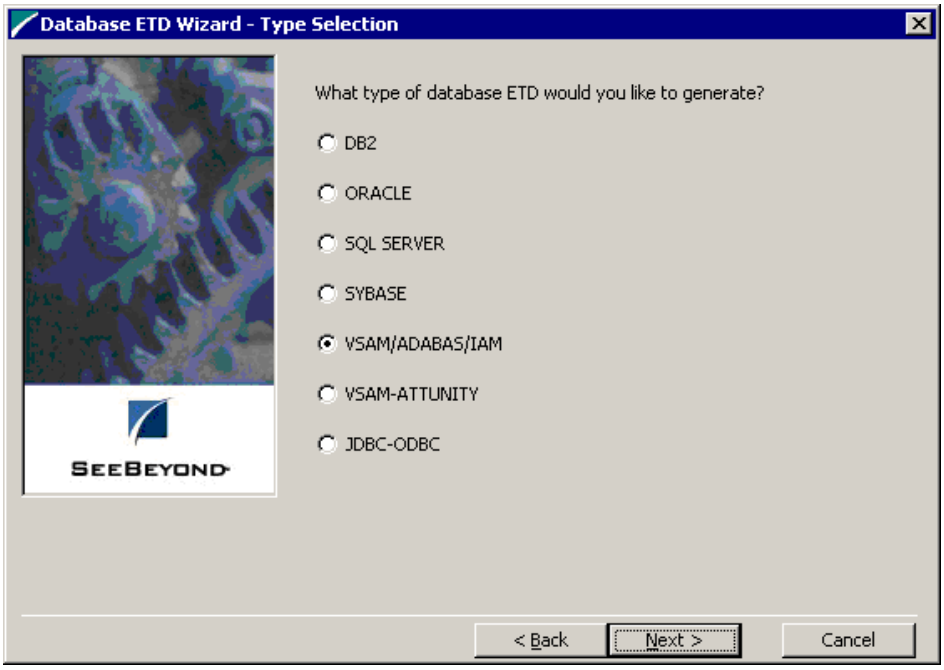
Figure 12 Database Builder Wizard: DSN Selection



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

9 Click **Next** to continue. The wizard's **Type Selection** dialog box appears (see **Figure 13 on page 38**).

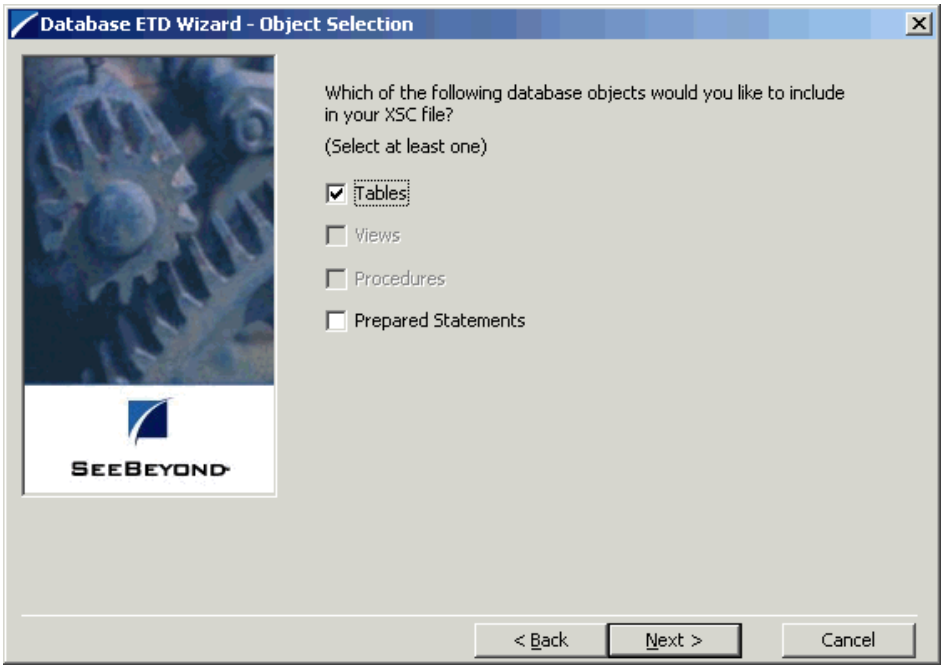
Figure 13 Database Builder Wizard: Type Selection



10 Select the type of database ETD you would like to generate as shown in the previous figure. The data source you selected in the wizard's **DSN Selection** dialog box is the default.

11 Click **Next** to continue. The wizard's **Object Selection** dialog box appears (see Figure 14).

Figure 14 Database Builder Wizard: Object Selection

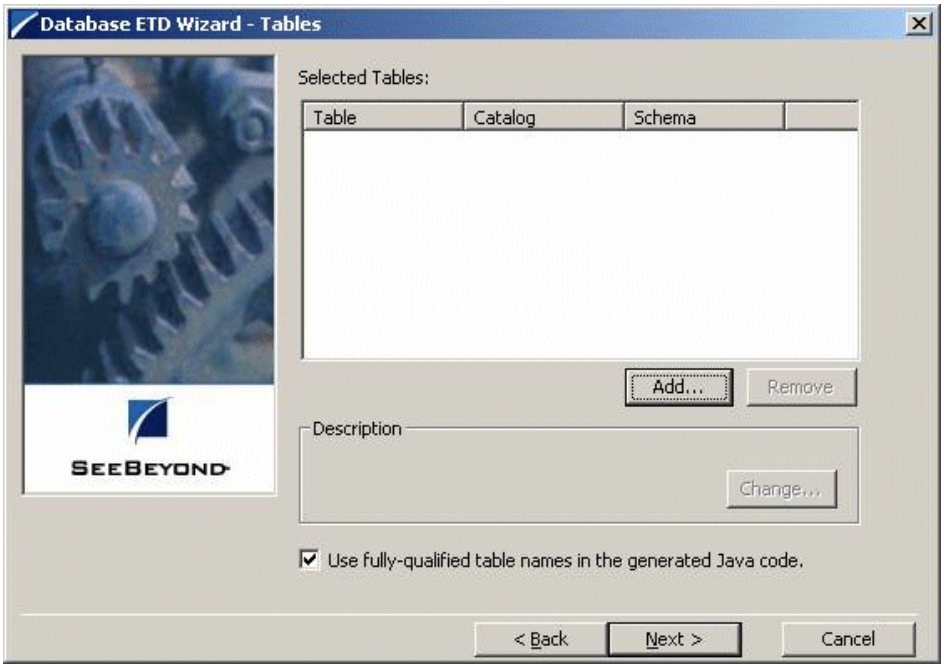


12 In the **Object Selection** dialog box, select **Tables** and/or **Prepared Statements**, depending on what you want to include in your **.xsc** file.

Note: *You cannot select* ***Views*** *or* ***Procedures****.*

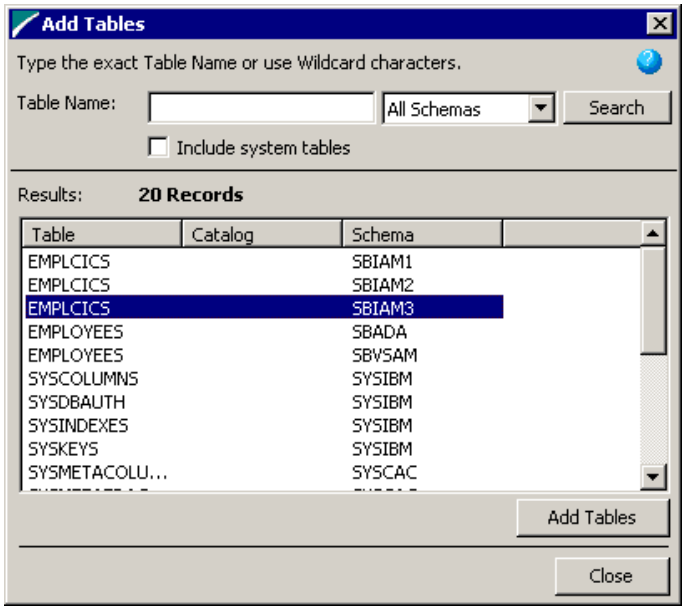
13 Click **Next** to continue. The wizard's **Tables** dialog box appears (see Figure 15). This dialog box allows you to select the tables you want to add to the ETD.

Figure 15 Database Builder Wizard: Tables



14 In this dialog box, click **Add Tables**. The **Add Tables** dialog box appears (see Figure 16).

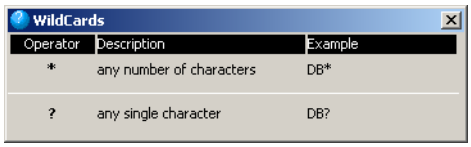
Figure 16 Add Tables Dialog Box



15 In the **Add Tables** dialog box, type the exact name of the database table or use wildcard characters to return table names.

16 To see a list of valid wildcard characters, click the round ball with a question mark located in its center (see Figure 17).

Figure 17 Wildcards



17 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** dialog box.

18 To select the name of the tables you wish to add to your .**xsc** file, double-click on the table name or highlight the table names and click **Add Tables**. You can also use adjacent selections or nonadjacent selections to select multiple table names.

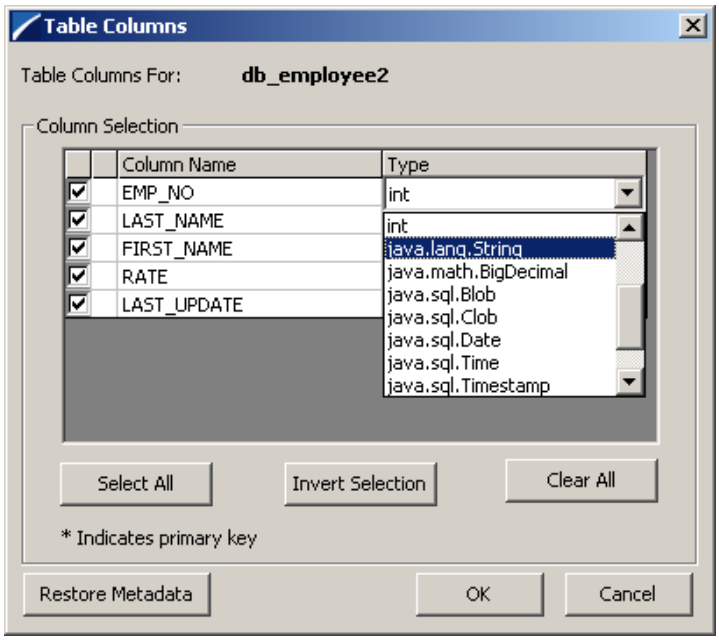
19 When you have finished, click **Close**.

20 In the wizard's **Tables** dialog box, review the tables you have selected. If you would like to change any of the tables you have selected, click **Change**.

21 In the **Columns Selection** dialog box, you can select or de-select 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.

22 Once you have completed your choices, click **OK** (see Figure 18).

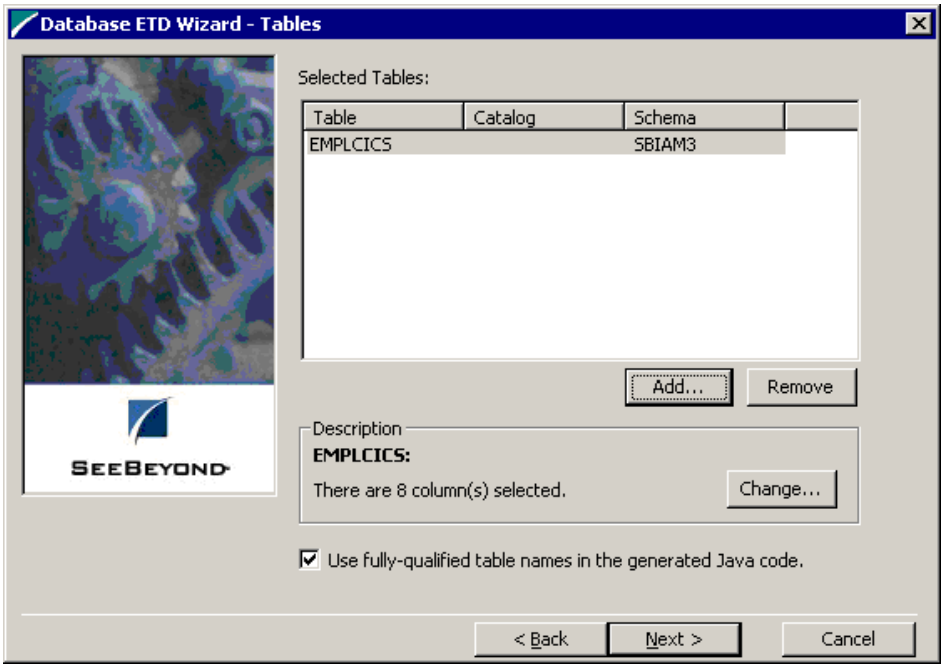
Figure 18 Columns Selection



The wizard then returns you to the **Tables** dialog box.

23 In this dialog box, review the tables you have selected (see Figure 19).

Figure 19 Database Builder Wizard: Tables with Table Names Added



24 If you do not want to use fully-qualified table names in the generated Java code, click to clear the check box.

25 When you are finished with this dialog box, click **Next** to continue.

26 If you selected **Prepared Statements** on the wizard's **Object Selection** dialog box, you are presented with the wizard's **Prepared Statement** dialog box. To add prepared statements to your .**xsc**. file, complete the following steps:

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

B Enter a prepared SQL statement.

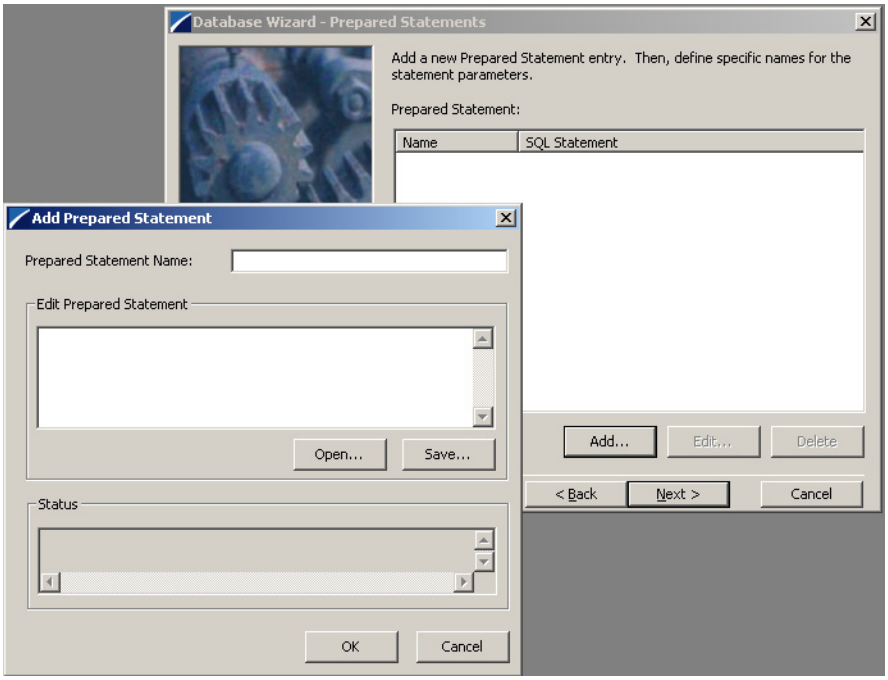
C Enter the **Prepared Statement Name** to be used by the statement.

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

E Click **OK** to return to the wizard's **Prepared Statements** dialog box.

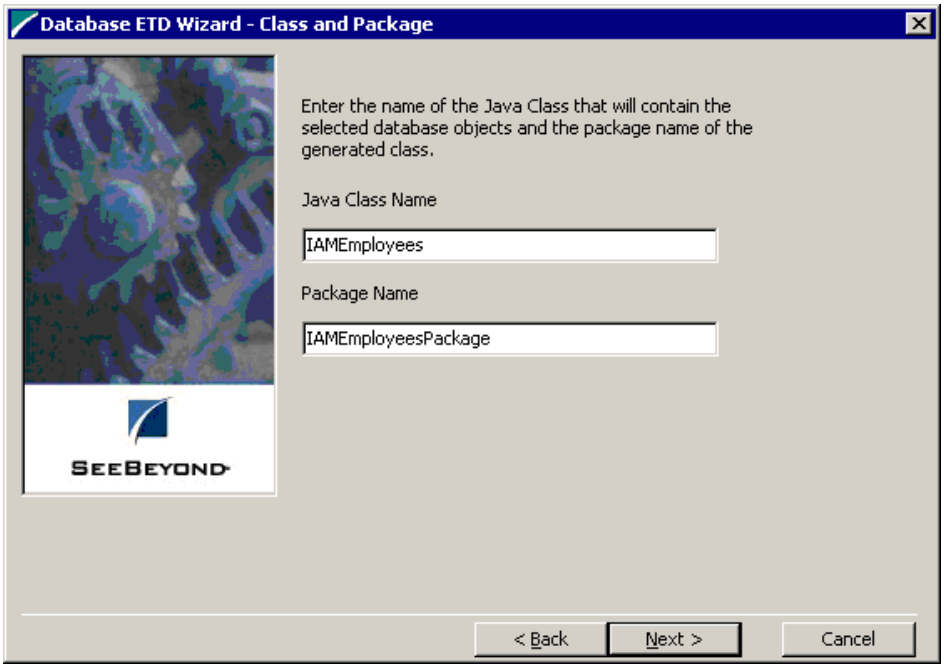
27 Repeat the sub-steps under step 26 to add additional prepared statements (see Figure 20) or click **Next** to continue.

Figure 20 Database Builder Wizard: Prepared Statements



28 The wizard's **Class and Package** dialog box appears (see Figure 21).

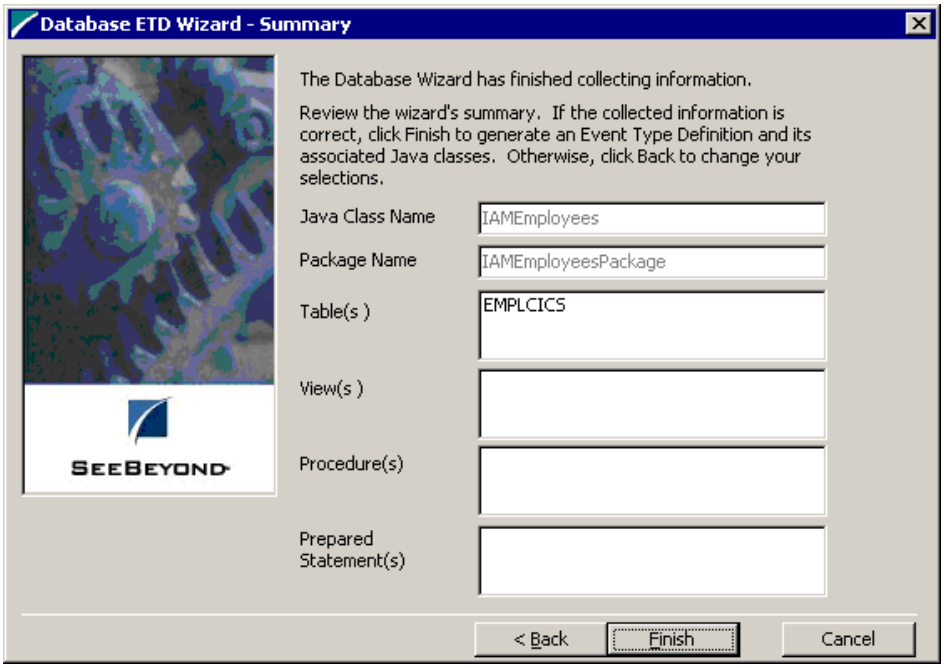
Figure 21 Database Builder Wizard: Class and Package



29 Enter the **Java Class Name** you want to contain the selected tables and/or prepared statements and the **Package Name** of the generated classes.

30 Click **Next** to view a summary of the information you have entered in the database wizard (see Figure 22).

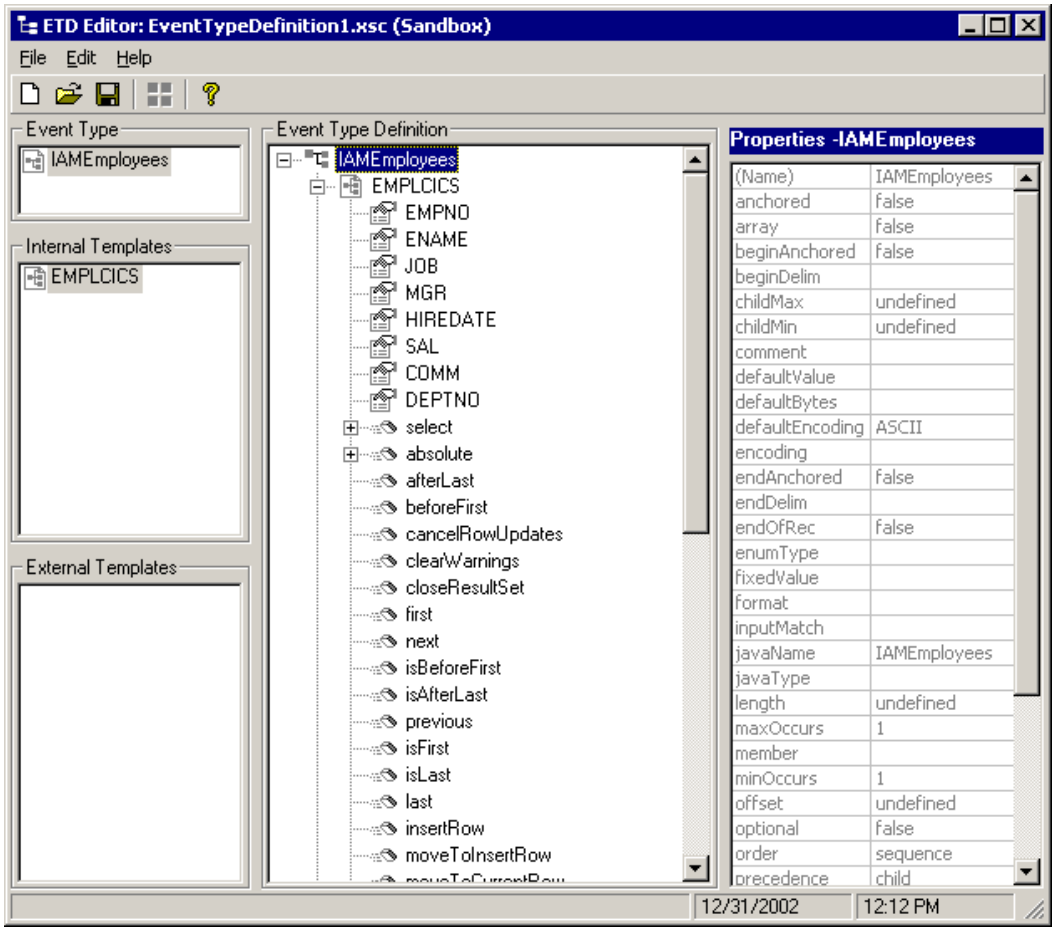
Figure 22 Database Builder Wizard: Summary



31 Click **Finish** to begin generating the ETD.

When the ETD has been generated, its node structure appears in the ETD Editor's Main Window as shown in **Figure 23 on page 45**.

Figure 23 ETD Editor Main Window with Generated ETD



32 Click the **Save** button to save the ETD in a **Save** dialog box. Use the dialog box to name the new ETD.

4.3.2 Generated ETDs

The Database Builder wizard can create editable ETDs for the e*Way. These types of ETDs can also be combined with each other. The types of ETDs are:

- **Table**: Contains fields for each of the columns in the selected table, as well as the methods required to exchange data with an external data source. To edit this type of ETD, you need to open the **.xsc** file in the ETD Editor.
- **Prepared Statement**: Contains a result set for a prepared statement. To edit this type of ETD, you need to open the **.xsc** file in the ETD Editor.

4.3.3 Editing an Existing .xsc File

You can use the Database Builder wizard to edit an existing **.xsc** file you have created.

To edit the .xsc file

1. From the **Options** menu of the e*Gate 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 Editor's **Tools** menu click **File** then **New**.
5. From the **New Event Type Definition** dialog box, select the **DB Wizard** icon and click **OK**.
6. On the Database Builder wizard's **Introduction** dialog box, select **Modify an existing XSC file** and browse to the appropriate ***.xsc*** file you want to edit.

   You are now able to edit the ***.xsc*** file.

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

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

4.4 Using Database ETDs

This section explains the types of ETDs used with the e*Way within the Collaboration Rules Editor. This section explains how to use these methods with:

- Tables
- Prepared statements

4.4.1 Using Tables

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

The select() Method

To perform a select operation on a table, add a new method in the Collaboration Rules Editor by clicking on the **method** button. In the **Method Properties** dialog box, define the method.

Add a rule to the method by clicking on the **rule{}** button. The **Rule Properties** dialog box appears. Select the node and drag and drop it into the dialog box. This method takes an empty string or a WHERE clause, for example:

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

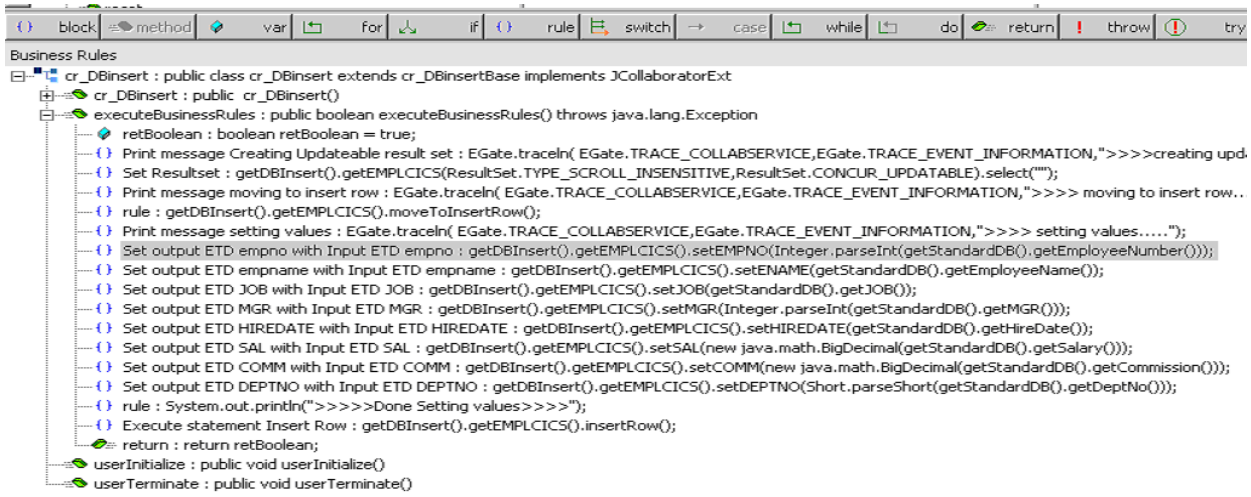
The insert() Method

To perform an insert operation on a table

1. From the e*Gate Collaboration Rules Editor, click the **method** button. Execute the **select()** method with the right concurrency, for example, CONCUR_UPDATABLE and scroll sensitivity, for example, TYPE_SCROLL_INSENSITIVE.
2. Move to the insert row by the **moveToInsertRow()** method.
3. Set the fields of the table ETD.
4. Insert the row by calling the **insertRow()** method.

   See Figure 24 for an example of how this method and code appear in the Collaboration Rules Editor.

Figure 24 insert() Method in Collaboration Rules Editor



The sample schema for this e*Way shows additional examples of how to use this method. See **"Creating Collaboration Rules" on page 62** for examples from the Collaboration Rules Editor.

The update() Method

Perform an update operation on a table as follows:

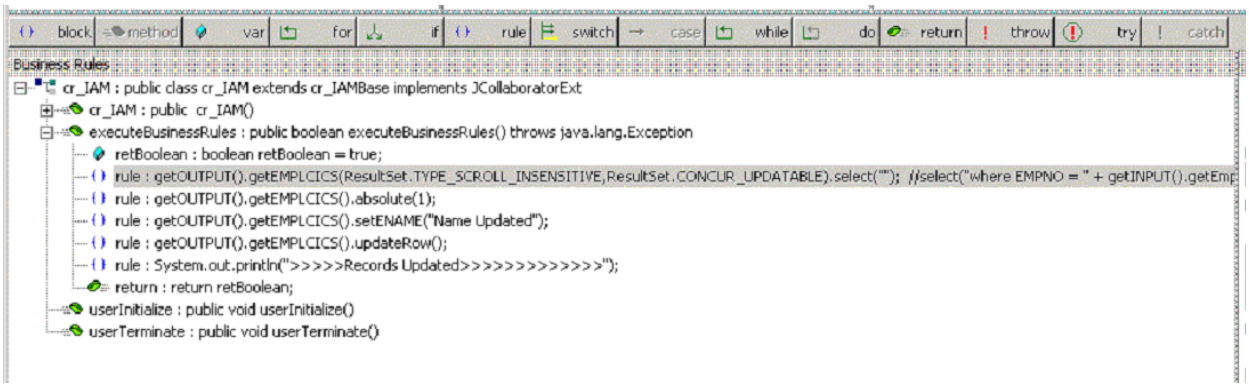
1. Execute the **select()** method with the right concurrency, for example, CONCUR_UPDATEABLE and scroll sensitivity, for example, TYPE_SCROLL_INSENSITIVE.
2. Move to the row that you want to update.
3. Set the fields of the table ETD.
4. Insert the row by calling **insertRow()**.

In this example, we move to the third record and update the EMP_NO and RATE fields. See the following code sample:

```
getDBEmp().getDB_EMPLOYEE(ResultSet.TYPE_SCROLL_INSENSITIVE,
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();
```

See Figure 25 for an example of how this method and code appear in the Collaboration Rules Editor.

Figure 25 Update() Method Business Rule



The delete() Method

Perform a delete operation on a table as follows:

1. Execute the **select()** method with the right concurrency, for example, CONCUR_UPDATEABLE and scroll sensitivity, for example, TYPE_SCROLL_INSENSITIVE.
2. Move to the row that you want to delete.
3. Set the fields of the table ETD.
4. Insert the row by calling **insertRow()**.

   In this example DELETE the first record of the result set. See the following code sample:

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

4.4.2 Using Prepared Statements

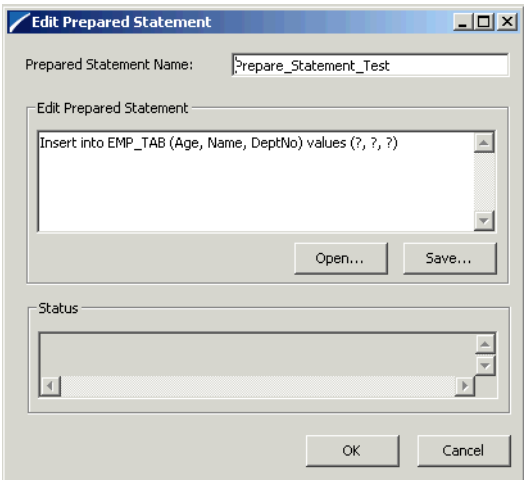
To create a prepared statement from the Database Builder wizard

1 Select the **Prepared Statement** option in the e*Gate ETD Editor's Database Builder wizard and complete the following steps:

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

B Enter a prepared SQL statement as shown in Figure 26. The question marks serve as place holders that are later defined as the parameters in statement.

Figure 26 Add Prepared Statement

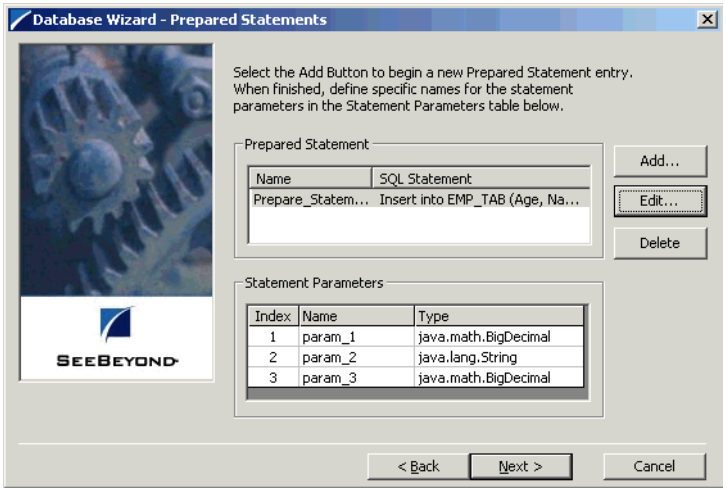


C Enter the **Prepared Statement Name** to be used by the statement.

D Use the **Open** or **Save** buttons to open pre-existing statements or create and save the current one.

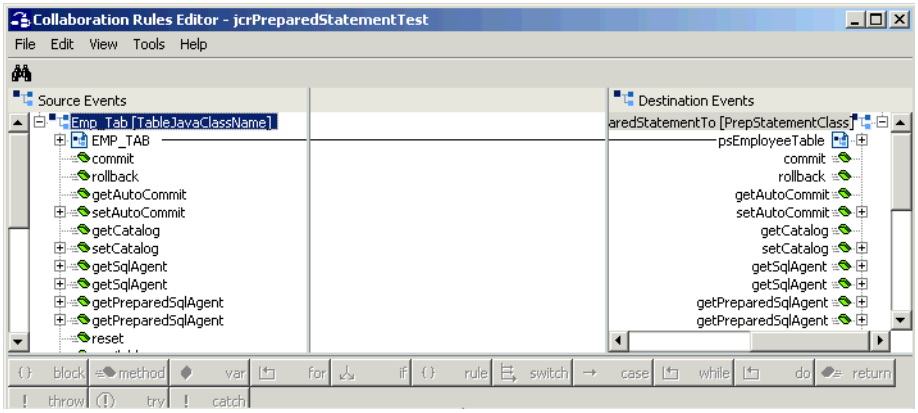
2 Figure 27 shows the statement along with a table of the parameters. You can give each parameter an appropriate name and data type. These parameters become nodes in the prepared statement ETD and fields within the generated Java object.

Figure 27 Prepared Statement Summary and Parameter Modification



3 From the ETD Editor, save the prepared statement you just created. Figure 28 shows the ETD with this prepared statement as viewed in the e*Gate Collaboration Rules Editor.

Figure 28 Collaboration Rules Editor



4 Map the fields you want to populate the table with from the **Source Events** to the **Destination Events**.

5 Create your rules by dragging and dropping each node into the **Rules** window. This particular example shows the age, name and department number of three employees.

6 You can verify the new content in the employee database by running the schema.

Multiple Deletes and Inserts

Performing multiple delete or insert operations using prepared statements requires specific configuration of the eXadas Data Integrator on the mainframe.

The parameter STATEMENT RETENTION needs to be set to DISCONNECT instead of SYNCPOINT. If this parameter is set to SYNCPOINT, the prepared statements are not kept across commits or rollbacks. This parameter has to be set to DISCONNECT for statements to be retained across commits and rollbacks.

For more information on the STATEMENT RETENTION parameter, see the appropriate eXadas Data Integrator documentation.

4.4.3 Database Configuration Node

The **Database Configuration** node allows you to manage the transaction mode through a Collaboration if you have set this mode to **Manual** in the e*Way Connection configuration.

4.5 Sample Schema: Polling from a Database

This section explains how to use the IAM e*Way in a sample schema implementation.

4.5.1 Implementation Overview

The name of the sample schema import file is **IAM_dbSelect1.zip**. You can find this sample on the CD-ROM at the following location:

samples\ewiam

Note: *See the **e*Gate Integrator Installation Guide** for a list that explains which of the CD-ROM disks contains the sample schemas.*

This sample allows you to observe an end-to-end data-exchange scenario involving e*Gate, the e*Way, and sample interfaces. This section explains how to implement this sample schema that uses the IAM e*Way.

You can also use the procedures given in this section to create your own schema based on the samples provided. It is recommended that you use a combination of both methods, creating your own schema like the sample, then importing the sample into e*Gate to check your results.

Before Importing or Running a Sample Schema

To import and run a sample schema, the IAM e*Way must be installed, as well as all the additional applications, as explained in **Chapter 2**. You must be able to connect to a IAM database.

Importing a Schema

This section explains how to import an e*Gate schema .**zip** file.

To import a sample schema

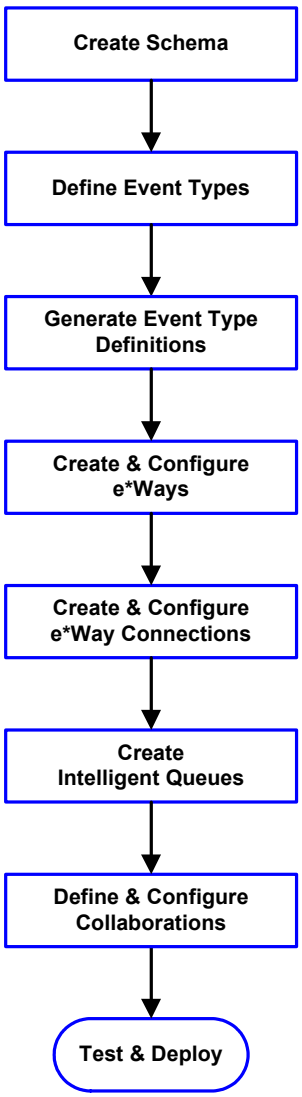
1. Copy the .**zip** file, **IAM_dbSelect.zip,** from the **samples\ewiam** directory in the install CD-ROM to your desktop or to a temporary directory, then unzip the file.
2. Start the e*Gate Schema Designer.
3. On the **Open Schema from Registry Host** dialog box, click **New**.
4. On the **New Schema** dialog box, click **Create from export**, and then click **Find**.
5. On the **Import from File** dialog box, browse to the directory that contains the sample schema. You can give the schema any name you want.
6. Click the .**zip** file then click **Open**.

   The schema is installed.

Basic Implementation Steps

To create and configure the sample schema

Use the following implementation sequence:



1 The first step is to create a new schema. The rest of these steps apply only to this schema.

2 The second step is to create and define the Event Types you are transporting and processing within the schema.

3 You need to associate the Event Types created in the previous step with Event Type Definitions (ETDs) you want to use in the schema.

4 The next step is to create and configure the required e*Ways.

5 You must create and configure the e*Way Connections.

6 Now you need to create Intelligent Queues (IQs) and IQ Managers (if necessary) to hold published Events.

7 You need to create the desired Collaboration Rules for your schema, along with their associated Business Rules.

8 Next, you need to define and configure the necessary Collaborations.

9 Finally, you must check and test your schema. Once you have verified that it is working correctly, you can deploy it in your production environment.

Running the Schema

When you have finished setting up the sample schema, you are ready to run it. See **"Running the Schema" on page 69** for details.

Sample Schema General Operation

This sample schema demonstrates the polling of records from a IAM file system and converting the records into e*Gate Events. The schema is set up to operate as follows:

1. The **ewIN** e*Way retrieves an Event (text file) containing the database select criteria and publishes it to the **aaa** Intelligent Queue (IQ).
2. The **ewIAM** e*Way retrieves the input Event Type (**et_Data**) from the **aaa** IQ. This action triggers this e*Way's **colIAM** Collaboration, which performs the following operations:
   - The information in **et_Data** is used to retrieve information from the database via the **ewcIAM** e*Way Connection. This e*Way Connection contains information used by the **colIAM** Collaboration to connect to the IAM database.
   - The information retrieved from the database is copied to the Event Type (**et_Data**) and published to the **bbb** IQ.
3. The **ewOUT** e*Way retrieves the Event Type (**et_Data**) from the **bbb** IQ then writes it out to a text file on the local file system.

External Database Tables

The sample uses a simple external IAM file system with a table called **EMPLCICS**. This table contains the columns explained in Table 3.

Table 3 EMPLCICS Table

Column	Format	Description
EMPNO	SQL_INTEGER	Employee number
ENAME	SQL_VARCHAR	Employee's name
JOB	SQL_CHAR	Employee's job title
MGR	SQL_INTEGER	Number for employee's manager
HIREDATE	SQL_CHAR	Employee's hire data
SAL	SQL_DECIMAL	Employee's salary
COMM	SQL_DECIMAL	Employee's commission
DEPTNO	SQL_SMALLINT	Department number of employee

4.5.2 Implementing the Sample Schema

This section explains how to create, configure, and implement the sample schema for the IAM e*Way. It follows the same steps outlined under the section **"To create and configure the sample schema" on page 53**.

Creating the Schema

The first step in deploying the sample implementation is to create a new schema.

To create the schema

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

   The Schema Designer then launches and displays the newly the created schema.

Adding Event Types and ETDs

The Event Types and corresponding ETDs used in this sample are shown in table.

Table 4 Schema Event Types and ETDs

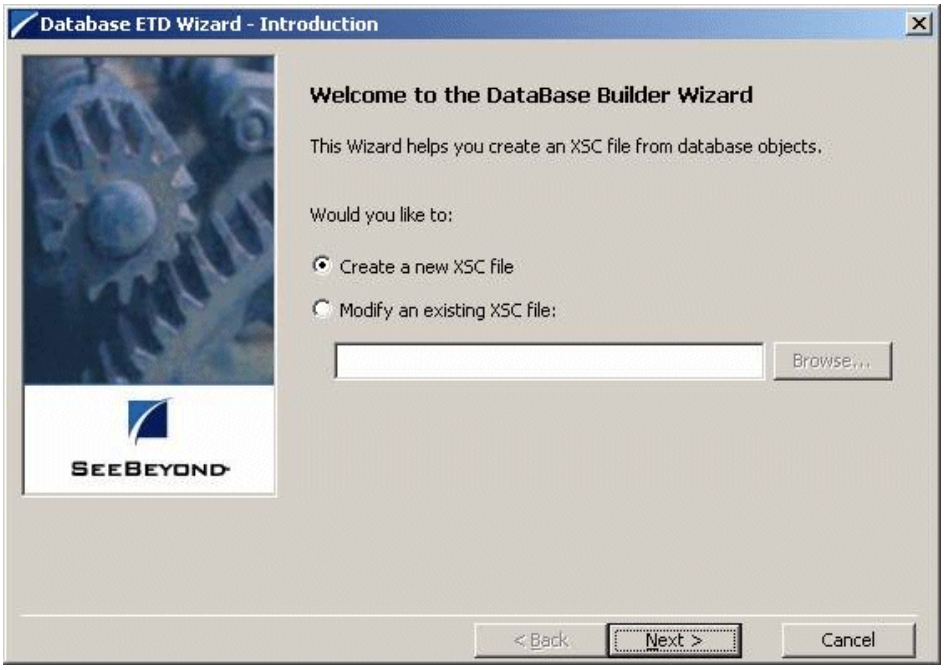
Event Type	ETD File Name	Description
et_IAM	et_IAM.xsc	Represents the layout of the employee records in the EMPLCICS table. The ETD is generated by using the ETD Editor's Database Builder wizard.
et_Data	et_Data.xsc	Passes data into and out of the e*Gate system.

To create the et_IAM Event Type and corresponding 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 **et_IAM** and click **OK**.
6. Double-click the new **et_IAM** Event Type to display its properties.
7. Select your **Data Source:** from the drop-down list and enter your **User Name** and **Password**.
8. From the **File** menu, choose **New**. The **New Event Type Definition** dialog box appears.
9. In the **New Event Type Definition** dialog box, click the **DB Wizard** icon and click **OK**.

   The Database Builder wizard's **Introduction** dialog box appears (see **Figure 29 on page 56**).

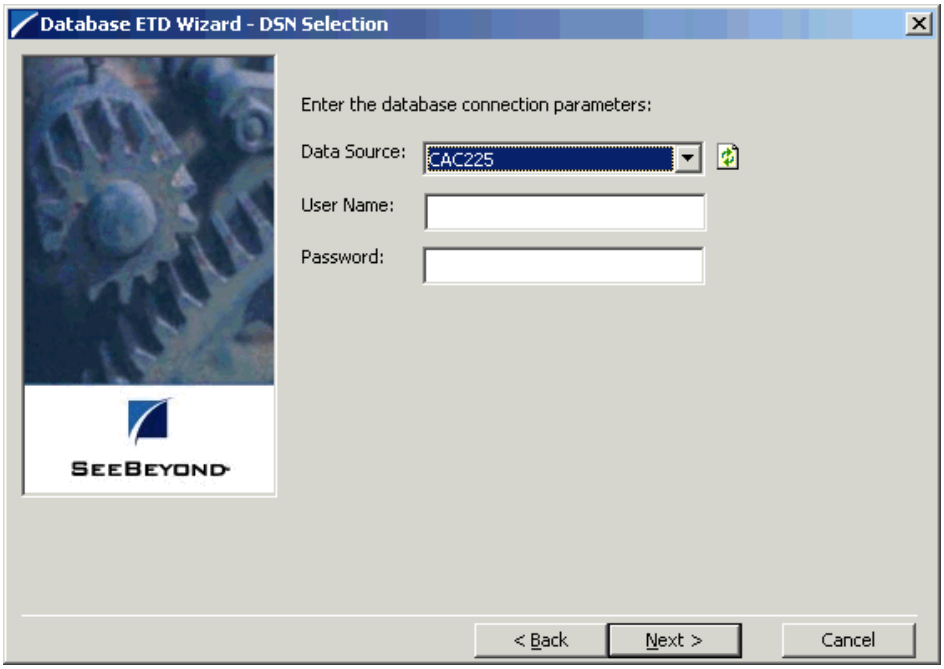
Figure 29 Database Builder Wizard Introduction



10 Select **Create a new .XSC file**. Click **Next** to continue.

The **DSN Selection** dialog box appears (see Figure 30).

Figure 30 Database Builder Wizard: DSN Selection



11 Enter the database DSN source and log-on information.

A Select the **Data Source** from the drop-down list of ODBC data sources.

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

12 Click **Next** to continue.

The wizard's **Type Selection** dialog box appears. The DSN source you selected on the previous window is the default selection for this window. Do not change this selection type unless instructed to do so by SeeBeyond support personal.

Note: *See* **"Database Builder Wizard" on page 34** *for complete instructions on how to use the Database Builder wizard.*

13 Click **Next** to continue.

14 This sample schema uses a table rather than a procedure. Select **Table** and click **Next** to continue.

15 From the wizard's **Tables** dialog box, click **Add Tables.** Enter the exact **Table Name** or enter any valid wildcards.

16 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.

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

18 The wizard's **Class and Package Name** dialog box appears. Enter the following class and package information:

A Enter your database name as the java class name.

B Enter **IAMEmployees** for the package name and click **Next** to continue.

19 Click **Finish** to complete the wizard. The wizard generates and displays the ETD.

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

21 In the **Save** dialog box, name the ETD **et_IAM.xsc** and click **OK**.

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

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

To create the et_Data Event Type and ETD

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

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

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

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

5 Click the **New** button to create a new ETD.

The Java ETD 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 introductory screen, then click **Next** to continue. The **Package Name** dialog box appears.

9 Enter **et_DataPackage** 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 et_Data ETD Properties

Node	Property	Value
Root Node	Name	et_Data
	Structure	Fixed
	Length	0
Child Node	Name	Data
	Structure	Fixed
	Length	0

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

14 Enter the name **et_Data.xsc** and click **OK**.

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

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

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

Adding and Configuring e*Ways

The sample schema uses the following e*Ways:

- **ewIN**: This e*Way retrieves an Event (text file) containing the database select criteria and publishes it to the **aaa** IQ.
- **ewOUT**: This e*Way retrieves the Event from the **bbb** IQ then writes it out to a text file on the local file system.
- **ewIAM**: This e*Way retrieves the input Event from the **aaa** IQ. This triggers the e*Way to request information from the external database (via the e*Way Connection) and publishes the results to the **bbb** IQ.

To create the ewIN e*Way

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

2 Enter **ewIN** 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 ewIN e*Way Parameter Settings

Section Name	Parameter	Value
General settings	AllowIncoming	YES
	AllowOutgoing	NO
	PerformanceTesting	Default
Outbound (send) settings	All	Default
Poller (inbound) settings	PollDirectory	\DATA\INPUT
	All others	Default
Performance testing	All	Default

7 Select **Save** from the **File** menu. Enter **ewIN** 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, click the **Start automatically** check box.

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

To create the ewOUT e*Way

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

2 Enter **ewOUT** 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 ewOUT e*Way Parameter Settings

Section Name	Parameter	Value
General settings	AllowIncoming	NO
	AllowOutgoing	YES
	PerformanceTesting	Default
Outbound (send) settings	OutputDirectory	\DATA\OUTPUT
	OutputFileName	output%d.dat
Poller (inbound) settings	All	Default
Performance testing	All	Default

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

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

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

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

To create the ewIAM e*Way

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

2 Enter **ewIAM** 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, to create a Multi-Mode e*Way.

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

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

Table 8 ewIAM e*Way Parameter Setting

Section Name	Parameter	Value
JVM Settings	All	Default

7 Select **Save** from the **File** menu. Enter **ewIAM.cfg** 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, click the **Start automatically** check box.

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

Adding and Configuring e*Way Connections

The sample schema uses the following e*Way Connection:

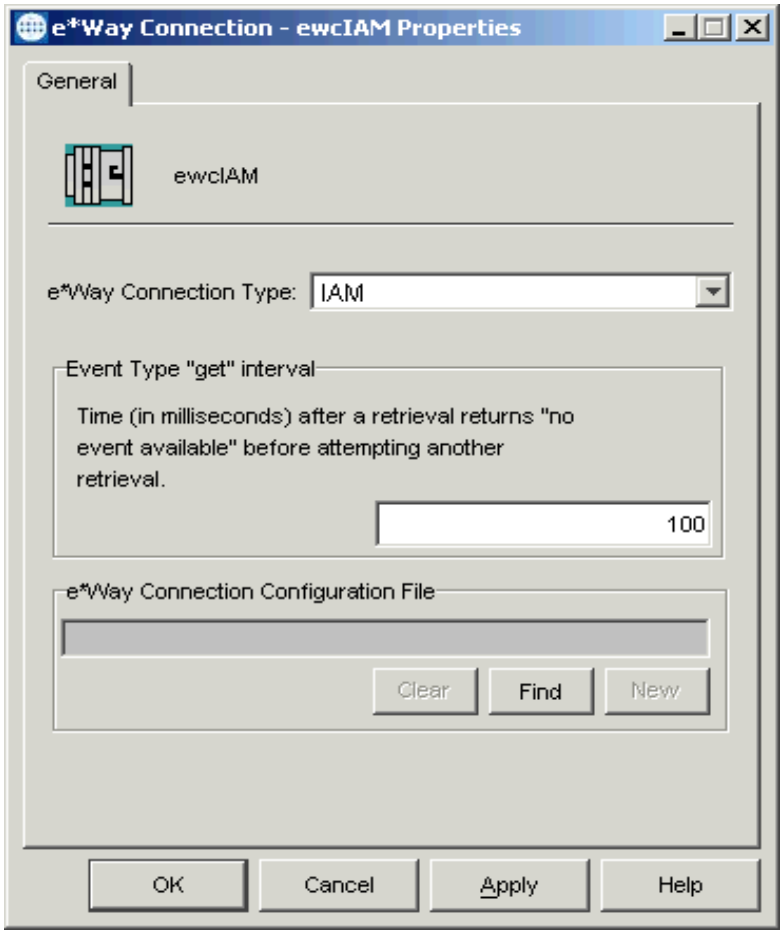
- **ewcIAM:** Connects the **DBselect** component to the external database and returns the requested records to be published to the **bbb** IQ.

To create the ewcIAM 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 **ewcIAM** 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.

   The **e*Way Connection Properties** dialog box appears (see Figure 31).

Figure 31 ewcIAM e*Way Connection Properties Dialog Box



5. Select **IAM** from the **e*Way Connection Type** drop-down list (see the previous figure).
6. Click **New** to create a new configuration file.

7 Enter the parameters for the e*Way Connection as shown in Table 9.

Table 9 ewcIAM e*Way Connection Parameter Settings

Section Name	Parameter	Value
DataSource	Driver	Default
	JDBC URL	A valid JDBC URL
	User name	Local settings
	Password	Local settings
connector	Type	Default
	Class	Default

8 Select **Save** from the **File** menu. Enter **ewcIAM** 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.

Adding IQs

The sample schema uses the following IQs:

- **aaa**: Queues the inbound Events for the **ewIAM** e*Way.
- **bbb**: Queues the outbound Events for the **ewOUT** 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 **aaa** and click **Apply** to save the IQ and leave the New IQ dialog box open.

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

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

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

Creating Collaboration Rules

The sample schema uses the following Collaboration Rules:

- **crPassThruIn**: Passes the **et_Data** Event Type into the schema without modifying the Event.
- **crPassThruOut**: Converts the inbound Event's selection criteria into a SQL statement, poll the external database, and returns the matching records as an outbound Event Type (**et_Data**).
- **crIAM**: Contains the logic required to communicate with the external database. This is a Java Collaboration Rule.

To create the crPassThruIn Collaboration Rule Component

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 **crPassThruIn** and click **OK**.
4. Click the **Properties** button to display the Collaboration Rule's properties.
5. Click the **Subscriptions** tab, select the **et_Data** Event Type, and click the right arrow.
6. Click the **Publications** tab, select the **et_Data** Event Type, and click the right arrow.
7. Click **OK** to save the Collaboration Rule.

To create the crPassThruOut Collaboration Rule Component

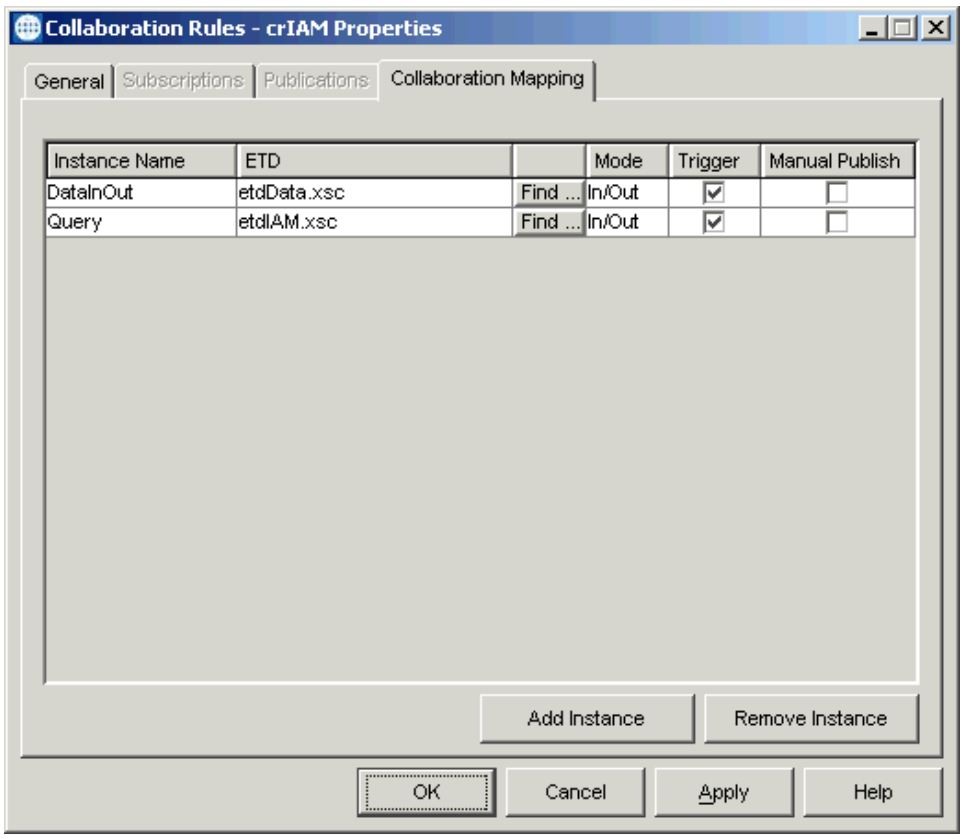
1. In the components pane of the Schema Designer, select the **Collaboration Rules** folder.
2. Click the **New Collaboration Rules** button to add a new Collaboration Rule.
3. Name the Collaboration Rule **crPassThruOut** and click **OK**.
4. Click the **Properties** button to display the Collaboration Rule's properties.
5. Click the **Subscriptions** tab, select the **et_Data** Event Type, and click the right arrow.
6. Click the **Publications** tab, select the **et_Data** Event Type, and click the right arrow.
7. Click **OK** to save the Collaboration Rule.

To create the crIAM Collaboration Rule Component

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 **crIAM** 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 and configure them as shown in Figure 32.

Figure 32 crIAM Collaboration Rules Properties: Collaboration Mapping



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

9 Click the **General** tab.

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

The Java Collaboration Rules Editor appears. Note that Source and Destination Events are already supplied based on the Collaboration Rule's Collaboration Mapping.

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

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

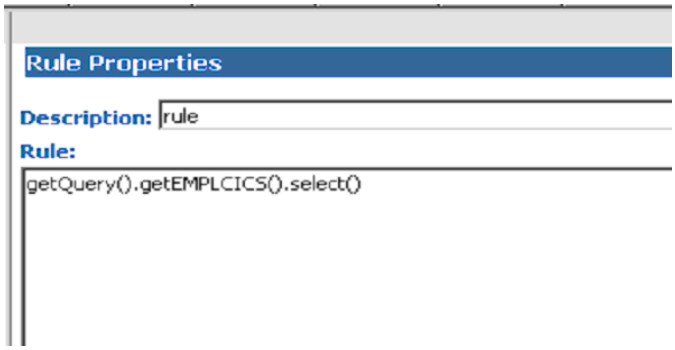
12 From the **Tools** menu, choose **Options**, and then click **Add File**. Select **eGate\client\classes\stcjdbcx.jar** and click **OK** to close each of the dialog boxes.

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

14 In the **Destination Events** pane, expand the first **et_Data** Event Type until the **select()** method is visible.

15 Drag the **select()** method into the **Rule** text box of the **Rule Properties** pane (see Figure 33).

Figure 33 Rule Properties 1

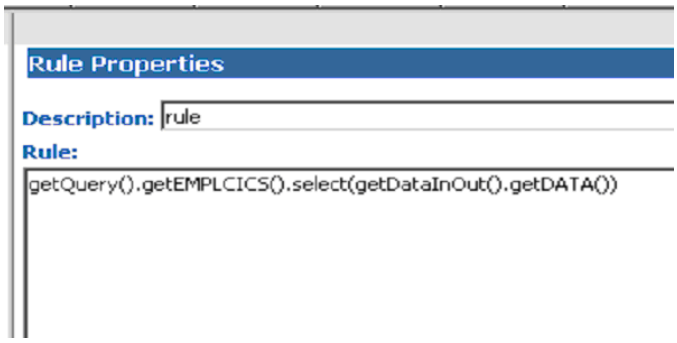


A dialog box appears. Click **OK** to close it without entering any criteria.

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

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

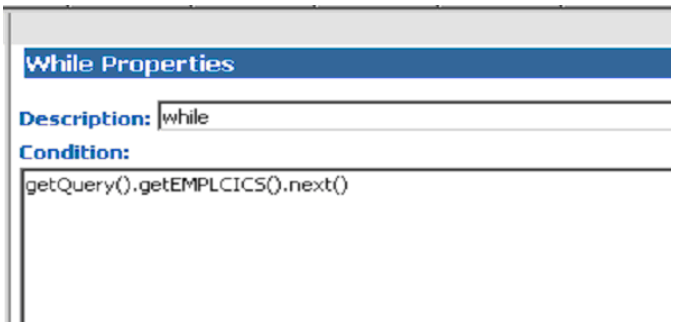
Figure 34 Rule Properties 1 (Continued)



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

19 Drag the **next()** method from the **Destination Events** pane into the **Condition** text box of the **While Properties** pane (see Figure 35).

Figure 35 While Properties

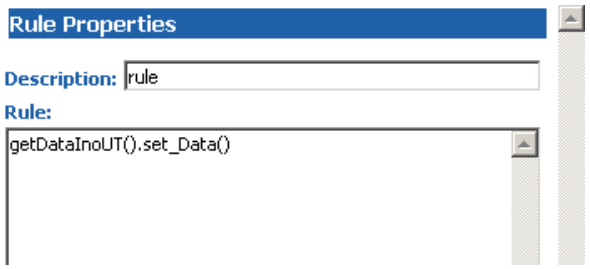


20 Select the newly edited **while** loop in the **Business Rules** pane and click the **rule** button to add a new rule as a *child* to the **while** loop.

21 In the **Destination Events** pane, expand the second **et_Data** Event Type until the **Data** node is visible.

22 Drag the **Data** node into the **Rule** text box of the **Rule Properties** pane. (see Figure 36).

Figure 36 Rule Properties 2

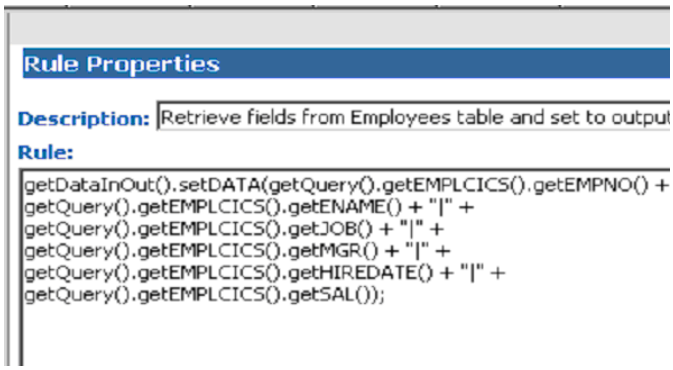


23 In the **Rule Properties** pane, position the cursor inside the parentheses of the **set_Data()** method. Then drag the following data nodes of the **Query** (**et_IAM** Event Type) Event (**EMPLCICS** table) from the **Source Events** pane into the parentheses:

- **EMPNO**
- **ENAME**
- **JOB**
- **MGR**
- **HIREDATE**
- **SAL**

24 Edit the text of the condition to add a newline character and pipe (|) delimiters (with quotation marks) between each of the data nodes as shown in Figure 37.

Figure 37 Rule Properties 2 (Continued)

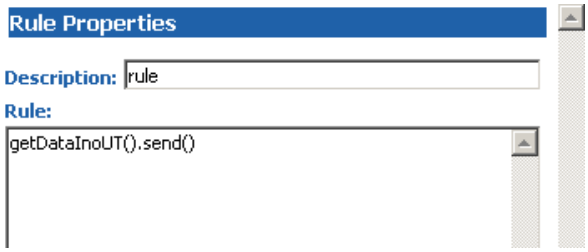


25 Select the newly edited rule in the **Business Rules** pane and click the **rule** button to add a new rule inside the while loop.

26 Drag the root node of the **et_Data** Event into the **rule** text box in the **Rule Properties** pane.

27 Edit the rule by adding a **send()** method as shown in Figure 38.

Figure 38 Rule Properties 3



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

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

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

30 From the **File** menu, choose **Close** to close the Collaboration Rules Editor and return to the **Collaboration Properties** dialog box.

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

31 Click **OK** to save and close the **crIAM** Collaboration Rule. The saved Collaboration Rules file is **crIAM.class**.

Adding and Configuring Collaborations

The sample schema has the following Collaborations:

- **colgetDATA**: Uses the **crPassThruIn** Collaboration Rule.
- **colputDATA**: Uses the **crPassThruOut** Collaboration Rule.
- **colIAM**: Uses and executes the **crIAM.class** Collaboration Rules file.

To add and configure the colgetDATA Collaboration

1. In the **Components** view of the e*Gate Schema Designer, select the **ewIN** e*Way.
2. Click the **New Collaboration** button to create a new Collaboration.
3. Enter the name **colgetDATA** and click **OK**.
4. Select the newly created Collaboration and click the **Properties** button. The **Collaboration Properties** dialog box appears.
5. Select **crPassThruIn** from the drop-down list of Collaboration Rules.
6. Click the upper **Add** button to add a new **Subscription**.
7. Select the **et_Data** Event Type and the **<EXTERNAL>** source.
8. Click the lower **Add** button to add a new **Publication**.
9. Select the **et_Data** Event Type and the **aaa** IQ destination.
10. Click **OK** to close the dialog box and save the Collaboration's properties.

To add and configure the colputDATA Collaboration

1. In the **Components** view of the Schema Designer, select the **ewOUT** e*Way.
2. Select the newly created Collaboration and click the **Properties** button. The **Collaboration Properties** dialog box appears.
3. Enter the name **colputDATA** and click **OK**.
4. Select the newly created Collaboration and click the **Properties** button.
5. Select **crPassThruOut** from the drop-down list of Collaboration Rules.
6. Click the upper **Add** button to add a new **Subscription**.
7. Select the **et_Data** Event Type and the **colIAM** source.
8. Click the lower **Add** button to add a new **Publication**.
9. Select the **et_Data** Event Type and the **<EXTERNAL>** destination.
10. Click **OK** to close the dialog box and save the Collaboration's properties.

To add and configure the colIAM Collaboration

1. In the **Components** view of the Schema Designer, select the **ewIAM** e*Way.
2. Click the **New Collaboration** button to create a new Collaboration.
3. Enter the name **colIAM** and click **OK**.
4. Select the newly created Collaboration and click the **Properties** button. The **Collaboration Properties** dialog box appears.

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

6 Click the upper **Add** button to add the following new **Subscription**:

Enter the **DataInoUT** Instance then select the **et_Data** Event Type and the **colgetDATA** source.

7 Click the lower **Add** button to add the following new **Publications**:

- Enter the **DataInoUT** Instance then select the **et_Data** Event Type and the **bbb** source.
- Enter the **QueryPS** Instance then select the **et_Data** Event Type and the **ewcIAM** source.

8 Click **OK** to close the dialog box and save the Collaboration's properties.

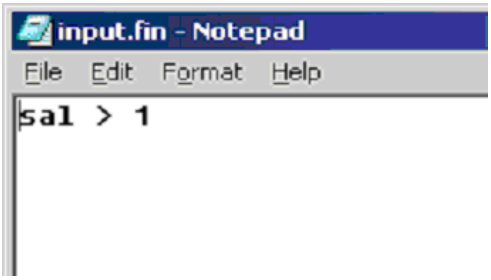
4.5.3 Running the Schema

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 is running, you can view the output text file to verify the results.

Sample input file

Use a text editor to create an input file to be read by the inbound file e*Way (**ewIN**). This simple input file contains the criteria for the **cr_IAM** Collaboration's **select** statement. An example of an input file is shown in Figure 39.

Figure 39 Sample Input File



To start the Control Broker

From a command prompt, type the following command all on one line:

```
stccb -ln logical name -rh registry -rs schema name -un user name
    -up user password
```

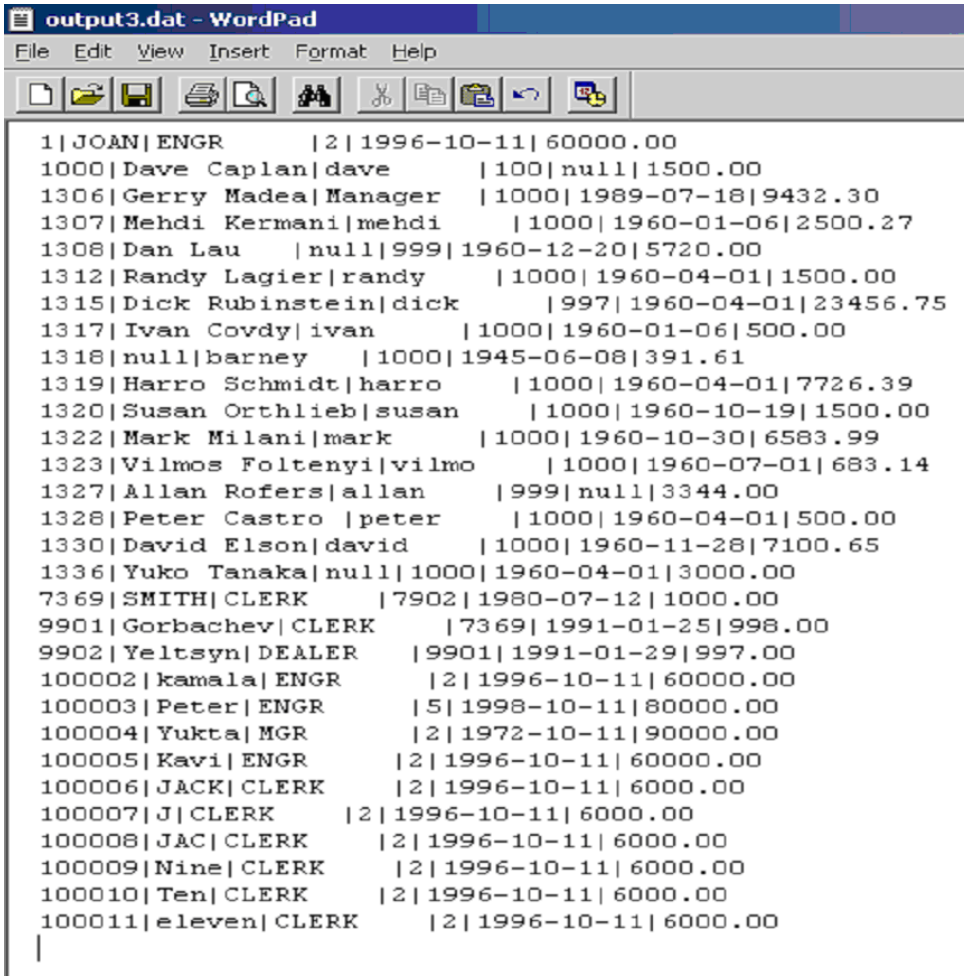
Where:

- ***logical name*** is the logical name of the Control Broker.
- ***schema name*** is the name of the sample schema.
- ***registry*** is the name of the Registry Host.
- ***user name*** and ***user password*** are a valid e*Gate user name/password combination.

To verify the results

Use a text editor to view the output file **C:\eGate\client\DATA\OUTPUT\output0.dat**. Figure 40 shows an example of the records that were returned by the sample schema.

Figure 40 Sample Output File



Chapter 5

IAM e*Way Methods

This chapter provides an overview of the Java classes and methods contained in the IAM e*Way Intelligent Adapter. These methods are used to extend the functionality of the e*Way.

5.1 IAM e*Way Methods and Classes: Overview

The IAM e*Way has been enabled by the Java programming language. Java methods have been added to make it easier to set information in the IAM e*Way Event Type Definitions (ETDs), as well as get information from them.

The nature of this data transfer depends on the configuration parameters (see **Chapter 3**) you set for the e*Way in the e*Gate Schema Designer's e*Way Configuration Editor window. These Java methods are organized into related groups or classes.

Note: *For more information on database ETD structures, their nodes, and attributes (within an e*Gate* ***.xsc*** *file), see* **Chapter 4**.

For the e*Way, the **stceway.exe** file (this file creates a Java-based Multi-Mode e*Way) is used to communicate between the e*Way and other e*Gate components. A Java Collaboration is utilized to keep the communication open between the e*Way and the external system or network.

Note: *See the* ***Standard e*Way Intelligent Adapter User's Guide*** *for more information on the Multi-Mode e*Way. This e*Way is a part of the e*Gate Integrator system.*

5.1.1 Using Java Methods

The e*Gate Schema Designer's Collaboration Rules Editor window allows you to call Java methods by dragging and dropping an ETD node into the **Rules** scroll box of the **Rules Properties** window.

Note: *The node name can be different from the Java method name.*

After you drag and drop, the actual conversion takes place in the **.xsc** file. To view the **.xsc** file, use the Schema Designer's ETD Editor or Collaboration Rules Editor windows.

For example, if the node name is **UpdateCount**, the associated **javaName** is **UpdateCount**. If you want to get the node value, use the Java method called **getUpdateCount()**. If you want to set the node value, use the Java method called **setUpdateCount()**.

5.1.2 Java Classes

The VASAM e*Way's methods are contained in the following classes:

- **com.stc.eways.jdbcx.StatementAgent Class** on page 72
- **com.stc.eways.jdbcx.PreparedStatementAgent Class** on page 82
- **com.stc.eways.jdbcx.PreparedStatementResultSet Class** on page 95
- **com.stc.eways.jdbcx.SqlStatementAgent Class** on page 121
- **com.stc.eways.jdbcx.CallableStatementAgent Class** on page 123
- **com.stc.eways.jdbcx.TableResultSet Class** on page 135
- **com.stc.eways.jdbcx.Session Class** on page 144

The rest of this chapter explains the methods contained in each of these classes.

5.2 com.stc.eways.jdbcx.StatementAgent Class

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

This class is an abstract class for another statement agent.

All Implemented Interfaces

ResetEventListener, SessionEventListener

Direct Known Subclasses

PreparedStatementAgent, SQLStatementAgent, TableResultSet

```
public abstract class StatementAgent
```

This class extends **java.lang.Object.**

The class implements **SessionEventListener** and **ResetEventListener**.

Methods of the StatementAgent Class

resultSetTypeToString

Retrieves the symbol string corresponding to the result set type enumeration.

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

Name	Description
type	The result set type.

Returns

The enumeration symbol string.

resultSetDirToString

Retrieves the symbol string corresponding to the result set direction enumeration.

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

Name	Description
dir	The result set scroll directions.

Returns

The enumeration symbol string.

resultSetConcurToString

Retrieves the symbol string corresponding to the result set concurrency enumeration.

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

Name	Description
concur	The result set concurrency.

Returns

The enumeration symbol string.

isClosed

Retrieves the statement agent's close status.

```
public boolean isClosed()
```

Returns

true if the statement agent is closed; otherwise **false**.

queryName

Supplies the name of the listener.

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

Specified By

queryName in interface SessionEventListener.

Returns

The listener's class name.

queryDescription

Retrieves 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 the interface **SessionEventListener**.

Name	Description
evt	Session event.

sessionClosed

Closes the session event handler.

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

Specified by

sessionClosed in the interface **SessionEventListener**.

Name	Description
evt	Session event.

resetRequested

Resets the event handler.

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

Specified by

resetRequested in the interface **ResetEventListener**.

Name	Description
evt	Requested Reset event.

Throws

java.sql.SQLException

getResultSetType

Returns the result set scroll type.

```
public int getResultSetType()
```

Returns

The result set type.

Throws

java.sql.SQLException

getResultSetConcurrency

Returns the result set concurrency mode.

```
public int getResultSetConcurrency()
```

Returns

The result set concurrency.

Throws

java.sql.SQLException

setEscapeProcessing

Sets the escape syntax processing.

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

Name	Description
bEscape	true to enable; false to disable.

Throws

java.sql.SQLException

setCursorName

Sets the result set cursor name.

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

Name	Description
sName	Cursor name.

Throws

java.sql.SQLException

getQueryTimeout

Returns the query timeout duration.

```
public int getQueryTimeout()
```

Returns

The number of seconds to wait before the time-out.

Throws

java.sql.SQLException

setQueryTimeout

Sets the query time-out duration.

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

Name	Description
nInterval	The number of seconds before the time-out.

Throws

java.sql.SQLException

getFetchDirection

Returns the result set fetch direction.

```
public int getFetchDirection()
```

Returns

The fetch direction of the result set: FETCH_FORWARD, FETCH_REVERSE, or FETCH_UNKNOWN.

Throws

java.sql.SQLException

setFetchDirection

Sets the result set fetch direction.

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

Name	Description
iDir	The fetch direction of the result set: FETCH_FORWARD, FETCH_REVERSE, or FETCH_UNKNOWN.

Throws

java.sql.SQLExeption

getFetchSize

Returns the result set pre-fetch record count.

```
public int getFetchSize()
```

Returns

The fetch size of the current **Statement Agent** 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 result set agent 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 result set agent.

Throws

java.sql.SQLException

getMaxFieldSize

Returns the maximum field data size.

```
public int getMaxFieldSize()
```

Returns

The maximum number of bytes that a result set agent 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 result set agent.

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. A 0 means no rows were affected or the operation was a DDL command. A -1 means the result is a result set agent, 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 result set agent that was produced by the call to the method **execute()**.

Throws

java.sql.SQLExcetpion

getMoreResults

Returns whether there are more result sets.

```
public boolean getMoreResults()
```

Returns

true if the next result is a result set agent; otherwise **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 an SQL **Warning** object.

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

Returns

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

Throws

java.sql.SQLException

clearWarnings

Clears 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	A class array for types of formal arguments for the method, in the declared order. It can be null if there are no formal arguments. However, it cannot invoke a constructor here.
args	An object array of formal arguments for the method, in the declared order. It can be null if there are no formal arguments. However, it cannot invoke a constructor here.

Returns

The object instance resulting from the method invocation. It can be null if nothing is returned (void return declaration).

Throws

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

5.3 com.stc.eways.jdbcx.PreparedStatementAgent Class

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

This class hosts the **PreparedStatement** object.

All Implemented Interfaces

ResetEventListener, SessionEventListener

Direct Known Subclasses

CallableStatementAgent

```
public class PreparedStatementAgent
```

This class extends **StatementAgent**.

Methods of the PreparedStatementAgent Class

sessionOpen

Opens the session event handler.

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

Overrides

sessionOpen in the class **StatementAgent**.

Name	Description
evt	Session event.

setNull

Nullifies the value of an indexed parameter.

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

Name	Description
index	A parameter index starting from 1.
type	A JDBC type defined by inava.sql.Types

Throws

java.sql.SQLException

setNull

Nullifies the value of an indexed parameter.

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

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

Throws

java.sql.SQLException

setObject

Sets the value of an indexed parameter with an object.

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

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

Throws

java.sql.SQLException

setObject

Sets the value of an indexed parameter with an object.

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

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

Throws

java.sql.SQLException

setObject

Sets the value of an indexed parameter with an object.

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

Name	Description
index	A parameter index starting from 1.
ob	An instance of a Java object containing the input parameter value.
iType	A JDBC type defined by inava.sql.Types
iScale	The number of digits to the right of the decimal point and 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	A parameter index starting from 1.
b	The Boolean 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	A 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	A 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	A 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	A 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	A 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	A 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	A parameter index starting from 1.
dec	The decimal 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	A 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 the time zone from the calendar.

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

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

Throws

java.sql.SQLException

setTime

Sets the time value of the indexed parameter.

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

Name	Description
index	A 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	A 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 time stamp value of the indexed parameter.

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

Name	Description
index	A parameter index starting from 1.
ts	The time stamp parameter value to be set.

Throws

java.sql.SQLException

setTimestamp

Sets the time stamp 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	A parameter index starting from 1.
ts	The time stamp parameter value to be set.
cal	The calendar object used to construct the time stamp.

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	A parameter index starting from 1.
s	The string parameter value to be set.

Throws

java.sql.SQLException

setBytes

Sets the byte array value of the indexed parameter.

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

Name	Description
index	A 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	A parameter index starting from 1.
is	The input stream 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	A parameter index starting from 1.
is	The input stream 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	A 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	A 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	A 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	A parameter index starting from 1.
clob	The clob value to be set.

Throws

java.sql.SQLException

setRef

Sets the reference value of the indexed parameter.

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

Name	Description
index	A parameter index starting from 1.
ref	The reference 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 a prepared SQL statement.

```
public void execute()
```

Throws

java.sql.SQLException

executeQuery

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

```
public ResultSetAgent executeQuery()
```

Returns

The result set agent or a null.

Throws

java.sql.SQLException

executeUpdate

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

```
public int executeUpdate()
```

Returns

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

Throws

java.sql.SQLException

5.4 com.stc.eways.jdbcx.PreparedStatementResultSet Class

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

This is the base class for the result set returned from a prepared statement execution.

All Implemented Interfaces

```
public abstract class PreparedStatementResultSet
```

This class extends **java.lang.Object.**

Constructors of PreparedStatementResultSet

- **PreparedStatementResultSet**

Methods of the PreparedStatementResultSet Class

isBeforeFirst on page 101
isFirst on page 100
isLast on page 101
last on page 100
next on page 99
previous on page 99
refreshRow on page 141
relative on page 100
getFetchDirection on page 97
getFetchSize on page 98
updateRow on page 120
wasNull on page 119

Constructor PreparedStatementResultSet

Constructs a **Prepared Statement Result Set** object.

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

Name	Description
rsAgent	The result set agent's underlying control.

getMetaData

Retrieves a **Result Set Metadata** object that contains result set properties.

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

Returns

The **Result Set Metadata** object.

Throws

java.sql.SQLException

getConcurrency

Gets the concurrency mode for the current **Result Set** object.

```
public int getConcurrency()
```

Returns

The concurrency mode.

Throws

java.sql.SQLException

getFetchDirection

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

```
public int getFetchDirection()
```

Returns

The 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	The fetch direction to use.

Throws

java.sql.SQLException

getFetchSize

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

```
public int getFetchSize()
```

Returns

The number of rows to fetch at a time.

Throws

java.sql.SQLException

setFetchSize

Gives the drivers a hint as to the number of rows to fetch each time.

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

Name	Description
nSize	The number of rows to fetch at a time.

Throws

java.sql.SQLException

getCursorName

Retrieves the name of the cursor associated with the current **Result Set** object.

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

Returns

The name of cursor.

Throws

java.sql.SQLException

close

Immediately releases a **Result Set** 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; otherwise **false**.

Throws

java.sql.SQLException

previous

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

```
public boolean previous()
```

Returns

true if successful; otherwise **false**.

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; otherwise **false**.

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; otherwise **false**.

Throws

java.sql.SQLException

---

first

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

```
public boolean first()
```

Returns

true if successful; otherwise **false**.

Throws

java.sql.SQLException

---

isFirst

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

```
public boolean isFirst()
```

Returns

true if the cursor is on the first row; otherwise **false**.

Throws

java.sql.SQLException

---

last

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

```
public boolean last()
```

Returns

true if successful; otherwise **false**.

Throws

java.sql.SQLException

---

isLast

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

```
public boolean isLast()
```

Returns

true if the cursor is on the first row; otherwise **false**.

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 the cursor before the first row; otherwise **false**.

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 the cursor is after the last row; otherwise **false**

Throws

java.sql.SQLException

getType

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

```
public int getType()
```

Returns

The 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	The column name.

Returns

The 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	The column index.

Returns

The object form of the 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	The column index.

Returns

The object form of the 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	The column index.
map	The type map.

Returns

The object form of the 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	The column index.
map	The 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	The column index.

Returns

The 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	The column name.

Returns

The Boolean value of the column.

Throws

java.sql.SQLException

getByte

Gets the byte value of the specified column.

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

Name	Description
index	The column index.

Returns

The 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	The column index.

Returns

The 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	The column name.

Returns

The 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	The column index.

Returns

The integer 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	The column name.

Returns

The integer 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	The column index.

Returns

The 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	The column name.

Returns

The 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	The column index.

Returns

The 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	The column name.

Returns

The 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	The column index.

Returns

The double value of the column.

Throws

java.sql.SQLException

getBigDecimal

Gets the big decimal value of the specified column.

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

Name	Description
index	The column index.

Returns

The big decimal value of the column.

Throws

java.sql.SQLException

---

getBigDecimal

Gets the big decimal value of the specified column.

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

Name	Description
index	The column name.

Returns

The 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	The column index.

Returns

The 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	The column name.

Returns

The 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	The column name.
calendar	The calendar to use.

Returns

The 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	The column index.

Returns

The 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	The column name.

Returns

The 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	The column index.
calendar	Calendar to use.

Returns

The 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	The column name.
calendar	The calendar to use.

Returns

Time value of the column.

Throws

java.sql.SQLException

---

getTimestamp

Gets the time stamp value of the specified column.

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

Name	Description
index	The column index.

Returns

The time stamp value of the column.

Throws

java.sql.SQLException

---

getTimestamp

Gets the time stamp value of the specified column.

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

Name	Description
index	The column name.

Returns

The time stamp value of the column.

Throws

java.sql.SQLException

---

getTimestamp

Gets the time stamp 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	The column index.

Returns

The time stamp value of the column.

Throws

java.sql.SQLException

getTimestamp

Gets the time stamp 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	The column name.
calendar	The calendar to use.

Returns

The time stamp 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	The column index.

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	The column name.

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	The column index.

Returns

The 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	The column name.

Returns

The 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	The column index.

Returns

The 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	The column name.

Returns

The 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	The column index.

Returns

The 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	The column name.

Returns

The 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	The column index.

Returns

The reader for the 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	The column index.

Returns

The 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	The column index.

Returns

The 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	The column name.

Returns

The 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	The column index.

Returns

The 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	The column name.

Returns

The clob value of the column.

Throws

java.sql.SQLException

getRef

Gets the reference value of the specified column.

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

Name	Description
index	The column index.

Returns

The reference value of the column.

Throws

java.sql.SQLException

getRef

Gets the reference value of the specified column.

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

Name	Description
index	The column name.

Returns

The reference value of the column.

Throws

java.sql.SQLException

wasNull

Checks to see whether the last value read was SQL NULL.

```
public boolean wasNull()
```

Returns

true if the value was SQL NULL; otherwise **false**

Throws

java.sql.SQLException

getWarnings

Gets the first SQL warning that has been reported for the current object.

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

Returns

The SQL warning.

Throws

java.sql.SQLException

clearWarnings

Clears any warnings reported on the current object.

```
public void clearWarnings()
```

Throws

java.sql.SQLException

getRow

Retrieves the current row number in the result set.

```
public int getRow()
```

Returns

The current row number

Throws

java.sql.SQLException

refreshRow

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

```
public void refreshRow()
```

Throws

java.sql.SQLException

insertRow

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

```
public void insertRow()
```

Throws

java.sql.SQLException

updateRow

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

```
public void updateRow()
```

Throws

java.sql.SQLException

deleteRow

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

```
public void deleteRow()
```

Throws

java.sql.SQLException

5.5 com.stc.eways.jdbcx.SqlStatementAgent Class

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

This class hosts a managed **Statement** object.

All Implemented Interfaces

ResetEventListener, SessionEventListener

```
public class SqlStatementAgent
```

This class extends **StatementAgent**.

Constructors of the SqlStatementAgent

- **SqlStatementAgent**
- **SqlStatementAgent**

Methods of the SqlStatementAgent Class

addBatch on page 123
execute on page 122
executeQuery on page 122
executeUpdate on page 123

Constructor SqlStatementAgent

Creates a new **SQLStatementAgent** with the scroll direction TYPE_FORWARD_ONLY and the concurrency CONCUR_READ_ONLY.

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

Name	Description
session	The connection session.

Constructor SqlStatementAgent

Creates a new **SQLStatementAgent**.

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

Name	Description
session	Connection session.
iScroll	The scroll direction: TYPE_FORWARD_ONLY, TYPE_SCROLL_INSENSITIVE, or TYPE_SCROLL_SENSITIVE.
iConcur	The concurrency: CONCUR_READ_ONLY or 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 result set agent, but **false** if it is an integer.

Throws

java.sql.SQLException

executeQuery

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

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

Name	Description
sSql	SQL statement.

Returns

The result set agent or a null.

Throws

java.sql.SQLException

executeUpdate

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

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

Name	Description
sSql	SQL statement.

Returns

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

Throws

java.sql.SQLException

addBatch

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

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

Name	Description
sSql	SQL statement.

Throws

java.sql.SQLException

5.6 com.stc.eways.jdbcx.CallableStatementAgent Class

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

The class hosts the **CallableStatement** interface.

All Implemented Interfaces

ResetEventListener, SessionEventListener

Direct Known Subclasses

StoredProcedureAgent

```
public abstract class CallableStatementAgent
```

This class extends **PreparedStatementAgent**.

Constructors of the CallableStatementAgent

- **CallableStatementAgent**
- **CallableStatementAgent**
- **CallableStatementAgent**

Methods of the CallableStatementAgent Class

Constructor CallableStatementAgent

Creates a new **CallableStatementAgent** with the scroll direction TYPE_FORWARD_ONLY and the concurrency CONCUR_READ_ONLY.

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

Name	Description
session	The 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	The 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	The scroll direction: TYPE_FORWARD_ONLY, TYPE SCROLL_INSENSITIVE, or TYPE_SCROLL_SENSITIVE
iConcur	The concurrency: CONCUR_READ_ONLY or CONCUR_UPDATEABLE

sessionOpen

Opens the session event handler.

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

Overrides

The **sessionOpen()** method in the class **PreparedStatementAgent**.

Name	Description
evt	The session event.

registerOutParameter

Registers the indexed OUT parameter with the specified type.

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

Name	Description
index	The parameter index starting from 1.
iType	A JDBC type defined by inava.sql.Types.

Throws

java.sql.SQLException

registerOutParameter

Registers the indexed OUT parameter with the specified type and scale.

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

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

Throws

java.sql.SQLException

registerOutParameter

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

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

Name	Description
index	The parameter index starting from 1.

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

Throws

java.sql.SQLException

wasNull

Returns whether 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 an object.

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

Name	Description
index	The parameter index starting from 1.

Returns

The object's value.

Throws

java.sql.SQLException

getObject

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

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

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

Returns

The object's value.

Throws

java.sql.SQLException

getBoolean

Gets the Boolean value of the indexed parameter.

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

Name	Description
index	The parameter index starting from 1.

Returns

The Boolean value.

Throws

java.sql.SQLException

getByte

Gets the byte value of the indexed parameter.

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

Name	Description
index	The parameter index starting from 1.

Returns

The byte value.

Throws

java.sql.SQLException

getShort

Gets the short value of the indexed parameter.

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

Returns

The short value.

Throws

java.sql.SQLException

getInt

Gets the integer value of the indexed parameter.

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

Name	Description
index	The parameter index starting from 1.

Returns

A integer value.

Throws

java.sql.SQLException

getLong

Gets the long value of the indexed parameter.

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

Name	Description
index	The parameter index starting from 1.

Returns

The long value.

Throws

java.sql.SQLException

getFloat

Gets the float value of the indexed parameter.

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

Name	Description
index	The parameter index starting from 1.

Returns

The float value.

Throws

java.sql.SQLException

getDouble

Gets double value of the indexed parameter.

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

Name	Description
index	The parameter index starting from 1.

Returns

The double value.

Throws

java.sql.SQLException

getBigDecimal

Gets the big decimal value of the indexed parameter.

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

Name	Description
index	The parameter index starting from 1.

Returns

The big decimal value as an object.

Throws

java.sql.SQLException

getDate

Gets date value of the indexed parameter.

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

Name	Description
index	The parameter index starting from 1.

Returns

The date value as an 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	The parameter index starting from 1.
cal	The Calendar object used to construct the time stamp.

Returns

The date value as an object.

Throws

java.sql.SQLException

getTime

Gets time value of the indexed parameter.

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

Name	Description
index	The parameter index starting from 1.

Returns

The time value as an 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	The parameter index starting from 1.
cal	The Calendar object used to construct the time stamp.

Returns

The time value as an object.

Throws

java.sql.SQLException

getTimestamp

Gets time stamp value of the indexed parameter.

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

Name	Description
index	The parameter index starting from 1.

Returns

The time stamp value as an object.

Throws

java.sql.SQLException

getTimestamp

Gets the time stamp value of the indexed parameter.

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

Name	Description
index	The parameter index starting from 1.
cal	The Calendar object used to construct the time stamp.

Returns

The time stamp value as an object.

Throws

java.sql.SQLException

getString

Gets string value of the indexed parameter.

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

Name	Description
index	The parameter index starting from 1.

Returns

The string value as an object.

Throws

java.sql.SQLException

getBytes

Gets byte array value of the indexed parameter.

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

Name	Description
index	The parameter index starting from 1.

Returns

The byte array value.

Throws

java.sql.SQLException

getArray

Gets the array value of the indexed parameter.

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

Name	Description
index	The parameter index starting from 1.

Returns

The array value as an object.

Throws

java.sql.SQLException

getBlob

Gets the blob value of the indexed parameter.

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

Name	Description
index	The parameter index starting from 1.

Returns

The blob value as an object.

Throws

java.sql.SQLException

getClob

Gets the clob value of the indexed parameter.

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

Name	Description
index	The parameter index starting from 1.

Returns

The clob value as an object.

Throws

java.sql.SQLException

getRef

Gets the reference value of the indexed parameter.

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

Name	Description
index	The parameter index starting from 1.

Returns

The reference value as an object.

Throws

java.sql.SQLException

5.7 com.stc.eways.jdbcx.TableResultSet Class

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

The class maps selected records of a table in the database.

All Implemented Interfaces

ResetEventListener, SessionEventListener

```
public abstract class TableResultSet
```

This class extends **StatementAgent**.

Methods of the TableResultSet Class

select

Selects table records.

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

Name	Description
sWhere	The where condition for the query.

Throws

java.sql.SQLException

next

Navigates one row forward.

```
public boolean next()
```

Returns

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

Throws

java.sql.SQLException

previous

Navigates one row backward. Call this method only on the **Result Set Agent** 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

Moves the cursor to the specified row number. Call this method only on the **Result Set Agent** 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

Moves the cursor forward or backward a specified number of rows. Call this method only on the **Result Set Agent** 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 backward.

Returns

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

Throws

java.sql.SQLException

first

Moves the cursor to the first row of the result set. Call this method only on the **Result Set Agent** 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

Checks whether the cursor is on the first row. Call this method only on the **Result Set Agent** 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

Moves to the last row of the result set. Call this method only on the **Result Set Agent** 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. Call this method only on the **Result Set Agent** 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

Moves the cursor before the first row. Call this method only on the **Result Set Agent** objects that are TYPE_SCROLL_SENSITIVE or TYPE_SCROLL_INSENSITIVE.

```
public void beforeFirst()
```

Throws

java.sql.SQLException

isBeforeFirst

Checks whether the cursor is positioned before the first row. Call this method only on the **Result Set Agent** 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

Moves the cursor after the last row. Call this method only on the **Result Set Agent** 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. Call this method only on the **Result Set Agent** 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 a **BinaryStream**.

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

Throws

java.sql.SQLException

getBinaryStream

Returns the column data as a **BinaryStream**.

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

Throws

java.sql.SQLException

getCharacterStream

Returns the column data as a **CharacterStream**.

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

Throws

java.sql.SQLException

getCharacterStream

Returns the column data as a **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 whether the current row has been inserted.

```
public boolean rowInserted()
```

Returns

true if the current row has been inserted; otherwise **false**.

Throws

java.sql.SQLException

rowUpdated

Returns whether the current row has been updated.

```
public boolean rowUpdated()
```

Returns

true if the current row has been updated; otherwise **false**.

Throws

java.sql.SQLException

rowDeleted

Returns whether the current row has been deleted.

```
public boolean rowDeleted()
```

Returns

true if the current row has been deleted; otherwise **false**.

Throws

java.sql.SQLException

wasNull

Returns whether the last data retrieved is NULL.

```
public boolean wasNull()
```

Returns

true if the last data retrieved is NULL; otherwise **false**.

Throws

java.sql.SQLException

5.8 com.stc.eways.jdbcx.Session Class

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

The class hosts a managed connection interface.

All Implemented Interfaces

```
public class Session
```

This class extends **java.lang.Object.**

Methods of the Session Class

setDbConnector

Sets the pointer that points back to the location of the associated database connector.

```
protected void setDbConnector(com.stc.eways.jdbcx.DbConnector dbConn)
```

Name	Description
DbConnector	The associated DbConnector.

connect

Connects to a database.

```
protected void connect()
```

Throws

java.sql.SQLException

open

Opens the session automatically in non-retro mode. If the session is in the retro mode, the method throws the exception.

```
public void open()
```

Throws

java.sql.SQLException

_open

Opens the session.

```
protected void _open()
```

Throws

java.sql.SQLException

releaseResources

Releases resources.

```
protected void releaseResources()
```

Throws

java.sql.SQLException

close

Closes the session.

```
public void close()
```

Throws

java.sql.SQLException

isClosed

Returns the session open status.

```
public boolean isClosed()
```

Throws

java.sql.SQLException

isPooled

Returns whether the session originates from a pooled connection.

```
public boolean isPooled()
```

Returns

true if the session is from a pool; otherwise **false**.

getAutoCommit

Retrieves the session auto commit property.

```
public boolean getAutoCommit()
```

Returns

The session's auto commit property.

Throws

java.sql.SQLException

setAutoCommit

Sets the session's auto commit property.

```
public void setAutoCommit(boolean bAuto)
```

Name	Description
bAuto	The auto commit property.

Throws

java.sql.SQLException

getCatalog

Returns the session's catalog property.

```
public java.lang.String getCatalog()
                    throws java.sql.SQLException
```

Throws

java.sql.SQLException

setCatalog

Sets the session's catalog property.

```
public void setCatalog(java.lang.String sCatalog)
```

Name	Description
sCatalog	The name of the session catalog.

Throws

java.sql.SQLException

isReadOnly

Returns the session's read-only status.

```
public boolean isReadOnly()
```

Throws

java.sql.SQLException

setReadOnly

Sets the session's read-only property.

```
public void setReadOnly(boolean bRead)
```

Throws

java.sql.SQLException

getTypeMap

Returns the session's type map.

```
public java.util.Map getTypeMap()
```

Throws

java.sql.SQLException

setTypeMap

Sets the session's type map.

```
public void setTypeMap(java.util.Map map)
```

Name	Description
map	The desired session type map.

Throws

java.sql.SQLException

getMetaData

Returns the session's database metadata.

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

Throws

java.sql.SQLException

getTransactionIsolation

Returns the session's transaction isolation level.

```
public int getTransactionIsolation()
```

Throws

java.sql.SQLException

setTransactionIsolation

Sets the session's transaction isolation level.

```
public void setTransactionIsolation(int iLevel)
```

Name	Description
iLevel	The session transaction isolation level number.

Throws

java.sql.SQLException

commit

Commits the session's transaction.

```
public void commit()
```

Throws

java.sql.SQLException

rollback

Rolls back the session's transaction.

```
public void rollback()
```

Throws

java.sql.SQLException

getDBMS

Retrieves the DBMS type.

```
public int getDBMS()
```

Returns

The DBMS type of the current session.

See Also

DBMS_TYPE_UNKNOWN, DBMS_TYPE_ORACLE, DBMS_TYPE_SYBASE, DBMS_TYPE_MSSQL, DBMS_TYPE_DB2, DBMS_TYPE_SEQUELINK, DBMS_TYPE_CROSSACCESS, and DBMS_TYPE_ATTUNITY.

getSupportsBatch

Registers an ETD Reset Event listener.

```
public boolean getSupportsBatch()
```

addResetEventListener

Adds an ETD Reset Event listener.

```
public void addResetEventListener
    (com.stc.eways.jdbcx.ResetEventListener l)
```

Name	Description
listener	A Reset Event listener.

removeResetEventListener

Unregisters an ETD Reset Event listener.

```
public void removeResetEventListener
    (com.stc.eways.jdbcx.ResetEventListener l)
```

Name	Description
listener	A Reset Event listener.

requestReset

Requests that a reset be done on the data content of a database ETD.

```
public void requestReset(com.stc.eways.jdbcx.ResetEvent revt)
```

Name	Description
revt	The target ETD.

connInvoke

Invokes a method of the database **Connection** object of an ETD.

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

Name	Description
methodName	The name of the method.
argsCls	The class array for types of formal arguments for the method, in the declared order. This value can be null if there are no formal arguments. However, it cannot invoke a constructor here.
args	The object array of formal arguments for the method in the declared order. This value can be null if there are no formal arguments. However, it cannot invoke a constructor here.

Returns

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

Throws

java.lang.Exception for whatever exception the invoked method throws.

getDbConnection

Gets the database **Connection** object of this ETD.

```
protected java.sql.Connection getDbConnection()
```

Returns

The database **Connection** object of this ETD.

setResultSetType

Sets the result set type.

```
protected void setResultSetType(int resultSetType)
```

Name	Description
resultSetType	The desired result set type.

getResultSetType

Gets the result set type.

```
protected int getResultSetType()
```

Returns

The result set type of the current session.

setConcurrencyType

Sets the concurrency type.

```
protected void setConcurrencyType(int concurrencyType)
```

Name	Description
concurrencyType	The desired concurrency type.

getConcurrencyType

Gets the concurrency type.

```
protected int getConcurrencyType()
```

Returns

The concurrency type of the current session.

setNewTypeFlag

Sets whether the current result set is a new result set type.

```
protected void setNewTypeFlag(boolean bNew)
```

Name	Description
bNew	The flag to indicate whether the result set type is modified.

getNewTypeFlag

Checks whether the current result set is a new result set type.

```
protected boolean getNewTypeFlag()
```

Returns

A Boolean value that indicates whether the result set type has been modified.

isOracleNativeForClob

Checks whether the database driver is an Oracle native driver with clob support.

```
protected boolean isOracleNativeForClob()
```

Returns

A Boolean value that indicates whether the driver is an Oracle native driver with clob support.

isRetroMode

Check whether the current mode is a retro mode, that is, a mode that does not have the Connection Manager's support.

```
protected boolean isRetroMode()
```

Returns

A Boolean value that indicates a mode with no Connection Manager support.

setLastActivityTime

Informs the database connector to set the time of the last activity.

```
protected void setLastActivityTime(long t)
```

Name	Description
t	The time value of the last activity.

setBrokenConnection

Breaks the database connection.

```
public void setBrokenConnection(boolean flag)
```

Name	Description
flag	The broken connection flag.

Index

U

W