

BEA WebLogic

Installing and Using WebLogic jDriver for SQL Server 5.1

Document 1.0
April, 2000

Copyright

Copyright © 2000 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, WebLogic, ObjectBroker, TOP END, and TUXEDO are registered trademarks of BEA Systems, Inc. BEA Builder, BEA Connect, BEA Manager, BEA MessageQ, Jolt and M3 are trademarks of BEA Systems, Inc.

All other company names may be trademarks of the respective companies with which they are associated.

Installing and using jDriver for MS SQL Server

Document Edition	Part Number	Date	Software Version
1.0		April 2000	jDriver for MS SQL Server 5.1

Contents

About This Document

What You Need to Know 1
e-docs Web Site 1
How to Print the Document 2
Related Information 2
Contact Us! 3
Documentation Conventions 3

1. Installing WebLogic jDriver for Microsoft SQL Server

Overview 7
- Which version should I use? 8
  - WebLogic jDriver for Microsoft SQL Server version 6.5 and 7.0 8
  - WebLogic jDriver for Microsoft SQL Server version 7.0 8
- Before you begin 8
  - Upgrade information 8
  - Evaluation licenses 9

Installation steps 9
- Editing an entry to the XML license file 12
- Setting up a connection pool 13
  - Configuring a connection pool with WebLogic Server 13
  - Using the connection pool in your application 14
- Verifying your SQL Server installation 14
- Setting a port for SQL Server connections 15
- Verifying the JDBC driver with dbping 16

More resources 16
- Documentation 16
- Code examples 17

2. Using WebLogic jDriver for Microsoft SQL Server

Introduction 19
What's in this document 19
Implementing with WebLogic jDriver for Microsoft SQL Server 20
Connecting to a SQL Server DBMS 20
Connection example 21
Adding other connection options 22
Setting properties for WebLogic JDBC use 22
Overview of data manipulation 23
Making a simple SQL query 23
Inserting, updating, and deleting records 24
Creating and using stored procedures and functions 24
Disconnecting and closing objects 27
Codeset support 27
Using WebLogic jDriver for Microsoft SQL Server in applets 28
JDBC extensions and limitations 31
Support for JDBC Extended SQL 31
cursorName() method not supported 32
java.sql.TimeStamp limitations 33
Querying metadata 33
Changing autoCommit mode 33
Statement.executeWriteText() methods not supported 33
Sharing a Connection object in multi-threaded applications 34
Execute keyword with stored procedures 34
More Resources 34
Code examples 35

3. Release Notes

About This Document

This document describes how to install and develop applications using jDriver for Microsoft SQL Server, BEA's type-2 Java Database Connectivity (JDBC) driver for the Microsoft SQL Server Database management system.

This document covers the following topics:

- Chapter 1, "Installing WebLogic jDriver for Microsoft SQL Server."
- Chapter 2, "Using WebLogic jDriver for Microsoft SQL Server."
- Chapter 3, "Release Notes."

What You Need to Know

This document is intended mainly for application developers who are interested in building applications requiring database access. It assumes a familiarity 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 "e-docs" Product Documentation page at http://e-docs.beasys.com.

How to Print the Document

You can print a copy of this document from Adobe Acrobat by using the File—>Print option.

If you do not have the Adobe Acrobat Reader, you can download it for free 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 website, use the following links:

Installing jDriver for Microsoft SQL Server:
http://www.weblogic.com/docs51/techstart/install_jmsq4.html

Using jDriver for Microsoft SQL Server:
http://www.weblogic.com/docs51/classdocs/API_jmsq4.html

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

Complete documentation for WebLogic Server is available online at *http://www.weblogic.com/docs51/resources.html.*

Contact Us!

Your feedback on the BEA WebLogic Enterprise documentation is important to us. Send us e-mail at **docsupport@beasys.com** if you have questions or comments. Your comments will be reviewed directly by the BEA professionals who create and update the jDriver for Microsoft SQL Server documentation.

In your e-mail message, please indicate that you are using the documentation for the BEA jDriver for Microsoft SQL Server.

If you have any questions about this version of BEA jDriver for Microsoft SQL Server, or if you have problems installing and running BEA jDriver for Microsoft SQL Server, contact BEA Customer Support through BEA WebSupport at **www.beasys.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	Item
boldface text	Indicates terms defined in the glossary.
Ctrl+Tab	Indicates that you must press two or more keys simultaneously.

Convention	Item
italics	Indicates emphasis or book titles.

`monospace text`	Indicates code samples, commands and their options, data structures and their members, data types, directories, and file names and their extensions. Monospace text also indicates text that you must enter from the keyboard.
	Examples:
	`#include <iostream.h> void main ( ) the pointer psz`
	`chmod u+w *`
	`\tux\data\ap`
	`.doc`
	`tux.doc`
	`BITMAP`
	`float`

`monospace boldface text`	Identifies significant words in code. *Example*: `void` **`commit`** `( )`
`monospace italic text`	Identifies variables in code. *Example*: `String` *`expr`*
UPPERCASE TEXT	Indicates device names, environment variables, and logical operators. *Example*s: LPT1 SIGNON OR
`{ }`	Indicates a set of choices in a syntax line. The braces themselves should never be typed.

`[ ]`	Indicates optional items in a syntax line. The brackets themselves should never be typed.
	Example:
	`buildobjclient [-v] [-o name ] [-f` *`file-list`*`]...`
	`[-l` *`file-list`*`]...`

Convention	Item
|	Separates mutually exclusive choices in a syntax line. The symbol itself should never be typed.

. . .	Indicates one of the following in a command line:
	■ That an argument can be repeated several times in a command line
	■ That the statement omits additional optional arguments
	■ That you can enter additional parameters, values, or other information
	The ellipsis itself should never be typed.
	Example:
	`buildobjclient [-v] [-o name ] [-f file-list]...`
	`[-l file-list]...`

. . .	Indicates the omission of items from a code example or from a syntax line. The vertical ellipsis itself should never be typed.

1 Installing WebLogic jDriver for Microsoft SQL Server

Overview

WebLogic jDriver for Microsoft SQL Server is a new name for the product previously referred to as jdbcKona/MSSQLServer4.

This document tells you how to install WebLogic jDriver for Microsoft SQL Server (formerly called FastForward for SQL Server), WebLogic's pure-Java Type 4 JDBC driver for Microsoft SQL Server. WebLogic jDriver for Microsoft SQL Server is a 100% pure Java implementation of the Java Database Connectivity (JDBC) API, the industry standard for relational database access from Java clients. It provides Java clients, including applets, with direct access to Microsoft SQL Server. The driver ships in two versions, one for SQL Server version 6.5 and 7.0, and another for SQL Server 7.0 only. Both versions function identically, except as noted below.

WebLogic jDriver for Microsoft SQL Server, like all Type 4 JDBC drivers, is pure-Java and 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.

Which version should I use?

WebLogic jDriver for Microsoft SQL Server version 6.5 and 7.0

- Supports SQL Server versions 4.2, 6.0, and 6.5.
- Supports SQL Server 7.0, with the following restrictions:
  - The SQL Server responds as if the connection were coming from a 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 is the normal behavior for SQL Server version 6.5.)
  - New SQL Server 7.0 data types are not supported.

WebLogic jDriver for Microsoft SQL Server version 7.0

- Supports only SQL Server 7.0. The SQL Server responds to requests made from this driver as requests from a SQL Server version 7.0 client and implements the semantics of a SQL Server 7.0. 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.

Before you begin

Upgrade information

If you are using WebLogic jDriver for Microsoft SQL Server with WebLogic Server 4.5 or later, you must also upgrade the driver for use with WebLogic Server.

If you have previously purchased a license for WebLogic jDriver for Microsoft SQL Server, *you must obtain an updated license file from your BEA contact person*. Instructions for updating your license are included in the Installation steps, below.

Evaluation licenses

The WebLogic jDriver for Microsoft SQL Server distribution you downloaded from WebLogic's website includes an evaluation license that allows you to test the product with 2 users for 30 days. To purchase additional licenses, contact sales@weblogic.com.

Installation steps

1. **Unpack the distribution.**

   If you have not already done so, unpack the zip file containing WebLogic jDriver for Microsoft SQL Server into the root directory of your computer. You must use a program such as WinZip at *http://www.winzip.com* that preserves the directory structure of the archive. The files unpack into the `weblogic` directory.

   The instructions in this document assume that you are installing WebLogic jDriver for Microsoft SQL Server on a Windows computer and that you unpacked the archive in the root directory of your C drive. If you're installing on a different operating system or in a different location, check your system documentation for help in adjusting the commands shown in these instructions for your own environment.

2. **Add the WebLogic jDriver for Microsoft SQL Server classes and license directory to your CLASSPATH variable.**

   To change your CLASSPATH temporarily at a Windows Command Prompt, use this command:

   ```
   $ set CLASSPATH=%CLASSPATH%;c:\weblogic\mssqlserver4\classes;
     c:\weblogic\mssqlserver4\license
   ```

   (If you are using the WebLogic jDriver for Microsoft SQL Server version 7.0, substitute *`mssqlserver4v70`* for *`mssqlserver4`* in the above command line.)

   To change your CLASSPATH permanently:

   a. Double-click the System icon in the Control Panel.

   b. Click the Environment tab.

c. In the lower panel, select the CLASSPATH variable. If no CLASSPATH variable is defined, add it.

d. Add `c:\weblogic\mssqlserver4\classes` and `c:\weblogic\mssqlserver4\license` to the value of the CLASSPATH variable. Use a semicolon (;) to separate the new path from the previous value of the variable, if any.

(If you are using the WebLogic jDriver for Microsoft SQL Server version 7.0, substitute *`mssqlserver4v70`* for *`mssqlserver4`* in the above classpath.)

Any program you launch after you update the CLASSPATH variable in the System control panel will have access to the new value.

For more help setting your CLASSPATH, read Setting classpath at *http://www.weblogic.com/docs51/admindocs/classpath.html*. If you're using WebLogic jDriver for Microsoft SQL Server from within an IDE like Symantec Cafe or JBuilder, the procedure for adding classes to the CLASSPATH may be different.

3. **Install** WebLogic jDriver for Microsoft SQL Server **for use with browser applets.**

If you're planning to use WebLogic jDriver for Microsoft SQL Server to access SQL Server databases via applets in a web browser, you must install the distribution on the computer executing the web server. Furthermore, the web server and SQL Server must be running on the same computer. This is a security restriction enforced by most web browsers, including Netscape Navigator and Internet Explorer.

The web server must also be able to find the WebLogic jDriver for Microsoft SQL Server class files. To ensure that the web server can find the files, install the WebLogic jDriver for Microsoft SQL Server distribution on your web server computer and then set the CLASSPATH on that computer as described in step 2. above. Be sure to restart the web server after you change the CLASSPATH so that the web server has access to the revised CLASSPATH value.

4. Install a license file.

With the release of version 5.1 of WebLogic jDriver for Microsoft SQL Server, a new type of license is shipped with the distribution. This license uses an XML file to store the license keys.

There are several issues to consider regarding your license for WebLogic jDriver for Microsoft SQL Server:

- If you downloaded an evaluation version of the driver, your distribution includes an evaluation license. Your driver is ready for evaluation use.
- If you have previously purchased a license for WebLogic jDriver for Microsoft SQL Server, *you must obtain an updated license file from your BEA contact person* and place that license in your WebLogic jDriver for Microsoft SQL Server installation. The procedure required to use the XML license differs depending on whether you are using WebLogic jDriver for Microsoft SQL Server as a standalone driver (without WebLogic Server), or as a two-tier driver from within WebLogic Server (using the t3, pool, JTS, or RMI type-3 drivers):

  Standalone use

  Copy your license key into your `WebLogicLicense.xml,` file. This file is located at *`weblogic`*`/mssqlserver4/license/WebLogicLicense.xml` (where *`weblogic`* is the directory containing your WebLogic jDriver for Microsoft SQL Server installation). For instructions on editing a WebLogic XML license file, see Editing an entry to the XML license file, below.

  (If you are using the WebLogic jDriver for Microsoft SQL Server version 7.0, substitute *`mssqlserver4v70`* for *`mssqlserver4`* in the above path.)

  Using with WebLogic Server

  If are using WebLogic jDriver for Microsoft SQL Server as a middle tier JDBC driver within WebLogic Server (via the pool, JTS, t3, or RMI type-3 drivers), copy your license key into the license file used by WebLogic Server. This file is located at *`weblogic`*`/license/WebLogicLicense.xml` (where *`weblogic`* is the directory containing your WebLogic Server installation). For instructions on editing a WebLogic XML license file, see Editing an entry to the XML license file below.

To purchase a permanent license for WebLogic jDriver for Microsoft SQL Server, contact sales@weblogic.com.

Editing an entry to the XML license file

1. Open the message containing the key(s) you received by email when you purchased a license from BEA Systems, Inc.
2. Open the `WebLogicLicense.xml` file in a text editor. This file is located in one of the following directories:
   - standalone driver: `weblogic/mssqlserver4/license`
   - standalone driver for SQL Server version 7.0: `weblogic/mssqlserver4v70/license`
   - using the driver with WebLogic Server: `weblogic/license`

   (Where *`weblogic`* is the directory containing your WebLogic jDriver for Microsoft SQL Server or WebLogic Server installation.)

   Do not edit this file in Microsoft Word or any other word-processing program that will save the file as a binary.
3. Copy the keys from the email and paste them **at the top** of the XML file.

   For example, if you have received a key for clustering, it looks something like this:

   ```
   <LICENSE PRODUCT="jdbcKona/MSSQLServer4"
         IP="000.000.900.900"
         UNITS="5"
         EXPIRATION="31-Mar-1999"
         KEY="w20f8s08480v0adpup43245paprtnp8ac"
   />
   ```
4. Save the `WebLogicLicense.xml` file into the same directory.
5. Save your license key information and a copy of your `WebLogicLicense.xml` file in a safe place outside the WebLogic distribution, preferably someplace you will remember. Although no one else will be able to use your license keys, you should probably save this information in a place protected from either malicious or innocent tampering by others. Please note that when you upgrade your WebLogic classes, you will want to keep your original `WebLogicLicense.xml` and add new keys to it for any new services you purchase.

Setting up a connection pool

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. Since the connections are shared among users, these connection pools eliminate the overhead of opening a new database connection for each user.

Your application then uses a multitier (Type-3) JDBC driver, such as the WebLogic **Pool**, **JTS** or **RMI** driver to connect to WebLogic Server. WebLogic Server then uses WebLogic jDriver for Microsoft SQL Server and one of the existing connections from the pool to connect to the SQL Server database on behalf of your application.

Configuring a connection pool with WebLogic Server

1. Include the WebLogic jDriver for Microsoft SQL Server classes in the WebLogic classpath used to start WebLogic Server (specified with the `weblogic.class.path` property on the command line). For more information on starting WebLogic Server, see Setting up and Starting WebLogic Server at *http://www.weblogic.com/docs51/install/startserver.html.*

2. Add an entry to the `weblogic.properties` file specifying the connection pool properties (driver name, url, server, password, ACLs, etc.). For example:

```
weblogic.jdbc.connectionPool.SQLPool=\
      url=jdbc:weblogic:mssqlserver4:myServer:myPort,\
      driver=weblogic.jdbc.mssqlserver4.Driver,\
      loginDelaySecs=1,\
      initialCapacity=4,\
      maxCapacity=10,\
      capacityIncrement=2,\
      allowShrinking=true,\
      shrinkPeriodMins=15,\
      refreshTestMinutes=10,\
      testTable=dual,\
      props=user=myUserName;password=secret;server=myServer;\


weblogic.allow.reserve.weblogic.jdbc.connectionPool.SQLPool=\
       guest,joe,jill
weblogic.allow.reset.weblogic.jdbc.connectionPool.SQLPool=\
       joe,jill
weblogic.allow.shrink.weblogic.jdbc.connectionPool.SQLPool=\
       joe,jill
```

For more information on setting properties for connection pools, see JDBC Connection Pools, at *http://www.weblogic.com/docs51/admindocs/properties.html#conpools,* in the WebLogic Administrators guide *Setting WebLogic Properties.*

3. Start WebLogic Server.

Using the connection pool in your application

Client-side applications

To use a connection pool in a client-side application, establish the database connection using the WebLogic JDBC/RMI driver. For more information, see

- Using WebLogic JDBC/RMI and WebLogic Clustered JDBC at *http://www.weblogic.com/docs51/classdocs/JDBC_RMI.html*.

Server-side applications

To use a connection pool in a server-side application (such as a servlet), establish your database connection using the WebLogic `pool` or `jts` drivers. For more information, see:

- Using connection pools with server-side Java (in *Using WebLogic HTTP Servlets*) at *http://www.weblogic.com/docs51/classdocs/API_servlet.html#pools0*
- Creating a startup connection pool at *http://www.weblogic.com/docs51/classdocs/API_jdbct3.html#startupconnpool*

Verifying your SQL Server installation

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
- *Hostname* or *IP number* of the machine where the SQL Server is running

- *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 on page 15.

Setting a port for SQL Server connections

You set the hostname and port for SQL Server connections by making an entry into 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 the logical server name, 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 with `telnet` that the server is listening on that port. Here is the syntax for the telnet command:

```
$ telnet hostname_or_IP_address port
```

For example, to test the SQL Server listening on port 1433 of the computer named "myhost", type:

```
$ telnet myhost 1433
```

If the server is *not* listening on the port, telnet will display an error message. If the server *is* listening on the port, telnet will display nothing; eventually, the host will drop the connection.

You can use `isql` to test your login information. Here's the `isql` syntax:

```
$ isql -Uusername -Ppassword -Sserver
```

Verifying the JDBC driver with dbping

You can use the WebLogic `dbping` Java application to verify that WebLogic jDriver for Microsoft SQL Server can connect to your SQL Server. To use `dbping`, make sure that the classes for WebLogic jDriver for Microsoft SQL Server are in your CLASSPATH (as described under Installation steps, on page 9), then type this command:

```
$ java utils.dbping MSSQLSERVER4 username password
   [database@]host[:port]
```

where:

username is the database user name
password is the user's password
database (optional) is the SQL Server database to use
host is the name or IP number of the computer running SQL Server
port (optional) is the TCP/IP port the SQL Server is listening on

For example, the following command pings the SQL Server "pubs" database on the computer named "myhost", using the default TCP/IP port, and the "sa" login with 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.

See Using WebLogic jDriver for SQL Server, at *http://www.weblogic.com/docs51/classdocs/API_jmsq4.html*, for more information about JDBC connections.

More resources

Documentation

- API reference *(http://www.weblogic.com/docs51/classdocs/packages.html)*

- Using WebLogic jDriver for SQL Server
  (http://www.weblogic.com/docs51/classdocs/API_jmsq4.html)
- WebLogic JDBC Options
  (*http://www.weblogic.com/docs51/classdocs/jdbcdrivers.html*)

Code examples

We also ship a number of code examples in the distribution that will help you get started. Code examples are located `in the examples` directory in the distribution.

2 Using WebLogic jDriver for Microsoft SQL Server

Introduction

WebLogic jDriver for Microsoft SQL Server is a new name for the product previously known as jdbcKona/MSSQLServer.

What's in this document

This document provides information on how to set up and use WebLogic jDriver for Microsoft SQL Server. WebLogic acquired this product, formerly known as FastForward, from Connect Software in April 1998.

WebLogic jDriver for Microsoft SQL Server is a Type 4, pure-Java, two-tier driver. It requires no client-side libraries since it connects to the database via a proprietary vendor protocol at the wire-format level. Unlike Type 2 JDBC drivers, Type 4 drivers make no "native" calls, so they can be used in Java applets.

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 four types of JDBC drivers, read the whitepaper Choosing a JDBC Driver at *http://www.weblogic.com/docs51/techoverview/jdbc.html*.

Within the WebLogic environment, you can use a Type 2 or a Type 4 two-tier driver to connect the WebLogic Server to a database, and then use WebLogic JDBC, WebLogic's pure-Java Type 3 multitier JDBC driver, for client connections to the WebLogic Server.

The API reference for JDBC, of which this driver is a fully compliant implementation, is available online in several formats at JavaSoft at *http://www.JavaSoft.com*.

Implementing with WebLogic jDriver for Microsoft SQL Server

Connecting to a SQL Server DBMS

Here is how to set up your application to connect to SQL Server using WebLogic jDriver for Microsoft SQL Server. In general, connecting happens in three steps:

1. Load the JDBC driver by calling `Class.forName().newInstance()` with the full class name of the WebLogic jDriver for Microsoft SQL Server JDBC driver class, and casting it to a `java.sql.Driver` object. This procedure properly loads and registers the JDBC driver. For example:

   ```
   Driver myDriver = (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 user name, 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, which 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);
   ```

In steps 1 and 3, you are describing the JDBC driver. In the first step, you use the full package name of the driver. Note that it is dot-delimited. In the third step, you identify the driver with its URL, which is colon-delimited. The URL must include at least *weblogic:jdbc:mssqlserver4*, and may include other information, including the server host name and the database name.

Connection example

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

You can combine the "db", "server", and "port" properties into one "server" property, as in this 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 MS SQL server, where they can be read in the `sysprocesses` table under the column names "program_name" and "hostname". The hostname value will be prepended with "WebLogic".

```
Driver myDriver = (Driver)

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

There are any number of variations on supplying information in the URL or in the Properties object. Whatever information you pass in the URL of the driver, you don't need to include in the Properties object. For more information, see Using URLs to set properties for a JDBC connection at *http://www.weblogic.com/docs51/classdocs/urls.html*.

Adding other 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 in this example:

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

You can use `Driver.getPropertyInfo()` to find out more about URL options at run-time.

Setting properties for WebLogic JDBC use

If you are using WebLogic JDBC in a multitier environment with a two-tier JDBC driver, you set connection properties in a slightly different way. See Using WebLogic JDBC at *http://www.weblogic.com/docs51/classdocs/API_jdbct3.htm*

Overview of data manipulation

This section provides an overview of the basic steps for executing SQL data queries, making inserts and updates, using stored procedures, and using callable statements. These steps follow basic JDBC methodology and are intended as a brief introduction to data manipulation with JDBC. For additional information, you should consult your SQL Server documentation as well as Java-oriented texts discussing JDBC. You may also wish to consult JavaSoft's JDBC tutorial at *http://java.sun.com/docs/books/tutorial/jdbc/index.html*.

Making a simple SQL query

The most fundamental task in database access is to retrieve data. With WebLogic jDriver for Microsoft SQL Server, this is a three-step process:

1. Create a Statement to send a 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());
for (int i = 1; i
```

Inserting, updating, and deleting records

We illustrate three common database tasks in this step: inserting, updating, and deleting records from a database table. We use a JDBC PreparedStatement for these operations; we create the PreparedStatement, then execute it and close it.

A PreparedStatement (subclassed from JDBC Statement) allows you to execute the same SQL over and over again with different values. PreparedStatements use the JDBC "?" syntax.

```
String inssql =
   "insert into emp(empid, name, dept) values (?, ?, ?)";
PreparedStatement pstmt = conn.prepareStatement(inssql);

for (int i = 0; i
```

We also use a PreparedStatement to update records. In this 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);

for (int i = 0; i
```

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);

for (int i = 0; i
```

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. First, 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, like `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 both `IN` and `OUT`, which requires both a `setXXX()` and a `registerOutParameter()` call to be done on the same parameter number.

In this example, we use a JDBC Statement to create a stored procedure, then we 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);
  for (int i = 0; i
```

In the next example, 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);
for (int i = 0; i
```

This next example uses a stored procedure named sp_getmessages (the code for this stored procedure is not included with this example) named `sp_getmessages`, which takes a message number as an input parameter and returns the message text in a ResultSet as an output parameter. Note that 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.

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

```
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);
```

We execute the stored procedure and check the return value to see if 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();
  }
```

Once we have finished processing the ResultSet, the `OUT` parameters and return status are available.

```
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

There are occasions on which you will want to call the `commit()` method to commit changes you have made to the database before you close the 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, and any call to the `rollback()` or `commit()` methods will end the current transaction and start a new one. Calling `commit()` before `close()` ensures that all of the transactions are completed before closing the Connection.

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 `try {}` block, and you should catch exceptions and deal with them appropriately. The final two lines of this example include a call to `commit` and then a call to `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. You can find the List of supported encodings at *http://www.JavaSoft.com/products/jdk/1.1/docs/guide/intl/encoding.doc.html* at the JavaSoft website.

For example, to use the cp932 codeset, create a Properties object and set the `weblogic.codeset` property before calling `Driver.connect()`, as in this example:

```
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);
```

Using WebLogic jDriver for Microsoft SQL Server in applets

Since WebLogic jDriver for Microsoft SQL Server is 100% Java, you can use it in Java applets. The `examples` directory includes Java applets to help you set up WebLogic jDriver for Microsoft SQL Server for use in applets. See the examples documentation (in the `examples` directory of your WebLogic jDriver for Microsoft SQL Server installation) for information about setting up and running the examples. If you're planning to use WebLogic jDriver for Microsoft SQL Server in an applet in a two-tier configuration, you will need to install the driver on your web server host. In addition, **the web server and the SQL Server must be located on the same machine**. Because an applet must communicate only with the host from which its classes were loaded, the SQL Server and the web server must be on the same IP host.

One way to overcome applet security restrictions is to use WebLogic jDriver for Microsoft SQL Server in a three-tier configuration with a server such as the WebLogic Server. In this configuration, applets connect to WebLogic Server via a WebLogic type-3 driver and WebLogic connects to SQL Server via WebLogic jDriver for

Microsoft SQL Server. For more information on using WebLogic with applets, read Using WebLogic for applet programming at *http://www.weblogic.com/docs51/techdeploy/applets.html*.

The SocketApplet applet (see `examples.jdbc.mssqlserver4.SocketApplet.html` in the `examples` directory of your WebLogic jDriver for Microsoft SQL Server installation) tests whether the driver can connect to the SQL Server socket. Once you have run `SocketApplet` successfully, you can use the `ConnectionApplet` applet to verify that you can *log into* the SQL Server.

To run SocketApplet:

1. Make sure that `c:\weblogic\mssqlserver4\classes` is in your `CLASSPATH`. (Or `c:\weblogic\mssqlserver4v70\classes`, if you are using WebLogic jDriver for Microsoft SQL Server version 7.0.)

2. At a command prompt, change to the `weblogic\mssqlserver4\examples\jdbc\mssqlserver4` directory (or `weblogic\mssqlserver4v70\examples\jdbc\mssqlserver4`, depending on which version of the driver you are using).

3. Edit the `SocketApplet.java` file. Make the following changes (you can search for the string "FIXME" to find the locations):

   - Replace "myhost" with the name of the computer running SQL Server.
   - Replace "1433" with the TCP/IP port number the SQL Server is listening on. The default port number is 1433, but you should check for the actual port your SQL Server is listening on.

   Save your changes and compile the applet:

   ```
   javac SocketApplet.java
   ```

   If you see compiler errors, make sure that the WebLogic jDriver for Microsoft SQL Server distribution is installed on the web server's computer and that `c:\weblogic\mssqlserver4\classes` (or `c:\weblogic\mssqlserver4v70\classes`, if you are using WebLogic jDriver for Microsoft SQL Server 7.0) is in the CLASSPATH when you start the web server. Copy `SocketApplet.html` and `SocketApplet.java` into a directory served by your web server.

   Load `SocketApplet.html` in your browser.

`SocketApplet` displays an error message if it was unable to connect to the SQL Server socket or a success message if it connected successfully.

Applet troubleshooting hints:

- The web server host name in the URL that you type in the browser must match the host name in the JDBC connection URL. For example, if you are running the browser, SQL Server and web server all on the same computer, you could refer to the computer by its name, IP number, or "localhost". Although these names all resolve to the same computer, some browsers require the names used to match.
- Be sure to load `SocketApplet.html` via a web server. Loading the page with a `file:///` URL will not work.
- If you are unable to connect with `SocketApplet`, make sure that you test the SQL Server connectivity with dbping (see *http://www.weblogic.com/docs51/techstart/dbping.html*).
- If you have more questions about applet security issues, read Troubleshooting Applet Security problems at *http://www.weblogic.com/docs51/techsupport/appletsecurity.html*.

To run ConnectionApplet:

1. Edit `ConnectionApplet.java`:
   - Replace "myhost" in the connection URL with the name of the computer running SQL Server
   - Add your own database user name and password

   Save your changes.

   Compile `ConnectionApplet.java`:

   ```
   javac ConnectionApplet.java
   ```

   Make sure that `c:\weblogic\mssqlserver4\classes` (or `c:\weblogic\mssqlserver4v70\classes`, if you are using WebLogic jDriver for Microsoft SQL Server 7.0) is in the `CLASSPATH` before you start the web server.

   Copy `ConnectionApplet.html` and `ConnectionApplet.class` into a directory served by your web server.

   Load `ConnectionApplet.html` with `Appletviewer`:

```
Appletviewer ConnectionApplet.html
```

The applet displays a message telling you that either the connection succeeded or that there was an exception. The exception message will help you diagnose problems.

The WebLogic jDriver for Microsoft SQL Server distribution includes several additional examples. Refer to the `examples` directory of your WebLogic jDriver for Microsoft SQL Server installation for information about these examples.

JDBC extensions and limitations

Support for JDBC Extended SQL

The JavaSoft JDBC specification includes *SQL Extensions*, also called *SQL Escape Syntax*. WebLogic jDriver for Microsoft SQL Server supports and 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 DBMSs as follows:

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

Here's an example that 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. Here is Extended SQL used as part of a CallableStatement:

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

This example shows that you can nest extended SQL expressions:

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

You can retrieve lists of supported Extended SQL functions from a DatabaseMetaData object. This example shows how to list all the functions a JDBC driver supports:

```
 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();
```

Refer to chapter 11 of the JDBC 1.2 specification at JavaSoft at *http://splash.javasoft.com/jdbc* for a description of Extended SQL.

cursorName() method not supported

The `cursorName()` method is not supported.

java.sql.TimeStamp limitations

The java.sql.TimeStamp class in the JavaSoft 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.

Querying metadata

You can only query metadata for the current database. 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.

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.exexecuteWriteText()` requires the DB-Library native libraries, and thus is not supported by the WebLogic jDriver for Microsoft SQL Server JDBC driver.

To read and write text and image data with streams, you can use the `prepareStatement.setAsciiStream()`, `prepareStatement.setBinaryStream()`, `ResultSet.getAsciiStream()`, and `ResultSet.getBinaryStream()` JDBC methods.

Sharing a Connection object in multi-threaded applications

WebLogic jDriver for Microsoft SQL Server allows you to write multithreaded applications where multiple threads can share a single Connection option. 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. The actual Statement that is cancelled depends on timing issues in the SQL Server. To avoid this unexpected behavior, 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.

More Resources

- WebLogic JDBC Options (*http://www.weblogic.com/docs51/classdocs/jdbcdrivers.html*)

  Contains information on other WebLogic JDBC drivers, additional documentation, support resources, and more.

- Using connection pools with server-side Java (in *Using WebLogic HTTP Servlets*) at *http://www.weblogic.com/docs51/classdocs/API_servlet.html#pools0*
- Creating a startup connection pool at *http://www.weblogic.com/docs51/classdocs/API_jdbct3.html#startupconnpool*
- JavaSoft's JDBC tutorial (*http://java.sun.com/docs/books/tutorial/jdbc/index.html*.)

Code examples

We also ship a number of code examples in the distribution that will help you get started. Code examples are located in the `weblogic/mssqlserver4/examples/` directory in the distribution. If you are using WebLogic jDriver for Microsoft SQL Server version 7.0, examples are located at `weblogic/mssqlserver4v70/examples/`

3 Release Notes

- WebLogic jDriver for Microsoft SQL Server is now availble in versions, one for both SQL Server 6.5 and 7.0, and one for SQL Server version 7.0. For details on which driver to use in your environment, see Which version should I use? on page 8.
- When establishing the database connection in your code, BEA now recommends, for performance reasons, that you *not* use the `DriverManager.getConnection()` method. In its place, construct a properties object and pass the driver class and properties to the `java.sql.Driver.connect()` method. This procedure is described in detail in the section Connecting to a SQL Server DBMS on page 20.