jBASE JDBC Driver

Version 1.0

Contents

DOCUMENTATION CONVENTIONS

This manual uses the following conventions:

Convention	Usage
BOLD	In syntax, bold indicates commands, function names, and options. In text, bold indicates keys to press, function names, menu selections, and MS-DOS commands.
UPPERCASE	In syntax, uppercase indicates JBASE commands, keywords, and options; BASIC statements and functions; and SQL statements and keywords. In text, uppercase also indicates JBASE identifiers such as filenames, account names, schema names, and Windows NT filenames and pathnames.
UPPERCASE *Italic*	In syntax, italic indicates information that you supply. In text, italic also indicates UNIX commands and options, filenames, and pathnames.
Courier	Courier indicates examples of source code and system output.
Courier Bold	**Courier Bold** In examples, courier bold indicates characters that the user types or keys (for example, <Return>).
[]	Brackets enclose optional items. Do not type the brackets unless indicated.
{}	Braces enclose nonoptional items from which you must select at least one. Do not type the braces.
ItemA | itemB	A vertical bar separating items indicates that you can choose only one item. Do not type the vertical bar.
. . .	Three periods indicate that more of the same type of item can optionally follow.

⇒ A right arrow between menu options indicates you should choose each option in sequence. For example, “Choose **File** ⇒ **Exit**” means you should choose **File** from the menu bar, and then choose **Exit**

Syntax definitions and examples are indented for ease in reading.

All punctuation marks included in the syntax—for example, commas, parentheses, or quotation marks—are required unless otherwise indicated.

Syntax lines that do not fit on one line in this manual are continued on subsequent lines. The continuation lines are indented. When entering syntax, type the entire syntax entry, including the continuation lines, on the same input line.

OVERVIEW

What is JDBC?

JDBC™ is a Java™ API for executing SQL statements. JDBC provides this standard API such that database developers and programmers can work with different databases seamlessly. This document is intended to be used with the SQL Query Engine for jBASE document and describes how java developers can use the standard JAVA API to communicate with a jBASE database using SQL. jBASE versions supported begin with 4.1.5. The 1.0 JDBC thin driver for jBASE 4.1.5 is a read-only driver and not meant for UPDATES/INSERTS/DELETES and other DML statements. In addition, this, driver version 1.0 does not support STORED PROCEDURES or FUNCTIONS. The driver is, however, a pure thin driver, meaning that jBASE itself is not necessary on the client. Since jBASE itself is not a relational database, there are some special tasks that need to be completed such that jBASE can act like a relational database. These tasks are explained in this document and the SQL Query Engine for jBASE document.

What is JavaObjex?

Throughout this document, you will see references to JavaObjex. What is it? JavaObjex listens on the database server for JDBC connections as well as other connections that want to do remote database access, retrieval, writing, etc. JavaObjex is also an API for client programs written in Java. More on javaObjex can be found in the Java OBjEX Javadocs.

FILE REQUIREMENTS

To compile a client program, there should be no external dependencies except those additional in the user client program (keep in mind that JDBC is part of the Java core API). To run, the CLASSPATH environment variable must point to the jBASE JDBC driver found at $JBCRELEASEDIR/java/lib/jbasejdbc.jar such that the driver can be loaded.

On the server, the javaObjexServer batch file sets up CLASSPATH dependencies. The user should have to do nothing additional. But for informational purposes, these dependencies are jbase.jar, jdom.jar, and javaobjex.jar. All of these are located in $JBCRELEASEDIR/java/lib and **should not be moved**.

In addition, the user should ensure nothing is in the ext directory of the JDK that is referenced in the javaObjexServer batch file.

CONNECTION

The standard way to connect to a database using a JDBC thin driver is through a URL. A java example is provided below connecting to an Oracle database. The string in bold represents the Oracle jdbc URL. (Please refer to the java.sql package under the JAVA API found here: **http://java.sun.com/j2se/1.4.2/docs/api/**)

```
try
{
Class.forName ("oracle.jdbc.driver.OracleDriver");
}
catch (ClassNotFoundException e)
{
e.printStackTrace ();
}
try
{
Connection conn =
DriverManager.getConnection
("jdbc:oracle:thin:@chdsk-rvincent:1521:ricksid",
"scott", "tiger");
}
catch( SQLException se )
{
}
```

The same connection mechanism applies to the jBASE JDBC driver. The structure of the URL is jdbc:jbasejo:thin:@<<Your computer Name or IP address where jBASE resides>>:3570:<<Your account name>>

Example: jdbc:jbasejo:thin:@mymachine:3570:myaccount

The Java OBjEX server must be running on the database server. The Java OBjEX server is the "listener" for JDBC connections. Later in this document is an explanation of how to start this server. Below is a client connection code example for jBASE.

```
 try
{
              Class.forName("com.jbase.jdbc.driver.JBaseJDBCDriver");
}

catch (ClassNotFoundException e)
{
e.printStackTrace ();
}
try
{
Connection conn =
DriverManager.getConnection
("jdbcjdbc:jbasejo:thin:@testserver:3570:TESTACCOUNT",
"test", "tiger");
}
catch( SQLException se )
{
}
```

The above URL represents a connection to a Java OBjEX server listening on the port 3570, running on a machine named testserver with an OS user named test on that server. The password does not actually refer to test, but TESTACCOUNT instead. This is because jBASE handles security differently. jBASE accounts are explained in the next section.

JBASE ACCOUNTS

jBASE Accounts are similar to SQL schemas, and in jBASE, are the mechanism for security and access to tables. They are a prerequisite to using the JDBC driver and must be set up before attempting to connect.

Setting up a jBASE Account

Set up an account in the jBASE SYSTEM file.)

C:\ >jed SYSTEM MYACCOUNT

File SYSTEM , Record 'MYACCOUNT' I

Command->

0001 D

0002 C:\users\myaccount

0003

0004

0005

0006

0007 I=B]n>(

0008

0009

0010

0011-X "JEDIFILEPATH=C:\jbase4\home"

1)Set JEDIFILENAME_SYSTEM and JEDIFILENAME_MD environment variables on the cmd prompt where javaobjex will be started. Alternatively, you can set these in the javaObjexServer batch file under $JBCRELEASEDIR/bin directory.

Note that Attribute 7 above is the encrypted password and must match that (non-encrypted) password that is passed through the JDBC url. An example of how to set the account password is shown below.

C:\ >jsh

jsh>password

Enter account name to modify password : MYACCOUNT

Enter old password for account MYACCOUNT :

Enter new password :

Re-enter new password :

2)Attribute 11 specifies the PATH to the tables that are accessable through the account. Any tables outside of this path will not be seen by the account MYACCOUNT. In addition, if use of the java.sql.DatabaseMetaData interface is necessary, the PATH to $JBCRELEASEDIR/sql/system must be specified. This is where jBASE SQL System tables are kept. Things such as FOREIGN KEY/PRIMARY KEY and TYPE information are stored here. This is necessary since jBASE is NOT a relational database.

STARTING JAVAOBJEX SERVER

Starting javaObjex server is simple. As long as the PATH environment variable is set to $JBCRELEASEDIR/bin, than the following should work:

> javaOBjEXServer start

TEMENOS javaOBjEX Server

Requested Action.....> START

Check if it's already running.....> NO

Starting.....> OK

Started and ready on <:3570>

For more on javaObjex server please refer to the manual <<where manual is>>

Notes: Ensure that nothing is in the ext directory of the jdk on your machine. (JARS in the ext directory can cause conflicts with javaObjex server). If there is a problem connecting, check that the JDBC driver (jbasejdbc10.jar) is present under $JBCRELEASEDIR/java/lib and that the CLASSPATH is set to this directory for the client program.

CLIENT EXAMPLE PROGRAM

The code for this example can be found under $JBCRELEASEDIR/samples/jdbc directory. The example assumes the following:

1)A setup of the MYACCOUNT as shown earlier. In addition, it assumes that the JEDIFILEPATH for this account points to the $JDBCRELEASEDIR/samples/SQL directory (where the tables are located).

2)javaObjexServer running on a server where the environment variables JEDIFILENAME_SYSTEM and JEDIFILENAME_MD have been set. It also assumes that javaObjexServer is running on port 3570.

3) You have substituted the TODOs in the sample program with your own values. (listing shown below)

Compile the program

>set CLASSPATH=%JBCRELEASEDIR%\java\lib\jbasejdbc10.jar;.

>javac JDBCTestJBase.java

Run the program

C:\data\jbase\src\jbase4\development\samples\jdbc>java JDBCTestJBase

```
Loading JDBC jBASE driver
Connecting to the local database
STALLED, JIM
JAMES, JIM
HUNT, DONNAYA
LAMB, JOHNO
MEERS, null
COOLRIDGE, CLIVE
SUE, JIM
```

```
import java.sql.*;
import java.io.*;


public class JDBCTestJBase
{
public static void main (String args []) throws SQLException,
IOException
{
  System.out.println ("Loading JDBC jBASE driver");
  try
  {
       Class.forName ("com.jbase.jdbc.driver.JBaseJDBCDriver");
  }
  catch (ClassNotFoundException e)
  {
       System.out.println ("Could not load the driver. Is your
classpath correct?");
       e.printStackTrace ();
  }
  System.out.println ("Connecting to the local database");
  Connection conn =
         DriverManager.getConnection
            ("jdbc:jbasejo:thin:@<<TODO put servername
here>>:3570:MYACCOUNT",<<TODO put server OS username here>>, <<TODO put
account password here>>);
  try
  {
       Statement stmt = conn.createStatement();


       ResultSet rset = stmt.executeQuery ("SELECT LASTNAME, FIRSTNAME
first FROM CUSTOMERS");
       while (rset.next())
      {
         // Print the name out
         String lastName =  rset.getString(1);
         String first = rset.getString("FIRST"); // case insensive,
referencing alias first
         System.out.println (lastName + ", " + first);
      }
      stmt.close();
```

```
        conn.close();

    }catch(SQLException se)
    {
        se.printStackTrace();
        System.out.println(".......................");

        String sErrorState = "Err Code: " + se.getErrorCode() +
             "  SQL  State:  "  +  se.getSQLState()  +  "  mesg:  "  +
se.getMessage();

       System.out.println(sErrorState);
       System.out.println(".......................");
       conn.close();
    }

}
```

JDBC LOGGING

The logging configuration file can be found at $JBCRELEASEDIR/config/javaobjex.properties. In this file you will find the lines:

```
#*********************************************************************
# LOGGING PROPERTIES for javaobjex CLIENT (used for jdbc)
#*********************************************************************
javaobjex.log.filename=jdbc.log
javaobjex.log.maxsize=1000000
javaobjex.log.typeconsole=0
javaobjex.log.level=0
```

These lines control the logging of the jdbc client. By default, the logs go to $JBASERELEASEDIR/logs. For the javaObjex server, the following property lines control the logging. Detailed explanations of each of these can be found in the javaobjex.properties file.

```
server.log.filename=javaobjex.log
server.log.maxsize=1000000
server.log.typeconsole=0
server.log.level=0
```

JDBC LIMITATIONS

As mentioned earlier, version 1.0 of the JDBC driver is read only. As well, it is highly recommended that the user read the SQL Query Engine for jBASE document for limitations on the jQL SQL engine itself. Since jBASE is a multi-valued database (hierarchal), interpretation of certain SQL commands becomes obscure. For example, select * from tablename where one of the attributes in this table is multi-valued will split the multivalues into rows in perhaps an undesired manner. This is because tables within tables are allowed in jBASE and the * in the SQL statement is a vague reference when this is the case. Please read this document for detailed explanations. In addition, there are other limitations described below.

Primary/Foreign Keys (Table SYS_TABLE_RELATIONS)

Because jBASE is not a relational database, primary/foreign key relationships must be set up manually if this information is required through the java.sql.DatabaseMetaData interface. The tables that reflect this information are located under $JBCRELEASEDIR/sql/system. The dictionary (SYS_TABLE_RELATIONS]D) directly reflects the Java API for the getExportedKeys function. If this information is not required (it is typically needed for GUI developers), the user has nothing to worry about**. IT IS NOT REQUIRED TO HAVE THIS INFORMATION IN ORDER FOR THE JDBC DRIVER TO WORK AND GENERALLY NOT USED**. The following functions within java.sql.DatabaseMetaData apply to the table SYS_TABLE_RELATIONS:

1) getCrossReference
2) getExportedKeys
3) getImportedKeys
4) getPrimaryKeys

Type Information (Table SYS_TYPE_INFO)

JDBC type information can be found in SYS_TYPE_INFO. **THIS DATA SHOULD NOT BE MODIFIED**. It applies to the function getTypeInfo() in the java.sql.DatabaseMetaData interface. All types map to the VAR type in jBASE.

Features Not Supported

Again, this version of the jDBC driver is “read-only”, so no update, delete, or insert capability is allowed and will throw errors if attempted.

Under the java.sql.DatabaseMetaData interface the following are unsupported:

getProcedures

getSuperTables

getSuperTypes

getVersionColumns

getIndexInfo

getUDTs

getAttributes

getProcedureColumns

All of these will throw SQLExceptions of “Unsupported Feature”.

SQL Locators

UDT’s (this includes use of SQLData, SQLInput, SQLOutput) and the following interfaces are not currently supported:

java.sql.Ref

java.sql.Array

java.sql.Blob

java.sql.Struct

Any calls to the ResultSet getters or setters with a reference to any of the above will throw a SQLException. As a side note, jBASE is different that Oracle and other DMBSs in that there is no limit to the size of data stored in an attribute (except for the max size of the file itself). Therefore, all data can be retrieved up to the limit of java.lang.String’s maximum size via a call to getString() on the ResultSet object.

Java.sql.Clob, however, is supported, but it must be represented in a dictionary item. An example follows.

```
File CUSTOMERS2]D , Record 'PETS'
Command->
0001 A
0002 12
0003 PETS
0004
0005
0006
0007
0008
0009 L
```

```
0010 20
0011
0012
0013
0014
0015
0016
0017
0018
0019
0020 JBASE_EDICT_START
0021 183
0022
0023
0024
0025
0026
0027
0028
0029
0030
0031
0032
0033
0034
0035
0036 JBASE_EDICT_END
```

Here, the value 183 in attribute 21 represents the data type CLOB. By assigning this, the user specifies to the jdbc driver that this field should not be materialized to the client until the appropriate method on the java.sql.Clob interface is called. This can save considerable time if large amounts of data are stored in this field as the server does not have to bring all of this data to the client when the ResultSet.next() method is called.

TIME/DATE

1) jBASE does not store time or timestamp information along with dates. If time information is required with a particular date, this information must be stored separately.

2) If any of the setDate, setTime, or setTimeStamp functions are used in the java.sql.PreparedStatement interface, there must be no Input or Output conversion set on the dictionary attribute. If Input or Output conversions are necessary, use setString() instead. A dictionary item with a blank attribute7 (InputConversion) and blank attribute 8(OutputConversion) is shown below:

```
File CUSTOMERS]D , Record 'BIRTHDATE'
Command->
0001 A
0002 15
0003 BIRTHDATE
0004
0005
0006
0007
0008
0009 R
0010 24
```

JDBC FAQ

When I try to start JavaObjex server I get an error that says that the jdk\jre\bin\java can't be found.

The JavaObjex server batch file relies on the environment variable JREDIR. Make sure this is pointing to a valid JDK or JRE. If you chose to install a JDK when you installed jBASE, it should be under $JBCRELEASEDIR/jdk.

When I start JavaObjex server the JEDIFILEPATH is set, but when I use JDBC it doesn't find the tables in this path.

The JEDIFILEPATH environment variable of the jBASE account used in the JDBC url overrides any JEDIFILEPATH set for the javaObjex server.

I get a ClassNotFoundException error similar to the below. What's wrong?

```
java.lang.ClassNotFoundException: com.jbase.jdbc.driver.JBaseJDBCDriver
        at java.net.URLClassLoader$1.run(Unknown Source)
        at java.security.AccessController.doPrivileged(Native Method)
        at java.net.URLClassLoader.findClass(Unknown Source)
        at java.lang.ClassLoader.loadClass(Unknown Source)
        at sun.misc.Launcher$AppClassLoader.loadClass(Unknown Source)
        at java.lang.ClassLoader.loadClass(Unknown Source)
        at java.lang.ClassLoader.loadClassInternal(Unknown Source)
        at java.lang.Class.forName0(Native Method)
        at java.lang.Class.forName(Unknown Source)
        at JDBCTestJBase2.main(JDBCTestJBase2.java:14)
Connecting to the local database
Exception in thread "main" java.sql.SQLException: No suitable driver
        at java.sql.DriverManager.getConnection(Unknown Source)
        at java.sql.DriverManager.getConnection(Unknown Source)
        at JDBCTestJBase2.main(JDBCTestJBase2.java:23)
```

Your CLASSPATH is not set to the JDBC driver. Please set the CLASSPATH environment variable to$JBCRELEASEDIR/java/lib/jbasejdbc10.jar.

I get a SQLException that says "Unable to logTo account". What's wrong?

The JEDIFILENAME_SYSTEM environment variable is not set where you ran JavaObjex server. Set this environment variable to the appropriate SYSTEM table where the account was created.

My SQL query doesn't work. Why?

If there is a SQLException and you don't know why or you are not convinced your query isn't working as you expect, try running it from jsh with the SQLSELECT command. An example is shown below.

jsh -->SQLSELECT a.FIRSTNAME, a.LASTNAME FROM CUSTOMERS a

SUMMARY

Faster Retrieval

Allowing users to use the JDBC API to access the jBASE database empowers the user to retrieve data at significantly improved performance than using other jQL verbs through the JavaObjex API or JBC programs. For one, there are far fewer network hits as data is retrieved in “clumps” of rows specifed by the fetchCount attribute in the java.sql.Statement implementation (the user can control this by calling setFetchSize()). As well, SQL is much more performant than the jQL verb SELECT.

External Tools

There are a host of tools that use JDBC thin drivers as their mechanism to communicate seamlessly with different databases. One such free tool on the internet is DBVisualizer (found at http://www.minq.se/products/dbvis).

Comment Sheet

Please give page number and description for any errors found:

Page	Error

Please use the box below to describe any material you think is missing; describe any material which is not easily understood; enter any suggestions for improvement; provide any specific examples of how you use your system which you think would be useful to readers of this manual. Continue on a separate sheet if necessary.

Copy and paste this page to a word document and include your name address and telephone number. Email to mailto:documentation@jbase.com