BEA WebLogic Server

Installing and Using WebLogic jDriver for Informix

BEA WebLogic Server 6.0
Document Date: March 6, 2001

Copyright

Restricted Rights Legend

Trademarks or Service Marks

Installing and Using WebLogic jDriver for Informix

Part Number	Document Date	Software Version
N/A	March 6, 2001	BEA WebLogic Server Version 6.0

Contents

About This Document

1. Installing WebLogic jDriver for Informix

2. Using WebLogic jDriver for Informix

About This Document

This document describes how to install and develop applications using WebLogic jDriver for Informix, BEA's type-4 Java Database Connectivity (JDBC) driver for the Informix Database management system.

This document is organized as follows:

- Chapter 1, "Installing WebLogic jDriver for Informix.".
- Chapter 2, "Using WebLogic jDriver for Informix."

Audience

This document is written for application developers who want to build e-commerce applications using the Java 2 Platform, Enterprise Edition (J2EE) from Sun Microsystems. It is assumed that readers are familar 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/docs60.

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 BEA corporate Web site provides all documentation for WebLogic Server.

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 your 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.
italics	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.
	Examples:
	`import java.util.Enumeration;`
	`chmod u+w *`
	`config/examples/applications`
	`.java`
	`config.xml`
	`float`

`monospace italic text`	Variables in code. *Example*: `String` *`CustomerName`*`;`

Convention	Usage
UPPERCASE TEXT	Device names, environment variables, and logical operators. *Example*s: LPT1 BEA_HOME OR
`{ }`	A set of choices in a syntax line.
`[ ]`	Optional items in a syntax line. *Example*: `java utils.MulticastTest -n name -a address [-p portnumber] [-t timeout] [-s send]`
`|`	Separates mutually exclusive choices in a syntax line. *Example*: `java weblogic.deploy [list|deploy|undeploy|update] password {application} {source}`
`...`	Indicates one of the following in a command line: ■ 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.

CHAPTER

1 Installing WebLogic jDriver for Informix

This document tells you how to install WebLogic jDriver for Informix, BEA's pure-Java Type 4 JDBC driver for Informix.

Specifically, it provides information about the following topics:

- Overview
- Before You Begin
- Installation Procedure
- Using Connection Pools
- Verifying Your Connection to the Informix Database
- Determining Your Database, Hostname, and Port
- References

Overview

WebLogic jDriver for Informix 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 with direct access to the Informix database management system (DBMS).

Before You Begin

This section explains when and how you need to upgrade your software for WebLogic Server Version 6.0.

Evaluation Licenses

The WebLogic jDriver licensing functionality is included in the license file located in the directory where you installed this version of WebLogic jDriver for Informix. For example:

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

Installation Procedure

WebLogic jDriver for Informix is bundled with your WebLogic Server distribution and is installed when you install WebLogic Server. The **`weblogic.jar`** file includes the Informix classes. You do not need to perform any steps for installation.

Note: The standalone WebLogic jDriver for Informix will be available in the next release.

Using Connection Pools

If you are using WebLogic jDriver for Informix with either WebLogic Server or WebLogic Express, you can set up a pool of connections to your Informix 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 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 Informix and one of the existing connections from the pool to connect to the Informix database on behalf of your application.

Configuring a Connection Pool with WebLogic Server

1. Include the WebLogic jDriver for Informix classes in the WebLogic classpath used to start WebLogic Server. For more information, see Starting and Stopping WebLogic Servers in the *Administration Guide* at http://e-docs.bea.com/wls/docs60/adminguide/startstop.html.
2. Use the Administration Console to set connection pools. To read about connection pools, see Connection Pools in the *Administration Guide* or, to go directly to the procedure, Create a JDBC Connection Pool in Online Help.
3. Start WebLogic Server.

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 is 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 using the WebLogic RMI driver. For more information about the RMI driver, see "Using WebLogic Multitier Drivers" 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 by using the WebLogic `pool` or `jts` drivers. For more information, see:

- "Programming Tasks" in *Programming WebLogic HTTP Servlets*

Verifying Your Connection to the Informix Database

Check your connection to the Informix database. You must have the following information:

- For the user: Valid username and password
- For the database: Database name, host name, and port number

If you do not have the required information about your database, see Determining Your Database, Hostname, and Port in the following section.

Once you have collected the required information, you can test your connection. At a command line, type:

```
java utils.dbping INFORMIX4 user password db@host:port
```

The arguments in this command line are defined as follows:

- *user* is the Informix username of a valid user for this database.
- *password* is the password for the user.
- *db@host:port*—Together, these three arguments show how to reach your Informix database:
  - *db* is the name of the database.
  - *host* is the name of the computer on which the Informix server is running.
  - *port* is the number of the TCP/IP port on which the Informix server is listening for connection requests.

  Note the syntax of the command: the database name is followed by @ (the *at* sign); the host name and port are separated by a colon.

For instructions for verifying your connection to a DBMS, see Testing Connections in Configuring WebLogic JDBC Features in *Programming WebLogic JDBC* at http://e-docs.bea.com/wls/docs60/jdbc/programming.html.

Determining Your Database, Hostname, and Port

Before you can connect to your Informix server installation, you must answer the following questions about it:

- What is the name of the database you will be accessing?
- What is the name of the host computer on which the Informix server is running?
- What is the TCP/IP address of the port on which the Informix server is listening for connection requests?

To avoid confusion, be careful not to use the word *server* to mean both the machine on which your database is running and the database instance itself. In this document we avoid this pitfall by using the following terms:

- *Host name* refers to the name of a machine.
- *Database name* refers to the name of an Informix instance.

To get information about the Informix server to which you want to connect, look in `$INFORMIXDIR/etc/sqlhosts` and find the appropriate entry under the SERVER column. The entry in this file will tell you the host name and *service name*—located in the far right column—for the connection. As long as you know the *service name*, you can determine the port number.

Then look in your `/etc/services` file (or, on a Windows NT platform, in `\Winnt \system32\drivers\etc\services`) to find the port number associated with the service name.

References

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

Documentation

- API reference at http://www.weblogic.com/docs60/samples/examples/jdbc/package-summary.html.
- "Using WebLogic jDriver for Informix" at http://e-docs.bea.com/wls/docs60/jdbc/API-jinf4.
- *Programming WebLogic JDBC* at http://e-docs.bea.com/wls/docs60/jdbc/index.html.

Code Examples

WebLogic Server provides code examples to help you get started. Code examples are located in the `samples/examples/jdbc/informix4` directory of your WebLogic Server distribution.

2 Using WebLogic jDriver for Informix

This section explains how to set up and use WebLogic jDriver for Informix. Specifically, it discusses the following topics:

- What Is the WebLogic jDriver for Informix?
- Mapping Types
- Connecting to an Informix DBMS
- Manipulating Data with JDBC
- WebLogic jDriver for Informix Conformance to JDBC
- References

What Is the WebLogic jDriver for Informix?

WebLogic jdriver for Informix is a Type 4, pure-Java, two-tier driver. It requires no client-side libraries because it connects to a database through a proprietary vendor protocol at the wire-format level. Consequently, unlike a Type 2 two-tier driver, it makes no *native* calls; it is written exclusively in Java.

A Type 4 driver is similar to a Type 2 driver in one respect, however. Because both types are two-tier drivers, any client used with either type of driver requires an in-memory copy of the driver to support its connection to the database.

WebLogic jDriver for Informix supports concurrent ResultSet execution. You are not required to close one ResultSet on a Connection before you can open and work with another. However, the driver cannot support both concurrent ResultSet execution *and* client-side caching.

WebLogic jDriver for Informix supports Informix OnLine versions 7.x and 9.x, with 7.x data types, plus the 9.x INT8 and SERIAL8 data types.

Mapping Types

The following table shows how to map:

- Informix types to WebLogic jDriver for Informix types
- WebLogic jDriver for Informix types to Java types

Informix	WebLogic jDriver for Informix	Java Type
Byte	Binary	use java.io.InputStream
Char	Char	java.lang.String
Date	Date	java.sql.Date
Datetime	Timestamp	java.sql.Timestamp
Decimal	Decimal	java.math.BigDecimal
Float	Decimal	java.math.BigDecimal
Integer	Integer	java.lang.Integer
Integer8	Long	java.lang.BigInt
Interval	InformixInterval	Literal string in Informix
Money	Decimal	java.math.BigDecimal
NChar	Char	java.lang.String
NVarchar	Varchar	java.lang.String

Informix	WebLogic jDriver for Informix	Java Type
Serial	Integer	java.lang.Integer
Serial8	Long	java.lang.BigInt
Smallfloat	Decimal	java.math.BigDecimal
Smallint	Smallint	java.lang.Integer
Text	Longvarchar	use java.io.InputStream
Varchar	Varchar	java.lang.String

Connecting to an Informix DBMS

This section presents instructions for coding the step of connecting to an Informix DBMS and a piece of sample code that shows how such a connection is made.

Connection Procedure

Complete the following three-step procedure to set up your application to connect to Informix using WebLogic jDriver for Informix:

1. Load and register the JDBC driver by doing the following:

   a. Call `Class.forName().newInstance()` with the full name of the WebLogic jDriver for Informix JDBC driver class.

   b. Cast it to a `java.sql.Driver` object.

   For example:

```
Driver myDriver = (Driver)
 Class.forName("weblogic.jdbc.informix4.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:informix4", props);
```

In steps 1 and 3, you describe 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 the following string: `weblogic:jdbc:informix4`. It may also include other information, such as the server host name and the database name.

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.informix4.Driver").newInstance();
Connection conn =
  myDriver.connect("jdbc:weblogic:informix4", 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");

Driver myDriver = (Driver)
  Class.forName("weblogic.jdbc.informix4.Driver").newInstance();
Connection conn  =
  myDriver.connect("jdbc:weblogic:informix4", props);
```

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

Additional Informix-Specific Properties for the Connection or Properties Object

This section describes other Informix-specific properties that you can set in the connection URL or Properties object. These properties give you more control over an Informix-specific environment. For more information, see your Informix documentation.

`weblogic.informix4.login_timeout_secs=`*`seconds_to_wait`*
: When an attempt to log in to an Informix server times out, WebLogic jDriver for Informix returns an SQLException. By default, the driver waits 90 seconds before it times out. You can modify the timeout period by setting this property to the number of seconds you want to wait before returning an SQLException.

`weblogic.informix4.delimited_identifiers=y`
: The Informix environment variable `DELIMIDENT` is used to enable and disable ANSI SQL Delimited Identifiers. The default is off (n).

`weblogic.informix4.db_money=`*`currency`*
: The Informix environment variable `DBMONEY` is used to set the display of the currency symbol. The default value is now `$.`, which can be overridden with this property.

`weblogic.informix4.db_date=`*`dateformat`*
: The Informix environment variable `DBDATE` allows a user to specify the input format of dates. It sets the Informix `DBDATE` environment variable at login.

The default value is `Y4MD`. Two-digit years (formats containing `Y2`) are not supported by the driver. Note that you cannot use this variable to format, correctly, a date obtained from a `ResultSet.getString()` statement. Instead, use `ResultSet.getDate()` to obtain a `java.util.Date` object and then format the date in your code.

The following sample code shows how these properties are used in a URL:

```
jdbc:weblogic:informix4:mydb@host:1493
?weblogic.informix4.delimited_identifiers=y
&weblogic.informix4.db_money=DM
&weblogic.informix4.db_date=Y4MD
```

Note: A URL is always entered as a single line. In the previous example, a single URL is presented on multiple lines to enhance readability.

Note the use of `?` and `&`, which are special characters for URLs.

The following sample code shows how these properties might be used with a Properties object:

```
Properties props = new Properties();
props.put("user",                                    "scott");
props.put("password",                                "tiger");
props.put("weblogic.informix4.delimited_identifiers", "y");
props.put("weblogic.informix4.db_money",             "DM");

Connection conn = myDriver.connect
   (jdbc:weblogic:informix4:myDB@myHost:8659",props);
```

Manipulating Data with JDBC

This section provides basic procedures for implementing the following tasks in a program:

- Making a Simple SQL Query
- Inserting, Updating, and Deleting Records
- Creating and Using Stored Procedures and Functions
- Disconnecting and Closing Objects

- Retrieving the SERIAL Column After an Insert
- Using the Informix INTERVAL Data Type
- Using ResultSetMetaData Methods
- Using Autocommit Mode

These procedures are limited to basic JDBC methodology; they are intended as a brief introduction to data manipulation with JDBC. For more information, see your Informix documentation and Java-oriented texts about JDBC. See also 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 Informix, 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 and save them in 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 <= md.getColumnCount(); i++) {
    System.out.println("Column Name: "     +
```

```
        md.getColumnName(i));
      System.out.println("Nullable: "        +
        md.isNullable(i));
      System.out.println("Precision: "       +
        md.getPrecision(i));
      System.out.println("Scale: "           +
        md.getScale(i));
      System.out.println("Size: "            +
        md.getColumnDisplaySize(i));
      System.out.println("Column Type: "     +
        md.getColumnType(i));
      System.out.println("Column Type Name: "+
        md.getColumnTypeName(i));
      System.out.println("");
    }

stmt.close();
```

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 < 100; i++) {
  pstmt.setInt(1, i);
  pstmt.setString(2, "Person " + i);
  pstmt.setInt(3, i);
  pstmt.execute():
}
  pstmt.close();
```

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 < 100; i++) {
  pstmt2.setInt(1, i);
  pstmt2.setInt(2, i);
  pstmt2.execute();
}
pstmt2.close();
```

Finally, we use a PreparedStatement to delete the records that were added and then updated.

```
String delsql = "delete from emp where empid = ?";
PreparedStatement pstmt3 = conn.prepareStatement(delsql);
for (int i = 0; i < 100; i++) {
  pstmt3.setInt(1, i);
  pstmt3.execute();
}
pstmt3.close();
```

Creating and Using Stored Procedures and Functions

You can use WebLogic jDriver for Informix 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);
  for (int i = 0; i < 5; i++) {
  cstmt1.setInt(1, i);
  cstmt1.execute();
  System.out.println(i + " " + cstmt1.getInt(1) + " "
   + cstmt1.getInt(2));
} cstmt1.close();
```

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);
  for (int i = 0; i < 5; i++) {
  cstmt2.setInt(2, i);
  cstmt2.execute();
```

```
  System.out.println(i + " " + cstmt2.getInt(1) +
                     " " + cstmt2.getInt(2));
}
cstmt2.close();
```

Now 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, in a ResultSet as an output parameter. Before `OUT` parameters and return status are available, you must execute the `Statement.execute()` and `Statement.getResult()` methods on all ResultSets returned by a stored procedure.

```
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:

- Parameter 1 (output only) is the stored procedure return value.
- Parameter 2 (input only) is the `msgno` argument to `sp_getmessage`.
- 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

Sometimes you may want to commit changes you have made to the database before closing a connection. You can do so by calling the `commit()` method.

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 starts 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 a final cleanup step in your application, in a `try {}` 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();
```

Retrieving the SERIAL Column After an Insert

You can obtain serial values after an insert by using the `Statement.getSerialNumber()` method, a WebLogic extension to JDBC in WebLogic jDriver for Informix. This method allows you to track the index order of rows as you add them to the table. You must create the table with a SERIAL column.

To use this extension, you must cast your Statement object explicitly to `weblogic.jdbc.informix4.Statement`.

The following simple code example shows how to use the `getSerialNumber()` method:

```
  weblogic.jdbc.informix4.Statement stmt =
    (weblogic.jdbc.informix4.Statement)conn.createStatement();
  String sql = "CREATE TABLE test ( s SERIAL, count INT )";
  stmt.executeUpdate(sql);

  for (int i = 100; i < 110 ; i++ ) {
    sql = "INSERT INTO test VALUES (0, " + i + ")";
    stmt.executeUpdate(sql);
    int ser = stmt.getSerialNumber();
    System.out.println("serial number is: " + ser);
}
  sql = "SELECT * from test";
  ResultSet rs = stmt.executeQuery(sql);
  while (rs.next()) {
    System.out.println("row: " + rs.getString(2) +
             " serial: " + rs.getString(1));
```

Using the Informix INTERVAL Data Type

To use the Informix INTERVAL data type, import `weblogic.jdbc.common.InformixInterval` and cast your objects to `weblogic.jdbc.common.InformixInterval`.

Use a literal string in the Informix INTERVAL format to enter an INTERVAL value in an SQL statement. Use `preparedStatement.setString()` to set an INTERVAL value parameter in a prepared statement.

For retrieving INTERVAL data from an Informix server, WebLogic jDriver for Informix supports three standard API methods on a ResultSet:

- `ResultSet.getString()` returns a String representation of the interval in the standard Informix format. Returns null if the interval is null.
- `ResultSet.getBytes()` returns the actual bytes returned by the server to represent the interval.

- `ResultSet.getObject()` returns an object of type `weblogic.jdbc.common.InformixInterval`. Returns null if the interval is null.

The `InformixInterval` interface provides the following public methods:

`String getString() throws SQLException`
Identical to `ResultSet.getString()`

`int getYear() throws SQLException`
Returns the signed year of the INTERVAL or zero if YEAR is not defined

`int getMonth() throws SQLException`
Returns the signed year of the interval or zero if MONTH is not defined

`int getDay() throws SQLException`
Returns the signed day of the interval or zero if DAY is not defined

`int getHour() throws SQLException`
Returns the signed hour of the interval or zero if HOUR is not defined

`int getMinute() throws SQLException`
Returns the signed minute of the interval or zero if MINUTE is not defined

`int getSecond() throws SQLException`
Returns the signed second of the interval or zero if SECOND is not defined

`int getFraction() throws SQLException`
Returns the actual value of the FRACTION times 10**5

Using ResultSetMetaData Methods

You can access the metadata returned by the Informix server (along with query results) by using the ResultSetMetaData methods. However, the Informix server does not return information for the following:

`getSchemaName(int)`
`getTableName(int)`
`getCatalogName(int)`

Using Autocommit Mode

Unlike other database system attributes, the autocommit mode of an Informix database cannot be set dynamically. It is defined when the database is created. You cannot change it with a call to the `Connection.setAutoCommit` method. Only non-ANSI, non-logged databases support the ability to change autocommit dynamically.

The JDBC specification states that the autocommit mode should be true by default but, with Informix, it is not possible to make `true` the default autocommit setting. Informix allows you only to identify the autocommit mode. To change this mode, you must first rebuild your database. (For more information, see "CREATE DATABASE" in the Informix documentation.)

The fact that the database must be rebuilt before the autocommit state can be changed affects how transactions and locking work. Various JDBC programs behave differently, depending on how the Informix database is created in each program.

Before you decide to depend on autocommit, you should know the setting of autocommit for the database that you will use. You can check the autocommit mode for a database with the `Connection.getAutoCommit()` method. This method returns true if autocommit is enabled. For Informix, this method returns false, by default, for an ANSI databases; for a non-ANSI database, it may return true or false, depending on how the database was created.

The following settings are supported by WebLogic jDriver for Informix when you call the `Connection.setAutoCommit()` method:

- For ANSI databases, only autocommit=false is supported.
- For non-ANSI databases, autocommit can be set to either true or false.
- For non-ANSI databases without logging, only autocommit=true is supported.

Your program must then operate in accordance with the state of your Informix database.

If you are using a non-ANSI database and you set autocommit to false, *all* transactional SQL must be implemented by using the `Connection.commit()` or `Connection.rollback()` method. You should *never* execute the explicit transaction controls BEGIN WORK, COMMIT WORK, or ROLLBACK WORK on a Statement, because WebLogic jDriver for Informix uses transaction commands internally to simulate an autocommit=false status. *You should always control a transaction using `commit()` and `rollback()` methods in the Connection class*.

For non-ANSI databases without logging, autocommit=false cannot be supported, because transactions are not supported. Consequently, only autocommit=true is supported for use with such databases.

Support for Informix-Specific Features

WebLogic jDriver for Informix includes support for other Informix-specific features that may not be part of the JDBC specification, but that provide additional power for a programmer writing a client application for an Informix database. These features include:

- Retrieving VARCHAR/CHAR Data as Bytes
- Codeset Support
- Using Unicode Streams in a Prepared Statement

They are described in the following sections.

Retrieving VARCHAR/CHAR Data as Bytes

WebLogic jDriver for Informix provides an extension to JDBC for Informix that allows programmers to retrieve VARCHAR and CHAR columns by using the `ResultSet.getBytes(String columnName)` and `ResultSet.getBytes(int columnIndex)` methods. Although this task is outside the scope of the JDBC specification, it was implemented in response to customer requests. No cast of the ResultSet is required to take advantage of this feature.

Codeset Support

As a Java application, WebLogic jDriver for Informix handles character strings as Unicode strings. To exchange character strings with a database that may operate with a different codeset, you must set the `weblogic.codeset` connection property to the proper JDK codeset. 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", "scott");
props.put("password", "tiger");

String connectUrl = "jdbc:weblogic:informix4:myDB@myHost:1493";

Driver myDriver = (Driver)
  Class.forName("weblogic.jdbc.informix4.Driver").newInstance();
  Connection conn =
     myDriver.connect(connectUrl, props);
```

Using Unicode Streams in a Prepared Statement

If you are using the `PreparedStatement.setUnicodeStream` method, you can create either your own InputStream object or a **`weblogic.jdbc.informix4.UnicodeInputStream`** object, using a String value in the constructor. The following sample code shows how to input a Unicode stream into an Informix TEXT column (using the `connectUrl` and `props` objects created earlier):

```
Driver myDriver = (Driver)
  Class.forName("weblogic.jdbc.informix4.Driver").newInstance();
  Connection c =
     myDriver.connect(connectUrl, props);

  PreparedStatement ps =
    c.prepareStatement("insert into dbTEST values (99,?)");

  String s = new String("\u93e1\u68b0\u897f");
  weblogic.jdbc.informix4.UnicodeInputStream uis =
    new weblogic.jdbc.informix4.UnicodeInputStream(s);

  try {
    ps.setUnicodeStream(1,uis,uis.available());
  }
  catch (java.io.IOException ioe) {
    System.out.println("--  IO Exception in setUnicodeStream");
  }
  ps.executeUpdate();
```

To retrieve data from a UnicodeInputStream use `java.io.InputStream`. For example:

```
InputStream uisout = rs.getUnicodeStream(2);
int i=0;
while (true) {
try {
 i = uisout.read();    // read 1 byte at a time from UnicodeStream
}
catch (IOException e) {
  System.out.println("-- IOException reading UnicodeStream");
}
```

For more information, check the full example provided with the WebLogic Server installation, in the `samples/examples/jdbc/informix4` directory.

WebLogic jDriver for Informix Conformance to JDBC

WebLogic jDriver for Informix is a complete implementation of the JDBC specification, except for those features described in the JDBC specification that are either unsupported or unavailable in Informix. Because there is often confusion about the implementation of the DatabaseMetaData interface, we list all its methods in this section. Most of these methods are supported; some are planned for a future release; and some (because of Informix limitations or implementations) will not be supported by WebLogic jDriver for Informix.

The following DatabaseMetaData methods are supported:

```
allProceduresAreCallable()
allTablesAreSelectable()
dataDefinitionCausesTransactionCommit()
dataDefinitionIgnoredInTransactions()
doesMaxRowSizeIncludeBlobs()
getCatalogSeparator()
getCatalogTerm()
getColumns()
getDatabaseProductName()
getDatabaseProductVersion()
getDefaultTransactionIsolation()
getDriverMajorVersion()
getDriverMinorVersion()
getDriverName()
```

```
getDriverVersion()
getExportedKeys()
getExtraNameCharacters()
getIdentifierQuoteString()
getImportedKeys()
getMaxBinaryLiteralLength()
getMaxCatalogNameLength()
getMaxCharLiteralLength()
getMaxColumnNameLength()
getMaxColumnsInGroupBy()
getMaxColumnsInIndex()
getMaxColumnsInOrderBy()
getMaxColumnsInSelect()
getMaxColumnsInTable()
getMaxConnections()
getMaxCursorNameLength()
getMaxIndexLength()
getMaxProcedureNameLength()
getMaxRowSize()
getMaxSchemaNameLength()
getMaxStatementLength()
getMaxStatements()
getMaxTableNameLength()
getMaxTablesInSelect()
getMaxUserNameLength()
getNumericFunctions()
getPrimaryKeys()
getProcedures()
getProcedureTerm()
getSchemas()
getSchemaTerm()
getSearchStringEscape()
getSQLKeywords()
getStringFunctions()
getSystemFunctions()
getTables()
getTableTypes()
getTimeDateFunctions()
getTypeInfo()
getURL()
getUserName()
isCatalogAtStart()
isReadOnly()
nullPlusNonNullIsNull()
nullsAreSortedAtEnd()
nullsAreSortedAtStart()
```

```
nullsAreSortedHigh()
nullsAreSortedLow()
storesLowerCaseIdentifiers()
storesLowerCaseQuotedIdentifiers()
storesMixedCaseIdentifiers()
storesMixedCaseQuotedIdentifiers()
storesUpperCaseIdentifiers()
storesUpperCaseQuotedIdentifiers()
supportsAlterTableWithAddColumn()
supportsAlterTableWithDropColumn()
supportsANSI92EntryLevelSQL()
supportsANSI92FullSQL()
supportsANSI92IntermediateSQL()
supportsCatalogsInDataManipulation()
supportsCatalogsInIndexDefinitions()
supportsCatalogsInPrivilegeDefinitions()
supportsCatalogsInProcedureCalls()
supportsCatalogsInTableDefinitions()
supportsColumnAliasing()
supportsConvert()
supportsCoreSQLGrammar()
supportsCorrelatedSubqueries()
supportsDataDefinitionAndDataManipulationTransactions()
supportsDataManipulationTransactionsOnly()
supportsDifferentTableCorrelationNames()
supportsExpressionsInOrderBy()
supportsExtendedSQLGrammar()
supportsFullOuterJoins()
supportsGroupBy()
supportsGroupByBeyondSelect()
supportsGroupByUnrelated()
supportsIntegrityEnhancementFacility()
supportsLikeEscapeClause()
supportsLimitedOuterJoins()
supportsMinimumSQLGrammar()
supportsMixedCaseIdentifiers()
supportsMixedCaseQuotedIdentifiers()
supportsMultipleResultSets()
supportsMultipleTransactions()
supportsNonNullableColumns()
supportsOpenCursorsAcrossCommit()
supportsOpenCursorsAcrossRollback()
supportsOpenStatementsAcrossCommit()
supportsOpenStatementsAcrossRollback()
supportsOrderByUnrelated()
```

```
supportsOuterJoins()
supportsPositionedDelete()
supportsPositionedUpdate()
supportsSchemasInDataManipulation()
supportsSchemasInIndexDefinitions()
supportsSchemasInPrivilegeDefinitions()
supportsSchemasInProcedureCalls()
supportsSchemasInTableDefinitions()
supportsSelectForUpdate()
supportsStoredProcedures()
supportsSubqueriesInComparisons()
supportsSubqueriesInExists()
supportsSubqueriesInIns()
supportsSubqueriesInQuantifieds()
supportsTableCorrelationNames()
supportsTransactionIsolationLevel()
supportsTransactions()
supportsUnion()
supportsUnionAll()
usesLocalFilePerTable()
usesLocalFiles()
```

Support for the following methods has been implemented and is now being tested:

```
getBestRowIdentifier()
getColumnPrivileges()
getTablePrivileges()
```

Support for the following methods is planned:

```
getIndexInfo()
supportsConvert()
```

The following methods will not be supported:

```
getCatalogs()
getCrossReference()
getProcedureColumns()
getVersionColumns()
```

References

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

Documentation

- Introduction to JDBC in *Programming WebLogic JDBC* at http://e-docs.bea.com/wls/docs60/jdbc/intro.html.

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

- Using connection pools with server-side Java in Programming Tasks in *Programming WebLogic HTTP Servlets* at http://e-docs.bea.com/wls/docs60/servlet/progtasks.html.

- Managing JDBC Connectivity in *Administration Guide* at http://e-docs.bea.com/wls/docs60/adminguide/jdbc.html.

  Describes administrative tasks for configuring JDBC connectivity, such as creating connection pools, datasources, and multipools.

- JavaSoft's JDBC tutorial at `http://java.sun.com/docs/books/tutorial/jdbc/index.html`.

Code Examples

To help you get started, WebLogic jDriver for Informix provides several code examples. You can find them in the `samples/examples/jdbc/informix4` directory of your WebLogic jDriver for Informix installation.