



BEA WebLogic Server™ and WebLogic Express™

**WebLogic jDriver for SQL
Server (Deprecated)**

Copyright

Copyright © 2003 BEA Systems, Inc. All Rights Reserved.

Restricted Rights Legend

This software and documentation is subject to and made available only pursuant to the terms of the BEA Systems License Agreement and may be used or copied only in accordance with the terms of that agreement. It is against the law to copy the software except as specifically allowed in the agreement. This document may not, in whole or in part, be copied, photocopied, reproduced, translated, or reduced to any electronic medium or machine readable form without prior consent, in writing, from BEA Systems, Inc.

Use, duplication or disclosure by the U.S. Government is subject to restrictions set forth in the BEA Systems License Agreement and in subparagraph (c)(1) of the Commercial Computer Software-Restricted Rights Clause at FAR 52.227-19; subparagraph (c)(1)(ii) of the Rights in Technical Data and Computer Software clause at DFARS 252.227-7013, subparagraph (d) of the Commercial Computer Software--Licensing clause at NASA FAR supplement 16-52.227-86; or their equivalent.

Information in this document is subject to change without notice and does not represent a commitment on the part of BEA Systems. THE SOFTWARE AND DOCUMENTATION ARE PROVIDED "AS IS" WITHOUT WARRANTY OF ANY KIND INCLUDING WITHOUT LIMITATION, ANY WARRANTY OF MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. FURTHER, BEA Systems DOES NOT WARRANT, GUARANTEE, OR MAKE ANY REPRESENTATIONS REGARDING THE USE, OR THE RESULTS OF THE USE, OF THE SOFTWARE OR WRITTEN MATERIAL IN TERMS OF CORRECTNESS, ACCURACY, RELIABILITY, OR OTHERWISE.

Trademarks or Service Marks

BEA, Jolt, Tuxedo, and WebLogic are registered trademarks of BEA Systems, Inc. BEA Builder, BEA Campaign Manager for WebLogic, BEA eLink, BEA Liquid Data for WebLogic, BEA Manager, BEA WebLogic Commerce Server, BEA WebLogic Enterprise, BEA WebLogic Enterprise Platform, BEA WebLogic Express, BEA WebLogic Integration, BEA WebLogic Personalization Server, BEA WebLogic Platform, BEA WebLogic Portal, BEA WebLogic Server, BEA WebLogic Workshop and How Business Becomes E-Business are trademarks of BEA Systems, Inc.

All other trademarks are the property of their respective companies.

Contents

About This Document

Audience	vii
e-docs Web Site	vii
How to Print the Document	vii
Related Information	viii
Contact Us!	viii
Documentation Conventions	ix

1. Installing WebLogic jDriver for Microsoft SQL Server

Overview	1-2
Before You Begin	1-2
Which Version Should I Use?.....	1-2
WebLogic jDriver for Microsoft SQL Server Version 7.0 and 2000.....	1-2
WebLogic jDriver for Microsoft SQL Server Versions 6.5 and 7.0	1-2
Checking Licensing Functionality	1-3
Installation Procedure	1-3
WebLogic jDriver for SQL Server Connection Properties	1-4
Using Connection Pools.....	1-4
Configuring a Connection Pool with WebLogic Server	1-5
Using the Connection Pool in Your Application	1-5
Client-Side Applications	1-5
Server-Side Applications.....	1-5

Verifying Your SQL Server Installation	1-6
Setting a Port for SQL Server Connections	1-6
Verifying the JDBC Driver With dbping	1-7
For More Information	1-8
Documentation	1-8
Code Examples	1-8
Using the SQL Server 2000 Driver for JDBC from Microsoft	1-8
.	1-8

2. Using WebLogic jDriver for Microsoft SQL Server

What Is the WebLogic jDriver for Microsoft SQL Server?	2-1
Connecting to an SQL Server DBMS	2-2
Using A Language Other Than English for a Connection	2-2
Connecting to a Database Using WebLogic Server in a Two-Tier Configuration	2-2
Connection Example	2-3
Adding Connection Options	2-4
Connecting Using WebLogic Server in a Multi-Tier Configuration	2-5
Manipulating Data with JDBC	2-5
Making Simple SQL Queries	2-6
Inserting, Updating, and Deleting Records	2-7
Creating and Using Stored Procedures and Functions	2-7
Disconnecting and Closing Objects	2-10
Codeset Support	2-11
JDBC Extensions	2-11
Support for JDBC Extended SQL	2-12
Querying Metadata	2-13
Sharing a Connection Object in Multithreaded Applications	2-13
Execute Keyword with Stored Procedures	2-13

JDBC Limitations	2-14
cursorName() Method Not Supported.....	2-14
java.sql.TimeStamp Limitations.....	2-14
Changing autoCommit Mode.....	2-14
Statement.executeWriteText() Methods Not Supported	2-14
References	2-15
Related Documentation	2-15

About This Document

This document describes how to configure and use WebLogic jDriver for Microsoft SQL Server, BEA's type-4 Java Database Connectivity (JDBC) driver for the Microsoft SQL Server Database management system.

This document is organized as follows:

- [Chapter 1, “Installing WebLogic jDriver for Microsoft SQL Server.”](#)
- [Chapter 2, “Using WebLogic jDriver for Microsoft SQL Server.”](#)

Audience

This document is written for application developers who are interested in building applications requiring database access. It is assumed that readers are familiar with SQL, general database concepts, and Java programming.

e-docs Web Site

BEA product documentation is available on the BEA corporate Web site. From the BEA Home page, click on Product Documentation or go directly to the WebLogic Server Product Documentation page at <http://e-docs.bea.com/wls/docs81>.

How to Print the Document

You can print a copy of this document from a Web browser, one main topic at a time, by using the File→Print option on your Web browser.

A PDF version of this document is available on the WebLogic Server documentation Home page on the e-docs Web site (and also on the documentation CD). You can open the PDF in Adobe Acrobat Reader and print the entire document (or a portion of it) in book format. To access the PDFs, open the WebLogic Server documentation Home page, click Download Documentation, and select the document you want to print.

Adobe Acrobat Reader is available at no charge from the Adobe Web site at <http://www.adobe.com>.

Related Information

The most up-to-date version of this document, *Installing and Using WebLogic jDriver for Microsoft SQL Server*, is also available in HTML format on the BEA e-docs website. To access the online HTML version on the BEA Web site, use the following link:

Installing and Using WebLogic jDriver for Microsoft SQL Server at <http://e-docs.bea.com/wls/docs81/mssqlserver4/index.html>

You can view or print the PDF versions of these documents using Adobe Acrobat, version 3.0 or higher.

Contact Us!

Your feedback on BEA documentation is important to us. Send us e-mail at docsupport@bea.com if you have questions or comments. Your comments will be reviewed directly by the BEA professionals who create and update the documentation.

In your e-mail message, please indicate the software name and version you are using, as well as the title and document date of your documentation. If you have any questions about this version of BEA WebLogic Server, or if you have problems installing and running BEA WebLogic Server, contact BEA Customer Support through BEA WebSupport at <http://www.bea.com>. You can also contact Customer Support by using the contact information provided on the Customer Support Card, which is included in the product package.

When contacting Customer Support, be prepared to provide the following information:

- Your name, e-mail address, phone number, and fax number
- Your company name and company address
- Your machine type and authorization codes
- The name and version of the product you are using

- A description of the problem and the content of pertinent error messages

Documentation Conventions

The following documentation conventions are used throughout this document.

Convention	Usage
Ctrl+Tab	Keys you press simultaneously.
<i>italics</i>	Emphasis and book titles.
monospace text	Code samples, commands and their options, Java classes, data types, directories, and file names and their extensions. Monospace text also indicates text that you enter from the keyboard. <i>Examples:</i> <pre>import java.util.Enumeration; chmod u+w * config/examples/applications .java config.xml float</pre>
<i>monospace</i> <i>italic</i> text	Variables in code. <i>Example:</i> <pre>String CustomerName;</pre>
UPPERCASE TEXT	Device names, environment variables, and logical operators. <i>Examples:</i> <pre>LPT1 BEA_HOME OR</pre>
{ }	A set of choices in a syntax line.
[]	Optional items in a syntax line. <i>Example:</i> <pre>java utils.MulticastTest -n name -a address [-p portnumber] [-t timeout] [-s send]</pre>

About This Document

Convention	Usage
	Separates mutually exclusive choices in a syntax line. <i>Example:</i> <pre>java weblogic.deploy [list deploy undeploy update] password {application} {source}</pre>
...	Indicates one of the following in a command line: <ul style="list-style-type: none">• An argument can be repeated several times in the command line.• The statement omits additional optional arguments.• You can enter additional parameters, values, or other information
.	Indicates the omission of items from a code example or from a syntax line.

Installing WebLogic jDriver for Microsoft SQL Server

This document tells you how to configure WebLogic jDriver for Microsoft SQL Server, BEA's pure-Java Type 4 JDBC driver for Microsoft SQL Server, and provides information about the following topics:

- [Overview](#)
- [Before You Begin](#)
- [Installation Procedure](#)
- [WebLogic jDriver for SQL Server Connection Properties](#)
- [Using Connection Pools](#)
- [Verifying Your SQL Server Installation](#)
- [Setting a Port for SQL Server Connections](#)
- [Verifying the JDBC Driver With dbping](#)
- [For More Information](#)

Note: The WebLogic jDriver for Microsoft SQL Server is deprecated. BEA recommends that you use the BEA WebLogic Type 4 JDBC MS SQL Server driver. For more information, see *BEA WebLogic Type 4 JDBC Drivers*.

Overview

WebLogic jDriver for Microsoft SQL Server is a pure Java implementation of the Java Database Connectivity (JDBC) API, the industry standard for relational database access from Java clients. It provides Java clients with direct access to Microsoft SQL Server. The driver is available in two versions: one for SQL Server versions 6.5 and 7.0, and another for SQL Server 7.0 only. Both versions function identically, except as noted in the following section, Before You Begin.

Like all Type 4 JDBC drivers, WebLogic jDriver for Microsoft SQL Server is pure Java; it requires no vendor-supported client libraries. WebLogic jDriver for Microsoft SQL Server communicates directly with SQL Server through a TCP/IP network, using the SQL Server *Tabular Data Stream* protocol, so DB-Library does not have to be installed on a client computer.

Before You Begin

This section describes the differences between two versions of WebLogic jDriver for Microsoft SQL Server.

Which Version Should I Use?

BEA offers two versions of the WebLogic jDriver for Microsoft SQL Server drivers. One version supports Microsoft SQL Server versions 7.0 and 2000 (the default), and the other version supports Microsoft SQL Server version 6.5 and 7.0.

WebLogic jDriver for Microsoft SQL Server Version 7.0 and 2000

- Supports SQL Server 7.0 and 2000. The SQL Server responds to requests made from this driver as it does to requests from an SQL Server version 7.0 and 2000 client and implements the semantics of an SQL Server 7.0 and 2000. For instance (in contrast to the semantics of SQL Server version 6.5), the DBMS, by default, creates columns that *allow* null values.
- Supports new data types introduced with SQL Server Version 7.0 and 2000.

This version of the WebLogic jDriver for Microsoft SQL Server is preconfigured with WebLogic Server. You do not need to add any entries to your `CLASSPATH` to use this driver.

WebLogic jDriver for Microsoft SQL Server Versions 6.5 and 7.0

- Supports SQL Server Versions 6.5.
- Supports SQL Server 7.0 with the following restrictions:

- The SQL Server responds as if the connection were coming from an SQL Server version 6.5 client, and implements the semantics of SQL Server version 6.5. For instance, when executing a `CREATE TABLE` SQL statement, the DBMS, by default, creates columns that *do not allow* null values. (This behavior is normal for SQL Server version 6.5.)
- New SQL Server 7.0 data types are not supported.

To use this version of the WebLogic jDriver for Microsoft SQL Server, you must add the path to `mssqlserver4v65.jar` in your `CLASSPATH`. For instructions, see [Installation Procedure on page 3](#).

Checking Licensing Functionality

To use WebLogic jDriver for Microsoft SQL Server, you must have the proper license. WebLogic jDriver for Microsoft SQL Server licensing functionality is included in the license file located in the BEA home directory where you installed this WebLogic Server. For example:

```
c:\bea\license.bea
```

If your license included WebLogic jDriver for Microsoft SQL Server when you installed or last updated your WebLogic Server license, no further action is required. If you are adding this functionality, you must get an updated license from your BEA sales representative. For instructions to update your license file, see “[Installing and Updating WebLogic Platform License Files](#)” in *Installing WebLogic Platform*.

Note: If you use WebLogic jDriver for Microsoft SQL Server when WebLogic Server is not running, you must include the path to the folder where `license.bea` resides in your `CLASSPATH`.

Installation Procedure

Microsoft SQL Server is bundled with your WebLogic Server distribution. For Version 7.0 and 2000, the `weblogic.jar` file includes the Microsoft SQL Server classes. No further steps are required. However, if you are using Version 6.5, you must pre-pend the `mssqlserver4v65.jar` file in the classpath as follows:

```
set CLASSPATH=%WL_HOME%\server\lib\mssqlserver4v65.JAR;  
%WL_HOME%\server\lib\weblogic.jar;%CLASSPATH%
```

WebLogic jDriver for SQL Server Connection Properties

Table 1-1 WebLogic jDriver for SQL Server Connection Properties

Property	Description
appname	This property is passed to the DBMS where it is written to the <code>sysprocesses</code> table in the <code>program_name</code> column.
db	The name of the database to which you want to connect.
hostname	This property is prepended with “WebLogic” and is passed to the DBMS where it is written to the <code>sysprocesses</code> table in the <code>hostname</code> column.
password	The password for the database user name.
port	The TCP port on which the database server listens for connections. The default is 1433.
server	The name or IP address of the database server.
user	The user name used to connect to your SQL Server database.
useVarChars	By default, the driver prepends an “N” to any varchar argument values sent to the DBMS so that the DBMS reads all 16 bits of data. Otherwise, the DBMS assumes American 7-bit characters and strips the high bit. When <code>useVarChars</code> is set to <code>true</code> , the driver does not prepend varchar arguments with an “N.” For customers who are not using multi-byte character sets, this option may improve performance. When set to <code>false</code> , an “N” is prepended to varchar arguments.

Using Connection Pools

If you are using WebLogic jDriver for Microsoft SQL Server with either WebLogic Server or WebLogic Express, you can set up a pool of connections to your SQL Server DBMS that will be established when WebLogic Server is started. Because the connections are shared among users,

these connection pools eliminate the overhead of opening a new database connection for each user.

Your application then looks up a `DataSource` on the JNDI tree and requests a connection from the connection pool. When finished with the database connection, your application returns it to the connection pool.

Configuring a Connection Pool with WebLogic Server

1. Set your environment and start WebLogic Server. For more information, see [Starting and Stopping Servers](#) in the *Administration Console Online Help* at <http://e-docs.bea.com/wls/docs81/ConsoleHelp/startstop.html>.
2. Use the Administration Console to set up connection pools. To read about connection pools, see [JDBC Components—Connection Pools, Data Sources, and MultiPools](#) in the *Administration Console Online Help* at http://e-docs.bea.com/wls/docs81/ConsoleHelp/jdbc.html#jdbc_components and [Configuring JDBC Connection Pools](#) in the *Administration Console Online Help* at http://e-docs.bea.com/wls/docs81/ConsoleHelp/jdbc_connection_pools.html.

Using the Connection Pool in Your Application

To use a connection pool, you must first establish a database connection. How you establish that connection depends on whether the application in which you want to use the connection pool as a client-side or a server-side application.

Client-Side Applications

To use a connection pool in a client-side application, establish the database connection by looking up a `DataSource` on the JNDI tree and then requesting a connection. For more information, see [Configuring and Using DataSources](#) in *Programming WebLogic JDBC*.

Server-Side Applications

To use a connection pool in a server-side application (such as a servlet), establish your database connection using a `DataSource` or by using the WebLogic `pool` or `jts` drivers. For more information, see “[Programming Tasks](#)” in *Programming WebLogic HTTP Servlets*

Verifying Your SQL Server Installation

Note: Verify that you are using version 6.5 or 7.0 of Microsoft SQL Server. Older versions of SQL Server do not properly support JDBC metadata functions and have limited data type support.

To connect to SQL Server, you need the following information:

- Username and password of a valid SQL Server account
- Host name or IP number of the machine on which the SQL Server is running
- Address of the TCP/IP port at which the DBMS is listening for connection requests

For Microsoft SQL Server, the default port number is 1433. Servers, however, can be configured to listen on any port number. Verify the port number from your configuration files. If you need help setting the port number, see the following section, “[Setting a Port for SQL Server Connections.](#)”

Setting a Port for SQL Server Connections

You set the host name and port for SQL Server connections by creating an entry in the SQL Server configuration files. In the configuration files, a logical *server name* is associated with a server machine name and port number. WebLogic jDriver for Microsoft SQL Server does not use a logical server name; it uses only the host name and the port number.

You must have administrator privileges to change your SQL Server settings. To set the port:

1. Run MS SQL Server Setup.
2. Select Change Network Support.
3. Select TCP/IP.
4. Select the port you want to use, such as 1433.

Once you have set the port, you can verify, by using `telnet`, that the server is listening on that port. Enter the following command:

```
$ telnet hostname_or_IP_address port
```

For example, to check whether the SQL Server is listening on port 1433 of a computer named `myhost`, type:

```
$ telnet myhost 1433
```

If the server is *not* listening on the port, `telnet` displays an error message. If the server *is* listening on the port, `telnet` displays nothing; eventually, the host drops the connection.

You can test your login information by entering the following command:

```
$ isql -Username -Ppassword -Sserver
```

Verifying the JDBC Driver With dbping

You can use `dbping`, a WebLogic Java application, to verify that WebLogic jDriver for Microsoft SQL Server can connect to your SQL Server. Use the following commands to set your environment and to execute `dbping`:

```
WL_HOME\server\bin\setWLSEnv.cmd
java utils.dbping MSSQLSERVER4 username password
  [database@]host[:port]
```

The arguments in this command line are defined as follows:

- `WL_HOME` is the directory where WebLogic Platform is installed, typically `c:\bea\weblogic700`.
- `username` is the name of the database user.
- `password` is the user's password.
- `database` (optional) is the SQL Server database to be used.
- `host` is the name or IP address of the computer on which SQL Server is running.
- `port` (optional) is the TCP/IP port on which the SQL Server is listening.

For example, the following command pings an SQL Server database called `pubs` on a computer named `myhost`, using the default TCP/IP port, the `sa` login, and a null password:

```
$ java utils.dbping MSSQLSERVER4 sa " " pubs@myhost
```

The output from the command includes code that you can use to connect to the database in a Java program.

For detailed instructions for using the `dbping` utility, see [Using the WebLogic Java Utilities](#) in the *WebLogic Server Command Reference* at

http://e-docs.bea.com/wls/docs81/admin_ref/utils.html.

For More Information

This section provides references to documents and code examples that maybe helpful to you.

Documentation

For more information about using JDBC and jDrivers with WebLogic Server, see [Programming WebLogic JDBC](http://e-docs.bea.com/wls/docs81/jdbc/index.html) at <http://e-docs.bea.com/wls/docs81/jdbc/index.html>.

Code Examples

WebLogic Server provides several code examples to help you get started. Code examples are located in the `SAMPLES_HOME\server\src\examples\jdbc\mssqlserver4` directory, where `SAMPLES_HOME` is the top-level directory for all samples and examples for the WebLogic Platform (`c:\bea\weblogic700\samples`, by default).

Using the SQL Server 2000 Driver for JDBC from Microsoft

The Microsoft SQL Server 2000 Driver for JDBC, available from Microsoft's MSDN Web site, is a Type 4 JDBC driver that supports a subset of the JDBC 2.0 Optional Package (see the driver documentation for details). This driver provides JDBC access to SQL Server 2000 through any Java-enabled applet, application, or application server.

The Microsoft SQL Server 2000 Driver for JDBC is available for download to all licensed SQL Server 2000 customers at no charge.

For information about downloading, configuring, and using the SQL Server 2000 Driver for JDBC from Microsoft, see [Installing and Using the SQL Server 2000 Driver for JDBC from Microsoft](#) in *Programming WebLogic JDBC* at

<http://e-docs.bea.com/wls/docs81/jdbc/thirdparty.html#sqlserver>.

Using WebLogic jDriver for Microsoft SQL Server

Note: The WebLogic jDriver for Microsoft SQL Server is deprecated. BEA recommends that you use the BEA WebLogic Type 4 JDBC MS SQL Server driver. For more information, see *BEA WebLogic Type 4 JDBC Drivers*.

This chapter explains how to set up and use WebLogic jDriver for Microsoft SQL Server, as described in the following sections:

- [What Is the WebLogic jDriver for Microsoft SQL Server?](#)
- [Connecting to an SQL Server DBMS](#)
- [Manipulating Data with JDBC](#)
- [Codeset Support](#)
- [JDBC Extensions](#)
- [JDBC Limitations](#)
- [References](#)

What Is the WebLogic jDriver for Microsoft SQL Server?

WebLogic jDriver for Microsoft SQL Server is a Type 4, pure-Java, two-tier JDBC driver that you can use to create the physical database connections in a connection pool. It requires no client-side libraries because it connects to the database via a proprietary vendor protocol at the wire-format level.

A Type 4 JDBC driver is similar to a Type 2 driver in many other ways. Type 2 and Type 4 drivers are two-tier drivers — each client requires an in-memory copy of the driver to support its connection to the database. For more information on the types of JDBC drivers, see [Introduction to WebLogic JDBC](#) in *Programming WebLogic JDBC* at <http://e-docs.bea.com/wls/docs81/jdbc/intro.html>.

Within the WebLogic Server environment, you can use either a Type 2 or a Type 4 two-tier driver to connect the WebLogic Server to a database, and then one of WebLogic's multitier drivers, the RMI, JTS, or Pool driver. These are pure-Java Type 3 multitier JDBC drivers, for client connections to the WebLogic Server.

The API reference for JDBC, for which this driver is compliant, is available online at [JavaSoft](#) at <http://java.sun.com>.

Connecting to an SQL Server DBMS

The following sections describe how to use WebLogic jDriver for Microsoft SQL Server to connect WebLogic Server to a database.

Using A Language Other Than English for a Connection

For Microsoft SQL Server databases that use a language other than English, you must specify the `language` property to reflect the language you use. For example, to specify the French language, add the following property when making a connection:

```
props.put("language","francais")
```

Failure to specify this property may result in exceptions, such as a "Primary Key Constraint Violation."

Connecting to a Database Using WebLogic Server in a Two-Tier Configuration

Complete the following procedure to set up your application to get a database connection through WebLogic Server using WebLogic jDriver for Microsoft SQL Server.

In steps 1 and 3, you describe the JDBC driver. In the first step, you use the full package name of the driver, which is dot-delimited. In the third step, you identify the driver with its URL, which is colon-delimited. The URL must include the following string:

`weblogic:jdbc:mssqlserver4`. It may also include other information, such as the server host name and the database name.

1. Load and register the JDBC driver by doing the following:
 - a. Call `Class.forName().newInstance()` with the **full class name of the WebLogic jDriver for Microsoft SQL Server JDBC driver class**.
 - b. Cast it to a `java.sql.Driver` object.

For example:

```
Driver myDriver = (java.sql.Driver)Class.forName
    ("weblogic.jdbc.mssqlserver4.Driver").newInstance();
```

2. Create a `java.util.Properties` object describing the connection. This object contains name-value pairs containing information such as username, password, database name, server name, and port number. For example:

```
Properties props = new Properties();
props.put("user", "scott");
props.put("password", "secret");
props.put("db", "myDB");
props.put("server", "myHost");
props.put("port", "8659");
```

3. Create a JDBC Connection object, which becomes an integral piece in your JDBC operations, by calling the `Driver.connect()` method. This method takes, as its parameters, the **URL of the driver** and the `java.util.Properties` object you created in step 2. For example:

```
Connection conn =
    myDriver.connect("jdbc:weblogic:mssqlserver4", props);
```

Connection Example

The following sample code shows how to use a `Properties` object to connect to a database named `myDB` on a server named `myHost`:

```
Properties props = new Properties();
props.put("user", "scott");
props.put("password", "secret");
props.put("db", "myDB");
props.put("server", "myHost");
props.put("port", "8659");
```

```
Driver myDriver = (Driver)
    Class.forName("weblogic.jdbc.mssqlserver4.Driver").newInstance();
```

Using WebLogic jDriver for Microsoft SQL Server

```
Connection conn =
    myDriver.connect("jdbc:weblogic:mssqlserver4", props);
```

You can combine the `db`, `server`, and `port` properties into one `server` property, as shown in the following example:

```
Properties props = new Properties();
props.put("user",      "scott");
props.put("password",  "secret");
props.put("server",    "myDB@myHost:8659");
// props.put("appname", "MyApplication");
// props.put("hostname", "MyHostName");
```

The last two properties, `appname` and `hostname`, are optional and are passed to the Microsoft SQL server, where they can be read in the `sysprocesses` table under the column names `program_name` and `hostname`. The `hostname` value is prepended with `WebLogic`.

```
Driver myDriver = (java.sql.Driver)
    Class.forName
        ("weblogic.jdbc.mssqlserver4.Driver").newInstance();
Connection conn =
    myDriver.connect("jdbc:weblogic:mssqlserver4", props);
```

Various methods can be used to supply information in the URL or in the Properties object. The information you pass in the URL of the driver does not need to be included in the Properties object.

Adding Connection Options

You can also add connection options to the end of the connection URL. Separate the URL from the connection options with a question mark, and separate options with ampersands, as shown in the following example:

```
String myUrl =
    "jdbc:weblogic:mssqlserver4:db@myhost:myport?
        user=sa&password=";
```

To find out more about URL options at run time, use `Driver.getPropertyInfo()`.

Connecting Using WebLogic Server in a Multi-Tier Configuration

To make a connection from your application to an SQL Server DBMS in a WebLogic Server multi-tier configuration, complete the following procedure:

1. To access the WebLogic RMI driver using JNDI, obtain a Context from the JNDI tree by looking up the JNDI name of your DataSource object. For example, to access a DataSource with JNDI name "myDataSource" that is defined in Administration Console:

```
try {
    Context ctx = new InitialContext();
    javax.sql.DataSource ds
        = (javax.sql.DataSource) ctx.lookup ("myDataSource");
} catch (NamingException ex) {

    // lookup failed
}
```

2. To obtain the JDBC connection from the DataSource object:

```
try {
    java.sql.Connection conn = ds.getConnection();
} catch (SQLException ex) {
    // obtain connection failed
}
```

For more information, see [Configuring and Using DataSources](#) in *Programming WebLogic JDBC*.

Manipulating Data with JDBC

This section is a brief introduction to data manipulation with JDBC, and provides basic procedures for implementing the following basic tasks in an application:

- [Making Simple SQL Queries](#)
- [Inserting, Updating, and Deleting Records](#)
- [Creating and Using Stored Procedures and Functions](#)
- [Disconnecting and Closing Objects](#)

For more information, see your Microsoft SQL Server documentation and Java-oriented texts about JDBC.

Note: The WebLogic jDriver for Microsoft SQL Server cannot handle question marks (“?”) in table names or column names. To avoid errors, do not use question marks in the table names and column names in your database.

Making Simple SQL Queries

The most fundamental task in database access is to retrieve data. With WebLogic jDriver for Microsoft SQL Server, you can retrieve data by completing the following three-step procedure:

1. Create a Statement to send an SQL query to the DBMS.
2. Execute the Statement.
3. Retrieve the results into a ResultSet. In this example, we execute a simple query on the Employee table (alias `emp`) and display data from three of the columns. We also access and display metadata about the table from which the data was retrieved. Note that we close the Statement at the end.

```
Statement stmt = conn.createStatement();
stmt.execute("select * from emp");
ResultSet rs = stmt.getResultSet();

while (rs.next()) {
    System.out.println(rs.getString("empid") + " - " +
        rs.getString("name") + " - " +
        rs.getString("dept"));
}

ResultSetMetaData md = rs.getMetaData();

System.out.println("Number of columns: " +
    md.getColumnCount());
stmt.close();
```

Inserting, Updating, and Deleting Records

In this section we show how to perform three common database tasks: inserting, updating, and deleting records from a database table. We use a JDBC `PreparedStatement` for these operations: first we create the `PreparedStatement`; then we execute and close it.

A `PreparedStatement` (subclassed from JDBC `Statement`) allows you to execute the same SQL repeatedly with different values.

In the following sample code, we create a `PreparedStatement`. Note the use of the `?` syntax:

```
String inssql =
    "insert into emp(empid, name, dept) values (?, ?, ?)";
PreparedStatement pstmt = conn.prepareStatement(inssql);
pstmt.setString(1, "12345");
pstmt.setString(2, "gumby");
pstmt.setString(3, "Cartoons");
```

Next, we use a `PreparedStatement` to update records. In the following example, we add the value of the counter `i` to the current value of the `dept` field:

```
String updsql =
    "update emp set dept = dept + ? where empid = ?";
PreparedStatement pstmt2 = conn.prepareStatement(updsql);

pstmt2.setString(1, "Cartoons");
pstmt2.setString(2, "12345");
```

Finally, we use a `PreparedStatement` to delete the records that we added and then updated:

```
String delsql = "delete from emp where empid = ?";
PreparedStatement pstmt3 = conn.prepareStatement(delsql);
pstmt3.setString(1, "12345");
```

Creating and Using Stored Procedures and Functions

You can use WebLogic `JDriver` for Microsoft SQL Server to create, use, and drop stored procedures and functions.

In the following sample code, we execute a series of Statements to drop a set of stored procedures and functions from the database:

```
Statement stmt = conn.createStatement();
try {stmt.execute("drop procedure proc_squareInt");}
catch (SQLException e) {}
try {stmt.execute("drop procedure func_squareInt");}
catch (SQLException e) {}
try {stmt.execute("drop procedure proc_getresults");}
catch (SQLException e) {}
stmt.close();
```

We use a JDBC Statement to create a stored procedure or function, and then we use a JDBC CallableStatement (subclassed from Statement) with the JDBC ? syntax to set IN and OUT parameters.

Stored procedure input parameters are mapped to JDBC IN parameters, using the CallableStatement.setXXX() methods, such as setInt(), and the JDBC PreparedStatement ? syntax. Stored procedure output parameters are mapped to JDBC OUT parameters, using the CallableStatement.registerOutParameter() methods and JDBC PreparedStatement ? syntax. A parameter may be set to both IN and OUT. If it is, calls to both setXXX() and registerOutParameter() on the same parameter number must be made.

In the following example, we use a JDBC Statement to create a stored procedure and then execute the stored procedure with a CallableStatement. We use the registerOutParameter() method to set an output parameter for the squared value.

```
Statement stmt1 = conn.createStatement();
stmt1.execute
("CREATE OR REPLACE PROCEDURE proc_squareInt " +
"(field1 IN OUT INTEGER, field2 OUT INTEGER) IS " +
"BEGIN field2 := field1 * field1; field1 := " +
"field1 * field1; END proc_squareInt;");
stmt1.close();

String sql = "{call proc_squareInt(?, ?)}";
CallableStatement cstmt1 = conn.prepareCall(sql);

// Register out parameters
```

```
cstmt1.registerOutParameter(2, java.sql.Types.INTEGER);
```

Next, we use similar code to create and execute a stored function that squares an integer:

```
Statement stmt2 = conn.createStatement();
stmt2.execute("CREATE OR REPLACE FUNCTION func_squareInt " +
             "(field1 IN INTEGER) RETURN INTEGER IS " +
             "BEGIN return field1 * field1; " +
             "END func_squareInt;");
stmt2.close();

sql = "{ ? = call func_squareInt(?)}";
CallableStatement cstmt2 = conn.prepareCall(sql);

cstmt2.registerOutParameter(1, Types.INTEGER);
```

In the following example we use a stored procedure named `sp_getmessages`. (The code for this stored procedure is not included with this example.) `sp_getmessages` takes a message number as an input parameter and returns the message text, as an output parameter, in a `ResultSet`. You must process all `ResultSets` returned by a stored procedure using the `Statement.execute()` and `Statement.getResult()` methods before `OUT` parameters and return status are available.

```
String sql = "{ ? = call sp_getmessage(?, ?)}";
CallableStatement stmt = conn.prepareCall(sql);

stmt.registerOutParameter(1, java.sql.Types.INTEGER);
stmt.setInt(2, 18000); // msgno 18000
stmt.registerOutParameter(3, java.sql.Types.VARCHAR);
```

First, we set up the three parameters to the `CallableStatement`:

1. Parameter 1 (output only) is the stored procedure return value.
2. Parameter 2 (input only) is the `msgno` argument to `sp_getmessage`.
3. Parameter 3 (output only) is the message text return for the message number.

Next, we execute the stored procedure and check the return value to determine whether the `ResultSet` is empty. If it is not, we use a loop to retrieve and display its contents.

```
boolean hasResultSet = stmt.execute();
while (true)
```

```
{
    ResultSet rs = stmt.getResultSet();
    int updateCount = stmt.getUpdateCount();
    if (rs == null && updateCount == -1) // no more results
        break;
    if (rs != null) {
        // Process the ResultSet until it is empty
        while (rs.next()) {
            System.out.println
                ("Get first col by id:" + rs.getString(1));
        }
    } else {
        // we have an update count
        System.out.println("Update count = " +
            stmt.getUpdateCount());
    }
    stmt.getMoreResults();
}
```

After we finish processing the `ResultSet`, the `OUT` parameters and return status are available, as shown in the following example:

```
int retstat = stmt.getInt(1);
String msg = stmt.getString(3);

System.out.println("sp_getmessage: status = " +
    retstat + " msg = " + msg);
stmt.close();
```

Disconnecting and Closing Objects

You may want to call the `commit()` method to commit changes you have made to the database before closing a connection.

When `autocommit` is set to `true` (the default JDBC transaction mode) each SQL statement is its own transaction. After we created the `Connection` for these examples, however, we set `autocommit` to `false`. In this mode, the `Connection` always has an implicit transaction associated with it; any call to the `rollback()` or `commit()` method ends the current transaction and start a new one. Calling `commit()` before `close()` ensures that all transactions are completed before the `Connection` is closed.

Just as you close `Statements`, `PreparedStatements`, and `CallableStatements` when you have finished working with them, you should always call the `close()` method on the connection as final cleanup in your application, in a `finally {}` block. You should catch exceptions and deal with them appropriately. The final two lines of this example contain calls to `commit` and `close` the connection:

```
conn.commit();
conn.close();
```

Codeset Support

As a Java application, WebLogic jDriver for Microsoft SQL Server handles character strings as Unicode strings. To exchange character strings with a database that may operate with a different codeset, the driver attempts to detect the codeset of the database and convert Unicode strings using a character set supported by the JDK. If there is no direct mapping between the codeset of your database and the character sets provided with the JDK, you can set the `weblogic.codeset` connection property to the most appropriate Java character set. For example, to use the cp932 codeset, create a Properties object and set the `weblogic.codeset` property before calling `Driver.connect()`, as shown in the following sample code:

```
java.util.Properties props = new java.util.Properties();
props.put("weblogic.codeset", "cp932");
props.put("user", "sa");
props.put("password", "");

String connectUrl = "jdbc:weblogic:mssqlserver4:myhost:1433";

Driver myDriver = (Driver)Class.forName
    ("weblogic.jdbc.mssqlserver4.Driver").newInstance();

Connection conn = myDriver.connect(connectUrl, props);
```

JDBC Extensions

This section describes the following extensions to JDBC:

- [Support for JDBC Extended SQL](#)
- [Querying Metadata](#)
- [Sharing a Connection Object in Multithreaded Applications](#)
- [Execute Keyword with Stored Procedures](#)

Support for JDBC Extended SQL

The JDBC specification includes *SQL Extensions*, also called *SQL Escape Syntax*. WebLogic jDriver for Microsoft SQL Server supports Extended SQL. Extended SQL provides access to common SQL extensions in a way that is portable between DBMSs.

For example, the function to extract the day name from a date is not defined by the SQL standards. For Oracle, the SQL is:

```
select to_char(date_column, 'DAY') from table_with_dates
```

Using Extended SQL, you can retrieve the day name for both Oracle and SQL Server DBMSs as follows:

```
select {fn dayname(date_column)} from table_with_dates
```

The following example code demonstrates several features of Extended SQL:

```
String query =
"-- This SQL includes comments and " +
    "JDBC extended SQL syntax.\n" +
"select into date_table values( \n" +
"    {fn now()},          -- current time \n" +
"    {d '1997-05-24'},    -- a date      \n" +
"    {t '10:30:29' },    -- a time      \n" +
"    {ts '1997-05-24 10:30:29.123'}, -- a timestamp\n" +
"    '{string data with { or } will not be altered}'\n" +
"-- Also note that you can safely include" +
"    " { and } in comments or\n" +
"-- string data.";
Statement stmt = conn.createStatement();
stmt.executeUpdate(query);
```

Extended SQL is delimited with curly braces ({}) to differentiate it from common SQL.

Comments are preceded by two hyphens, and are ended by a newline (\n). The entire Extended SQL sequence, including comments, SQL, and Extended SQL, is placed within double quotes and passed to the `execute()` method of a `Statement` object.

The following code sample shows how Extended SQL can be used as part of a `CallableStatement`:

```
CallableStatement cstmt =
    conn.prepareCall("{ ? = call func_squareInt(?) }");
```

The following example shows how you can nest extended SQL expressions:

```
select {fn dayname({fn now()})}
```

You can retrieve lists of supported Extended SQL functions from a `DatabaseMetaData` object. The following example shows how to list the functions supported by a JDBC driver:

```
DatabaseMetaData md = conn.getMetaData();
System.out.println("Numeric functions:      " +
    md.getNumericFunctions());
System.out.println("\nString functions:      " +
    md.getStringFunctions());
System.out.println("\nTime/date functions: " +
    md.getTimeDateFunctions());
System.out.println("\nSystem functions:      " +
    md.getSystemFunctions());
conn.close();
```

For a description of Extended SQL, see Chapter 11 of the JDBC 1.2 specification.

Querying Metadata

You can query metadata for the current database only. The metadata methods call the corresponding SQL Server stored procedures, which only operate on the current database. For example, if the current database is *master*, only the metadata relative to *master* is available on the connection.

Sharing a Connection Object in Multithreaded Applications

WebLogic `jDriver` for Microsoft SQL Server allows you to write multithreaded applications in which multiple threads share a single `Connection` object. Each thread can have an active `Statement` object. However, if you call `Statement.cancel()` on one thread, SQL Server may cancel a `Statement` on a different thread. Which `Statement` is cancelled depends on timing issues in the SQL Server. To avoid unexpected cancellations, we recommend that you get a separate `Connection` for each thread.

Execute Keyword with Stored Procedures

A Transact-SQL feature allows you to omit the `EXECUTE` keyword on a stored procedure when the stored procedure is the first command in the batch. However, when a stored procedure has parameters, WebLogic `jDriver` for Microsoft SQL Server adds variable declarations (specific to

the JDBC implementation) before the procedure call. Because of this, it is good practice to use the `EXECUTE` keyword for stored procedures. Note that the JDBC extended SQL stored procedure syntax, which does not include the `EXECUTE` keyword, is not affected by this issue.

JDBC Limitations

This section describes the following limitations to JDBC:

- [cursorName\(\) Method Not Supported](#)
- [java.sql.TimeStamp Limitations](#)
- [Changing autoCommit Mode](#)
- [Statement.executeWriteText\(\) Methods Not Supported](#)

cursorName() Method Not Supported

The `cursorName()` method is not supported.

java.sql.TimeStamp Limitations

The `java.sql.TimeStamp` class in the JDK is limited to dates after 1970. Earlier dates raise an exception. However, if you retrieve dates using `getString()`, WebLogic jDriver for Microsoft SQL Server uses its own date class to overcome the limitation.

Changing autoCommit Mode

Call `Connection.setAutoCommit()` with a `true` or `false` argument to enable or disable chained transaction mode. When `autoCommit` is `false`, the WebLogic jDriver for Microsoft SQL Server driver begins a transaction whenever the previous transaction is committed or rolled back. You must explicitly end your transactions with a `commit` or `rollback`. If there is an uncommitted transaction when you call `setAutoCommit()`, the driver rolls back the transaction before changing the mode, so be sure to commit any changes before you call this method.

Statement.executeWriteText() Methods Not Supported

The WebLogic Type 2 JDBC drivers support an extension that allows you to write text and image data into a row as part of an SQL `INSERT` or `UPDATE` statement without using a text pointer. This extension, `Statement.executeWriteText()` requires the DB-Library native libraries. Because the WebLogic jDriver for Microsoft SQL Server is a Type 4 JDBC driver, it does not

use the DB-Library native libraries to communicate with the database, and therefore, cannot support the `Statement.executeWriteText()` extension.

Use the following methods to read and write text and image data with streams:

- `prepareStatement.setAsciiStream()`
- `prepareStatement.setBinaryStream()`
- `ResultSet.getAsciiStream()`
- `ResultSet.getBinaryStream()`

References

This section provides references to documents and code examples that may help you learn about using BEA WebLogic jDriver for Microsoft SQL Server.

Related Documentation

- [Introduction to JDBC](http://e-docs.bea.com/wls/docs81/jdbc/intro.html) in *Programming WebLogic JDBC* at <http://e-docs.bea.com/wls/docs81/jdbc/intro.html>.
Contains information about other WebLogic JDBC drivers, additional documentation, support resources, and more.
- [Programming Tasks](#) in *Programming WebLogic HTTP Servlets* at `{DOCRoot}/servlet/progtasks.html`.
Contains information about using connection pools with server-side Java.
- [Configuring JDBC](http://e-docs.bea.com/wls/docs81/ConsoleHelp/jdbc.html) in the *Administration Console Online Help* at <http://e-docs.bea.com/wls/docs81/ConsoleHelp/jdbc.html>.
Describes administrative tasks for configuring JDBC connectivity, such as creating connection pools, datasources, and multipools.
- [Sun's JDBC tutorial](http://java.sun.com/docs/books/tutorial/jdbc/index.html)
at <http://java.sun.com/docs/books/tutorial/jdbc/index.html>.

Using WebLogic jDriver for Microsoft SQL Server