StarSQL™ for Java User's Guide

Version 2.7

Statement of Limitations on Warranty & Liability

StarQuest Ventures makes no representations or warranties about the suitability of the software and documentation, either expressed or implied, including but not limited to the implied warranties of merchantability, fitness for a particular purpose, or non-infringement. StarQuest Ventures shall not be liable for any damages suffered by licensee as a result of using, modifying, or distributing this software or its derivatives.

v2.7_01.03.2020

Contents

Introduction 7
Overview of JDBC 7
Overview of StarSQL for Java 9
Automatic Package Binding 9
Data Source Support 10
DISTINCT UDT Support 10
LOB Data Type Support 10
Java EE Support 10
Derby Support 10
SSL Support 11
Global Transaction Support 11
Scrollable Cursor Support 11
Encrypted Password Support 11
European Date and Decimal Notation Support 12
International Language Support 12
Tracing and Logging Information 12
StarSQL for Java Operating Environment 12
System Requirements 12
Java Environment 12
Host Database Management Systems 13
Common Network Topologies 13
Documentation 16
Quick Path to Using StarSQL 16
StarSQL Product Documentation 16
User's Guide 16
Release Notes 16
Contacting StarQuest 17
Support 17
Sales and Service 17

Installing StarSQL for Java 19
Installation on a UNIX-Based Computer 19
Installation on AIX, HP-UX, Linux, or Solaris 19

Installation on Mac OS X 20
Installation on z/OS 20
Installation on IBM i 20
PASE 20
QSHELL 21
Installation on a Windows-Based Computer 21
Setting the Classpath and Environment Variables 21
Setting the Classpath 22
Setting Environment Variables 22
Licensing StarQuest Products 23
Node-Locked License 23
Floating License 24
Configuring a License 24
License Configuration File Format 25
Configuring a Client to Use a Floating License 26
Configuring a Client to Use a Node-Locked License 28
Online Licensing 28
Manually Adding a License Key 28

Using StarSQL for Java 29

Configuring and Loading the StarSQL for Java Driver 29
StarSQL for Java Configuration Properties 29
accounting Parameter 32
catalogFilter Parameter 32
collection Parameter 32
commitProcedureCall Parameter 33
createTable Parameter 33
databaseName Parameter 34
dateFormat Parameter 34
decimalDelimiter Parameter 35
defaultQualifier Parameter 35
description Parameter 35
diagnosticsLevel Parameter 36
drdaTrace Parameter 36
dynamicRules Parameter 36
fullyMaterializeCLOB Parameter 37
heldCursors Parameter 37
keepDynamic Parameter 38
newPassword Parameter 38
password Parameter 38
portNumber Parameter 38
productID Parameter 38

pwdEncryption Parameter . . . 39
sendUnicode Parameter . . . 39
serverName Parameter . . . 40
ssl Parameter . . . 40
typdefovr Parameter . . . 40
user Parameter . . . 40
Using the JDBC Driver Interface . . . 41
Using the Connection URL . . . 41
Using Data Sources . . . 42
Displaying StarSQL Driver Information . . . 44
Testing a StarSQL/DRDA Connection . . . 44
The StarSQL for Java Sample Applications . . . 46
Running the Sample Applications . . . 47
Running the CreateDS Sample Application . . . 47
Specifying Connection Information . . . 51
Running the CatalogApp Sample Application . . . 52
Running the LobTestApp Sample Application . . . 54
Running the QueryApp Sample Application . . . 59
Building the Sample Applications . . . 62
Building the Sample Applications Using a Batch or Script File . . . 62
Building the Sample Applications Using an IDE . . . 63
Managing Two-Phase Commit Transactions . . . 66

Preparing Hosts for StarSQL Access . . . 69
Preparation Required for All Hosts . . . 69
User Accounts . . . 69
Permissions . . . 70
Preparing DB2 on a z/OS Host . . . 70
Configuring DDF . . . 70
Starting DDF . . . 71
Supporting Password Management Using DRDA Flows . . . 72
Using StarSQL with Stored Procedures . . . 72
Registering Stored Procedures . . . 72
Calling Stored Procedures . . . 73
Preparing a DB2 for i Host . . . 74
Creating a Library for SQL Packages . . . 74
Determining the Database Name . . . 74
Enabling DRDA Over TCP/IP . . . 75
Registering Stored Procedures on DB2 for i . . . 76
Configuring Support for the SSL Protocol . . . 76
Considerations for Specific IBM i Releases . . . 77
IBM i 6.1 Issues . . . 77

OS/400 v5r4 Issues 77
Preparing a DB2 LUW Host. 78
Enabling DRDA Support for TCP/IP. 78
Using db2 Commands to Specify the DRDA Port 78
Enabling Encryption 79
Using db2 Commands to Enable Encryption 79
Locating the Database Name 80
Preparing a Derby Network Server Host 80
Setting Network Server Properties. 80
Enabling Remote Connections 81
Configuring Support for the SSL Protocol 81
Configuring the StarSQL for Java for Derby Connections 82
Preparing a DB2 Server for VSE & VM 82

Binding Packages 85
Using Dynamic SQL Packages 85
Permissions Required for Packages 88
For Binding Packages 88
For Using Packages 89
Granting Use Permissions 90
For DB2 for z/OS 90
For DB2 for LUW (Linux, UNIX & Windows) 90
For DB2 for i. 90
Derby Held Cursors Packages 91
National Language Support 93
Determining the Default Character Set of the Client 94
Determining Which Character Set Conversions are Supported 94
Troubleshooting Tips and Techniques 101
Troubleshooting Communication Problems 101
Configuring the Logging Facility 102
Disabling the Logging Facility 105
Sample Logging Configuration File 105
Optimizing Java on an IBM i Computer 106

Glossary 109

CHAPTER 1

Introduction

StarSQL is available as an ODBC driver for Windows- and UNIX-based computers, and as a JDBC driver for any computer that has the Java Runtime Engine (JRE) or Java Virtual Machine (JVM) installed.

The StarSQL software is not copy protected, rather the usage is limited based upon the maximum number of concurrent connections ("CCs") licensed. The StarQuest license allows for you to install and run any of the StarSQL drivers on any number of client computers, subject to terms of the license grant. CCs may be made available for clients on a single computer ("Node-locked License") or any computer on the network ("Floating License").

The StarSQL for Java driver is a type 4 (Direct-to-Database pure Java) JDBC driver that provides direct access to the entire range of IBM DB2 database systems. It converts JDBC calls into the Distributed Relational Database Architecture (DRDA) protocol used by IBM DB2 host database systems. It uses the TCP/IP network protocol to communicate directly to host systems that implement DRDA over TCP/IP.

StarSQL for Java provides a native-protocol pure Java driver ("type 4" JDBC driver) that implements Oracle's JDBC application program interface (API). This version of StarSQL for Java supports the JDBC 3.0 API.

Overview of JDBC

JDBC is a Java API for executing SQL statements, which provides connectivity between Java-based applications or applets to database systems. The Java API consists of a set of classes and interfaces written in the Java programming language.

JDBC provides a standard API, based on the ANSI SQL-92 standard, that allows database developers to directly invoke SQL commands. The JDBC API is expressed as a series of abstract Java interfaces that allow an application programmer to open

connections to particular databases, execute SQL statements, and process the results. The StarSQL for Java driver provides the implementations of the abstract classes provided by the JDBC API.

There are four types of JDBC drivers, as described in Table 1 on page 8.

Table 1. JDBC Driver Types

Driver Type	Description	Pure Java	Net Protocol
Type 1: JDBC-ODBC Bridge	Provides JDBC access via ODBC drivers. Note that the Oracle JDBC-ODBC Bridge has been deprecated as of JRE 8.	No	Direct
Type 2: Native-API	Converts JDBC calls into calls on the client API for a specific DBMS. The client code typically is a library of platform-specific code (such as C or C++) that is accessed via the Java Native Interface (JNI).	No	Direct
Type 3: JDBC-Net	Translates JDBC calls into a DBMS-independent net protocol, which a server then translates to a DBMS protocol. The server could be any of the four driver types. The net server middleware connects the Java clients to a variety of databases.	Yes	Requires connector (RMI, CORBA, etc.)
Type 4: Native Protocol	Converts JDBC calls into the network protocol used by specific database systems, which allows client applications direct access to the database system.	Yes	Direct

A type 4 native-protocol/all-Java driver provides significant advantages over the other driver types. Since there is no translation of database requests to ODBC or a native connectivity interface, a type 4 driver provides better performance than types 1 and 2. And a type 4 driver requires no special software on the client or host as with a type 3 driver. The StarSQL for Java driver provides fast, direct, platform-independent communication between client applications and DRDA host database systems.

Overview of StarSQL for Java

StarSQL for Java allows communication with any host that supports DRDA protocols over TCP/IP. StarSQL for Java supports the JDBC 3.0 API, and includes the following major features:

- automatic package binding
- support for data sources
- support for the DISTINCT user-defined type (UDT)
- support for LOB (large object) data types
- support for the Java Enterprise Edition (Java EE)
- support for Derby, the Apache Software Foundation's open-source relational database
- support for the Secure Sockets Layer (SSL) protocol
- support for global (two-phase) transactions
- support for scrollable cursors
- support for encrypted passwords
- support for European date formats and decimal notation in numeric values
- international language support
- extensive DRDA tracing and logging information

This release of StarSQL for Java supports stored procedures but does not support static SQL.

Automatic Package Binding

StarSQL for Java uses the same dynamic SQL packages on the host as the StarSQL ODBC driver, unless you want to use features that are unique to the Java driver, such as the dateFormat property (see "dateFormat Parameter" on page 34). On the initial connection with the host, the StarSQL for Java driver searches for the required packages and, if it does not find them, it creates them automatically. The names and locations of the packages that are bound varies, depending on the configuration settings in the JDBC data source that are in effect when the initial connection is made.

Packages also can be created explicitly using the StarAdmin utility that is available from StarQuest as a separate download.

Data Source Support

An application can use a data source to load StarSQL for Java and connect to a database. A data source is associated with a JDBC provider that supplies the specific JDBC driver implementation class. You can create multiple data sources that are associated with the same JDBC provider. See "Using Data Sources" on page 43 for more information about defining data sources.

DISTINCT UDT Support

StarSQL for Java supports the SQL3 DISTINCT user-defined type (UDT). A `DISTINCT` data type shares its internal representation with a built-in data type—the source type to which it is mapped. Typically, a `SQLData` implementation defines a field for each attribute of a SQL structured type or a single field for a SQL `DISTINCT` type. When a `DISTINCT` data type is retrieved from a data source, it is mapped as an instance of the `SQLData` class, and can be manipulated like other objects in the Java programming language.

LOB Data Type Support

Large objects (LOBs) can contain text documents, images, or even movies, and can be stored directly in the DBMS. The StarSQL for Java driver supports working with DB2 LOB data types.

Java EE Support

The Java Enterprise Edition (Java EE) technology and its component-based model provide Web services and an infrastructure that can help to integrate enterprise applications. StarSQL for Java supports the Java EE environment, allowing you to use it in conjunction with popular Web servers such as IBM WebSphere® and Oracle/BEA WebLogic®. You can configure data sources using the Web server that allow Enterprise Java Beans (EJBs) to use StarSQL for Java to communicate with the DB2 system.

Derby Support

Derby is a full-featured, open source relational database system that is based on Java and SQL standards. IBM contributed the original Java database, Cloudscape, as open source to the Apache Software Foundation (ASF) in 2004 and continues to include it with some of its Web-oriented software bundles. The ASF open-source Apache Derby project flourished with support from community members such as IBM, Sun Microsystems, and Oracle to receive widespread adoption. A supported implementation

of Derby is available from Oracle as Java DB. See http://www.oracle.com/technetwork/java/javadb/documentation/index.html and http://db.apache.org/derby/ for more information.

SSL Support

The StarSQL for Java driver can be configured to support the Secure Sockets Layer (SSL) protocol. The SSL protocol operates above TCP/IP and below higher-level protocols such as HTTP, LDAP, or IMAP. It allows an SSL-enabled server to authenticate itself to an SSL-enabled client, allows the client to authenticate itself to the server, and allows both computers to establish an encrypted connection. To enable the StarSQL for Java driver to use SSL protocol, configure the SSL property, as described in "StarSQL for Java Configuration Properties" on page 29. You also need to configure the host system to use the SSL protocol, as described in the chapter "Preparing Hosts for StarSQL Access" on page 69.

Global Transaction Support

In a distributed processing environment where a unit of work spans more than one database, using the two-phase commit protocol can help ensure the integrity of the transaction and databases. When the two-phase commit protocol is in effect, the transaction must complete successfully before it is committed. If the transaction fails, all of the updates are rolled back and the databases remain in the state they were before the transaction was begun. StarSQL for Java includes a Transaction Log Manager application to help you manage two-phase commit transactions over a TCP/IP network, as described in "Managing Two-Phase Commit Transactions" on page 67.

Scrollable Cursor Support

StarSQL for Java supports scrollable cursors to DB2 for z/OS and to DB2 LUW.

Encrypted Password Support

The default behavior of StarSQL for Java is to send passwords in clear text to the host computer. Many versions of DB2 support password encryption or successfully negotiate the use of password encryption (see "StarSQL for Java Operating Environment" on page 12 for details about which versions). If the host database supports encrypted passwords, you can enable the password encryption feature as described in "pwdEncryption Parameter" on page 39.

European Date and Decimal Notation Support

StarSQL for Java supports applications that use European date formats in SQL strings and the getString() and setString() methods, as described for the "dateFormat Parameter" on page 34. It also supports using a comma as the decimal notation in numeric data types, as explained for the "decimalDelimiter Parameter" on page 35.

International Language Support

StarSQL provides character set conversion to support most languages. "National Language Support" on page 93 lists the supported languages.

Tracing and Logging Information

StarSQL for Java allows you to capture trace and log information about the DRDA and API communications. To trace DRDA operations, you load the StarSQL for Java driver with the drdaTrace property enabled, as described in "drdaTrace Parameter" on page 37. The section "Configuring the Logging Facility" on page 102 describes how to configure the logging facility to capture the desired level of information to a file and/or to display logging messages on the computer console.

StarSQL for Java Operating Environment

You can install and use the StarSQL for Java driver on any computer that has the Java Development Kit (JDK) or Java Runtime Environment (JRE) installed. The data accessed through StarSQL for Java resides in a DB2 database on a host computer. StarSQL for Java requires a physical network and network software for communication between the desktop computer and the host.

System Requirements

This section describes the Java environment and the host systems that can be used with the StarSQL for Java driver.

Java Environment

StarSQL for Java runs on any system with a Java Virtual Machine (JVM) that is equivalent to that contained in the Java Development Kit (JDK) or the Java Runtime Environment (JRE) v1.5 or later.

You can download the latest JRE from http://www.oracle.com/technetwork/java/index.html or use another compatible JRE or JDK, such as the IBM JRE.

Host Database Management Systems

StarSQL for Java can connect to any of the following host databases:

- Apache Derby (JavaDB) 10.3 or later
- DB2 for z/OS v9 & later
- DB2 for i (formerly known as DB2/400, DB2 UDB for iSeries, and DB2 for i5/OS) v5r4 & later
- DB2 for Linux, UNIX, and Windows (LUW) v9.7 and later

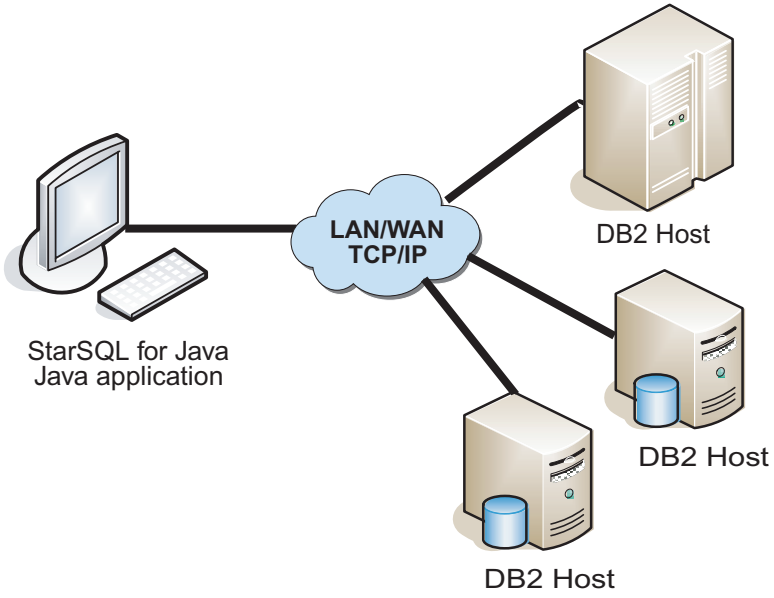
"Preparing Hosts for StarSQL Access" on page 69 provides information on configuring the host system.

Common Network Topologies

The following diagrams illustrate some common network topologies used with StarSQL for Java.

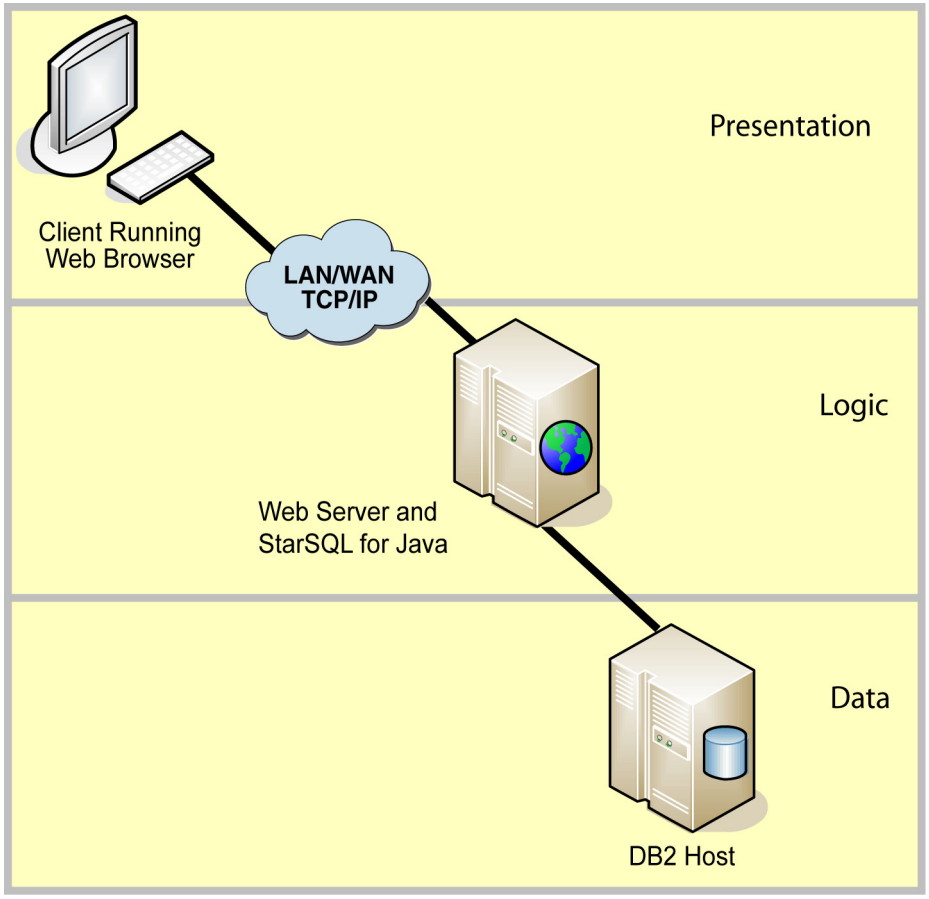
Figure 1 shows StarSQL for Java installed on a client computer for use with a locally installed Java application. In this scenario StarSQL for Java can access data source configuration information on the client computer to connect to any host that supports DRDA over TCP/IP.

Figure 1. StarSQL for Java on Client Computer



As depicted in Figure 2, the StarSQL for Java driver also can used in a three-tier architecture, in which the user interface (Presentation layer), the functional process logic (Logic layer), and the data storage and access (Data layer) are developed and maintained as independent modules. With this configuration the StarSQL for Java driver is downloaded with a Java applet from an application web server such as Apache, ColdFusion, JBoss, or WebSphere. This client-server architecture restricts the client connection to the host from which the applet is downloaded due to Java security constraints on downloaded applets.

Figure 2. StarSQL for Java Provides Middleware for 3-Tier Architecture



Documentation

There are many sources of information that can help you install, configure, and use the StarSQL driver. The following sections describe the information available from StarQuest and provides references to other information that may be particularly useful.

Quick Path to Using StarSQL

StarQuest provides StarSQL Quick Start Guides that provide step-by-step instructions for quickly installing and using the StarSQL ODBC and JDBC driver on a particular computing platform. The procedures in the Quick Start Guides are appropriate for the most common environments and describe the fastest way to install and configure the software you need to begin using the driver. If you need to customize the StarSQL driver settings or have an environment for which the default values are not appropriate you can refer to the product documentation for details.

All the Quick Start Guides are listed at http://west.comww.starqu/docs/Supportdocs/browseQuickStarts.shtml.

StarSQL Product Documentation

The StarSQL for Java product documentation consists of the following components:

- this User's Guide
- Release Notes
- Technical documents (Info Center website)

User's Guide

This User's Guide provides information about installing the StarSQL software, and configuring a client license to use the driver. It also describes how to use the StarSQL driver and the utilities and programs that are included with it. Licensing is managed by StarLicense server software, which includes separate documentation for installing and configuring a StarLicense server.

Release Notes

The Release Notes contain important information about using StarSQL for Java in specific environments, known limitations or issues, and a history of changes to the driver software.

Contacting StarQuest

Please use the following methods to contact StarQuest Ventures if you need to obtain a license key, or have suggestions or need information about StarQuest products.

Support

To obtain a license key for your product, send an email to support@starquest.com with the following information:

- TCP/IP address or Host ID of the computer on which the license will be installed
- Number of connections purchased
- Company Name
- Contact Name
- Phone Number
- Email Address

StarQuest Support will send a reply email that provides the license key for your organization's use of the product. Since the license is unique to the computer on which it will be installed, you must contact StarQuest should you need to move the license from one computer to another.

Additional technical support may be available subject to the prices, terms, and conditions specified in your organization's maintenance contract with StarQuest Ventures, Inc.

Sales and Service

If you have ideas for product enhancements or need more information about how StarQuest products can provide solutions for connecting Mac OS X, Windows, and UNIX applications to IBM host resources, please contact us via any of the following methods.

Address	StarQuest Ventures, Inc. 548 Market St, #22938 San Francisco, CA 94104-5401
Telephone	415-669-9619 Option 1: Sales Option 2: Technical Support

Fax	415-669-9639
Email	support@starquest.com
World Wide Web	www.starquest.com support.starquest.com

CHAPTER 2

Installing StarSQL for Java

This chapter describes how to install StarSQL for Java and set the environment variables required for using the driver. As a pure Java, type 4 driver, StarSQL for Java typically is installed on a client machine as a .jar file. When used with locally installed Java applications, StarSQL for Java can access the client file system to retrieve data source configuration information. It also can connect to any host that supports DRDA over TCP/IP. StarSQL for Java can be downloaded with a Java applet by a Web browser, however, security constraints placed on applets might restrict the driver's access to just the host from which it was downloaded.

Follow the instructions below, as appropriate to the type of operating system the computer is running, to install StarSQL for Java. Be sure to review the Release Notes included in the distribution for important information about installing or upgrading the StarSQL for Java driver.

Installation on a UNIX-Based Computer

The procedures for installing StarSQL for Java on a UNIX-based computer vary depending on the version of UNIX and the type of computer. The following sections describe how to install StarSQL for Java for the various UNIX platforms.

Installation on AIX, HP-UX, Linux, or Solaris

StarSQL for Java is distributed as an rpm file for installation by the RPM Package Manager, and as a tar file that uses a `setup` script for installation on UNIX or Linux computers that do not support RPM. To use either of these methods you run the `setup` shell script. The `setup` script extracts the contents of the `starjdbc.tar` file or invokes rpm when using the RPM installer.

1. Edit the `setup` shell script if you want to install the StarSQL JDBC driver to a location other than the default directory (`/usr/lpp/starsql_java` for AIX, `/usr/share/starsql_java` for Linux, `/opt/starsql_java` for HP-UX or Solaris).

2. Copy the files **setup** and **starsql_jdbc.tar** to the UNIX computer.
3. Enter the following command to run the **setup** shell script.

```
# ./setup
```

After installing the driver, you can delete the `setup` and `starsql_jdbc.tar` files.

Installation on Mac OS X

To install StarSQL for Java on a Mac OS X computer, use the Finder to open the folder that contains the UNIX installer and double-click **setup-mac.command**. Alternatively, you can open a Terminal window from Applications/Utilities and use the standard UNIX installation and operation procedures. The default directory for installing StarSQL for Java on a Mac OS X computer is `/Applications/starsql_java`.

Installation on z/OS

To install StarSQL for Java under UNIX System Services on IBM OS/390 or z/OS, use the UNIX installer and run **setup.ebcdic**. The installer converts text files, such as shell scripts and Java source to EBCDIC so they can be used by OS/390 or z/OS.

1. Copy the files **setup.ebcdic** and **starsql_jdbc.tar** to the computer.
2. Enter the following command to run the **setup.ebcdic** shell script.

```
# ./setup.ebcdic
```

After installing the driver, you can delete the `setup.ebcdic` and `starsql_jdbc.tar` files.

Installation on IBM i

On an IBM i computer you can use either QSHELL or the Portable Application Solutions Environment (PASE) to extract the contents of the StarSQL for Java tarfile without conversion, as shown below. Do *not* use the **tar** command with QSHELL as it will corrupt the JAR files by attempting to convert the binary files.

PASE

1. Enter the command **CALL QP2TERM** to start PASE.
2. Enter the command **tar xf /tmp/starsql_java_unix.tar** to extract the distribution.
3. Using the UNIX installer, enter the following command to run the **setup** shell script.

```
# ./setup
```

QSHELL

1. Enter the command **QSH** or **STRQSH** to start QSHELL.
2. Enter the command **pax -C 819 -r -f /tmp/starsql_java_unix.tar** to extract the distribution.
3. Using the UNIX installer, enter the following command to run the **setup** shell script.

```
# ./setup
```

Installation on a Windows-Based Computer

To install StarSQL for Java on a Windows computer, be sure you are logged in as Administrator or a member of the Administrator group as the installation program needs to add a variable to the Windows environment variables.

1. Double click the StarSQL for Java SETUP program to launch the Windows installation program.
2. Respond to the installation prompts as appropriate. If you select the Custom Setup option you can choose which components to install and the location. The default installation directory is `C:\Program Files\StarQuest\StarSQL_Java\`. Make a note of the location if you change it.
   Since the StarSQL for Windows installer is a 32-bit installer, the actual installation directory on a 64-bit system will be `C:\Program Files (x86)\StarQuest\StarSQL_Java\`.

The Windows installation program automatically sets environment variables that are needed to run StarSQL for Java. If you are installing from a terminal client session, you may need to log in again for the environment variables to take effect. Command windows must be re-opened for the environment variable change to take effect.

On a 64-bit system, the Windows installer will install both 32-bit and 64-bit versions of the licensing support files.

Setting the Classpath and Environment Variables

To use the StarSQL for Java driver, you must have a compatible JDK or JRE installed (see "System Requirements" on page 12 for details). You can use the **java -version** command to display information about the installed JRE or JVM. If the path to your JDK or JRE is not specified as an environment variable, add the `bin` directory to your PATH environment variable (for example, `C:\Program Files\Java\jre1.8.0_161\bin`), or provide the complete path as you enter the **java -version** command, as shown below.

```
C:\Program Files\Java\jre1.8.0_161\bin>java -version
```

The **java -version** command displays information similar to the following:

```
java version "1.8.0_161"
Java(TM) SE Runtime Environment (build 1.8.0_161-b12)
Java HotSpot(TM) 64-Bit Server VM (build 25.161-b12, mixed
mode)
```

Setting the Classpath

You need to specify the location of the jar files for the StarSQL for Java driver and any applications that you want to use with it. You can specify these locations in the CLASSPATH environment variable, or with the `classpath` argument of the **java** command. If you install StarSQL for Java on a Mac OS X computer, the installation program automatically creates a symbolic link to StarSQL_JDBC.jar in /Library/Java/Extensions.

The location of the jar file for the StarSQL for Java driver may be specified in the CLASSPATH variable on a Windows-based computer as shown below:

```
.;"C:\Program Files\StarQuest\StarSQL_Java\StarSQL_JDBC.jar"
```

On a UNIX-based computer, the CLASSPATH variable may be specified as shown below:

```
CLASSPATH=/opt/starsql_java/StarSQL_JDBC.jar
```

You also can use the `CLASSPATH` argument of the **java** command to specify the location of class or jar files. The following examples specify the location of the StarSQL for JDBC driver, assuming that you install it to the default location.

On a Windows-based computer:

```
java -classpath "C:\Program
Files\StarQuest\StarSQL_Java\StarSQL_JDBC.jar"
```

On a UNIX-based computer:

```
java -classpath $CLASSPATH:/opt/starsql_java/StarSQL_JDBC.jar.
```

Setting Environment Variables

To run the StarSQL for Java sample applications, set the environment variable $STARSQL_JAVA (%STARSQL_JAVA% for Windows) to the location where StarSQL for Java is installed. The Windows installation automatically sets this environment variable. For UNIX users this environment variable can be set in a system-wide profile file or in each user's .profile, .login, or .cshrc file.

To use the sample applications that are distributed with StarSQL for Java with an X Windows server running on a Mac OS X, UNIX, or Windows computer, you must set the DISPLAY environment variable, such as:

```
export DISPLAY=myworkstation:0
```

On an IBM i computer, you also need to configure the Java system properties to use the Native Abstract Windowing Toolkit (NAWT). Create a file in your home directory named `SystemDefault.properties`, or create a system-wide file named `/QIBM/UserData/Java400/SystemDefault.properties` that contains the following statement:

```
os400.awt.native=true
```

To invoke Java using the Portable Application Solutions Environment (PASE) on an IBM i computer, set the QIBM_JAVA_PASE_ALLOW_PREV environment variable as shown below.

```
export QIBM_JAVA_PASE_ALLOW_PREV=1
```

Licensing StarQuest Products

All StarQuest products are licensed for use. Each product setup contains a client module used to configure the specific license option used to enforce the use of the product. The licensing options allow you to use a node-locked license or a floating license.

Node-Locked License

A node-locked license allows you to use the Product on a single computer: Node-locked licenses are only available for computers using Microsoft Windows Operating Systems. With a node-locked license:

- The computer is identified by a unique Host ID.
- The product can run only on the identified computer.
- The product usage may not exceed the limits allowed by the license.

In addition to the setup of the StarQuest product you will be provided with a unique registration code that should be used for the activation of the software license. It is also possible to use the client module to display the HOSTID to request a software license via email or telephone.

The software license should be activated online using the supplied registration code, or manually after communicating a HOSTID with StarQuest and receiving an email response containing a license string.

Floating License

A floating license allows multiple computers using a StarQuest product to share use of the software license. The software license can be used on any computer within a network provided that the number of concurrent requests does not exceed the limit allowed by the license. All StarQuest products for UNIX and Mac OS X ***must*** use a floating license. StarQuest products for Windows ***may*** use a floating license. Generally, only one license server need be installed on a network, to service any number of clients.

For a floating license, in addition to the setup of StarQuest product you will be provided with a StarLicense Server setup and a unique registration code to activate the license server. The *StarLicense Server for UNIX User's Guide* contains details about installing and configuring the server software, but in general:

- The StarLicense Server software should be installed on a network server.
- The network server is usually identified by a unique, static IP address.
- The StarLicense Server should be activated online or via e-mail.
- The StarLicense Server controls the total number of concurrent connections within the network.

For a client to obtain a license from a StarLicense server the parameters of the network server where the StarLicense Server is installed are specified in the appropriate client license module on any computer using the StarQuest product.

Configuring a License

StarQuest issues a unique License Key that specifies the number of connections that are allowed and the duration of the license. StarSQL for Java uses a file named `StarLicense.properties` to obtain information about the StarQuest license to use for the connection.

The StarSQL driver uses the following order and locations to find the `StarLicense.properties` file.

1. the location specified by the CLASSPATH (java.class.path)
2. in the JRE library directory (java.home/lib)

If it fails to find a `StarLicense.properties` file, StarSQL attempts to check out a license from a StarLicense server with a hostname of **starlic** (Windows computers) or 127.0.0.1 (UNIX computers) that listens for connections on port 4999.

You can edit the `StarLicense.properties` file to configure StarSQL to use a different StarLicense server, or to use a node-locked license on the computer.

- From a Windows computer, select the **Edit StarLicense.properties** shortcut from the **StarSQL for Java** program group.
- From a UNIX, Mac OS X, OS/390, or OS/400 computer, use a text editor to edit the `StarLicense.properties` file.

The next section describes the format and valid configuration values.

License Configuration File Format

The StarLicense.properties file contains a global variable and two sections, as shown below:

```
DEFAULT_PRODUCTID
[ClientServerN]
[ClientLicense]
```

You can precede any line with a # sign to include comments in the file or to comment out (disable) a particular line in the configuration.

Setting the global variable DEFAULT_PRODUCTID in the StarLicense.properties file limits the licenses that can be checked out to that product license type. The following table shows the product ID values that you can specify for DEFAULT_PRODUCTID to control how the license connections can be used.

Table 2. Default Product IDs for License Checkout

DEFAULT_PRODUCTID Value	Description
SQ	The default value, SQ, checks out a StarSQL license for any type of DRDA-based host.
CA	Requires that the connection be to a DB2 for i host.
UX	Requires that the connection be to a DB2 for LUW host.
JV	Requires the connection be to a Derby host.

The [ClientServer*N*] section is for specifying a license server, while the [ClientLicense] section is for specifying that you want to use a node-locked license.

The `StarLicense.properties` file that is included in the distribution for Windows-based computers is configured with the following entry to use a node-locked license by default:

```
[ClientLicense]
Nodelocked=true
```

The `StarLicense.properties` file that is included in the distribution for UNIX-based computers is configured by default to use a StarLicense daemon running on the local computer for license checkout. The IP address 127.0.0.1 that is specified for the hostname is a standard loopback network connection address that is commonly used to access a network service that is running on the same computer.

```
[ClientServer0]
hostname=127.0.0.1
port=4999
```

Refer to the appropriate section below, depending on whether you want to configure the client to use a floating license on a StarLicense server within the same TCP/IP network or use a node-locked license on the local computer.

Configuring a Client to Use a Floating License

Edit the [ClientServer*N*] section of the StarLicense.properties file to specify a StarLicense server to use for licensing StarSQL connections.

The StarLicense.properties file must contain the following section to identify the license server.

```
[ClientServerN]
hostname=yourhostname
port=nnnn
primary=true|false
```

If you have more than one StarLicense server, you can specify multiple [ClientServer*N*] headers in the StarLicense.properties file, replacing *N* with a number that represents that server definition. The number of the ClientServer definition determines the order in which the servers are used, regardless of where the header appears in the properties file (unless there is a duplicate number, in which case the last definition supersedes the previous definition for that header). The [ClientServer*N*] header without the optional number defaults to the equivalent of [ClientServer0].

Each server header section can contain the keywords described in Table 3.

Table 3. Keywords for [ClientServer*N*] Header

Keyword	Description
hostname	Specify the name of the host, either as an IP address (xxx.xxx.xxx.xxx) or a DNS name (myhost.mydomain.com). The default hostname is `starlic`.
port	The number of the port used to connect to the StarLicense server specified by the hostname keyword. The default port number is `4999`.
primary	Indicates whether the StarLicense server specified by the hostname keyword is a primary license server. When a license is requested, it is requested from primary servers prior to attempting to connect to secondary servers. You can specify True, which indicates the server specified by the hostname keyword acts as a primary license server, or False, which indicates the host is a secondary license server. The default is `True`.

Following is an example StarLicense.properties file that defines three StarLicense servers. The first two servers are defined as primary license servers, with the first server listening for license requests on port 50001 and the second listing for requests on the default port of 4999. The third StarLicense server, [ClientServer2], is defined to act as a secondary license server by specifying False for the `primary` keyword.

```
[ClientServer0]
hostname=starlicserv1.organization.com
port=50001

[ClientServer1]
hostname=215.168.44.101

[ClientServer2]
hostname=215.168.64.100
primary=false
```

The StarLicense Server documentation provides additional information about configuring and managing StarLicense servers.

Configuring a Client to Use a Node-Locked License

An alternative to using a StarLicense server to obtain a license is to configure the client computer with a node-locked license for using the StarSQL for Java driver. This section describes how to configure a node-locked license on a Windows workstation. Node-locked licensing is not currently available for the Mac OS X, OS/390, OS/400, or UNIX platforms—from those platforms you must connect to a StarLicense server to obtain a floating license, as described in the previous section.

Online Licensing

To request a node-locked License Key over the Internet you can simply start the StarLicense Configuration utility and provide the Registration Key that you received from StarQuest Support for using StarSQL for Java.

1. Select StarLicense Configuration from the StarSQL for Java program group.
2. Click on the License Online tab.
3. Select whether to use the Host ID or an IP address for the type of license lock. If you use an IP address it must be a static IP address (not one dynamically assigned by a DHCP server).
4. Enter the Registration Key that was provided in the download confirmation email and click Get License.

When the request successfully completes, the StarSQL license appears in the License Keys list of the Licenses tab. A Registration Key can produce several License Keys, depending on the products you are registered to use.

Manually Adding a License Key

You also can request a License Key via email from StarQuest Customer Support and manually add it to the License Key list. Refer to "Contacting StarQuest" on page 17 for the address and a list of the information you need to supply in your license request.

After you receive the License Key, follow the instructions below to add it.

1. Select **License Configuration** from the **StarSQL for Java** program group.
2. Select the Licenses tab of the License Configuration dialog.
3. Click the Add button to display the Add License dialog.
4. Enter the license key that was provided by StarQuest and click the Add button of the Add License dialog to save the license key to the Licenses tab.

CHAPTER 3

Using StarSQL for Java

This chapter describes how to configure and load the StarSQL for Java driver, and how to run the sample applications.

Configuring and Loading the StarSQL for Java Driver

StarSQL for Java provides a class which implements the java.sql.Driver interface that is used by the generic DriverManager class to locate a specific JDBC driver. The classname for StarSQL for Java is **com.starsql.jdbc.SQDriver**.

You can configure and load StarSQL for Java using any of the following methods:

- the getPropertyInfo method, which is part of the java.sql.Driver interface
- via a connection URL
- via a data source

Applications can load and configure StarSQL for Java at connection time, passing parameters as part of the URL, in a java.util.Properties object, or in a data source. The property information specifies the parameters that StarSQL for Java needs to establish a connection to a database.

StarSQL for Java Configuration Properties

The property parameter names and values are case sensitive. If a parameter is specified in both the URL and the Properties object, the value specified in the URL takes precedence. For example, if an application passes a user ID and password when getting a connection using a data source, the user ID and password values that are passed supersede those defined in the data source.

Table 4 shows the property parameters that StarSQL for Java supports, which parameters are required, and whether a default value is assumed if the parameter is not specified. Some parameters are dependent on how the host DBMS is configured, such as for security, or the type of host StarSQL will connect to, such as parameters specific

to Derby. Each parameter is described in more detail beginning on page 32.

Table 4. Property Information Parameters

Name	Required	Default Value	Valid Values
accounting			
catalogFilter			
collection		STARSQL	
commitProcedureCall		True	True | False
createTable			
databaseName	Yes		
dateFormat		standard JDBC format (yyyy-mm-dd)	eur | eurSlash | usa eur = dd.mm.yyyy eurSlash = dd/mm/yy usa = mm/dd/yyyy
decimalDelimiter	Must be set to `host` for connection to Derby host.	period	comma | host | period
defaultQualifier			
description			
diagnosticsLevel		0	0 | 1 | 2
drdaTrace		False	True | False
dynamicRules		Run	Run | Bind
fullyMaterializeCLOB		True	True | False
heldCursors		False	True | False

Name	Required	Default Value	Valid Values
keepDynamic		True	True | False
newPassword			
password	depends on host DBMS		
portNumber		446	
productID		SQ	CA | JV | SQ | UX CA = IBM i host JV = Derby host SQ = any DRDA-based host UX = LUW (UDB) host
pwdEncryption		False	True | False
sendUnicode	Must be set to True for connection to Derby host.	True	True | False
serverName	Yes		
ssl			SSL | SSLv3 | TLS | TLSv1
typdefovr		1252,0,0	any supported CCSID values (see National Language Support on page 93)
user	depends on host DBMS		

For prior releases of the StarSQL for Java driver, the properties shown in Table 5 were available. Although the current release of StarSQL for Java still accepts these properties, they may not be valid in future releases. At your earliest convenience, use the current properties instead of the deprecated properties.

Table 5. Deprecated URL Parameters

Deprecated Parameter	Current Parameter
create.table	createTable
dynamic.rules	dynamicRules
held.cursors	heldCursors
host.name	serverName
new.password	newPassword
package.collectionID	collection
pwd.encryption	pwdEncryption
rdb.name	databaseName

accounting Parameter

Some DRDA host systems allow you to specify an accounting string for charging back the mainframe resources that particular users consume when connected to the host through StarSQL. The accounting string is passed in the PRDDTA parameter of the ACCRDB DRDA command. It is stored as an accounting record on the mainframe.

The string that you specify for the `accounting` parameter is appended to the accounting string as a user-defined suffix.

catalogFilter Parameter

You can use the catalogFilter parameter to limit the amount of data retrieved from catalog calls when passing a "%" or NULL value. The value of this parameter, which can be a comma delimited set of values, limits the results to the specified catalogs. For examples, to limit the results of a getTables catalog call to those tables in catalogs Sales and Marketing, specify the parameter as catalogFilter=SALES,MARKETING.

collection Parameter

The collection parameter specifies the location on the DRDA host of the collection required by StarSQL to execute Dynamic SQL. The default is STARSQL.

commitProcedureCall Parameter

The commitProcedureCall parameter specifies whether to send a Commit on the execute chain when invoking a stored procedure from an application using the JDBC auto-commit mode. The default value is True. Setting commitProcedureCall to False allows compatibility with versions of StarSQL for Java that were released prior to 2004 when invoking a stored procedure in autoCommitMode did not send a Commit.

createTable Parameter

Set the createTable string to contain a directive that you want to append to SQL **create table** statements. The directive indicates where you want the table to be created.

For example, if **`createTable=in database DSNTEST`** and an application passes the following statement:

```
create table test (num integer)
```

the StarSQL driver passes the statement to DB2 as:

```
create table test (num integer) in database DSNTEST
```

Valid values for the createTable keyword depend on the host where the data source is located, as shown in Table 6.

Table 6. Supported CreateTable Directives

DRDA Host	Valid Table Creation Directives	Comment
DB2 for z/OS or DB2 for VSE & VM	**IN DATABASE** *DATABASE_NAME* **IN** *DATABASE_NAME.TABLESPACE*	
DB2 for i	**IN NODE** *GROUP_NAME*	A node group name is a qualified or unqualified name that designates a nodegroup, which is a group of IBM i systems across which a table is distributed.
DB2 for LUW	**IN** *TABLESPACE_NAME*	A tablespace name is a long qualifier that identifies a distinct table space that is described in the DB2 catalog.

databaseName Parameter

For the databaseName parameter, specify the name of the database to which you want to connect. This parameter is known as the RDB name on DB2 for i, the DB2 Location Name on DB2 z/OS, and the Database Name on DB2 LUW.

dateFormat Parameter

The dateFormat parameter allows an application to use the specified date format when passing dates as SQL strings and setting and retrieving dates using the setString() and getString() methods. If the dateFormat parameter is not explicitly set, the StarSQL for Java driver assumes dates will be in the standard IBM-ISO date format, which is yyyy-mm-dd.

Note

The JDBC API provides mechanisms for localizing an application for a specific geographical, political, or cultural region. There is no need to set the dateFormat parameter of the StarSQL for Java driver if you use the standard JDBC methods for localizing an application.

The values that you can specify for the dateFormat parameter are **`eur`**, **`eurSlash`**, and **`usa`**. The **`eur`** and **`usa`** date formats can be specified for any of the supported hosts (see page 13). Specifying the **`eurSlash`** format affects only the data returned with getString()—SQL strings and setString() values must use the IBM-ISO date format.

To use the **`eur`**, **`eurSlash`**, or **`usa`** date formats you must bind specific packages on the host. The following table shows the format for each of the dateFormat values, and which package must be bound to use that format. The SW*xxx* portion of the package name varies depending on the type of host and the configuration properties, as described in "Binding Packages" on page 85. For example, if the default package used to connect to a z/OS host is SWRC5000 (Read Committed isolation level with heldCursors and dynamicRules set to Yes) and you want to use the **`usa`** date format, you must bind the package SWRC5400.

dateFormat value	date format	Required Package
eur	dd.mm.yyyy	SW*xxx*800
eurSlash	dd/mm/yy	SW*xxx*C00
usa	mm/dd/yyyy	SW*xxx*400

These packages are unique to StarSQL for Java, so you must use the StarSQL for Java driver to bind them. To bind the package, connect to the host (using an ID that has authority to bind packages) from a JDBC application that uses the StarSQL for Java driver configured with the desired dateFormat value. Then execute a simple SELECT statement from the JDBC application. After the package is bound you can then grant EXECUTE privileges on the new package to other users as needed.

decimalDelimiter Parameter

The decimalDelimiter parameter allows an application to pass decimal values using a comma instead of a period for the decimal notation in numeric values that are passed in SQL strings and with the setString() and getString() methods. The decimalDelimiter parameter must be set to **`host`** to connect to a Derby host. For a DB2 host, if the decimalDelimiter parameter is not explicitly set to **`comma`**, StarSQL for Java uses a period as the decimal delimiter. The decimalDelimiter parameter setting affects the SQL data types DECIMAL, DOUBLE PRECISION, FLOAT, NUMERIC, and REAL PRECISION.

The host server imposes syntax restrictions when using decimal commas in SQL strings. For example, an additional space character is required to distinguish the decimal value, as compared for the following SQL statements.

SQL statement with period notation for decimal values:

```
insert into mytable values (1234,6.78)
```

SQL statement with comma notation for decimal values:

```
insert into mytable values (1234, 6,78)
```

defaultQualifier Parameter

For the defaultQualifier parameter you can specify a value that you want to use to qualify all unqualified SQL statements. On DB2 for i, this qualifier refers to IBM i library. On other DRDA hosts it refers to an Owner or Authorization Identifier.

For example, if you set defaultQualifier=MYQUAL, the SQL statement "SELECT * FROM MYTABLE" would be converted to "SELECT * FROM MYQUAL.MYTABLE".

description Parameter

The optional description parameter is applicable only when you are connecting to the host using a data source. It provides a brief description of the data source that can be displayed by datasource tools.

diagnosticsLevel Parameter

The diagnosticsLevel parameter controls whether the driver requests additional DRDA level diagnostics information. The default value is 0, which does not request the additional diagnostics information. Set this parameter only under the advice of a StarQuest Technical Support engineer should additional diagnostics information be needed to troubleshoot a problem.

drdaTrace Parameter

DRDA tracing is turned off by default. To trace DRDA operations, load the StarSQL for Java driver with the **drdaTrace** parameter enabled in the URL or data source definition. The following code excerpt shows establishing a connection to a host with DRDA tracing turned on.

```
//    Find the class...
        Class.forName( "com.starsql.jdbc.SQDriver" );
//    Open a connection...
        Connection myConnection =
          DriverManager.getConnection(
            "jdbc:StarSQL_JDBC://starhost
               /SAMPLE;drdaTrace=True",
            "staruser",
            "starpass" );
```

The DRDA trace is saved to the application's working directory with an .sqd file extension; be sure the userID running the application has permission to write in that directory. If it is difficult to grant that permission, or if the application is running as a Windows service, we recommend using the DRDA trace facilities of StarPipes instead.

The .sqd trace files can be read with the DRDA Trace Viewer application that is supplied with the StarSQL for Windows ODBC driver software.

dynamicRules Parameter

The dynamicRules parameter applies only to applications that use Dynamic SQL on a data source on DB2 for z/OS. Set this option to RUN or BIND as desired.

On DB2 for z/OS, the permissions required to run applications that use Dynamic SQL depend on the value of the dynamicRules parameter.

If dynamicRules=RUN, which is the default value, the System Administrator needs to grant the user explicit permissions to read and write the columns and tables accessed by the application.

If dynamicRules=BIND, the StarSQL user executes with the permissions of the owner of the dynamic SQL package. If dynamicRules is set to BIND, the following SQL statements cannot be executed, regardless of the actual permissions of the package owner:

- SET CURRENT SQLID
- GRANT
- REVOKE
- ALTER
- CREATE
- DROP
- Any SQL statement that cannot be prepared as dynamic SQL

fullyMaterializeCLOB Parameter

When data that is sent between the host and client computers contains Character Large Objects (CLOBs) with multi-byte characters, the DRDA host returns the length of the CLOB in bytes rather than number of characters. Setting the fullyMaterializeCLOB parameter to True, which is the default value if it is not explicitly set, causes the StarSQL for Java driver to retrieve the complete CLOB value and convert it to the local CCSID prior to determining the length.

With the fullyMaterializeCLOB parameter set to True, retrieving a very large CLOB may exhaust the Java Virtual Machine (JVM) memory. In this case, set fullyMaterializeCLOB to False. If fullyMaterializeCLOB is set to False and the CLOB contains multi-byte characters, the Clob.getSubString() method may not work as expected. The work around is to have the application call the Clob.getCharacterStream() method rather than the getSubString() method to retrieve the CLOB value.

heldCursors Parameter

Normally a cursor is closed when its transaction commits. A transaction can involve one or more SQL commands that are committed as a unit—either the entire transaction or none of it is committed to the database. If the heldCursors parameter is enabled (heldCursors=True), the cursor remains open after the commit. This enables an application to fetch rows from a result set, commit the transaction, and then continue fetching additional rows on the same result set.

keepDynamic Parameter

The keepDynamic parameter enables (keepDynamic=True) or disables (keepDynamic=False) optimization for prepared statements by tuning the KeepDynamic bind option (DB2 for OS/390 v5 and above and DB2 UDB v6 for Windows and UNIX).

newPassword Parameter

You can pass a value for the newPassword parameter to host systems that support resetting the password. You must specify valid entries for the `user` and `password` properties for the `newPassword` parameter to be valid.

password Parameter

The password parameter must contain a valid password for the user account that is accessing the host database.

portNumber Parameter

For the portNumber parameter, specify the port number that the DB2 server is listening on for DRDA communications. The default DRDA port is 446, which is the default for DB2 for z/OS and DB2 for i. Note that the default port for DB2 for LUW is 50000, so you will need to set the port explicitly. The default port that host systems use for the SSL protocol is 448, so you may need to set the port explicitly if you want to use SSL. The following example shows a connection URL that enables SSL and sets the port to 448.

```
set URL=jdbc:StarSQL_JDBC://servername:448/databasename;ssl=TLS
```

productID Parameter

The productID parameter causes the StarSQL for Java driver to check out a license using a specific type of StarQuest product license. Specifying the productID in the URL or data source allows you to use different licenses for different connections from the same system or application. The default value of SQ checks out a license for any DRDA-based host. Setting the productID to one of the other valid values requires that the connection use a license for the specific type of host—CA requires a license for connecting to a DB2 for i host, JV requires a license for connecting to a Derby host, and UX requires that the connection be to a DB2 LUW host.

If no productID is specified for the license type, the default SQ is used unless a different value is specified for the DEFAULT_PRODUCTID property in the `StarLicense.properties` file. If an organization has only one type of license, such as for connections to a DB2 for i host, setting the DEFAULT_PRODUCTID property

avoids the need to specify the productID parameter in the URL or data source. See "License Configuration File Format" on page 25 for information about setting the DEFAULT_PRODUCTID property.

pwdEncryption Parameter

Some host databases support encrypting the login user ID and/or password for increased security. Refer to "Host Database Management Systems" on page 13 for details about which versions of the host database systems support DRDA password encryption.

By default, this release of the StarSQL for Java driver sends the user ID and password in clear text to avoid problems connecting to hosts that do not support encryption. To enable password encryption for StarSQL for Java, set pwdEncryption=True when you load the driver.

Table 7 shows examples of how to enable encryption depending on which method you use to load the StarSQL for Java driver. Since the default value is False, password encryption is disabled if you do not specify the pwdEncryption parameter.

Table 7. Configuring Password Encryption

Connection Method	To Enable Encryption
using JDBC driver interface	connectionProperties(“pwdEncryption”, “True”);
using a connection URL	set URL=jdbc:StarSQL_JDBC://*servername*:446/*databasename*;pwdEncryption=True
using a data source	ds->setPwdEncryption(new Boolean(True))

sendUnicode Parameter

Setting the sendUnicode parameter to True causes StarSQL for Java to send Multi-byte Character Set (MBCS) data to the host using CCSID 1208, and Double-byte Character Set (DBCS) and GRAPHIC data using CCSID 1200. Set the sendUnicode parameter to True only if the host is capable of accepting Unicode data. The sendUnicode parameter must be set to True to connect to a Derby host.

The default value for the sendUnicode parameter depends on the default locale that is set for the JVM or JRE. If the locale is set to a Group 2 language (Chinese, Japanese, or Korean), the default for the sendUnicode parameter is True. If the locale is set to any other language, the default for the sendUnicode parameter is False.

serverName Parameter

For the serverName parameter, specify the host name or IP address of the host as it is defined in the network.

ssl Parameter

The ssl parameter controls whether the StarSQL for Java driver uses the Secure Sockets Layer (SSL) protocol for encrypted communications between the host and client computers. By default the StarSQL for Java driver uses standard Sockets protocol. To use SSL protocol, you must:

- Set the ssl parameter for the StarSQL for Java driver to a valid string that indicates which named secure socket protocol to use.
- Configure the host system to use the SSL protocol, as described in "Configuring Support for the SSL Protocol" on page 76.

You also may need to explicitly set the port number, as some hosts use a default port of 448 for SSL rather than the default DRDA port of 446. See "portNumber Parameter" on page 39 for details.

typdefovr Parameter

Use the typdefovr keyword to explicitly set the single-byte character set (SBCS) CCSID that StarSQL uses to send character data to the DRDA host. When typdefovr is set, StarSQL sends the data using the CCSID specified in the typdefovr with each DRDA request.

The typdefovr keyword value can be any SBCS CCSID that StarSQL supports. (For the complete list of supported CCSIDs, refer to the Oracle Java documentation).

If a SBCS CCSID is specified for typdefovr, StarSQL sends all SQL statements and parameters as single-byte character strings defined by that CCSID. To support double-byte (DBCS) and mixed-byte (MBCS) character sets, set the sendUnicode property of the StarSQL driver to True (see "sendUnicode Parameter" on page 40) rather than specifying the typdefovr parameter.

user Parameter

The user parameter specifies the user ID to use for logging on to the host database.

Using the JDBC Driver Interface

Applications can call the getPropertyInfo method provided as part of the StarSQL JDBC driver interface to determine which parameters are supported. StarSQL for Java returns information for the parameters shown in Table 4 on page 30.

The following code excerpt shows establishing a connection to a host named starhost, with an database name of SAMPLE, as user staruser with a password of starpass. Since no port value is specified, the default port value of 446 is used for the DRDA connection.

```
//    Find the class...
        Class.forName( "com.starsql.jdbc.SQDriver" );
//    Open a connection...
        Connection myConnection =
          DriverManager.getConnection(
            "jdbc:StarSQL_JDBC://starhost/SAMPLE",
             "staruser", "starpass" );
```

Using the Connection URL

Applications can load and configure the StarSQL for Java driver by passing the property parameters in a URL. The supported parameters are shown in Table 4 on page 30, and the syntax of the URL is as follows:

```
jdbc:StarSQL_JDBC[://servername[:port]/
databasename][;attribute-name=attribute-value]
```

The *servername* (in the form of a DNS name or an IP address) and *databasename* parameters are required, but a default port of 446 is used if you do not specify a port.

You can specify an unlimited number of *attribute-name=attribute-value* pairs. Separate each name=value pair with a semi-colon (;). For example, the following connection URL establishes a connection to a remote database named DB2V7B, using port 448 with a username and password of staruser and starpass.

```
jdbc:StarSQL_JDBC://db2v7b.mydomain.com:448/
DB2V7B;user=staruser;password=starpass
```

Note

To specify multiple attribute-name pairs on a UNIX computer, you must enclose the URL in quotation marks (such as, URL="jdbc:StarSQL_JDBC://amelia:446/SAMPLE;pwdEncryption=True; defaultQualifier=Sales,Marketing") because UNIX interprets the semicolon (;) between the parameters as the next command to invoke.

If there are duplicate name=value pairs, the final occurrence of the pair determines the value passed to the Properties object.

Using Data Sources

Many commercial applications include support for data sources. Data sources provide the configuration parameters the StarSQL for Java driver needs to connect to the host database. To set up the StarSQL for Java driver as a JDBC provider and define a data source, you need to supply the following driver-specific information.

Table 8. JDBC Provider Information

Configuration Property	Value for StarSQL for Java Driver
Classpath	StarSQL_JDBC.jar
Implementation Classname	com.starsql.jdbc.SQDriver

After you configure the StarSQL for Java driver to be a JDBC provider, you create a data source that provides the specific connection information.

StarSQL for Java provides classes that implement the three JDBC data source types for applications that need data source connectivity rather than URL-based connections. JDBC defines three data source types, for which the StarSQL for Java driver provides classes for using, as shown in Table 9. Use the java.sql.DataSource type for standard JDBC connections. Use the java.sql.ConnectionPoolDataSource for data source connectivity where connections are pooled, which allows for improved performance. Use the javax.sql.XADataSource for data source connectivity in a transaction-oriented environment. XA data sources also are pooled connections.

Table 9. StarSQL for Java Classes for Using Data Sources

Data Source Type	StarSQL for Java Implementation Class
java.sql.DataSource	com.starsql.datasource.SQDataSource
java.sql.ConnectionPoolDataSource	com.starsql.datasource.SQConnectionPoolDataSource
javax.sql.XADataSource	com.starsql.datasource.SQXADataSource

The Initial Context for a data source provides access to naming directory services using the JNDI. The StarSQL for Java driver supports any JNDI Service Provider that conforms to the JNDI specification.

There are many service providers that are publicly available, such as COS Naming for access to CORBA naming services, the Directory Services Markup Language (DSML), Novell for access to the NetWare Directory Services (NDS), Windows for access to the Windows Registry, and XNam for reading naming information from XML files. Refer to the Oracle Java website for a a list of publicly available service providers. Following is information about specifying two commonly used service provider interfaces, which use the file system context factory and an LDAP server.

- To use a file data source, set the Initial Context to the file system context factory, **com.sun.jndi.fscontext.RefFSContextFactory**, and specify the Provider URL in the form of "file:/directory," which is where the .bindings file that keeps track of the JNDI names is located.
- To use a data source on an LDAP server, set the Initial Context to **com.sun.jndi.ldap.LdapCtxFactory** and specify the Provider URL in the form of "ldap://ldap.mydomain.com:port".

Regardless of the host platform and service provider that you use, you will need the host information described in Table 10 to specify the database you want to access using StarSQL for Java. You may need to obtain this information from the DB2 administrator.

Table 10. Host Information Required for Data Source Configuration

Data Source Information Item	Information Needed
SQL Package Collection ID (see "collection Parameter" on page 33)	location of SQL packages required by StarSQL for Java
Database Server Name (see "databaseName Parameter" on page 34)	relational database name or location name
User ID and Password (see "user Parameter" on page 41 and "password Parameter" on page 39)	user id and password for connecting to the host
TCP/IP connection information (see "serverName Parameter" on page 40 and "portNumber Parameter" on page 39)	host name and port

StarSQL for Java includes a sample application, CreateDS, that you can use to create a simple data source definition. Refer to the section, "Running the Sample Applications" on page 48 for details about using the CreateDS sample application.

Displaying StarSQL Driver Information

StarSQL for Java includes a program named ShowVersion that loads the driver, from the default CLASSPATH or one specified in the script or batch file, and displays information about the driver.

- From a Mac OS X computer, use the Finder to open `Applications/starsql_java/samples`, double-click `samples.command`, and select ShowVersion.
- From other UNIX computers, run the `runapp.sh ShowVersion` script or the `samples.sh` script.
- From a Windows computer, select **Show Version** from the **StarSQL for Java–>Samples** program group.

The ShowVersion program displays information about the driver, such as shown below:

Testing a StarSQL/DRDA Connection

After you install and configure StarSQL for Java, perform the procedures in this section to test that the client can connect to the DRDA host using StarSQL. To establish connectivity to a database, the host must be configured to accept connections from StarSQL. Refer to "Preparing Hosts for StarSQL Access" on page 69 if your host is not already configured for StarSQL connections.

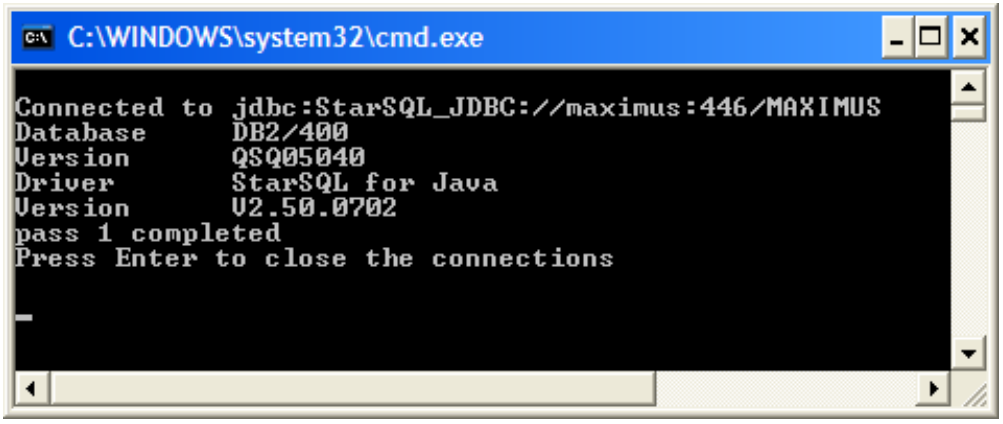
You can use the **SimpConn** program included with StarSQL for Java to verify that a client can connect to a specific URL using a valid host account. The SimpConn program checks out a license, either from the local workstation or from a license server, to connect to the host. You must specify a username and password of an account that is valid on the host and has permission to access the database. (See "Preparation Required for All Hosts" on page 69 for more information about user accounts and permissions.)

The SimpConn program also accepts a parameter, N=*n*, that specifies the number of times to run the connection test. Since the SimpConn program does not release connections until it ends, you can run multiple iterations of the program to test the number of simultaneous connections the license allows. The SimpConn program initially is configured to run once (N=1).

1. Edit the `simpconn.command` (for Mac OS X), `simpconn.sh` (for other versions of UNIX) or `simpconn.bat` file (for Windows) to specify a valid host, database, and user account. You may need to change the file permissions from Read Only to Write to save the changes.
2. Run the `simpconn.sh` script from UNIX (double-click `simpconn.command` from Mac OS X), or the `simpconn.bat` file for Windows.

The SimpConn program runs in a command window and, upon successful connection to the host, returns information about the connection and the database and driver versions. The following sample output shows that the StarSQL for Java driver connected to a DB2 for i database on a host named MAXIMUS using the SimpConn program:

Figure 3. Using SimpConn to Connect to a Host



The StarSQL for Java Sample Applications

StarSQL for Java includes several sample applications that use the StarSQL for Java driver to show typical connection, statement execution, and handling of result sets.

Note

The sample applications are *not* designed to support the full JDBC functionality nor are they intended to be used as test tools or productivity applications. They are sample programs intended only to help you test that the StarSQL for Java driver is installed and working correctly.

- **CreateDS** is a program that allows you to create a simple file data source definition. You can connect to a file data source that you create with the CreateDS application from the CatalogApp, LobTestApp, and QueryApp sample applications.
- **CatalogApp** is a database browser that displays the metadata (schema, table name, column name, and the column data type) for a database and the contents for that table.
- **LobTestApp** is a program that illustrates how the StarSQL for Java driver supports BLOB and CLOB data types.
- **QueryApp** provides a graphical interface for querying and displaying data from a database.

As described earlier in this chapter, StarSQL for Java also includes the following two sample applications to provide information about the StarSQL for Java driver and test whether a client can connect to a host.

- **ShowVersion** displays information about the version of StarSQL for Java that is installed. See "Displaying StarSQL Driver Information" on page 45 for details about running the ShowVersion sample application.
- **SimpConn** is a program that tests whether a client can connect to a specific host database. See "Testing a StarSQL/DRDA Connection" on page 45 for details about running the SimpConn sample application.

The source code for all of the sample applications is provided with the StarSQL for Java software. Refer to "Building the Sample Applications" on page 63 for guidance on how to build the sample applications.

Running the Sample Applications

The `Samples` directory contains a batch file (RunApp.bat) for running the sample applications under Windows, and a shell script (runapp.sh) for running the sample applications under UNIX. You need to modify the batch or script file if you installed StarSQL for Java in a location other than the default installation directory and have not set the STARSQL_JAVA environment variable. The script tests whether the STARSQL_JAVA environment variable has been set and, if not, uses the default installation directory. Refer to "Installing StarSQL for Java" on page 19 for information about the default installation location for each platform and how to set the STARSQL_JAVA environment variable.

To use the RunApp batch or script file, specify the case-sensitive sample application classname (CatalogApp, CreateDS, LobTestApp, QueryApp, or TransactionLogManager) when you enter the command. For example, the following command runs the CatalogApp sample application:

```
runapp CatalogApp
```

Note

When you run these sample applications on UNIX, text may appear truncated at the bottom of text fields if the font specified in the font properties for the graphical interface is not available. This affects only the appearance, not the functionality of the application.

The UNIX shell scripts `samples.command` (for Mac OS X) and `samples.sh` present a menu of the available sample applications and utilities.

On an OS/400 computer you can run the sample applications under QSHELL, PASE, or xterm under PASE; however, the appearance of the samples menu has been optimized for using xterm under PASE.

Running the CreateDS Sample Application

Two versions of the CreateDS application are provided—one that has a graphical user interface (GUI) and one that can be run from a command line interface (CLI). The CreateDS sample application creates a file data source, which uses the file system context factory. (See "Using Data Sources" on page 43 for more information about context factories.)

Running the CreateDS GUI Version

To create a file data source using the GUI version of the CreateDS sample application, follow the steps below. When you create a data source using the CreateDS application, it creates a data source file named `.bindings` in the `\java` directory on Windows or the `/tmp` directory on UNIX.

1. From a Mac OS X computer, use the Finder to open `Applications/starsql_java/samples`, double-click `samples.command`, and select CreateDS.

   From other UNIX computers, run the `runapp.sh CreateDS` script or the `samples.sh` script.

   From a Windows computer, select **CreateDS** from the **StarSQL for Java–>Samples** program group.

The CreateDS application window appears so you can enter appropriate values for the data source.

Figure 4. GUI Version of the CreateDS Sample Application

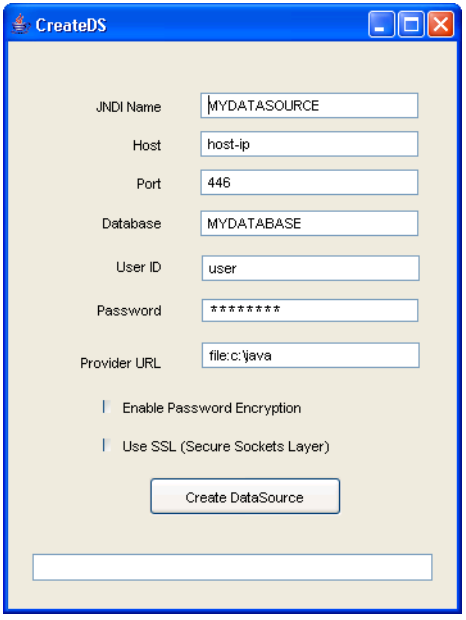


Table 11 describes the properties you can specify for the data source.

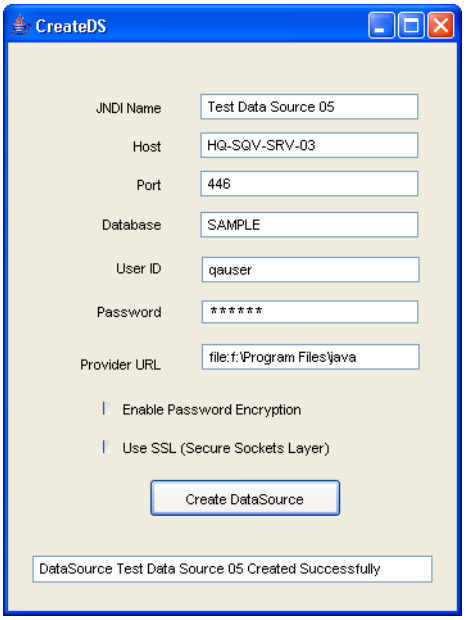
Table 11. Data Source Properties

Data Source Information Item	Description
JNDI Name	A name that is used to bind the data source in the Java Naming and Directory Interface.
Host	The DNS name or IP address of the host to which you want to connect.
Port	The port you want to use for DRDA communications.
Database Server Name	The name of the relational database to which you want to connect.
User ID and Password	The user id and password required to access the host computer.
Provider URL	The path to the .bindings that stores the JNDI names. This typically is C:/java for Windows-based computers or /tmp for UNIX-based computers.
Enable Password Encryption	An option that determines whether the data source allows encrypted passwords. If you turn on password encryption, be sure the host supports it ("Host Database Management Systems" on page 13 describes which database systems support password encryption).
Use SSL	Enable the SSL option if you want the StarSQL for Java driver to use Secure Sockets Layer protocol when connecting to the host using this data source. The host also must be configured to use SSL, as described in "Configuring Support for the SSL Protocol" on page 76.

2. Enter appropriate values for the data source and click **Create DataSource**.

The CreateDS application creates a file data source, showing status and error messages in the text field at the bottom of the CreateDS window. For example, the following illustration shows that a data source named Test Data Source 05 that connects to the SAMPLE database was successfully created.

Figure 5. Successful Creation of File Data Source



Running the CreateDS CLI Version

The `CreateDS.cli` directory contains files for running the command line interface (CLI) of the CreateDS sample application. When you run the CLI version of CreateDS, a data source file named `.bindings` is created in the `\java` directory on Windows or the `/tmp` directory on UNIX.

Note

The target directory for the data source file (\java on Windows or the /tmp directory on UNIX) must already exist before you run the command version of the CreateDS application.

To run the CLI version of CreateDS, follow the steps below.

1. Edit the `createds.command` script file (for Mac OS X), the `createds.sh` script file (for other versions of UNIX) or the `CreateDS.bat` file (for Windows) to specify the paths to your installed JDK and StarSQL for Java software and to provide the appropriate name for the data source. See "Data Source Properties" on page 50 for a description of the values you can specify.
2. From a UNIX computer, run the `createds.sh` (`createds.command` for Mac OS X) script.

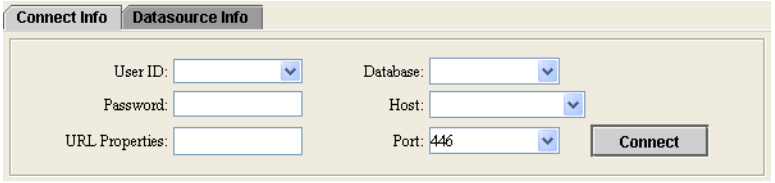
   From a Windows computer, run the `CreateDS.bat` batch file.

Using the values specified in the script or batch file, the CreateDS application creates the data source file in the target directory.

Specifying Connection Information

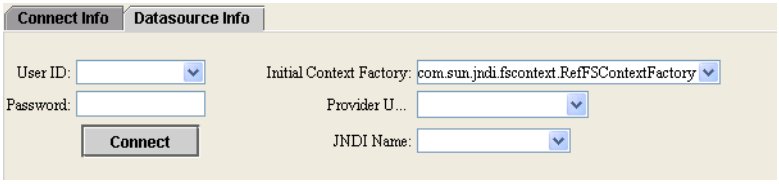
The sample CatalogApp, LOBTestApp, and QueryApp applications display two tabs for establishing a connection to a host database. The Connect Info tab prompts for the information needed to connect to the database. Enter your userID, password, database name, server name, and port id into the input fields and click the **Connect** button to establish a connection to the host database. In the URL Properties you can specify any of the configuration properties shown in Table 4 on page 30 for establishing the connection. For example, entering `portNumber=448;ssl=TLS` for the URL Properties would connect using the SSL protocol on port number 448.

Figure 6. Using the Connect Info Tab



The Datasource Info tab allows you to specify a data source to use for connecting to the database.

Figure 7. Using the Datasource Info Tab



If the data source specifies the user ID and password, leave these fields blank in the Datasource Info tab. (If the user ID and password are specified in the data source, the connection is established using that information.)

If you are using a file data source, such as a data source created with the CreateDS application, leave the Initial Context Factory set to the file system context, `com.sun.jndi.fscontext.RefFSContextFactory`. For the Provider URL, specify the path to the .bindings file, such as `file:c:/java`. For the JNDI Name, enter the name of the data source.

Running the CatalogApp Sample Application

The sample CatalogApp application uses the StarSQL for Java driver to display metadata (schema, table name, column name, and the column data type) and table results for a database. Follow the steps below to run the CatalogApp sample application.

1. From a Mac OS X computer, use the Finder to open `Applications/starsql_java/samples`, double-click `samples.command`, and select CatalogApp.

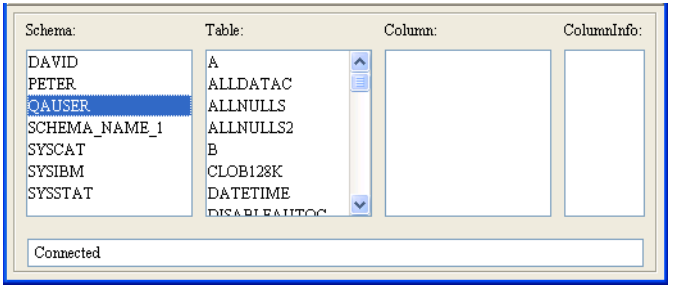
   From other UNIX computers, run the `runapp.sh CatalogApp` script or the `samples.sh` script.

   From a Windows computer, select **CatalogApp** from the **StarSQL for Java–>Samples** program group.

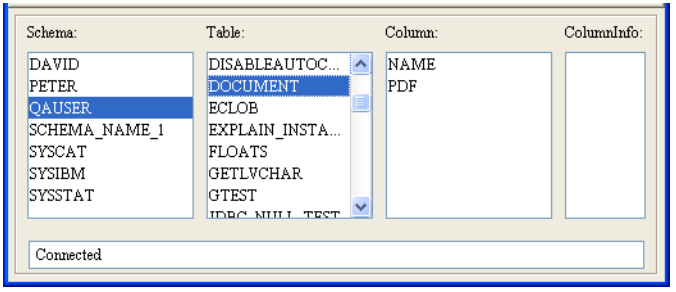
2. Establish a connection to the host database, using either the Connect Info or Datasource Info tabs. (See "Specifying Connection Information" on page 52 for more information.)

Once the connection is established, the schema for that database appears in the pane labeled Schema. The text box at the bottom of the window provides status and error messages.

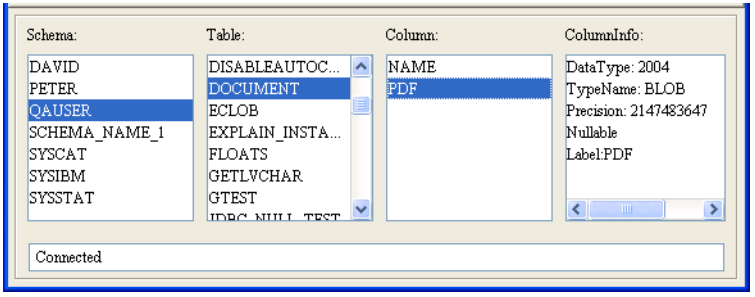
3. To display the table names for a schema, click a schema name. All the tables for that schema appear in the pane labeled Table.



4. Click on a table name to display the column names for that table in the pane labeled Columns.

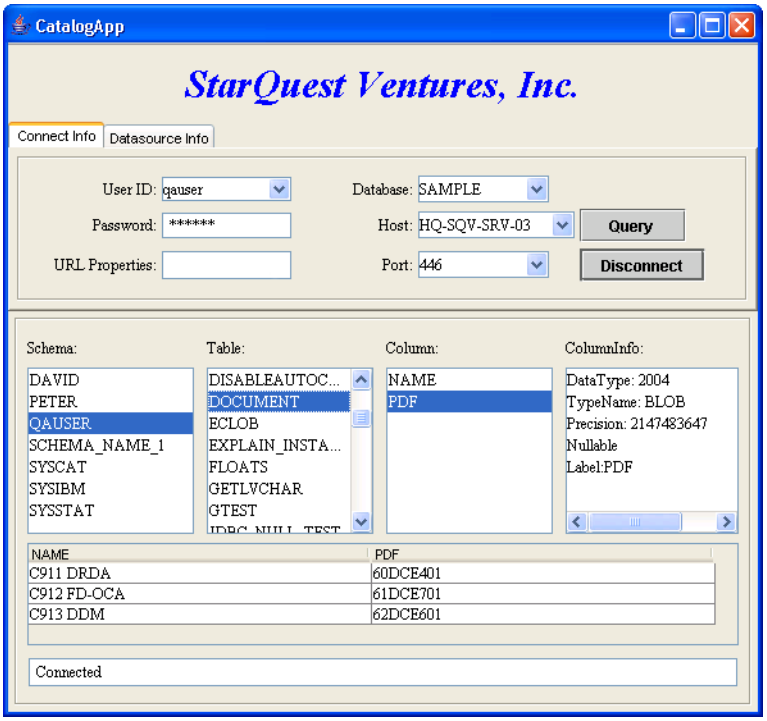


5. Click on a column name to display the SQL data types for the column in the pane labeled ColumnInfo.



6. To display the result set for a table, select the table name in the pane labeled Table and click the **Query** button. The result set appears in the bottom pane, as illustrated for the DOCUMENT table in Figure 8.

Figure 8. Table Query Results Using CatalogApp



7. When you are finished using the CatalogApp application, click the **Disconnect** button to disconnect from the host database and then click the **Close** button to close the application windows.

Running the LobTestApp Sample Application

The LobTestApp application demonstrates how StarSQL for Java works with LOB (large object) data types. It allows you to create tables that contain BLOB (binary large objects) or CLOB (character large objects) data types, and then retrieve data from them.

To use the LobTestApp program, the database you are connecting to must support LOBs (see "Host Database Management Systems" on page 13 for which versions support LOBs).

Note

The LobTestApp sample application was developed and tested with LOBs in a DB2 for z/OS database. Due to differences in how tablespaces are created on different hosts that allow LOB data, the LobTestApp may require modification to successfully work with LOB data on other host systems.

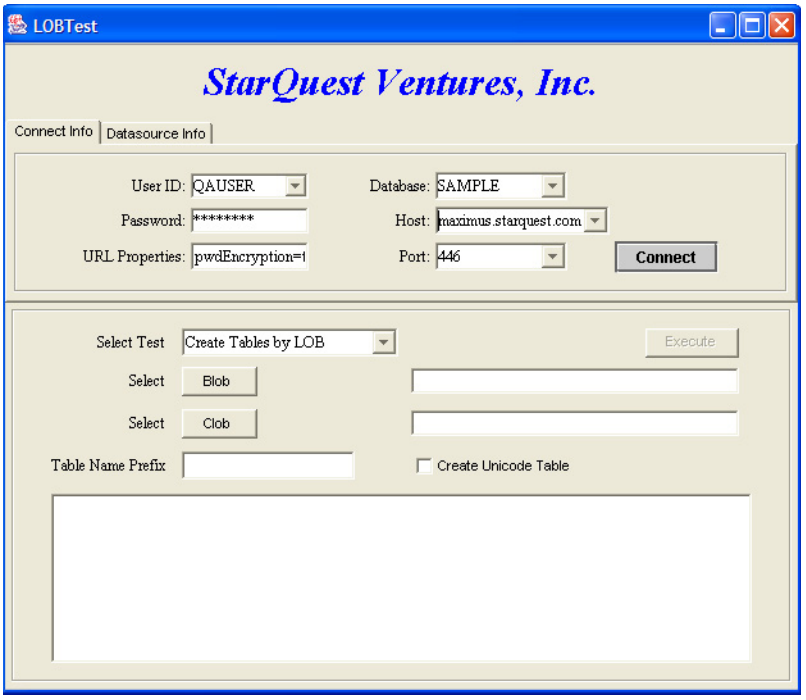
Follow the steps below to run the LobTestApp program.

1. From a Mac OS X computer, use the Finder to open `Applications/starsql_java/samples`, double-click `samples.command`, and select LobTestApp.

   From other UNIX computers, run the `runapp.sh LobTest` script or the `samples.sh` script.

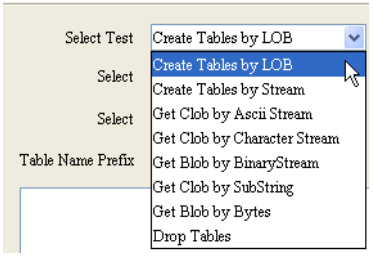
   From a Windows computer, select **LobTestApp** from the **StarSQL for Java–>Samples** program group.

Figure 9. LobTestApp Sample Application



2. Establish a connection to the host database, using either the Connect Info or Datasource Info tabs. (See "Specifying Connection Information" on page 52 for more information.)

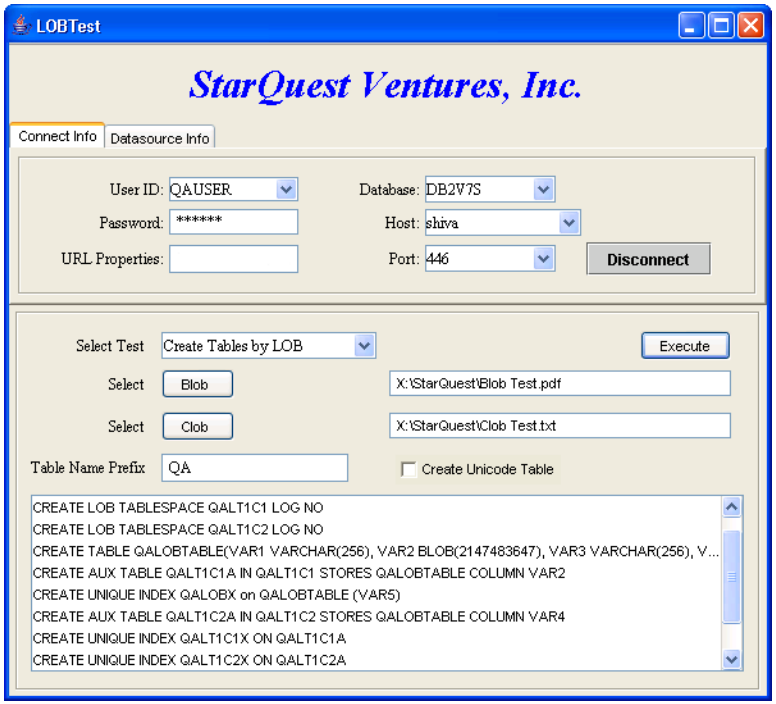
3. Select which Create test you want to execute from the drop-down menu next to Select Test. You must create tables, either by LOB or by stream, before you can execute the Get tests.



4. Click on BLOB and select a file for which you want to create a table. With BLOB data types, the file can be any format (such as .pdf, .jpg, and .doc).
5. Click on CLOB and select a file for which you want to create a table. For CLOB data types, the file can contain only ASCII text or control characters.
6. For the Table Name Prefix, enter a few characters to prefix the table name to ensure that the tables are given unique names. Make a note of the prefix, as you need to enter the same prefix to get data from the tables.
7. Enable the Create Unicode Table option if you want to create the table using the Unicode character set and the host is capable of accepting Unicode data. This option typically is used when creating tables that contain DBCS and MBCS data.
8. Click the **Execute** button to create the tables.

As the test executes, the results are displayed in the bottom pane of the LOBTest window, as shown in Figure 10.

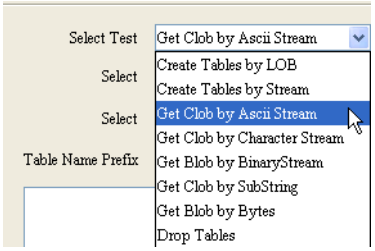
Figure 10. Using LOBTest to Create Tables



After you create the tables you can use LOBTest to get BLOB or CLOB data from them.

1 . If you disconnected from the host, reconnect as described in "Specifying Connection Information" on page 52.

2. Select which Get test you want to execute from the drop-down menu next to Select Test.



3. Select a file with BLOB or CLOB data, as appropriate for whether you selected a Get BLOB or Get CLOB test.

4. For the Table Name Prefix, enter the same characters that you used to create the tables.

5. Click the **Execute** button to get the selected data.

When the Get test completes successfully, it writes a file with the data to the `samples` subdirectory where StarSQL for Java is installed. The output file should match the original file that was used to create the tables. The filename that you specified for the BLOB or CLOB data is prefixed with `Gotn-` for the output file. You can compare the contents of the original file and the output file to ensure that LOB data is being created and retrieved properly.

When you are finished using the LOBTest sample application you may want to drop the tables that you created for the test.

6. From the Select Test drop-down menu, select Drop Tables and click **Execute**.

7. Click the **Disconnect** button to disconnect from the host database and then click the **Close** button to close the application windows.

Running the QueryApp Sample Application

QueryApp is a Java-based graphical program that retrieves the result set for a table in a database. Follow the steps below to run the QueryApp sample application.

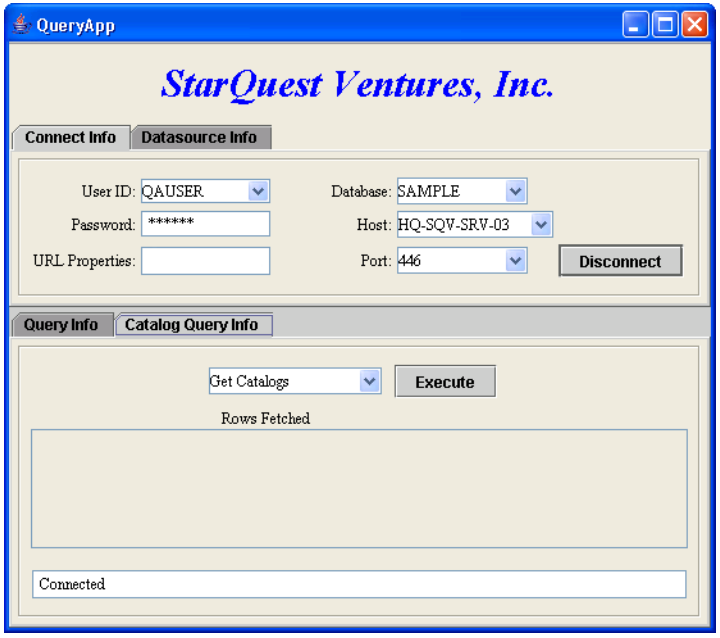
1. From a Mac OS X computer, use the Finder to open `Applications/starsql_java/samples`, double-click `samples.command`, and select QueryApp.

   From other UNIX computers, run the `runapp.sh QueryApp` script or the `samples.sh` script.

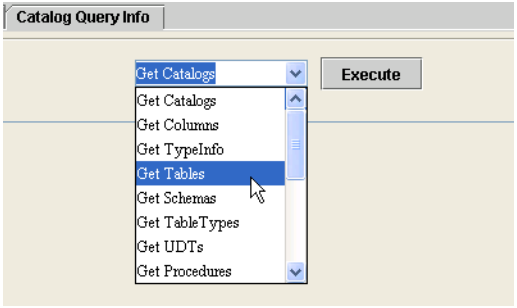
   From a Windows computer, select **QueryApp** from the **StarSQL for Java–>Samples** program group.

2. Establish a connection to the host database, using either the Connect Info or Datasource Info tabs. (See "Specifying Connection Information" on page 52 for more information.)

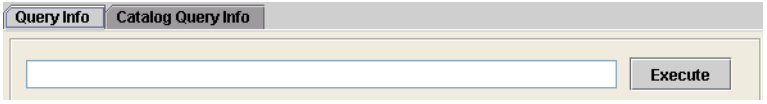
Figure 11. QueryApp Sample Application



3. On the Catalog Query Info tab you can select a Get operation from the drop-down menu and click **Execute** to execute the operation.

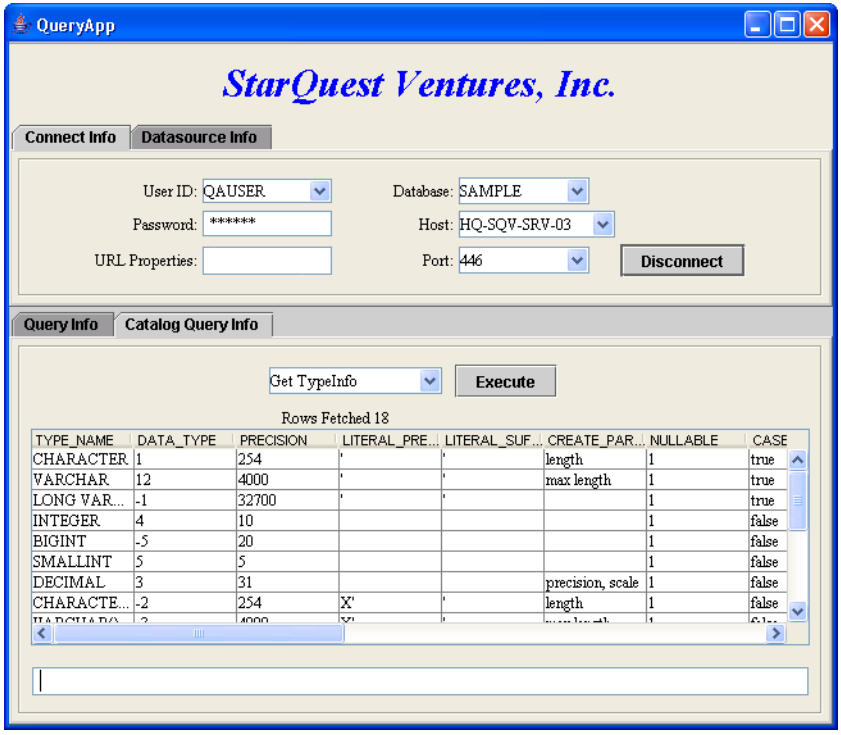


4. On the Query Info tab you can type a valid SELECT statement in the text field and click the **Execute** button to execute the statement.



The Execute button changes to Cancel while the operation is being performed, and then the QueryApp application displays the result set in the bottom panel, as illustrated in Figure 12. If the query returns multiple result sets, click the **Next** button to display the next result set. Status and error messages appear in the text field at the bottom of the window.

Figure 12. Using QueryApp to Issue Catalog Queries



5. When you are finished using the QueryApp sample application, click **Disconnect** to disconnect from the host, and click the **Close** button to exit the application.

Building the Sample Applications

All the sample applications that are provided with StarSQL for Java are available in `\Program Files\StarQuest\StarSQL_Java\Samples` for Windows, and `$STARSQL_JAVA/samples` for UNIX. The `Samples` directory contains a `java_src` directory which provides the source code for the sample applications.

The command line sample applications, CreateDS.cli and simpconn have their own build scripts. You can build the GUI sample applications (CatalogApp, CreateDS, LobTest, and QueryApp) using either of the following methods:

- run the batch file, `build.bat`, on a Windows computer, or the script file `build.sh` from a UNIX computer. These files use the javac and jar tools of the JDK to build the sample applications.
- import the sample application source files into an Integrated Development Environment (IDE) tool, such as the open source Eclipse SDK (available at http://www.eclipse.org)

Regardless of how you build the sample applications, keep the following guidelines in mind:

- The sample applications use shared code, which is distributed in the apps\com\StarSQL\apps directory where StarSQL for Java is installed. Set up a project for the shared code if you use an IDE to build the sample applications.
- The CreateDS project relies upon the StarSQL_JDBC.jar file to get the datasource information that is specific to StarSQL. The other sample applications do not need access to the StarSQL_JDBC.jar until they are run.
- Output the .jar file for each sample application to the same directory so they can share the Connection.properties and Datasource.properties that are created when a connection is successful.

Building the Sample Applications Using a Batch or Script File

From a Windows computer you can build the sample applications by running the `build` batch file, and from a UNIX computer you can run the `build.sh` script. Be sure the PATH variable specifies the appropriate path to the Java SDK \bin subdirectory so the batch file can run the Java executable programs and compiler (java, jar, and javac).

Building the Sample Applications Using an IDE

The procedures in this section describe how to build the sample applications using the Eclipse Platform v3.0 on a computer running Windows XP Pro. The details may differ if you are using a different version of Eclipse, a different operating system, or a different IDE, but the general procedures should be applicable.

1. Copy the `java_src` directory from the StarSQL for Java installation directory to a working directory (the following procedures name the working directory `samples`).

   On a Windows computer:

   ```
   C> mkdir C:\samples
   C> xcopy %STARSQL_JAVA%\samples\java_src\*
        C:\samples /S
   ```

   On a UNIX computer:

   ```
   $ mkdir $HOME/samples
   $ cp -R $STARSQL_JAVA/samples/java_src/*
        $HOME/samples
   ```

2. Start Eclipse and select File–>Switch Workspace to create a new workspace.

3. Set up a project for the shared code, which is distributed in the `apps` subdirectory (such as C:\samples\apps if you copied the java_src files to a `samples` directory on drive C:).

   a. Select File–>New–>Project.

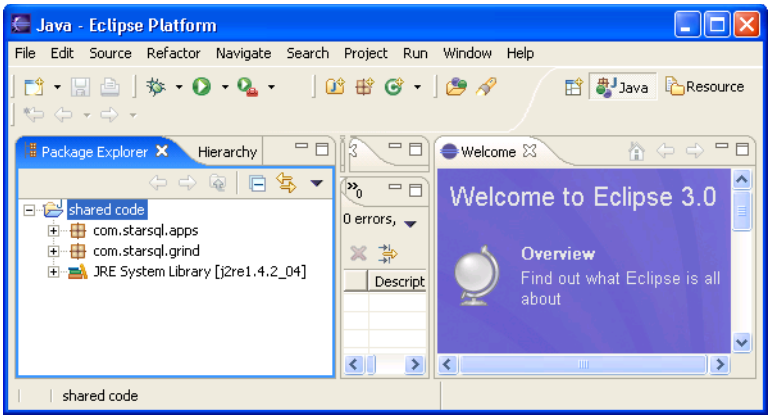
   b. In the New Project wizard, select Java Project and click Next.

   c. Enter a name for the project, such as Shared Code. For the Location, select the Create Project at External Location option and browse to the `apps` subdirectory of the working directory to which you copied the java_src files. Click Next to proceed.

   d. In the Java Settings pane of the New Project wizard, click **Finish**. When the Confirm Perspective Switch dialog appears, click **Yes** to approve switching to the Java Perspective.

After you set up the project for the shared code and switch to the Java Perspective, two Java packages—com.starsql.apps and com.starsql.grind—appear in the Package Explorer pane, as shown in Figure 5.

Figure 5. StarSQL for Java Packages in the Eclipse Platform



4. Create a new project for each of the sample applications you want to build.
   a. Select File–>New–>Project.
   b. In the New Project wizard, select Java Project and click Next.
   c. Enter a name for the project, such as CreateDS. For the Location, select the Create Project at External Location option and browse to the appropriate subdirectory of the working directory to which you copied the java_src files (such as C:\samples\CreateDS). Click Next to proceed.
   d. In the Java Settings window, click the Projects tab and select the project for the shared code (the project you created in step 3 on page 64) to include it on the build path for each of the other sample application projects.
   e. If you are creating a project to build the CreateDS sample application, click the Libraries tab of the Java Settings pane and click the Add External JARs button. Browse to the StarSQL for Java installation directory and select the StarSQL_JDBC.jar file.
   f. Click **Finish** to create the project.
5. Export the project to create a JAR file for the Java sample application.
   a. Select the desired project from the Package Explorer pane.
   b. Right-click and select the Export command.
   c. Select JAR file for the export destination and click Next.

d. Select the resource that corresponds to the sample application you are building. Also specify the export destination, including an appropriate name for the JAR file, and click Next.

e. In the JAR Packaging Options window, enable the option for saving the description of the JAR in the workspace, specify an appropriate name for the .jardesc file, and click Next.

f. For the Main Class field of the JAR Manifest Specification window, click **Browse** and select the class that corresponds to the sample application you are building. Click **Finish** to export the JAR file to the specified location.

6. Run the sample application.

a. To execute the application from within Eclipse, select the project, right-click, and select the Run–>Run command.

b. In the Run window select Java Application and click **New** to add a configuration for the project. Enter an appropriate name for the configuration. On the Main tab, ensure that the correct Project is selected, clicking **Browse** to select the correct project if necessary.

For the Main Class field, click **Search** and select the default package for the sample application.

Click the Classpath tab, select User Entries and click the **Add External JARs** button. Navigate to the StarSQL for Java installation directory and select the StarSQL_JDBC.jar file. If you want to connect using a datasource when you run the sample application, also add the `fscontext.jar` and `providerutil.jar` files from the `ext` subdirectory.

c. Click **Apply** to save the configuration, and click **Run** to run the sample application.

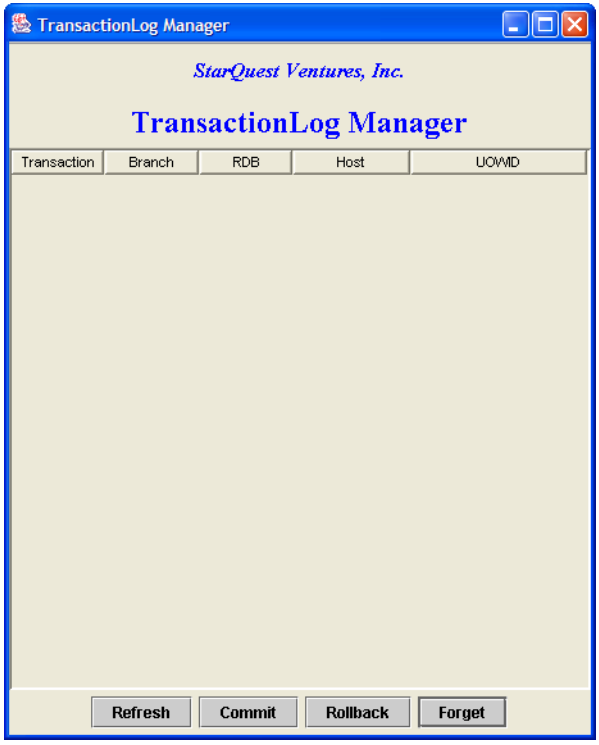
Managing Two-Phase Commit Transactions

A transaction is a collection of activities that involve changes to the database, all of which must be executed successfully if the changes are to be committed to the database, and none of which may be committed if any one or more of the activities fail. If one or more of the activities fail, the transaction is considered "in-doubt" because its integrity is questionable. Typically the Transaction Manager resolves in-doubt transactions without user intervention.

StarSQL for Java includes a Transaction Log Manager application that you can use to clear the StarSQL for Java transaction log or resolve in-doubt transactions. If the Transaction Manager log or the DB2 transaction log is not available or is corrupt, follow the instructions below to run the StarSQL for Java Transaction Log Manager.

- From a Mac OS X computer, use the Finder to open `Applications/starsql_java/samples`, double-click `samples.command`, and select TransactionLogManager.
- From other UNIX computers, run the `runapp.sh TransactionLogManager` script or the `samples.sh` script.
- From a Windows computer, select **Transaction Log Manager** from the **StarSQL for Java** program group.

Figure 4. Transaction Log Manager Application



Any transactions that are in-doubt appear in the display. Select the transaction and choose whether you want to commit it, roll it back, or forget it. Click the **Refresh** button to refresh the list of in-doubt transactions.

CHAPTER 4

Preparing Hosts for StarSQL Access

This chapter describes how to prepare host systems to enable StarSQL to provide access to the host databases.

It covers:

- Preparation required for all hosts
- Preparing a DB2 for z/OS host
- Preparing a DB2 for i host
- Preparing a DB2 for Linux, UNIX, and Windows (LUW) host
- Preparing a Derby Network Server host
- Preparing a DB2 Server for VSE & VM

These sections cover details of the DB2 environment that are pertinent to StarSQL for Java. For complete documentation of installation and configuration on the host, consult IBM's DB2 documentation, especially IBM's *DRDA Connectivity Guide*.

Preparation Required for All Hosts

Each StarSQL user must have an account on the host and have permission to access the necessary packages.

User Accounts

To connect to a DB2 database, each StarSQL for Java user needs an account on the host database. An account consists of a user ID and password.

You need to provide the host user account information to each StarSQL for Java user who needs to connect to the database.

Permissions

Usually a database administrator (DBA) is responsible for packages on the host, including binding them and granting permissions to use them. Depending on the host platform and the type of package used by the Java application, the DBA may need to grant StarSQL for Java users explicit permissions to access data used by the application.

Preparing DB2 on a z/OS Host

Preparing DB2 on a z/OS host for access with StarSQL for Java primarily involves configuring the Distributed Data Facility (DDF), which is a component of DB2 for z/OS. Its primary task is to process DRDA requests. DDF must be active for a desktop to connect to DB2 using StarSQL or any other DRDA requestor or client.

Configuring DDF

If your organization has not implemented distributed database capabilities, DDF may not be configured and activated. The DSNTINST CLIST provides two panels—DSNTIPR and DSNTIP5 for customizing a DB2 for z/OS subsystem to use native DRDA TCP/IP support. The DSNTIP5 panel is specific for TCP/IP. However, to use native TCP/IP support, you also must have APPC support configured and active because DB2 uses the network ID and the LU name to identify units of work. You specify the LU name that identifies the DB2 subsystem to VTAM and to uniquely identify logical units of work, in the DSNTIPR panel.

The values specified on these panels are used to generate the JCL that stores them in the DB2 bootstrap data set (BSDS) communication record.

If you are installing DB2, use the DDF panel DSNTIPR and DSNTIP5 to provide the following parameters. To change the DDF parameters after installation, run a customized configuration job DSNTIJUZ to update the BSDS.

- DDF Location Name. This name must be specified for the databaseName parameter that StarSQL for Java uses to connect to the host.
- Password used when connecting DB2 to VTAM, if a password is required.

- IP port to use for TCP/IP access. To enable support for TCP/IP, set the DRDA port in the DDF to 446.
- IP port to use for two-phase commit. The RESYNC PORT parameter in the DSNTIP5 panel specifies a TCP/IP port number for processing requests for two-phase commit re-synchronization. The RESYNC PORT must be different than the DRDA PORT.

For more information about establishing connectivity between client computers and DB2 for z/OS over a TCP/IP network, the IBM Redbook, *WOW! DRDA Supports TCP/IP: DB2 Server for OS/390 and DB2 Universal Database*, may be useful in addition to the installation guide for your version of DB2 for z/OS. For complete information about configuring DDF, consult IBM's DB2 for z/OS installation documentation and the IBM Redbook, *Distributed Functions of DB2 for z/OS and OS/390*.

Starting DDF

Use the following command, which requires authority of SYSOPR or higher, to start DDF:

```
-START DDF
```

When DDF starts successfully, the following messages are displayed:

```
DSNL003I - DDF IS STARTING
DSNL004I - DDF START COMPLETE LOCATION locname LU
netname.luname
```

If DDF has not been properly installed, the START DDF command fails and displays the following message:

```
DSN9032I - REQUESTED FUNCTION IS NOT AVAILABLE
```

If DDF has already been started, the START DDF command fails and displays the following message:

```
DSNL001I - DDF IS ALREADY STARTED
```

The following command shows whether DDF is running and, if so, the parameters that it is using:

```
-DIS DDF
```

Supporting Password Management Using DRDA Flows

There are two host requirements to support the ability of StarSQL for Java users to change their host passwords through StarSQL for Java on DB2 for z/OS, set Extended Security to YES (EXTSEC=YES). The default is NO. This can be done using either

- the DSNTIPR (DDF) panel on the DB2 installation dialog.
- a customized configuration job DSNTIJUZ, with the option EXTSEC=YES specified.

Using StarSQL with Stored Procedures

Stored procedures are application programs that reside on the host and are invoked via DB2. They are usually written in a traditional programming language like COBOL, RPG, or C. They may contain SQL statements for accessing the DB2 database or they may be used to access non-DB2 resources.

You invoke a stored procedure using the SQL Call statement, and receive output data in a result set or in output parameters. The Call statement is executed as any other SQL, with a JDBC Statement, PreparedStatement, or CallableStatement.

Registering Stored Procedures

The stored procedure should be registered on the host so that calling application can obtain information using the SQLProcedures and SQLProcedureColumns functions.

Use the CREATE PROCEDURE command to register the stored procedure in the system. The CREATE PROCEDURE command automatically updates the SYSIBM.SYSROUTINES catalog table.

```
CREATE PROCEDURE SYSPROC.STARPING (
IN REGION CHAR(8) CCSID EBCDIC,
IN PROGRAM CHAR(8) CCSID EBCDIC,
IN TRANSID CHAR(4) CCSID EBCDIC,
IN COMMLEN SMALLINT,
INOUT COMMAREA VARCHAR(32700) FOR BIT DATA,
OUT RC INTEGER,
OUT ABCODE CHAR(4) CCSID EBCDIC
)
PARAMETER STYLE GENERAL
LANGUAGE C
EXTERNAL NAME 'STARPING'
RESULT SETS 0
DETERMINISTIC
NO SQL
NO DBINFO
NO COLLID
ASUTIME NO LIMIT
NO WLM ENVIRONMENT
STAY RESIDENT NO
PROGRAM TYPE MAIN
SECURITY DB2
COMMIT ON RETURN YES
```

Calling Stored Procedures

To get a result set from the stored procedure call, use the following SQL statement syntax for calling the stored procedure:

```
Call MyProc( ?, ?, ?, ?)
```

Preparing a DB2 for i Host

This section discusses setting up a DB2 for i host for supporting a connection through StarSQL for Java.

It covers:

- creating a library/collection for SQL packages
- determining the database name
- enabling DRDA over TCP/IP
- supporting password encryption, LOB data types, and two-phase commit transactions

For complete information about setting up DB2 for i, consult IBM's installation documentation.

Creating a Library for SQL Packages

On DB2 for i, required SQL packages are stored in a collection or library. You may need to create the collection or library on the host.

Use the CRTLIB command to create a new library for the SQL packages used by StarSQL for Java. You can do this from a 5250 terminal session with a user ID that has QSECOFR privileges. The library does not have to be a SQL collection, but it must be accessible to all StarSQL users.

For example, the following command creates a new library named STARSQL:

```
CRTLIB STARSQL
```

Record the name of the library, as it must be specified as the `collection` property when configuring a data source to be used by StarSQL for Java.

Determining the Database Name

Determine the Relational Database (RDB) name of the DB2 for i. From the DB2 command line, enter:

```
WRKRDBDIRE
```

Look for an entry with a Remote Location value of *LOCAL. If such an entry does not exist, create it with the 1=ADD option. A common convention is to use the same name as the DB2 for i system name for the database name.

Make a note of the database name as you need to specify it as the `databaseName` property when configuring the data source used by the client computers.

Enabling DRDA Over TCP/IP

The Distributed Data Management (DDM) server allows client computers to access the DB2 functions. The DDM server supports remote SQL access, record level access, and remote journals. To initiate a DDM server job using TCP/IP communications a DRDA application or DDM source system connects to the well-known port number for TCP/IP, port 446 or 447. The DDM listener program, upon accepting the connection request, issues an internal request to attach the client's connection to a DDM server job.

The DDM listener program runs in a batch job in the QSYSWRK subsystem. There is one listener program that serves potentially many DDM server jobs. If you have access to iSeries Navigator you can verify whether DDM is configured by selecting TCP/IP from the Network–>Servers menu.

Follow the steps below if you need to configure a host running DB2/400 v4r2 and later to accept DRDA requests over TCP/IP:

1. Log on to the DB2 host.
2. Change the DDM TCP/IP Attributes to automatically start the listener program by entering:

   ```
   CHGDDMTCPA AUTOSTART(*YES)
   ```

3. Start the TCPIP DDM Server by entering:

   ```
   STRTCPSVR SERVER(*DDM)
   ```

When you are logged on to the DB2 host, you can examine which port DB2 is using to listen for DRDA requests using either the WRKSRVTBLE or the WRKTCPSTS command.

To use the WRKSRVTBLE command:

1. Enter WRKSRVTBLE.
2. Look for the DRDA entry with the port number.

To use the WRKTCPSTS command:

1. Enter WRKTCPSTS.
2. Choose option 3, “Work with TCP/IP connection status.”
3. Find the entry with port “drda” and press “F14=Display port numbers.” The default port number for DRDA is 446.

Registering Stored Procedures on DB2 for i

This section describes issues regarding stored procedures that are specific to IBM i hosts. Refer to "Using StarSQL with Stored Procedures" on page 72 for general information about using stored procedures.

Following is sample SQL for registering a stored procedure on an IBM i system. It assumes that the COBOL program MYLIB.MYPRGM already exists on the system. This statement modifies the QSYS2.SYSPROCS and QSYS2.SYSPARMS catalog tables for you.

```
CREATE PROCEDURE MYLIB.MYPROC (INOUT PARM1 CHAR(10))
EXTERNAL NAME MYLIB.MYPGM LANGUAGE COBOL GENERAL
```

In the above example, the procedure name is MYLIB.MYPROC, which references the COBOL program MYLIB.MYPGM. The program takes one input parameter called PARM1 which is a char field of length 10. This procedure does not return a result set.

Refer to the *IBM SQL Reference and SQL Programming Guide* for the appropriate version of IBM i for more information on registering a stored procedure and the full syntax of the CREATE PROCEDURE statement.

Configuring Support for the SSL Protocol

To configure an IBM i host system to use the Secure Sockets Layer (SSL) protocol you must have the following components for an AS/400 host system:

- Digital Certificate Manager - 5722-SS1 (v5rx), 5761-SS1 (6.1), or 5770-SS1(7.x) option 34
- TCP/IP Connectivity Utilities - 5722-TC1(v5r4), 5761-TC1 (6.1), or 5770-SS1 (7.x)

- IBM HTTP Server - 5722-DG1 (v5rx), 5761-DG1 (6.1) or 5770-DG1 (7.x)

Following are general procedures for configuring SSL on the IBM i host. Refer to your IBM documentation for details, especially the IBM i system documentation and the "*IBM iSeries Wired Network Security OS/400 V5R1 DCM and Cryptography Enhancements*" Redbook.

1. Start the Admin HTTP instance and use a browser to configure the Digital Certificate Manager.
   a. Create a local Certificate Authority or obtain a certificate from a public Internet Certificate Authority.
   b. Create a *SYSTEM certificate store.
   c. Use "Manage Applications" to assign a server certificate to the OS/400 DDM/DRDA server.
   d. After you assign the certificate, restart the DDM/DRDA server:
      ```
      ENDTCPSVR *DDM
      STRTCPSVR *DDM
      ```
2. If necessary, set the port on which the DDM/DRDA server listens for SSL conversations. The default port for SSL is 448.

Considerations for Specific IBM i Releases

In general, it is a good idea to stay current on PTF packages and the DB2 Group PTF, and to use the latest version of StarSQL. The following sections describe more specific issues for particular versions of OS/400.

IBM i 6.1 Issues

If you are using V6R1, ensure that the following PTFs are either installed or superseded:

- cum C8064610 or later
- PTF 5761SS1-SI30581

OS/400 v5r4 Issues

If you are running V5R4 of the i5/OS, ensure that the following PTFs are either installed or superseded:

- PTF 5722SS1-SI23461
- PTF 5722SS1-SI24317

If you are using StarSQL to invoke Java stored procedures on an AS/400, make sure the following PTF is installed:

- PTF 5722SS1-SI22551

If your host needs to support BLOB data, make sure the following PTFs are installed:

- PTF 5722SS1-SI22324
- PTF 5722SS1-SI22335
-

Preparing a DB2 LUW Host

This section provides details for setting up a DB2 for Linux, UNIX, and Windows host to support a connection through StarSQL for Java. It covers:

- enabling DRDA support for TCP/IP
- locating the database name

For complete information about setting up DB2 for LUW, consult IBM's installation documentation.

Enabling DRDA Support for TCP/IP

When using TCP/IP to connect to a DB2 for LUW host, make sure that the host has a static IP address. You may experience problems installing DB2 LUW on a computer using DHCP. To be recognized, the system requires an entry in the DNS (Domain Name Server) or an entry in the HOSTS file.

Although you can configure StarSQL for Java to communicate with DB2 using any available port, port 446 is the standard port used for DRDA communications and is the default value that StarSQL uses if another is not specified. DB2 LUW uses a default value of 50000 for DRDA communications.

If there is a fire wall that monitors network traffic to the host, be sure that it allows DRDA communications to pass through the port that you configure for DRDA requests.

Using db2 Commands to Specify the DRDA Port

Issue the following db2 commands to change the port that DB2 uses for DRDA communications.

1. Enter the following command to determine on which port DB2 is listening to for TCP/IP communications.

```
db2 get dbm configuration
```

In the dbm configuration, look for “TPC/IP Service Name (SVCENAME).” It will have a value similar to “db2c_DB2.” This is the symbolic name of the connection port.

2. Find the symbolic name of the connection ports in the services file, which is typically located in /etc/services.
3. Edit the services field and change the value of the TCP/IP connection port to 446 and set the interrupt port to 447.
4. Restart the DB2 instance for the changes to take effect.

To verify that DB2 is listening on the correct port, from either the client or the server enter:

```
telnet <host> 446
```

If DB2 is listening on that port, no error is returned to the telnet window. If DB2 is not listening on that port, you will see an error similar to the following and may need to contact your DB2 administrator for the correct port number to use:

```
Could not open connection to the host, on port 446: Connect
failed.
```

Click the Close (X) icon to close the telnet window.

Enabling Encryption

If you configure the StarSQL for Java driver to send encrypted user IDs and passwords (see "pwdEncryption Parameter" on page 39), be sure to enable the database for encryption. On a DB2 for Linux, UNIX, and Windows host you enable encryption by setting the Server Connection Authentication (SRVCON_AUTH) parameter to “Server encrypt” and restarting the instance.

Using db2 Commands to Enable Encryption

Issue the following db2 commands to enable encryption.

1. To enable encryption from a db2 command window, enter the following command:

   ```
   db2 update dbm cfg using SRVCON_AUTH
   SERVER_ENCRYPT
   ```

2. Stop and restart the instance by issuing the following commands:

   ```
   db2stop
   db2start
   ```

Locating the Database Name

When an administrator sets up a DB2 for LUW system, they assign names to DB2 databases. You will need to enter the Database Name for the `databaseName` property when you configure data sources on the desktop. You can display a list of the databases that have been created by issuing the following command:

```
db2 list database directory
```

Contact your database administrator if you need to determine which databases should be accessible to StarSQL for Java clients.

Preparing a Derby Network Server Host

The Derby Network Server is part of the Derby software distribution and provides a framework for multi-user connectivity to Derby databases across a network. Derby must be started in the Network Server mode (vs. the embedded mode) for client computers to connect to a Derby database using StarSQL for Java.

The default of the Derby Network Server is to start with user authentication disabled. Refer to the Derby documentation for information about enabling user authentication and running under the Java security manager to help avoid security problems.

Setting Network Server Properties

Typically the Java system property `derby.system.home` specifies the system directory, which contains subdirectories that hold the databases that are available to the Derby Network Server. If you do not explicitly set the `derby.system.home` property when starting Derby, the default is the directory in which Derby was started. You need to specify at least the name of the database you want to access with the StarSQL for Java driver and which port to use. Contact your database administrator if you do not know which Derby databases should be accessible to StarSQL for Java clients.

The Network Server properties can be specified three ways:

- on the command line
- in the `.bat` or `.ksh` files, loading the properties by executing **java -D**
- in the `derby.properties` file

Properties in the command line or in the .bat or .ksh files take precedence over the properties in the derby.properties file. Arguments included on commands that are issued on the command line take precedence over property values.

The properties that are particularly of interest for using the StarSQL for Java driver are those that allow remote connections and determine which protocol to use.

```
derby.drda.host=hostname
derby.drda.portNumber=portnumber
derby.drda.sslMode=SSL | TLS || SSLv3 | TLSv1
```

Enabling Remote Connections

The default of the Derby Network Server is to start with user authentication disabled. Before you enable remote connections ensure that you are running under the security manager and that user authorization is enabled. Refer to the Derby documentation for details about enabling user authentication and running under the Java security manager.

The `derby.drda.host` property causes the Network Server to listen on a specific interface, allowing multiple instances of Network Server to run on a single computer with a unique host:port combination. By default the Derby Network Server listens only on the loopback IP address (127.0.0.1) and port 1527, which restricts access to the local computer. To make the Derby Network Server accessible to other computers on the network, specify a particular interface (host name or IP address) and specify a port number other than 1527 on which to listen for connections. The Derby Network Server must listen for connections on the same port that the StarSQL for Java driver is configured to use, which is port 446 by default.

The following shows how to use a java command to start the Derby Network Server to listen on port 446 for connection requests from any host name or IP address.

```
java -jar $DERBY_HOME%\lib\derbyrun.jar server start
          -h 0.0.0.0 -p 446
```

The -h and -p options can be specified regardless of how you choose to start Derby Network Server. Refer to the Derby documentation for details about the additional methods, such as running the startNetworkServer script or using a java command to directly invoke the NetworkServerControl class.

Configuring Support for the SSL Protocol

The Derby Network Server supports network security with Secure Socket Layer/Transport Layer Security (SSL/TLS). With SSL/TLS the client-server communication protocol is encrypted and both the client and the server may, independently of each other, require certificate-based authentication of the other part. You can configure SSL for the Derby Network Server to be Off, or to use SSL encryption with no peer authentication (basic), or to use SSL encryption and peer authentication (peerAuthentication). To enable SSL support, use the `ssl` option on the command line when you start Derby or specify the `derby.drda.sslMode` property in `derby.properties`.

For example, the following command would start Derby Network Server using the "basic" SSL encryption option:

```
java -jar derbyrun.jar server start -ssl basic
```

You also need to use the **keytool** to create a server key pair, and specify the key store and password when starting the server. For SSL operation the server always needs a key pair. If the server runs in peer authentication mode, then each client needs its own key pair. The **keytool** is located in the JRE bin directory. Entering the following command prompts for the information needed to generate a key pair.

```
keytool -genkey myDatabase -keystore serverKeyStore.key
```

The key pair is located in a file which is called a key store and the JDK's SSL provider needs the system properties `javax.net.ssl.keyStore` and `javax.net.ssl.keyStorePassword` to access the key store. Specify the key store and password when starting the server, such as shown below:

```
java -Djavax.net.ssl.keyStore=serverKeyStore.key \
     -Djavax.net.ssl.keyStorePassword=myPassword \
     -jar %DERBY_HOME%\lib\derbyrun.jar server start
                       -h 0.0.0.0 -p 446 -ssl basic
```

Refer to the Derby documentation at http://db.apache.org/derby/docs/10.4/adminguide/cadminsslkeys.html for additional details about key and certificate handling.

Configuring the StarSQL for Java for Derby Connections

To use the StarSQL for Java driver to connect a client computer to a Derby host, StarSQL must be configured with the `sendUnicode` parameter set to **`True`** and the `decimalDelimiter` parameter set to **`host`**. Refer to "StarSQL for Java Configuration Properties" on page 29 for discussion of these and other parameters that can be configured for the StarSQL for Java driver.

Preparing a DB2 Server for VSE & VM

Support for the TCP/IP network protocol was made available beginning with DB2 for VSE v7.1 and DB2 for VM v6.1. Version 6.1 of DB2 for VM also was the first release to support stored procedures. Contact StarSQL Customer Support if you want to use StarSQL over an SNA network to access a DB2 Server for VSE or VM.

When you configure the StarSQL data source, specify the **dbname** parameter as the Database Server Name in the data source. For VSE the database name typically is defined in the **DBNAME** directory to allow the database administrator to ensure that a unique TCP/IP port number is assigned for each DB2 Server for VSE. The application server must be able to determine on which TCP/IP port to listen for connections. The

DBNAME directory contains an entry for each DB2 server that specifies the **dbname** and the TCP/IP port number to use (the TCPPORT parameter). The TCP/IP port also can be specified using the TCPPORT initialization parameter, as described in the *DB2 Server for VSE & VM Operation* manual.

For VM the **dbname** parameter is specified in the CMS Communications Directory in the CMS file of type **NAMES**.

Refer to your IBM documentation for details about configuring DB2 Server to support DRDA connections over a TCP/IP network. The *System Administration Guide* for VSE v7.3 is publication number SC09-2981, and for VM v7.3 the publication number is SC09-2980.

CHAPTER 5

Binding Packages

A SQL package is an object that DB2 uses to process a SQL statement. Different packages are required, depending on configuration parameters, to execute dynamic SQL. Note that Derby uses JDBC's Prepared Statement, and does not provide SQL commands for dynamic SQL. Derby does use standard packages that are provided by the DBMS to determine whether to use held cursors, as further described in "Derby Held Cursors Packages" on page 91.

For DB2 host systems the StarSQL for Java driver assumes that the dynamic SQL package already exists in the host database. If the StarSQL for Java driver cannot find the required packages when they are needed, it creates them automatically.

The names and locations of the packages that are bound vary depending on the values that are specified for certain configuration settings in the data source definition, or in the connection URL, when the initial connection is made.

The StarSQL for Java user requires certain permissions to bind and use packages on the host. Typically, the package will be created by an administrator and other users will be granted permission to use the packages.

You can use the StarAdmin utility, which is available from StarQuest, to bind packages on a host. Packages that are bound using StarAdmin can be used by any version (Java, Linux, UNIX, or Windows) of the StarSQL driver.

Using Dynamic SQL Packages

There are several packages that can be used for executing dynamic SQL. The default dynamic package varies by host platform. Some dynamic packages are used based on the desired transaction isolation level and the heldCursors, keepDynamic, and dynamicRules settings configured in the data source or in the connection URL.

The isolation level is set when the application calls the setTransactionIsolation() method on the java.sql.Connection object. The section "StarSQL for Java Configuration Properties" on page 29 describes the heldCursors, keepDynamic, and dynamicRules properties. If you set the dateFormat configuration property to use a date format other than the standard JDBC format (yyyy-mm-dd), you must bind a different package, as described on page 34.

Table 12 shows the standard dynamic packages that can be bound.

Table 12. Dynamic Packages

Package Name	Isolation Level and heldCursors Setting
SWRC0000	Read Committed / heldCursors no
SWNC0000	Commit None/heldCursors no
SWRR0000	Repeatable Read / heldCursors no
SWRU0000	Read Uncommitted / heldCursors no
SWTS0000	Serializable / heldCursors no
SWRC1000	Read Committed / heldCursors yes
SWRR1000	Repeatable Read / heldCursors yes
SWRU1000	Read Uncommitted / heldCursors yes
SWTS1000	Serializable / heldCursors yes

Table 13 through Table 15 show the packages that can be bound when using the keepDynamic option, or if the dynamicRules option is set to BIND. These tables are only applicable for hosts running DB2 for OS/390 v5.1 and later.

Table 13. Dynamic Packages on DB2 for z/OS with keepDynamic=Yes

Package Name	Isolation Level and heldCursors Setting
SWRC4000	Read Committed / heldCursors no
SWRR4000	Repeatable Read / heldCursors no
SWRU4000	Read Uncommitted / heldCursors no
SWTS4000	Serializable / heldCursors no
SWRC5000	Read Committed / heldCursors yes
SWRR5000	Repeatable Read / heldCursors yes
SWRU5000	Read Uncommitted / heldCursors yes
SWTS5000	Serializable / heldCursors yes

Table 14. Dynamic Packages on DB2 for z/OS with dynamicRules=BIND and keepDynamic=No

Package Name	Isolation Level and heldCursors Setting
SWRC2000	Read Committed / heldCursors no
SWRR2000	Repeatable Read / heldCursors no
SWRU2000	Read Uncommitted / heldCursors no
SWTS2000	Serializable / heldCursors no
SWRC3000	Read Committed / heldCursors yes

Package Name	Isolation Level and heldCursors Setting
SWRR3000	Repeatable Read / heldCursors yes
SWRU3000	Read Uncommitted / heldCursors yes
SWTS3000	Serializable / heldCursors yes

Table 15. Dynamic Packages on DB2 for z/OS with dynamicRules=BIND and keepDynamic=Yes

Package Name	Isolation Level and heldCursors Setting
SWRC6000	Read Committed / heldCursors no
SWRR6000	Repeatable Read / heldCursors no
SWRU6000	Read Uncommitted / heldCursors no
SWTS6000	Serializable / heldCursors no
SWRC7000	Read Committed / heldCursors yes
SWRR7000	Repeatable Read / heldCursors yes
SWRU7000	Read Uncommitted / heldCursors yes
SWTS7000	Serializable / heldCursors yes

Permissions Required for Packages

This section describes the permissions that are required to bind packages and to use packages.

For Binding Packages

If the required dynamic SQL package does not already exist, StarSQL for Java automatically creates it on the first execution of an SQL statement.

The user who performs the package binding must have permission to bind packages in the host database. On DB2 for i, the DBA must grant CREATE permission for this user for the specified library. On other hosts, the DBA must grant CREATE IN COLLECTION and BINDADD permissions. The binding will fail if the user does not have these permissions.

After the StarSQL for Java packages have been bound, the DBA administrator can revoke these permissions.

For Using Packages

After the packages have been bound, the DBA must grant StarSQL for Java users permission to execute them. On DB2 for i, users must be granted *USE permission on the packages, which can usually be done by the package owner. On other hosts, the DBA must grant StarSQL for Java users EXECUTE or RUN permissions on the packages.

On all hosts, except for DB2 for z/OS, for applications that use dynamic SQL, the StarSQL for Java user needs explicit permissions to read (SELECT) and write (UPDATE/INSERT/DELETE) the columns and tables accessed by the application. The DBA needs to grant these permissions to “public” or to the various groups.

On DB2 for z/OS, the required permissions depend on the setting of the dynamicRules property. If the dynamicRules property is set to RUN, which is the default, the StarSQL for Java user needs explicit permissions to read and write the columns and tables accessed by the application. If the dynamicRules property is set to BIND, the user has the permissions of the package owner, except that the following SQL statements cannot be executed regardless of the permissions of the package owner:

- SET CURRENT SQLID
- GRANT
- REVOKE
- ALTER
- CREATE
- DROP
- Any SQL statement that cannot be prepared as a dynamic SQL statement.

See "dynamicRules Parameter" on page 37 for details about setting the dynamicRules property.

Granting Use Permissions

To grant EXECUTE authority on the SQL packages you can execute SQL statements similar to those shown in the following sections

For DB2 for z/OS

Using SPUFI on the host or the SQL pass through mode of a JDBC-enabled application, such as the sample QueryApp application, execute the following SQL statements:

```
GRANT EXECUTE ON PACKAGE
STARSQL.SWRC5000
TO PUBLIC
```

For DB2 for LUW (Linux, UNIX & Windows)

Using the DB2 Command Line Processor or the SQL pass through mode of a JDBC-enabled application, such as the sample QueryApp application, execute the following SQL statements:

```
GRANT EXECUTE ON PACKAGE
STARSQL.SWRC0000
TO PUBLIC
```

For DB2 for i

There are several methods available in the IBM i environment for granting EXECUTE privileges. You can:

- use STRSQL, which is the interactive SQL interpreter, available in the Query Manager and SQL Development Kit, Licensed Program 57xx-ST1.
- use the SQL pass through mode of a JDBC-enabled application, such as the sample QueryApp application
- use CRTQMQRY/STRQMQRY or RUNSQLSTM to run a file that contains the SQL commands
- use the db2 command of the QSH environment

Regardless of which method you use, execute the following SQL statements to grant EXECUTE authority on the SQL package:

```
GRANT EXECUTE ON PACKAGE
STARSQL.SWNC0000
TO PUBLIC
```

In addition, you can use IBM i authority commands to grant *USE authority for all users (*PUBLIC) to the library packages:

1. From a 5250 session, enter **WRKLIB <***package library name***>**.
2. Select Option 12 (work with objects).
3. Select Option 2 (edit authority on each object one at a time).
4. Change **PUBLIC *EXCLUDE** to **PUBLIC *USE**.

Another alternative is to use the GRTOBJAUT command from a 5250 session to grant *USE authority for the library and EXECUTE authority for the packages to all users. The following example assumes that the package library is named "STARSQL."

```
GRTOBJAUT OBJ(STARSQL) OBJTYPE(*LIB) USER(*PUBLIC)
AUT(*USE)
GRTOBJAUT OBJ(STARSQL/*ALL) OBJTYPE(*ALL)
USER(*PUBLIC) AUT(*USE)
```

Derby Held Cursors Packages

Derby uses packages that are supplied by the DBMS to reflect the requested isolation level and whether the cursor is held. Setting the transaction isolation level for a connection allows a user to specify how severely the user's transaction should be isolated from other transactions. If a connection does not specify its isolation level, it inherits the default isolation level for the Derby system. To override the inherited default value, use the methods of `java.sql.Connection` or use the WITH clause to change the isolation level for the current statement only (not the transaction).

The following table shows the `java.sql.Connection` isolation levels that are supported and the package that is used depending on whether the cursor is held open after a commit takes place.

Table 16. Derby Isolation Level and Held Cursor Packages

Cursor Isolation Level	Held	Not Held
TRANSACTION_NONE	SYSLH0000	SYSSLN000
TRANSACTION_READ_UNCOMMITTED	SYSLH100	SYSLN100

Cursor Isolation Level	Held	Not Held
TRANSACTION_READ_COMMITTED	SYSLH200	SYSLN200
TRANSACTION_REPEATABLE_READ	SYSLH300	SYSLN300
TRANSACTION_SERIALIZABLE	SYSLH400	SYSLN400

Appendix A

National Language Support

StarSQL for Java is a pure Java JDBC driver that converts JDBC calls into the Distributed Relational Database Architecture (DRDA) protocol used by most host database systems. These host database systems use a variety of character encodings to support the diverse languages and cultures of the world.

Java applications use the Unicode Standard to represent all characters. The Unicode Standard defines a unique number for every character, regardless of the platform or language. This version of StarSQL for Java supports the JDBC 3.0 API, and uses Unicode to support double-byte and multi-byte character sets.

When connected to a host configured for single-byte character set (SBCS) data, StarSQL for Java sends SQL statements and input data to DB2, converting Unicode data to the host-specified CCSID for SBCS data. IBM developed the Coded Character Set Identifier (CCSID) numbering system to represent a character set, a code page, an encoding scheme, and additional coding-related information.

To determine the desired language for the client computer, StarSQL for Java uses the locale that is configured for the JVM. A *locale* defines a particular combination of language and region. Data from the host is automatically converted to Unicode.

Character encoding conversion supports converting text between Unicode and other character encodings when receiving and sending data between a host and client computer. Java provides a set of classes that convert many standard character encodings to and from Unicode. The character encodings that these classes support are listed in the Oracle Java documentation. Contact StarQuest Customer Support to request that additional languages or codesets be supported by StarSQL for Java.

If the locale for the JVM specifies that the client is using a double-byte language (Chinese, Japanese, or Korean), the StarSQL for Java driver sends mixed-byte character set (MBCS) data to the host using CCSID 1208 (UTF-8), and double-byte character set (DBCS) and GRAPHIC data using CCSID 1200 (UTF-16).

For SBCS data you can control which CCSID is used by setting the typdefovr property (described on page 41) of StarSQL for Java. If the host computer is configured to support Chinese, Japanese, or Korean characters sets, using EBCDIC or Unicode to represent the DBCS and MBCS character sets, you must set the sendUnicode property of the StarSQL driver to True (see "sendUnicode Parameter" on page 40).

Determining the Default Character Set of the Client

StarSQL determines the local client's code page according to the default locale that is set for the JVM or JRE. To determine the name of the default character set for a local system you can run the following program.

```
import java.io.*;
import java.nio.charset.*;

public class Encode3 {
    public static void main(String args[])
      throws IOException {
            FileWriter filewriter =
                new FileWriter("out");
      String encname =
           filewriter.getEncoding();
      filewriter.close();
      System.out.println(
            "default charset is: " + encname);

   }
}
```

Running this program should display a result such as:

```
default charset is: Cp1252
```

If the default character set is for a Group 2 language (Chinese, Japanese, or Korean), StarSQL for Java sends Unicode data to the host. For all other languages, StarSQL for Java sends the data using the coded character set specified by the typdefovr (page 41) property.

To control the SBCS CCSID that StarSQL for Java uses to send outbound data, you can modify the typfefovr value for the StarSQL data source as described in "typdefovr Parameter" on page 41.

Determining Which Character Set Conversions are Supported

StarSQL for Java uses the Java classes and libraries to convert from one character set to another. Translation errors can occur if the JVM/JDK that is installed does not include the necessary character converters. You can run the following program to determine which character sets are supported by the Java environment.

```
import java.nio.charset.Charset;
import java.util.Iterator;
import java.util.Map;

public class AvailableCharSets {
 public static void main(String[] args) {
Map charSets = Charset.availableCharsets();
Iterator it = charSets.keySet().iterator();
while (it.hasNext()) {
    String csName = (String) it.next();
    System.out.print(csName);
    Iterator aliases = ((Charset)
    charSets.get(csName)).aliases() .iterator();
    if (aliases.hasNext())
    System.out.print(": ");
    while (aliases.hasNext()) {
    System.out.print(aliases.next());
    if (aliases.hasNext())
    System.out.print(", ");
    }
    System.out.println();
    }
 }
}
```

Running this program outputs a list of the character sets for which Java converters are available, such as shown in the following excerpt.

```
Big5: csBig5

Big5-HKSCS: big5-hkscs, Big5_HKSCS, big5hkscs

EUC-JP: eucjis, x-eucjp, csEUCPkdFmtjapanese, eucjp,
Extended_UNIX_Code_Packed_Format_for_Japanese, x-euc-jp, euc_jp

EUC-KR: ksc5601, 5601, ksc5601_1987, ksc_5601, ksc5601-1987,
euc_kr, ks_c_5601-1987, euckr, csEUCKR

GB18030: gb18030-2000

GBK: windows-936, CP936

ISO-2022-JP: jis, jis_encoding, csjisencoding, csISO2022JP,
iso2022jp

ISO-2022-KR: ISO2022KR, csISO2022KR

ISO-8859-1: iso-ir-100, 8859_1, ISO_8859-1, ISO8859_1, 819,
csISOLatin1, IBM-819, ISO_8859-1:1987, latin1, cp819, ISO8859-1,
IBM819, ISO_8859_1, l1

ISO-8859-13: iso8859_13
```

ISO-8859-15: 8859_15, csISOlatin9, IBM923, cp923, 923, L9, IBM-923, ISO8859-15, LATIN9, ISO_8859-15, LATIN0, csISOlatin0, ISO8859_15_FDIS, ISO-8859-15

ISO-8859-2: l2, iso-ir-101, ISO_8859-2:1987, ISO_8859-2, latin2, csISOLatin2, iso8859_2

ISO-8859-3

ISO-8859-4: iso-ir-110, l4, latin4, csISOLatin4, iso8859_4, ISO_8859-4:1988, ISO_8859-4

ISO-8859-5: cyrillic, iso8859_5, ISO_8859-5, iso-ir-144, csISOLatinCyrillic

ISO-8859-6

ISO-8859-7: greek8, ECMA-118, sun_eu_greek, ELOT_928, ISO_8859-7:1987, iso-ir-126, ISO_8859-7, iso8859_7, greek, csISOLatinGreek

ISO-8859-8

ISO-8859-9: iso-ir-148, latin5, l5, ISO_8859-9, ISO_8859-9:1989, csISOLatin5, iso8859_9

JIS_X0201: JIS_X0201, X0201, JIS0201, csHalfWidthKatakana

JIS_X0212-1990: jis_x0212-1990, iso-ir-159, x0212, JIS0212, csISO159JISX02121990

KOI8-R: koi8, cskoi8r

Shift_JIS: shift-jis, x-sjis, ms_kanji, shift_jis, csShiftJIS, sjis, pck

TIS-620

US-ASCII: ISO646-US, IBM367, ASCII, cp367, ascii7, ANSI_X3.4-1986, iso-ir-6, us, 646, iso_646.irv:1983, csASCII, ANSI_X3.4-1968, ISO_646.irv:1991

UTF-16: UTF_16

UTF-16BE: X-UTF-16BE, UTF_16BE, ISO-10646-UCS-2

UTF-16LE: UTF_16LE, X-UTF-16LE

UTF-8: UTF8

windows-1250: cp1250

windows-1251: cp1251

windows-1252: cp1252

windows-1253: cp1253

windows-1254: cp1254

windows-1255

windows-1256

windows-1257: cp1257

windows-1258

windows-31j: csWindows31J, windows-932, MS932

x-EUC-CN: gb2312, EUC_CN, euccn, euc-cn, gb2312-80, gb2312-1980

x-euc-jp-linux: euc_jp_linux, euc-jp-linux

x-EUC-TW: cns11643, euc_tw, EUC-TW, euctw

x-ISCII91: iscii, ST_SEV_358-88, iso-ir-153, csISO153GOST1976874, ISCII91

x-JIS0208: JIS_C6626-1983, JIS0208, csISO87JISX0208, x0208, JIS_X0208-1983, iso-ir-87

x-Johab: johab, ms1361, ksc5601-1992, ksc5601_1992

x-MS950-HKSCS: MS950_HKSCS

x-mswin-936: ms936, ms_936

x-windows-949: windows949, ms_949, ms949

x-windows-950: windows-950, ms950

StarSQL for Java includes built-in conversions for some CCSIDs (such as 37, 273, 277, 278, 280, 284, 285, 297, 500, 871, 1140 through 1149, and 1252). StarSQL for Java uses the following character set names when it needs to access the Java-supplied converters.

```
367 = "ASCII"
420 = "Cp420"
424 = "Cp424"
620 = "TIS620"
813 = "ISO8859_7"
819 = "ISO8859_1"
850 = "Cp850"
852 = "Cp852"
855 = "Cp855"
856 = "Cp856"
857 = "Cp857"
858 = "Cp858"
869 = "Cp869"
870 = "Cp870"
874 = "Cp874"
875 = "Cp875"
912 = "ISO8859_2"
915 = "ISO8859_5"
916 = "ISO8859_8"
918 = "Cp918"
920 = "ISO8859_9"
921 = "ISO8859_13"
922 = "Cp922"
923 = "ISO8859_15"
930 = "Cp930"
933 = "Cp933"
935 = "Cp935"
937 = "Cp937"
939 = "Cp939"
```

```
1006 = "Cp1006"
1025 = "Cp1025"
1026 = "Cp1026"
1046 = "Cp1046"
1097 = "Cp1097"
1098 = "Cp1098"
1112 = "Cp1112"
1122 = "Cp1122"
1123 = "Cp1123"
1124 = "Cp1124"
1200 = "UnicodeBigUnmarked"
1201 = "UnicodeBigUnmarked"
1202 = "UnicodeLittleUnmarked"
1208 = "UTF8"
1250 = "Cp1250"
1253 = "Cp1253"
1254 = "Cp1254"
1255 = "Cp1255"
1256 = "Cp1256"
```

Appendix B

Troubleshooting Tips and Techniques

This appendix provides details about using the tracing and logging utilities to help troubleshoot communication problems, along with other tips and techniques for maximizing use of StarSQL for Java.

Troubleshooting Communication Problems

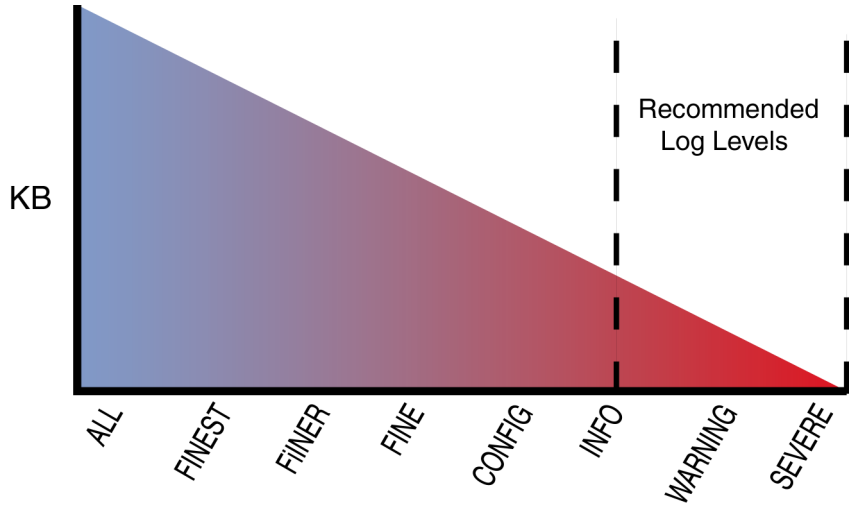
The StarSQL for Java driver provides a facility for tracing API activity and logging DRDA communications between the driver and the DRDA host. You can log driver events, errors, API calls, and other internal driver activity to the console or a file, or to both. Typically only messages of high severity are logged to the console and you should not need to enable other logging unless you are working with a StarQuest Technical Support engineer to troubleshoot a problem.

The amount and the type of information that is logged is determined by the log level, which can be different for the console messages and the file messages. For example, you can log informational messages to file while you have only severe messages display on the computer monitor.

StarSQL for Java supports seven levels of logging. As illustrated in Figure 5, the amount of information that is logged increases as the levels of logging become more broad. The ALL level logs all activity, including DRDA communication between the driver and host. The FINER level logs API calls and other internal method calls.

The broad levels of logging can significantly degrade performance of all computers involved in the communications. If you need to enable logging to troubleshoot a problem, remember to disable it as soon as you have logged the necessary messages.

Figure 5. Logging Levels



If you do not configure logging, only messages with a level of SEVERE are logged to the console. If you want to enable a different level of logging, or log messages to file, you must configure the logging facility.

Configuring the Logging Facility

The following steps describe how to configure the logging facility.

1. Copy the sample configuration file, `sqlogging.properties`, that is provided with StarSQL for Java from the StarSQL installation directory to a working directory.
2. Use a text editor to change the values of the `sqlogging.properties` file as desired. Table 17 describes the properties that you can configure to control logging.
3. Copy the customized `sqlogging.properties` file to the `lib` directory where the JRE is installed. This typically is a directory such as C:\Program Files\Java\jdk1.5.0_09\jre\lib on Windows, or /usr/jdk1.5.0_09/jre/lib on UNIX. If you have more than one JVM installed, copy the configured `sqlogging.properties` file to each `lib` directory.

Table 17 describes the properties that you can configure to control logging. The logging facility uses two handlers. The ConsoleHandler writes messages to the computer monitor, and the FileHandler writes messages to file. The .level property specifies

which types of events are logged to the file handler and the console handler if a logging level is not specified for them. If .level is set to OFF, no messages are logged to the console or file. If both the global .level parameter and the FileHandler.level or the ConsoleHandler.level parameters are set, the most restrictive level of logging prevails.

For example, if the global `.level` parameter is set to ALL, setting `FileHandler.level` to INFO results in logging messages of level INFO or more severe to the log file. If the global `.level` parameter is set to SEVERE, setting `FileHandler.level` to INFO results in logging SEVERE messages to the log files.

Table 17. Properties for Configuring the Logging Facility

Logging Configuration Property	Description	Default Value	Valid Values
.level	Sets the global logging level, which is used unless the FileHandler.level or ConsoleHandler.level is set to a more restrictive (detailed) level of logging. No logging occurs if `.level` is set to OFF.	OFF	OFF | ALL | FINEST | FINER | FINE | CONFIG | INFO | WARNING | SEVERE
FileHandler.pattern	Specifies a pattern for generating the output file name.	“%h/starsql_java%g.log”	see information that follows this table
FileHandler.limit	Sets the maximum character limit of the log file. When this limit is reached, the file is closed and the next file in rotation is opened for logging. If FileHandler.count is set to 1, prior messages are overwritten when the file reaches its maximum size.	0 (unlimited)	0 to a specific number of bytes

Logging Configuration Property	Description	Default Value	Valid Values
FileHandler.count	Sets the number of files to be used in the rotation of log files. When a file reaches the maximum size allowed by the FileHandler.limit, the file is closed and the next file is opened for writing log messages.	1	1 - maximum allowed by operating system
FileHandler.level	Sets the logging level of the FileHandler. If this property is set, messages for the corresponding level of logging are written to file. The FileHandler.pattern property sets the location and name of the file, and the FileHandler.count sets the number of files that are used to log messages.	OFF	OFF | ALL | FINEST | FINER | FINE | CONFIG | INFO | WARNING | SEVERE
ConsoleHandler.level	Sets the logging level of the ConsoleHandler. If this property is set, messages for the corresponding level of logging are written to the computer monitor.	SEVERE	OFF | ALL | FINEST | FINER | FINE | CONFIG | INFO | WARNING | SEVERE

For the FileHandler.pattern property, the pattern can be a string that includes the following special placeholders, which are replaced at runtime:

/	the local pathname separator
%t	the system temporary directory
%h	the value of the "user.home" system property
%g	the generation number to distinguish rotated logs
%u	a unique number to resolve conflicts
%%	translates to a single percent sign "%"

If you specify a "%g" field and a file count (FileHandler.count) greater than one, the generation number is added to the end of the log filenames. The default pattern is "%h/starsql_java%g.log", which names the initial log file starsql_java0.log and places it in the home directory defined by the java.home system property. If the FileHandler.count is greater than 1, a number is appended to the filename, such as starsql_java1.log, starsql_java2.log, starsql_java3.log, if there are four files used in rotation.

Disabling the Logging Facility

After you have captured the logging information that you need, turn off the logging activity either by setting the global `.level` logging parameter to OFF, or by deleting the **sqlogging.properties** file from the `lib` directory. If you have more than one JVM installed, remember to delete the **sqlogging.properties** file or set the `.level` to OFF in each of the `lib` directories.

Sample Logging Configuration File

Following is the sample `sqlogging.properties` configuration file that is provided with StarSQL for Java. It configures the FileHandler to use five files in rotation, each with a maximum of approximately 150000 bytes, to log messages of level WARNING or above to the home directory. It also configures the ConsoleHandler to log messages of level SEVERE to the computer monitor.

```
############################################################

#Default Logging Configuration File

############################################################

#Global properties

############################################################

# Default global logging level.
# The global logging level can be overridden by the
# FileHandler.level property and the ConsoleHandler.level
# property, but only to a more restrictive level. Setting the
# global level property to OFF tuns off all logging (to a file
```

```
# or the console).
# Valid values are OFF, ALL, FINEST, FINER, FINE, CONFIG, INFO,
# WARNING, and SEVERE (most restrictive).

# .level = OFF

.level = WARNING

############################################################

#FileHandler configuration properties.

############################################################

# Write messages of level WARNING and above to 5 rotating files
# in the user's home directory.

FileHandler.pattern = %h/starsql_java%g.log

FileHandler.limit = 150000

FileHandler.count = 5

FileHandler.level = WARNING

############################################################

#ConsoleHandler configuration property.

############################################################

# Display messages of level SEVERE on the computer monitor.

ConsoleHandler.level = SEVERE
```

Optimizing Java on an IBM i Computer

This section describes how performance can be improved if you run Java applications on an IBM i computer. The StarSQL for Java driver and most of the sample applications are supplied as JAR files. You can improve the performance of Java applications by running the Create Java Program (CRTJVAPGM) control language command before you use the StarSQL for Java driver. The CRTJVAPGM command creates a Java program object that contains optimized native instructions for the IBM i server and associates the Java program object with the class file, JAR file, or ZIP file.

You can set the level of optimization that you want the CRTJVAPRGM to use. The Display Java Program (DSPJVAPGM) command shows the attributes of a Java program object, which shows the level of optimization that the Java program was created with. Running CRTJVAPGM with a high level of optimization may take a long time as the StarSQL_JDBC.jar file contains a large number of classes.

StarSQL for Java includes a shell script, `crtjva.sh`, that runs the CRTJVAPGRM command on the StarSQL_JDBC.jar and sample applications with an optimization level set to 20.

If you do not explicitly run the `crtjva.sh` script or the CRTJVAPGM command before you use StarSQL for Java, the system implicitly runs the CRTJVAPGM with no optimization (*INTERPRET). Controlling the optimization level at runtime using the **java -opt** option is not recommended because it can be a lengthy process.

For more information about improving the performance of Java on an IBM i computer, refer to the "Java performance considerations" section of the *IBM iSeries Developer Kit for Java* and *Java—Deploying your Java Application to the iSeries*.

Glossary

APAR - Authorized Program Analysis Report A request for correction of a problem caused by a defect in a current unaltered release of a program.

APPL An APPL statement defines the DB2 subsystem to VTAM for the purposes of remote access. It is required for any configuration that involves a DB2 host on an IBM mainframe (z/OS).

authorization identifier On DB2 for z/OS and DB2 for VM and VSE, the authorization identifier is the user's login ID or an assigned Authorization Identifier, which corresponds to a group with which the user is associated.

BLOB An abbreviation for binary large object, which is a collection of binary data that is stored as a single entity in a database management system. BLOBs are used primarily to hold multimedia objects, such as images, videos, and sound, though they also can be used to store programs or fragments of code. The BLOB data type is relatively new and may not be supported by older versions of database management systems.

BSDS The BSDS (bootstrap data set) is a VSAM data set that contains name and status information for DB2, as well as RBA range specifications, for all active and archive log data sets, passwords for the DB2 directory and catalog, and lists of conditional restart and checkpoint records.

CCSID CCSID (Coded Character Set Identifier) represents a character set, code page, encoding scheme, and additional coding-related information. Used to support international code sets.

classpath A list of directories or JAR files where classes are searched for. The standard **java** command takes a **-classpath** option. The term also is used to refer to the locations where classes are searched for in other kinds of applications, such as applets or web applications. Classes are typically packaged in JARs.

CLOB A sequence of characters with a size ranging from 0 bytes to 2 gigabytes less 1 byte. In general, character large object values are used when a character string might exceed the limits of the VARCHAR data type.

collection A collection is a location on the host for database objects, such as user tables and catalog tables, which are collected together under a single qualifying name.

CoS Class of Service (CoS) is the ability of switches and routers to prioritize traffic into different queues and classes.

data source A data source object enables JDBC applications to obtain a DBMS connection from a connection pool. Each data source object binds to the JNDI tree and points to a connection pool or MultiPool. Applications look up the data source on the JNDI tree and then request a connection from the data source.

DBMS A database management system, such as DB2, Oracle, and SQL Server, that controls the organization, storage, management, and retrieval of data in a database.

DDF DDF (Distributed Data Facility) is the vehicle that DB2 for z/OS uses to send and receive remote procedure calls.

DRDA DRDA (Distributed Relational database Architecture) supports access to distributed data by which an application can explicitly connect to another location, using an SQL statement, to execute packages that have been previously bound at that location.

Dynamic SQL Dynamic SQL refers to SQL statements that are prepared and executed within an application program while the program is executing. A dynamic SQL statement can change during program execution. Contrasted with Static SQL.

Enterprise Java Beans (EJB) is a Java API that defines a component architecture for multi-tier client/server systems.

held cursor A held cursor is a cursor that is not automatically closed when its transaction commits.

J2EE (Java 2 Platform, Enterprise Edition). The edition of the Java 2 platform that combines a number of technologies (such as enterprise beans, JSP pages, and XML) in one architecture for building enterprise server applications.

JAR (Java Archive file). A platform-independent file format that bundles classes, images, and other files into one compressed file.

Java Beans is a specification that defines how Java objects interact. An object that conforms to this specification is called a JavaBean, and is similar to an ActiveX control. It can be used by any application that understands the Java Beans format.

JDBC (Java Database Connectivity) is a call-level API that enables Java programs to interact with any SQL-compliant database. JDBC is similar to ODBC, but is designed specifically for Java programs, whereas ODBC is language-independent.

JDK (Java Development Kit) is a software development environment for writing applets and applications in the Java programming language.

JNDI (Java Naming and Directory Interface) is an interface that provides information about users, machines, networks, and services to applications.

JNI (Java Native Interface) is a Java programming interface, or API, that allows developers to access the languages of a host system and determine the way Java integrates with native code.

JRE (Java Runtime Environment) is a subset of the Java Development Kit (JDK) for end-users and developers who want to redistribute only the runtime environment. The JRE consists of the Java Virtual Machine (JRM), the Java core classes, and supporting files.

JVM (Java Virtual Machine) is the software that executes the byte codes in Java class files.

LOB A generic acronym for “large object,” which refers to the BLOB, CLOB, and DBCLOB data types.

ODBC (Open Database Connectivity) is a call-level interface developed by Microsoft Corporation that allows a single application to access DBMSs from different vendors using a single interface.

Open Edition Open Edition is a component of the z/OS operating system that supports TCP/IP processing in certain versions of z/OS.

package A package is an object on the host containing a set of SQL statements that have been bound statically and are available for processing.

PTF Program Temporary Fix - a method used by IBM for distributing software program fixes quickly.

RDB Name An RDB name is a unique identifier for an RDBMS within a network.

Secure Sockets Layer (SSL) A protocol that provides secure communications between client and server computers. Any amount of data can be sent securely over the SSL connection. Many Web sites use the SSL protocol to obtain confidential user information, such as credit card numbers, and by convention use a URL that starts with https: instead of http:.

static SQL Static SQL refers to SQL statements in an application program that are created before the application executes. After being created, a static SQL statement does not change, although values of host variables specified by the statement might change. Contrasted with Dynamic SQL.

system catalogs The system catalogs are tables in the host database that DB2 uses to keep track of the system. On some RDBMSs, they are called the system tables.

TCP/IP Transmission Control Protocol/Internet Protocol is a common communications protocol on which the Internet and most commercial networks are based.

user id A user id is a unique identifier that enables a user to log on to a computer.

VTAM VTAM (Virtual Telecommunications Access Method) provides network communications on IBM z/OS systems.

VSAM Virtual Storage Access Method - A data storage system used in z/OS. VSAM was designed to organize data more efficiently and to improve access time by searching indexes instead of actual files.

Symbols

.level property 103

A

accounting parameter 30, 32
accounts, user 40, 69
Apache Derby 15, 35, 39
APPL statement 109
application server, using StarSQL with 15
AS/400
    host, preparing a 74
    registering stored procedures on 76
authentication 82
Authorization identifier 109
autoCommitMode 30, 33

B

binding, packages 9, 34, 85

C

CA license code 25
CatalogApp sample application 46, 52
catalogFilter parameter 30, 32
CCSID 12, 109
    for national language support 93
    in typdefovr 40
    setting the 31
character encoding 93
character sets, international 12, 93
characters, multi-byte 30, 37
charge backs 32
classes for using data sources 42
classname for StarSQL driver 29
classpath 21, 109
client code page 94
CLOB with multi-byte 30, 37
code page, client 94
ColdFusion application server 15
collection 110
    package on host 74
    parameter 30, 32
command(s)
    CRTLIB 74

for setup script 20, 21
java classpath 22
java version 22
WRDSRVTBLE 75
WRKTCPSTS 76
commas in numeric values, using 30, 35
commitProcedureCall parameter 30, 33
configuration keywords 29
configuring
data sources 43
host system for SSL 76, 81
logging 102
the DDF 70
the StarSQL JDBC driver 29
Connect Info tab, sample application 51
connection
data source 42
enabling remote Derby 81
information, specifying 51
license 38
parameters 29
testing the StarSQL/DB2 44
URL, using the 41
ConsoleHandler.level 104
contacting StarQuest 17
conversion tables 12, 93
COS 110
create.table parameter 32
CreateDS sample application 47
createds shell script 51
createTable parameter 30, 33
crtjva.sh shell script 106
CRTJVAPGM command, running the 106
CRTLIB command 74
cursor, holding a 37

D

data source(s) 10, 110
description parameter 35
host information for 43
implementation classes 42
Initial Context for a 42
using 42

using CreateDS to create 49
data type
DISTINCT 10
LOB 10
data, converting 12, 93
database
name 34, 74
names, listing 80
supported host 13
databaseName parameter 30, 34
Datasource Info tab, sample application 52
date formats, using European 30, 34
dateFormat parameter 30, 34
DB2 host(s)
DB2/UDB, preparing a 78
DRDA port on 78
enabling DRDA on a 75
on MVS 70
preparing an AS/400 74
supported versions 13
DBMS versions 13
DDF 71, 110
decimalDelimiter parameter 30, 35
defaultQualifier parameter 30, 35
deprecated properties 31
Derby 35, 39
configuring SSL support for 81
decimalDelimiter parameter 30, 35
enabling remote connections 81
host, preparing a 80
Network Server host 80
setting decimalDelimiter for 82
setting sendUnicode for 31, 39, 82
derby.drda.host property 81
derby.drda.portNumber property 81
derby.drda.sslMode property 81
derby.system.home property 80
description parameter 30, 35
diagnosticsLevel parameter 30, 36
directive, table 33
DISTINCT data type 10
Distributed Data Facility (DDF), configuring the 70

DRDA 75, 110
diagnostics 36
operations, tracing 12, 36
port number 38
standard 93
DRDA host, testing connection to a 44
drdaTrace parameter 30, 36
driver
as a JDBC provider 42
interface 41
type of JDBC 7
DSN 44
DSNJU003 utility 70
DSNTIPR panel 70
dynamic SQL 32, 36, 110
dynamicRules property 89
packages 85
dynamic.rules parameter 32
dynamicRules parameter 30, 36

E

enabling DRDA 75
enabling encryption for DB2 UDB 79
encrypted user ID and password 79
encryption for DB2 UDB, enabling 79
encryption, password 11, 32, 39
Enterprise Java Beans 10, 110
environment variable
LANG 94
environment variables, setting 21
European date formats, using 30, 34
European numeric notation, using 30, 35

F

FileHandler.count 104
FileHandler.level 104
FileHandler.limit 103
FileHandler.pattern 103
fullyMaterializeCLOB parameter 30, 37

G

getPropertyInfo method 41
global logging level 103

global transactions 11

H

Held cursor 85, 110
held.cursors parameter 32
heldCursors parameter 30, 37
host.name parameter 32
host(s)
    configuring SSL support on a 76, 81
    connecting from the sample applications to a 51
    DB2 on an OS/390 70
    DB2/400 74, 75
    Derby 80
    information for data sources 43
    MVS 70
    passwords, changing 72
    permissions to 70
    preparation of 69
    systems supported 13
    UDB 78
    user accounts 38, 69
hostname keyword 27

I

ID, StarQuest product 31, 38
IDs, user 69
implementation classes, data source 42
in-doubt transactions 67
Initial Context, establishing the 42
installing stored procedures 76
international language support 12, 93
IP port, setting the 78

J

J2EE 10, 110
JAR file 22, 110
Java
    Beans 10, 110
    environment, required 12
    optimizing on an OS/400 106
    version command 22
Java DB 35, 39
JBoss application server 15

JDBC 7, 111
driver interface, using 41
provider information 42
JDK 111
JNDI 43, 111
JNI 111
JRE 12, 111
JV license code 25
JVM 12, 111

K

keepDynamic parameter 31, 38
key pair 82
key store 82
key, obtaining a license 17
keytool 82

L

LANG environment variable 94
language support, international 12, 93
library, package 74
license codes 25
license key
obtaining 17
licensing 38
limiting result sets 32
listing database names 80
loading the driver 29
via a data source 42
via connection URL 41
via getPropertyInfo method 41
LOB data types 10
LobTestApp sample application 54
local code page 94
local license, using a 28
locale 31, 39, 93, 94
location name 40
logging messages 12, 101

M

Mac OS X, installing StarSQL on 20
mainframe resources, charging for 32
management, password 72, 76

messages, logging 12
multi-byte characters 30, 37
MVS host, preparing DB2 on an 70

N

national languages, support for 93
Native Protocol driver 8
network topologies, example 13
new.password parameter 32
newPassword parameter 31, 38
numeric values, notation for 30, 35

O

obsolete properties 31
ODBC 7, 111
ODBC compliance 93
OpenEdition 111
optimizing Java 106
optimizing prepared statements 38
OS/390, installing StarSQL on 20
OS/400 computer, installing StarSQL on 20
OS/400 computer, optimizing Java on an 106

P

package.collectionID parameter 32
package(s) 111
    binding 9, 34, 85
    libraries for 74
    permissions for 70
packages 85
page, code 94
parameters, property 29
PASE, using StarSQL with 23, 47
password
    encryption 11, 32, 39
    management 72
    parameter 31, 38
    setting a new 38
    user 69
password encryption 79
password management 76
pattern, filename 104
permissions for packages 70

port
for DRDA requests 38, 75
for SSL 38
for TCP/IP connections 78
keyword for license 27
portNumber parameter 31, 38
prepared statements, optimizing 38
preparing
a DB2/400 host 74
a DB2/UDB host 78
a Derby host 80
DB2 on an MVS host 70
hosts for StarSQL 69
primary keyword 27
product license codes 25
productID parameter 31, 38
programs, sample
CatalogApp 46
CreateDS 47
LobTestApp 54
QueryApp 46
ShowVersion 44
simpconn 44
properties
Derby Network Server 80
StarSQL 29
protocol, supporting the SSL 81
pwdEncryption parameter 31, 39

Q

QSHELL, using StarSQL with 47
qualifying SQL statements 35
QueryApp sample application 46, 59

R

RDB name 40, 74, 111
rdb.name parameter 32
registering store procedures 76
requirements, system 12
result sets, limiting 32
runapp.command shell script 47
runapp.sh shell script 47

S

sample
    applications 46
    logging configuration file 105
samples.command shell script 47
samples.sh shell script 47
Secure Sockets Layer
    see SSL protocol
sendUnicode 31, 39
server
    name, database 40
server key pair 82
serverName parameter 31, 40
service providers, JNDI 43
shell scripts
    createds.command 51
    createds.sh 51
    crtjva.sh 106
    runapp.command 47
    runapp.sh 47
    samples.command 47
    samples.sh 47
    simpconn.command 45
    simpconn.sh 45
ShowVersion program 44
simpconn program 44
SQ license code 25
SQL statement table directive 33
ssl parameter 31, 40
SSL protocol 11, 81
    configuring on AS/400 76
    configuring on Derby 81
    configuring StarSQL for Java to use 40
    default port for 38
StarQuest
    contacting 17
    product IDs 31, 38
StarQuest, contacting 17
StarSQL
    overview 7
StarSQL for Java
    activity, logging 101

and application servers 15
classname 29
configuring 29
license 38
national language support 93
preparing hosts for 69
provider information 42
system requirements 12
testing 44
version information 44
static SQL 112
stored procedures
password management 76
registering 76
using StarSQL with 72
support, contacting technical 17
system requirements 12

T

tables, creating 33
TCP/IP 12
enabling DRDA over 75
setting the IP port 78
UDB host address for 78
technical support, StarQuest 17
testing StarSQL 44
tracing DRDA and API operations 12, 101
Transaction Log Manager application 66
transactions
in-doubt 67
two-phase commit 11, 66
typdefovr parameter 31, 40

U

UDB host 78
Unicode
and character sets 93
and Derby 31, 39
and Java applications 93
data support 39
UNIX, installing StarSQL on 19
URL, syntax for connection 41
user

accounts 69
parameter 31, 40
user id 112
user-defined data types 10

V

variables, setting environment 21
version
of Java 22
of StarSQL for Java 44
VTAM 112

W

WebSphere application server 15
Windows, installing StarSQL on 21
workstation licensing 28
WRDSRVTBLE command 75
WRKTCPSTS command 76

X

X Windows server, using StarSQL with an 23

Z

z/OS, installing StarSQL on 20