HP Driver for JDBC User's Manual

HP 3000 MPE/iX Computer Systems

Edition 1

Manufacturing Part Number: 36216-90217
E0399

U.S.A. March 1999

Notice

The information contained in this document is subject to change without notice.

Hewlett-Packard makes no warranty of any kind with regard to this material, including, but not limited to, the implied warranties of merchantability or fitness for a particular purpose. Hewlett-Packard shall not be liable for errors contained herein or for direct, indirect, special, incidental or consequential damages in connection with the furnishing or use of this material.

Hewlett-Packard assumes no responsibility for the use or reliability of its software on equipment that is not furnished by Hewlett-Packard.

Restricted Rights Legend

Use, duplication, or disclosure by the U.S. Government is subject to restrictions as set forth in subparagraph (c) (1) (ii) of the Rights in Technical Data and Computer Software clause at DFARS 252.227-7013. Rights for non-DOD U.S. Government Departments and Agencies are as set forth in FAR 52.227-19 (c) (1,2).

Acknowledgments

Java is a U.S. trademark of Sun Microsystems, Inc.

Hewlett-Packard Company
3000 Hanover Street
Palo Alto, CA 94304 U.S.A.

Contents

Contents

Figures

Figures

Tables

Tables

Preface

Java Database Connectivity (JDBC) is a standard Application Programming Interface (API) for database access from Java. HP Driver for JDBC is an implementation of the standard JDBC API. It consists of Hewlett Paclard's fully-java Driver for JDBC, network protocol and database server interface components for concurrent access to IMAGE/SQL and ALLBASE/SQL databases

This manual provides information about installing, configuring and using HP Driver and server components for JDBC.

- Chapter 1 , "Introduction," presents a general summary of Java Database Connectivity (JDBC) and HP Driver for JDBC components.
- Chapter 2 , "Requirements," provides information about Java, MPE, HP-UX, ALLBASE/SQL and IMAGE/SQL versions required to use HP Driver for JDBC.
- Chapter 3 , "Installation," describes the installation steps.
- Chapter 4 , "Using the HP Driver for JDBC," shows how to use HP Driver for JDBC from Java programs.
- Chapter 5 , "ALLBASE/SQL Specifics," provides information specific to ALLBASE/SQL and HP Driver for JDBC.
- Chapter 6 , "Troubleshooting," discusses common problems encountered and their solutions.
- Appendix A , "Data Types," describes ALLBASE/SQL & JDBC data types.
- Appendix B , "JDBC Monitor," contains information about JDBC Monitor for HP-UX and MPE/iX.
- Appendix C , "HP JDBC Server," presents a detailed description of HP JDBC Server.
- Appendix D , "Simple Client Source Code," contains the Java source code for a demonstration application.
- Appendix E , "HP ALLBASE/SQL JDBC File Lists," identifies all the files that are part of this product.

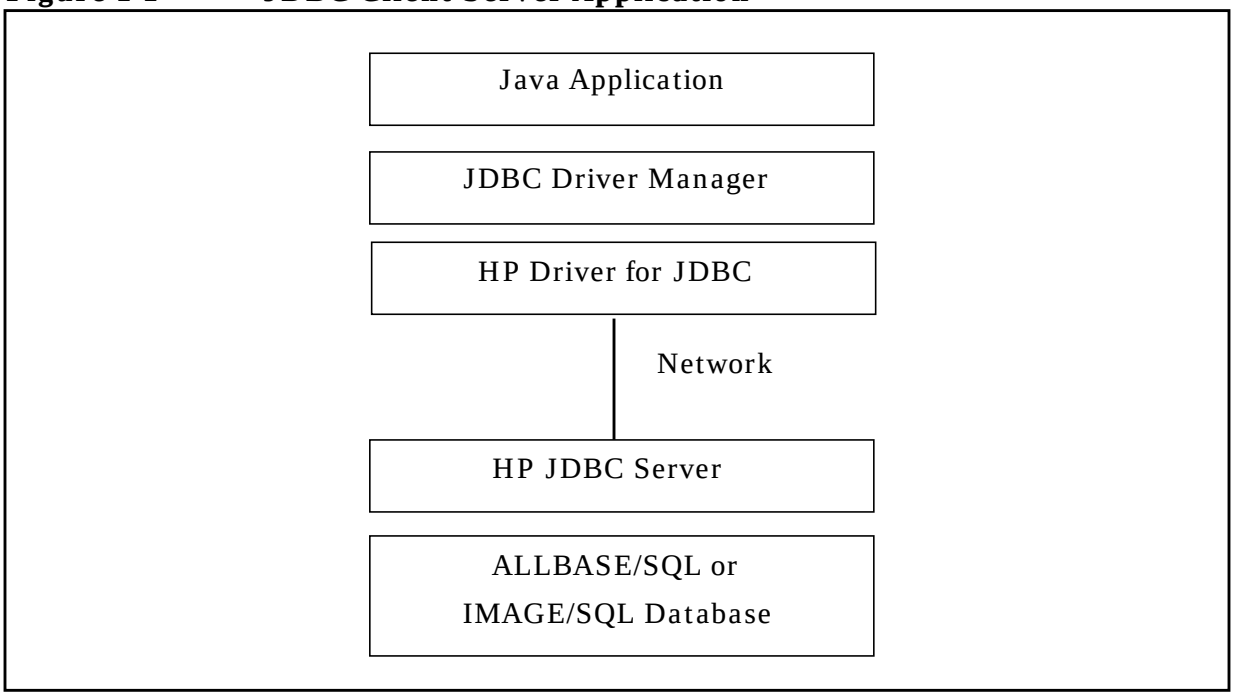
1 Introduction

Java Database Connectivity (JDBC) is a Java API that enables development of Java applications and applets with a wide range of relational databases. It consists of a set of classes and interfaces written in Java. JDBC allows developers to write database applications using a standard, pure Java API.

JDBC

A typical JDBC Java application consists of a Java application or applet, the JDBC Driver Manager, a vendor specific JDBC driver, and a database. The JDBC Driver Manager is provided with the Java SDK and its primary function is to load and register the vendor-specific JDBC driver with the Java applications and then get out of the way. The following diagram shows the various components of a client-server JDBC application.

Figure 1-1 JDBC Client-Server Application



The HP Driver for JDBC is a vendor specific JDBC Driver that allows Java applications to connect to ALLBASE/SQL or IMAGE/SQL databases on MPE/iX or ALLBASE/SQL on HP-UX. The HP Driver for JDBC is a Type 3 (Network-Protocol) driver, meaning the driver translates the JDBC API into a DBMS-independent protocol on the client-side, and then translates to the ALLBASE/SQL protocol on the server. The driver components on the client-side are written in 100% Pure Java, which provides the complete compatibility with all Java Virtual Machines on all platforms.

The document will only describe the specifics necessary to use the HP Driver for JDBC, and not JDBC and JDBC drivers in general. For a comprehensive discussion of JDBC, refer to:

JDBC Database Access with Java: A Tutorial and Annotated Reference, Graham Hamilton, Rick Cattell, and Maydene Fisher, Addison-Wesley 1997.

HP JDBC Components

There are three components supplied with the HP JDBC product, the JDBC Driver, the JDBC Monitor, and the JDBC Server. The user is responsible for writing a Java application or applet that uses JDBC on the client, as well as providing the ALLBASE/SQL or IMAGE/SQL database on the server.

The **HP Driver for JDBC** is a set of Java classes that implement the `java.sql.*` interfaces and provide an implementation of a JDBC driver that can communicate with an ALLBASE/SQL or IMAGE/SQL database. The HP Driver for JDBC typically will reside on the client side of the user application. It provides the translation from the Java language and the JDBC API to the HP proprietary network protocol.

The **JDBC Monitor** is a component that is installed on the JDBC server host that manages all client JDBC Driver connections to the server host. It is typically started as a daemon when the server machine is booted. All JDBC client connections are made through the JDBC Monitor. The monitor performs validation of the userid and password that are passed in the client connection request and spawns JDBC Server processes to serve each of the client connections. Once the server process is spawned, the monitor returns to wait for the next client connection.

The **JDBC Server** is the server process that is spawned by the JDBC Monitor to service a client connection. It handles the translation from the HP proprietary network protocol to the ALLBASE/SQL calls. There is at least one JDBC Server process for each client connection to the server host. More than one JDBC Server process may be used to handle multiple client statements using the same connection. This component also handles the translation from JDBC SQL to ALLBASE SQL and conversion of the database data from ALLBASE/SQL format to JDBC format.

Both the JDBC Monitor and the JDBC Server must be installed on the same host where the ALLBASE/SQL or IMAGE/SQL databases reside.

2 Requirements

HP Driver for JDBC can be used to access ALLBASE/SQL or IMAGE/SQL from Java Clients running under various operating systems. This chapter specifies Java Client, Database Server and Operating System requirements for successfully installing and running HP Driver for JDBC.

Java Requirements

The HP JDBC Client components (the JDBC driver itself) requires a Sun-compliant JDK version 1.1 and above, which includes a JDBC version 1.2. Install the JDK from Sun or from your platform vendor. Individual platform vendors may have their own requirements for the platform host. For example, Java only runs on HP-UX 10.20 and MPE/iX 6.0 and above; HP-UX 9.x and MPE/iX 5.5 are not supported.

Java and JDK components only need to be installed on the client platform. The JDBC server platforms use native components and must be installed on the same host where the ALLBASE/SQL or IMAGE/SQL databases reside.

HP-UX Server Requirements

The HP JDBC Server components require HP-UX version 10.20 or greater. HP-UX components are required only if your JDBC Server platform is the HP-UX operating system.

MPE/iX Server Requirements

The HP JDBC Server components require MPE/iX version 6.0 or greater. MPE/iX components are only required if your JDBC Server platform is the MPE/iX operating system.

ALLBASE/SQL or IMAGE/SQL Requirements

The HP JDBC Server components require an ALLBASE/SQL G3.01 or IMAGE/SQL G3.00 or greater.

3 Installation

This chapter takes you through the step-by-step process of installing and configuring HP Driver for JDBC and its server components.

HP-UX Server Components

The installation of the JDBC Server components must be done by a system administrator who has “root” capability on the server host where the database resides. A temporary directory, `/tmp/jdbc`, is used to stage the installation scripts, and product tar file.

The server components are distributed as a UNIX tape archive (TAR) file, `hpjdbc_XXX.tar`, where `XXX` represents the release number of the product.

Once the tar file is copied into a staging directory, `/tmp/jdbc`, extract the installation shell script from the tar file:

```
$ tar xf hpjdbc_XXX.tar install.sh
```

Run the installation script `install.sh` to perform the actual installation. The installation script will create the necessary directories, `/opt/allbase/jdbc`, extract the server files, and the client archive files packaged in three formats, and set up the JDBC monitor startup scripts. Once the installation is complete, you may delete the tar file and the installation script.

For a complete list of the files that are installed, please refer to Appendix E , “HP ALLBASE/SQL JDBC File Lists.”

If existing files are found during the installation, they are backed up by renaming them with a “`.1`” (dot one) suffix (any existing file with the “`.1`” suffix is overwritten). This allows the installer to manually revert to the previous installation, if desired.

MPE/iX Server Components

The installation of the HP JDBC product must be done by a system administrator on the server host where the database resides..

The components of HP JDBC are distributed as an archive file, `HFSFILES.JDBC.SYS`. Stream the JDBC installation job `I00IJDBC`, to perform the actual installation:

```
:stream I00IJDBC.JDBC.SYS
```

The installation script will extract the server files and client archive files and set up the JDBC monitor startup scripts.

For a complete list of the files that are installed, please refer to Appendix E , “HP ALLBASE/SQL JDBC File Lists.”

Java Client Components

The HP JDBC product must first be installed on the server host before the client can be installed. This is because the client files are bundled with the server product.

The HP Driver for JDBC components consist of the Driver Java class files and a sample JDBC client source file. These are the only components required on the client-side. The driver components are packaged in three formats (the same files contents are in each package), UNIX tape archive (TAR), Java archive (JAR), and Windows ZIP (ZIP).

Setting up HP Driver for JDBC client involves:

- Downloading the appropriate HP JDBC archive file to a temporary directory.
- Extracting the HP JDBC archive file onto your client platform.

Downloading the HP JDBC Archive File

On an HP-UX server the files are located in `/opt/allbase/jdbc/driver` with the following names:

`driver.jar` Driver files in Java archive package.

`driver.tar` Driver files in UNIX tape archive package.

`driver.zip` Driver files in a Windows ZIP package.

On an MPE/iX server the files are located in `@.JDBC.SYS` with the following names:

`DRIVERJ` Driver files in Java archive package.

`DRIVERT` Driver files in UNIX tape archive package.

`DRIVERZ` Driver files in a Windows ZIP package.

Using FTP.ARPA.SYS

For the **HP 3000** server, use `MANAGER.SYS,JDBC` as the username:

From your client platform:

ftp *mpe-server*

```
>binary
>get DRIVERT /* or DRIVERJ or DRIVERZ */
```

For the **HP 9000** server, use any login as the username:

From your client platform:

ftp *hpux-server*

```
>binary
>get /opt/allbase/jdbc/driver.tar
/* or driver.jar or driver.zip */
```

Using Reflection Software

See the software description on how to transfer the appropriate file (transfer the file in binary mode).

Extracting the HP Driver for JDBC Class Files

The HP Driver for JDBC class files must be installed in your Java class path so that the Java compiler and the Java class loader can find them.

Example on the Win32 Platform

```
CLASSPATH=C:\JDK1.1.4\LIB;.
```

You would then install the HP Driver for JDBC class files in the directory:

```
C:\JDK1.1.4\LIB\
```

Each of the package formats include the directory information, so they should be copied into the class path directory.

Once the package is copied onto the client machine, extract the contents of the package by using the appropriate command or application for that package format.

For example:

- `tar` command to unpack the TAR package: `$tar xvf driver.tar`
- `jar` command to unpack the JAR package: `$jar xvf driver.jar`

UNZIP or WINZIP to unpack the ZIP package: Follow the unzip steps provided in the UNZIP/WINZIP package.

For the HP 3000 system, a job stream, `I01IJDBC.JDBC.SYS`, is provided to install the HP Driver for JDBC client under the `/usr/local/java/latest/lib/` directory of the MPE/iX POSIX domain.

Example on HP 3000

```
CLASSPATH=/usr/local/java/latest/lib:.
```

In order to use the HP Driver for JDBC client on the MPE/iX system, you need to have the JAVA/iX installed. JAVA/iX is shipped with MPE/iX starting in Release 6.0.

The following sub-directories for the Driver class files should be automatically created:

```
com/hp/jdbc/allbase
```

```
com/hp/jdbc/allbase/samples
```

For a complete list of the files that are installed, please refer to Appendix E , “HP ALLBASE/SQL JDBC File Lists.”

Configuring HP JDBC Server

The HP JDBC Server components behave the same on both the HP-UX and MPE/iX platforms. Thus both the configuration file and log file are the same (except for filenames).

The following example shows the standard HP JDBC default server configuration file, servcfg. This is a text file that resides on the server host in the same directory as the HP JDBC Server executable files and can be used to alter some of the server's behavior.

```
LOGFILE /opt/allbase/jdbc/logs/servlog
TIMEOUT 7200


#LOGGING ERROR
#LOGGING CONNECTION
#LOGGING INFO
#LOGGING WARNING
#LOGGING FATAL
#LOGGING IN
#LOGGING OUT
#LOGGING INHEX
#LOGGING OUTHEX
#LOGGING DEBUG
#LOGGING TIMESTAMP
LOGGING NONE
# If LOGGING NONE is not commented out, it must be the
# last line for it to have the desired effect.
```

The first configuration option LOGFILE refers to the path and filename for the HP JDBC Server log file. On HP-UX this is normally set to /opt/allbase/jdbc/logs/servlog. On MPE/iX this is normally set to SERVLOG.JDBC.SYS.

The second configuration option TIMEOUT specifies the number of seconds that the HP JDBC Server can remain idle before terminating.

The next set of lines are the LOGGING levels used to determine the type of information to be logged to the server log file. The meanings of the various logging levels are discussed in the Troubleshooting Section, under "Server Logging" of the *HP Driver for JDBC User's Manual*.

The server configuration file is read in anew each time a new server process is started. Thus once the file is edited and saved, the changes will take place beginning with the next server process.

If logging is turned on in this file, it will be in effect for all subsequent server processes.

For a complete listings of JDBC Server log file, please refer to Appendix C , “HP JDBC Server.”

Starting and Stopping HP JDBC Monitor

HP JDBC Monitor is the server-side counterpart of HP Driver for JDBC, present on the client-side. For a JDBC application to connect to a database, the JDBC Monitor must be running on the database server. JDBC Monitor can be started manually or automatically at system startup by including the JDBC Monitor start commands in the system startup scripts.

HP-UX Monitor Startup and Shutdown

The startup and shutdown of the HP-UX JDBC Monitor is normally controlled by the system startup and shutdown scripts. Thus as long as the machine is up and running, the JDBC Monitor will also be up.

The only time the Monitor should ever be brought down is to install a newer version. In this case, use the `monctrl` command to kill the Monitor process. Do not use the UNIX `kill` command, as this could render the server unable to start a new Monitor process. The `monctrl` command to shut down the monitor is:

`monctrl kill [`*`portnumber`*`]`

The *`portnumber`* parameter is necessary only if the monitor you wish to shut down is not running on the default port number 31700.

To restart the monitor after it has been accidentally terminated or was shut down, use the `monctrl` command to start it up:

`monctrl start [`*`portnumber`*`]`

Again, the *`portnumber`* parameter is necessary only if you wish to start the monitor on a port number other than the default port number 31700.

You cannot start a monitor on the same port number as a currently running monitor. You also can not re-start a monitor on its original port number until all child processes that were spawned by the previous monitor process are terminated, and the port released. If the monitor is intentionally terminated or accidentally terminates, all of its child processes must be terminated before it can be restarted. For this reason, the monitor *must not* be terminated by using the UNIX `kill` command. Always use the `monctrl` tool to kill the monitor. The tool will search out all the child processes and kill them first, before killing the monitor.

MPE/iX Monitor Startup and Shutdown

The startup and shutdown of the MPE/iX HP JDBC Monitor is normally done by the startup and shutdown stream jobs, JSTRTMON and JSTOPMON, which are normally included in MPE/iX system startup and shutdown scripts. Thus as long as the machine is up and running, the JDBC Monitor will also be up.

The only time the monitor should ever be brought down is to install a newer version. In this case, use the JSTOPMON stream job or ABORTJOB command to kill the monitor process. The ABORTJOB command to shut down the monitor is:

```
: ABORTJOB #JXX
```

where XX is the job number.

To restart the monitor after it was accidentally terminated or shut down, use the JSTRTMON stream job to start it up.

```
: STREAM JSTRTMON.JDBC.SYS
```

On the MPE/iX platform, each client connection is serviced by a separate process that is in the same session as the monitor. Thus, if the monitor is terminated or dies, the existing server process already serving the client requests will also die. This could result in client applications experiencing dropped connections.

For a complete listings of JDBC Monitor log file, please refer to Appendix B , “JDBC Monitor.”

4 Using the HP Driver for JDBC

This chapter describes how to use HP Driver for JDBC from Java Programs. Information about setting up environment variables, compiling Java programs, loading the HP Driver for JDBC, and connecting to a database is provided.

Java Class Path

Before the HP Driver for JDBC can be used, the Java class path must be set to include the HP Driver for JDBC class files. Without this setting, both the Java compiler and the Java Run Time Environment will not be able to locate the HP Driver for JDBC. Please refer to your *JDK* documentation for more explicit details as to how to set the Java class path. In general, the Java class path must include the directory in which the HP Driver for JDBC class files have been installed.

If the HP Driver for JDBC class files are installed in:

```
/opt/java/lib/com/hp/jdbc/allbase
```

then the Java class path must include the directory:

```
/opt/java/lib
```

In most cases, this would mean that the CLASSPATH environment variable must be set to something resembling:

Example on the HP 9000

```
CLASSPATH=/opt/java/lib:.
```

Sample JDBC Client

The HP Driver for JDBC comes with the source code to two JDBC applications, a sample client and a simple client. Both can be used to test the installation of JDBC components on the client and server.

The simple client is called `SimpleClient` and is a bare-bones text-based application that makes a connection to an ALLBASE/SQL or IMAGE/SQL database, and allows the user to send SQL statements and retrieve the results. There are no frills in the application, so as to make the code as simple as possible. This client should mainly be used for educational purposes. The rest of this discussion will focus on the more robust application.

The sample client is called `SampleClient` and is a more robust version of the simple client. It shows how to use a dialog box to obtain user input, and also better formats the result set output.

The first step using the sample client is to build it by compiling the Java source code. Normally, you would do this by using the Java compiler command (the command you use may be different, depending on your JDK):

```
javac SampleClient.java
```

If this step fails, then you most likely do not have the HP Driver for JDBC class files in your class path. Set your CLASSPATH environment variable or use the `-classpath` Java compiler option to include the directory in which the driver class files are located. For more information, refer to your *JDK* documentation on including other class files.

Once the Java compiler has compiled the source file, the following Java class file will be created:

```
SampleClient.class
```

To run the sample client, follow the instructions provided with your Java SDK for your client platform. Make sure that your Java CLASSPATH includes the directory in which the HP Driver for JDBC class files have been installed, so that the Java Virtual Machine is able to load them. For most platforms, the command to run the sample client is:

```
java SampleClient [-w] [-t]
```

Follow the prompts given by the application to connect to your database and to execute SQL statements.

The sample client can be used as a starting point for developing your own applications or for troubleshooting connection problems with the database.

The `-w` command line switch will cause the sample client to display a Java dialog box to accept user input of the connection information. Normally the sample client uses text-mode input, which displays the user password on the screen.

The `-t` command line switch will cause the application to turn on client-side tracing and prompt the user for the trace flags and the trace output filename.

Loading the HP Driver for JDBC

The Java method that is used to load all JDBC drivers is the `class.forName` method. To load the HP Driver for JDBC, the code is:

```
class.forName("com.hp.jdbc.allbase.JdbcDriver");
```

This loads the HP Driver for JDBC and registers it with the JDBC Driver Manager. Once a driver has been loaded and registered with the Driver Manager, it is ready to be used to connect to a database.

Connection Parameters

The following information is required for all HP JDBC connections to an ALLBASE/SQL or IMAGE/SQL database:

Host name — The name of the server host on which the ALLBASE/SQL or IMAGE/SQL database and JDBC Monitor is running. This can either be the name of the server host, or the IP address. For example, `'myserv.mycomp.com' or '20.45.12.122'`.

Port number — The port number on which the JDBC Monitor is listening. This is specified when the monitor is started on the server host in the startup command or script. The default port number for use by the JDBC Monitor is 31700. If a port number is not specified, this default value is used.

User Name — A valid user name for the server host. The JDBC Monitor performs user validation on the server host for all client connections. For the HP-UX server platform, the user name is the same as the HP-UX user id. For the MPE/iX server platform, the user name can be composed of a user, group, account, and session id. The user name is masked by the JDBC Driver before it is transferred over the network to prevent casual detection.

User Password — The password(s) that match the provided user name. For the HP-UX server platform this is the password for the provided HP-UX user id. For the MPE/iX platform, this is composed of the group and account passwords for the group and account provided as the user name. The user password is masked by the JDBC Driver before it is transferred over the network to prevent casual detection.

Database Name — This is the name of the ALLBASE/SQL or IMAGE/SQL database file on the server. On HP-UX, this is the fully qualified path and filename of the database file. On MPE/iX, this is either the fully qualified three-part filename of the database, or the database name if a `user.account,group` name was specified.

NOTE Database name for MPE/iX usually includes the location of the DBE. If the username does not include the same logon group, the full location must be provided.

These values must be validated before a connection is made to the JDBC Server. The sample client provided with the HP Driver for JDBC can be used to test the correctness of these values.

Once these values have been determined, the JDBC Driver connection URL can be synthesized. There are two primary JDBC Driver class methods that can be used to connect to the database. The first one requires that all connection parameters be provided in a Java String URL, including the user name and password. However, this information can be logged on the client, thus recording the user name and password. Remember that the user name and passwords can be used to log on to the server host.

The second connection method also takes a Java String URL, but allows for the user name and password to be specified as separate arguments, which are not logged. This would be the more secure method of connecting to the database.

The choice of method is entirely up to the application developer. The HP Driver for JDBC supports both methods.

URL Syntax (Including User Name and Password)

The first connection method specifies all connection parameters in a URL string including the user name and password.

```
java.sql.DriverManager.getConnection(url)
```

where:

```
String url =
      "jdbc:allbase://host[:port]/database?UID=uid&PWD=pwd"
      "[&TRACE=trace]";
```

host	Name or IP address of the sever host.
port	Optional port number on which the JDBC Monitor is listening. If not specified, the default part number 31700 is used.
database	ALLBASE/SQL or IMAGE/SQL database name.
uid	Server host userid that is authorized to access the database.
pwd	Server host password that matches the user id provided above.
trace	Optional trace values separated by the vertical bar "|" character. For more information on tracing, see the Troubleshooting section.

NOTE This connection method may result in the user name and password being logged in various places, as URLs are commonly logged on various web servers and proxy servers. The URL is also logged if tracing is turned on. For this reason, the second connection method is preferred.

URL Syntax (Without User Name and Password)

The second connection method specifies the user id and password as method arguments, so this information is not present in the URL.

```
java.sql.DriverManager.getConnection(url, uid, pwd)
```

where:

```
String url =
      "jdbc:allbase://host[:port]/database[?TRACE=trace]";
String uid = "uid";
String pwd = "pwd";
```

host	Name or IP address of the sever host
port	Optional port number on which the JDBC Monitor is listening. If not specified, the default part number 31700 is used.
database	ALLBASE/SQL or IMAGE/SQL database name.
uid	Server host userid that is authorized to access the database.
pwd	Server host password that matches the user id provided above.
trace	Optional trace values separated by the vertical bar "|" character. For more information on tracing, see the Troubleshooting section.

HP-UX Userid and Password

For the JDBC Monitor on HP-UX, the userid and password in the client URL connection string must correspond to a valid HP-UX user name and password for the server host. However, the user accounts may be configured to not allow shell logons, to prevent someone from using the user name and password to actually log on to the server host.

For example, a user with a HP-UX user name of "user1" and password "pass1" would use the method call:

```
java.sql.DriverManager.getConnection(
      "jdbc:allbase://host/user1/mydatabase",
      "user1",
      "pass1");
```

MPE/iX Userid and Password

The MPE/iX operating system supports user names, account names, group names, and session names. There are also user passwords, account passwords, and group passwords. Various combinations of these names and passwords are supported by the MPE/iX JDBC Monitor for user validation. The format for the MPE/iX userid is:

```
[sessionname,]username.accountname[,groupname]
```

The group and session names are optional. The format for the MPE/iX password is:

```
userpass.accountpass[.grouppass]
```

If a group name was specified in the userid and there is a group password, the group password must also be specified in the password parameter.

In a simple example, a user with an MPE/iX user name of "user1" and an account name of "acct1", with the respective passwords of "upass1" and "apass1" would use the method call:

```
java.sql.DriverManager.getConnection(
      "jdbc:allbase://host/dbname",
      "user1.acct1",
      "upass1.apass1");
```

In a more complicated example, a user with an MPE/iX user name of "user1", account name of "acct1", group name of "group1", session name of "sess1" with the respective passwords of "upass1", "apass1", and "gpass1" would use the method call:

```
java.sql.DriverManager.getConnection(
      "jdbc:allbase://host/dbname",
      "sess1,user1.acct1,group1",
      "upass1.apass1.gpass1");
```

5 ALLBASE/SQL Specifics

This chapter describes details specific to HP Driver for JDBC and ALLBASE/SQL. Data type mapping, data type conversions, unsupported data types, unsupported SQL statements and support for stored procedures are discussed.

ALLBASE/SQL to JDBC Data Type Mapping

Table 5-1 shows what the HP Driver for JDBC will report as the JDBC data type for each ALLBASE/SQL data type. These are the `java.sql.Types` values that will be returned from the `java.sql.ResultSetMetaData.getColumnType` method. An "X" in the column indicates the data type mapping.

A description of each ALLBASE/SQL and JDBC data type can be found in Appendix A , "Data Types."

Table 5-1 Data Type Mapping

JDBC DATA TYPES (vertical) / **ALLBASE/SQL DATA TYPES (horizontal)**	TINYINT	SMALLINT	INTEGER	BIGINT	REAL	FLOAT	DOUBLE	DECIMAL	NUMERIC	BIT	CHAR	VARCHAR	LONGVARCHAR	BINARY	VARBINARY	LONGVARBINARY	DATE	TIME	TIMESTAMP
SMALLINT (16-bits)		X																	
INTEGER (32-bits)			X																
REAL					X														
FLOAT(1...24)					X														
FLOAT(25...53)							X												
DOUBLE PRECISION							X												
DECIMAL								X											
NUMERIC									X										
CHAR											X								
VARCHAR												X							
DATE																	X		
TIME																		X	
DATETIME																			X
INTERVAL											X								

ALLBASE/SQL to JDBC Data Type Conversions

Table 5-2 shows the supported data type conversions between ALLBASE/SQL and JDBC. For conversions from JDBC to Java, please refer to a JDBC book or the Java JDBC documentation. Those conversions are generic to all JDBC Drivers.

A description of each ALLBASE/SQL and JDBC data type can be found in Appendix A , “Data Types.”

Suggested conversions are denoted by a capital letter “X” in the conversion grid. Conversions which are supported, but which may result in a loss of precision, overflow, or rounding, are denoted by a lowercase letter “x” in the conversion grid.

Table 5-2 Data Type Conversions

JDBC DATA TYPES (vertical) / **ALLBASE/SQL DATA TYPES (horizontal)**	TINYINT	SMALLINT	INTEGER	BIGINT	REAL	FLOAT	DOUBLE	DECIMAL	NUMERIC	BIT	CHAR	VARCHAR	LONGVARCHAR	BINARY	VARBINARY	LONGVARBINARY	DATE	TIME	TIMESTAMP
SMALLINT (16-bits)	x	X	X	X	X	X	X	X	X	x	X	X	X	x	x	x			
INTEGER (32-bits)	x	x	X	X	x	x	x	X	X	x	X	X	X	x	x	x			
REAL	x	x	x	x	X	X	X	X	X	X	X	X	X	x	x	x			
FLOAT(1...24)	x	x	x	x	x	x	X	X	X	X	X	X	X	x	x	x			
FLOAT(25...53)	x	x	x	x	x	x	X	X	X	X	x	x	x	x	x	x			
DOUBLE PRECISION	x	x	x	x	x	x	X	X	X	x	X	X	X	x	x	x			
DECIMAL	x	x	x	x	x	x	x	X	X	x	X	X	X	x	x	x			
NUMERIC	x	x	x	x	x	x	x	X	X	x	X	X	X	x	x	x			
CHAR	x	x	x	x	x	x	x	x	x	x	X	X	X	x	x	x			
VARCHAR	x	x	x	x	x	x	x	x	x	x	X	X	X	x	x	x			
DATE											X	X	X	x	x	x	X		x
TIME											X	X	X	x	x	x		X	x
DATETIME											X	X	X	x	x	x	x	x	X
INTERVAL											X	X	X	x	x	x			

The conversion of any ALLBASE/SQL data type to the `java.sql.Types.BIT` data type is such that only the value of zero, in either numeric or character format, will be converted to the `java.sql.Types.BIT` value of 0. All other values will be converted to the `java.sql.Types.BIT` value of 1. Thus only the integer value 0, the floating-point value 0.0, the decimal value 0 (not 0.0), and the character string “0” will be converted to a bit value of 0. Everything else is converted to a bit value of 1.

The conversion of an ALLBASE/SQL data type to a JDBC data type that has a smaller degree of precision (such as conversion from ALLBASE/SQL `INTEGER` to `java.sql.Types.SMALLINT`) will follow the Java VM rules of casting one data type to another. This may result in different values on different Java platforms. There is also no warning generated when this occurs. It is up to the application developer to choose the appropriate JDBC data type.

The conversion of an ALLBASE/SQL character data type to a JDBC numeric data type uses the Java numeric conversion routines and any necessary numeric casting. Thus this could result in a `java.lang.NumberFormatException` being shown when the conversion is performed. Again, the application developer should take the necessary precautions.

Unsupported ALLBASE/SQL Data Types

The ALLBASE `LONG BINARY`, and `LONG VARBINARY` data types are currently not supported by the HP Driver for JDBC.

Acceptable SQL Syntax

The HP JDBC Server will accept either ODBC SQL statement syntax, or ALLBASE/SQL statement syntax. All SQL statements are first parsed for ODBC 2.0 SQL syntax. If the statement conforms to the ODBC 2.0 SQL syntax, it is translated to ALLBASE/SQL before being passed to the ALLBASE/SQL DBMS. If the statement does not conform to the ODBC 2.0 SQL syntax rules, it is assumed to be an ALLBASE/SQL statment, and is passed without modification to the ALLBASE/SQL SQL DBMS.

Unsupported ALLBASE/SQL SQL Statements

Table 5-3 shows a list of unsupported ALLBASE/SQL statement types in HP JDBC. Note that, in many cases, JDBC provides a standardized method of performing the same action. For example, the ALLBASE/SQL "COMMIT WORK" statement is not supported, since the user should be using the `java.sql.Connection.commit` method.

Table 5-3 Unsupported Statements

ADVANCE	BEGIN DECLARE SECTION	BEGIN WORK
CLOSE	COMMIT WORK	CONNECT
DECLARE CURSOR	DELETE WHERE CURRENT	DESCRIBE
DISCONNECT	END DECLARE SECTION	EXECUTE
EXTRACT	FETCH	INCLUDE
OPEN	PREPARE	RELEASE
ROLLBACK WORK	SET CONNECTION	SET SESSION
SET TRANSACTION	SETOPT	START DBE
STOP DBE	SQLEXPLAIN	UPDATE WHERE CURRENT
TERMINATE USER	WHENEVER	

Dynamic/Parameterized SQL Statements

The HP Driver for JDBC supports the use of parameterized SQL statements through the `java.sql.PreparedStatement` interface. The SQL statements must use a question mark (?) as the marker character for passing the parameters.

For example, a `SELECT` statement with parameters in the where clause would look like:

```
SELECT NAME, ADDRESS FROM ADDRBOOK WHERE NAME=?
```

Another example is an `INSERT` statement:

```
INSERT INTO ADDRBOOK(NAME, ADDRESS) VALUES (?,?)
```

Stored Procedures

The HP Driver for JDBC supports the following types of ALLBASE/SQL stored procedures through the `java.sql.CallableStatement` interface:

1. Procedures that return one or more result sets.
2. Procedures that take one or more input arguments.
3. Procedures that return one or more output arguments.
4. Procedures that have a return status value.

Procedures may be of one or more of these combinations; for example, the procedure may have parameterized input and a return status, and return one or more result sets.

Stored procedures are executed by first creating a `java.sql.CallableStatement` object with the execute procedure SQL statement. The execute procedure SQL statement in one or more of the following formats (each format corresponds to one of the cases listed above):

1. `EXECUTE PROCEDURE PROCNAME`
2. `EXECUTE PROCEDURE PROCNAME(10, ?)`
3. `EXECUTE PROCEDURE PROCNAME(? OUTPUT, ? OUTPUT ONLY)`
4. `EXECUTE PROCEDURE ? = PROCNAME`

Input arguments can be either explicitly specified in the SQL statement, or dynamic parameters can be used for passing values when the statement is executed.

Output only arguments must be specified with a dynamic parameter, otherwise the output value will not be returned.

6 Troubleshooting

This chapter lists most frequently encountered problems in using HP Driver for JDBC and how to overcome them. Ways to enable client tracing and server logging to identify problems are also given.

Connection Process

Most problems arise during the initial connection of the JDBC client to the database. For this reason, we shall first completely describe the steps that occur during the connection process, and mention some common problems that can arise.

Step 1. Loading the HP Driver for JDBC Class Files

- If this step fails, the most likely cause is that the Java CLASSPATH environment variable has not been set correctly.
- Other possible causes include the driver files not being installed, or unreadable or corrupt driver files.

Step 2. Loading the Network Components

- If this step fails, it could mean a bad network connection, or the client not being connected to the network.

Step 3. Loading the Specified Server Name

- If this step fails, the most likely cause is either an incorrect or invalid server name, an unreachable server (network down, or outside firewall), or a down server. The user can try to use other methods to reach the server such as ping, telnet, or ftp.

Step 4. Connecting to the 31700 or the Specified Port Number

- If this step fails, it means that there either was no process listening on that port number, or something other than the JDBC Monitor was listening on that port number. A server administrator can verify that the JDBC Monitor process is running and accepting connections on the default or other port number.

Step 5. Validating the Userid and Password

- If this step fails, the most likely cause is an incorrect userid, or invalid passwords for the userid. You can attempt to use ftp or telnet to verify that the userid and passwords are valid.

Step 6. Spawning a Server Process by the JDBC Monitor

- If this step fails (some resource limit is reached on the server), the monitor logs should indicate the reason for the failure.

Step 7. Switching to the Effective User

- Once the process is spawned, the monitor switches the effective user id to the user that was supplied in the user validation. This ensures that the user only has the privileges assigned to the user. If this step fails, the most likely cause is that the monitor was not started by "root," the superuser for HP-UX, or a user with the PM capabilities for MPE/iX. The user can log on to the server and verify that the monitor process was started by the correct user and has the proper capabilities.

Step 8. Executing the JDBC Server by the Spawned Process

- If this process fails, the JDBC Server executable could be corrupt, have incorrect permissions, be missing, or be incompatible to the server operating system (installed components for incorrect platform). The user can log on to the server and manually verify the server installation files.

Step 9. Attempting to Connect to the Specified Database

- If this step fails, the client should receive an ALLBASE/SQL error message through the `java.sql.SQLException` that is shown. Common causes of this failing are incorrect database name or a corrupt database. Make sure the case-sensitivity of the database name is also checked.

If all the preceding steps are successful, the Driver Manager will return a connection to the database.

Client Tracing

Tracing of the JDBC Driver client class files is invoked by adding tracing commands to the connection URL. The connection URL can be altered to both invoke tracing, and to control the type of information that is traced. The tracing information is sent to a Java stream, which must be specified by the application using the `java.sql.DriverManager.setLogStream` method.

Both the tracing level and tracing output must be specified before any tracing can be done.

Note that the use of tracing will impact performance of the application. The greater the detail in the tracing, the slower the performance of the application.

The URL syntax to invoke tracing is:

`"jdbc:allbase://server[:port]/database?TRACE=trace"`

where *trace* is any vertical bar (|) separated combination of the values:

`ARGUMENTS`	Trace HP Driver for JDBC class method arguments. Input arguments and return values are all traced. Only the methods called by the application are traced.
`ARGUMENTS_ALL`	Trace all HP Driver for JDBC class methods that are called by both the application and the driver itself.
`TIME`	Include the time in `HHMMSSFFF` format on all tracing output lines where `HH` is the hour from 00 to 23, `MM` is the minute from 00 to 59, SS is the second from 00 to 59, and `FFF` is the millisecond from 000 to 999. The information appears in the third column of the tracing output.
`DATE`	Include the date in `YYYYMMDD` format on all tracing output lines where `YYYY` is the year from 0000 to 9999, `MM` is the month from 01 to 12, `DD` is the day of the month from 00 to 31. This information appears in the second column of the tracing output.

TIMESTAMP	Include both the date and the time on all tracing output lines. This is just a combination of the DATE and TIME tracing values.
OBJECT	Include the object hash code in the format HHHHHHHH on all tracing output lines where HHHHHHHH is the eight digit hexadecimal value of the hash code. This information appears in the first column of the tracing output.
THREAD	Include the executing thread name on all tracing output lines. This information appears as the fourth column of the tracing output.
DEBUG	Trace debugging statements from the driver. This will produce a large amount of output, so it should be used sparingly.
CONNECTION	Trace driver connection information.
NETWORK	Trace driver network information.
ERROR	Trace driver errors.
WARNING	Trace driver warnings.
SQL	Trace SQL statements that are passed to the driver.
ALL	Trace everything.

For example, the URL to turn on tracing of all connections with time stamp information would be:

```
“jdbc:allbase://server/database?TRACE=CONNECTION|TIMESTAMP”
```

Do not use any space characters between tracing levels and the vertical bar separator as some browsers and web servers do not allow space characters in a URL.

NOTE Remember that the application must also set the tracing stream using the `java.sql.DriverManager.setLogStream` method.

Here is a sample output of a JDBC trace log using the trace flag ALL:

```
0f457ce2 19980917 110116.300 main JdbcConnection.createStatement(void)
0f457ce2 19980917 110116.312 main JdbcConnection.createStatement() =
   JdbcStatement@0f458ef4
0f458df3 19980917 110116.903 main
   JdbcResultSet.new(SolidResultSet@0f458e9b)
0f458df3 19980917 110116.907 main JdbcResultSet.new() =
   JdbcResultSet@0f458df3
0f458df3 19980917 110117.088 main JdbcResultSet.getMetaData(void)
0f458df3 19980917 110117.152 main JdbcResultSet.getMetaData() =
   JdbcResultSetMetaData@0f458c9f
0f458df3 19980917 110117.164 main JdbcResultSet.next(void)
0f458df3 19980917 110117.167 main JdbcResultSet.next() = true
0f458df3 19980917 110117.170 main JdbcResultSet.getString(1)
0f458df3 19980917 110117.199 main JdbcResultSet.getString() = 33
0f458df3 19980917 110117.202 main JdbcResultSet.wasNull(void)
0f458df3 19980917 110117.205 main JdbcResultSet.wasNull() = false
0f458df3 19980917 110117.212 main JdbcResultSet.next(void)
0f458df3 19980917 110117.215 main JdbcResultSet.next() = false
0f458df3 19980917 110117.218 main JdbcResultSet.close(void)
0f458df3 19980917 110117.221 main JdbcResultSet.close() = void
0f457ce2 19980917 110118.087 main JdbcConnection.close(void)
0f457ce2 19980917 110118.090 main JdbcConnection.close:
 Closing JdbcConnection
   object. Disconnecting all connections.
0f457ce2 19980917 110118.093 main  JdbcConnection.close: Closing connection
   number 0
0f457ce2 19980917 110118.099 main  JdbcConnection.close:
 JdbcConnection object
   closed
0f457ce2 19980917 110118.102 main JdbcConnection.close() = void
```

The first column shows the object hash code. Note that the object hash code can be matched with the objects returned by various methods. The second column shows the date, September 17, 1998. The third column shows the time, 11:01 AM. The fourth column shows the thread name; in this case there is only the main thread. The last column contains the tracing information. In each line, the class and method are logged.

Server Logging

The server configuration file controls logging of the JDBC Server, see "Configuring HP JDBC Server" in Chapter 3 , "Installation," or "HP JDBC Server Configuration Files" in Appendix C , "HP JDBC Server." The following is a list of the various logging levels that can be set in the server configuration file.

`NONE`	Turns off all preceding logging levels.
`CONNECTION`	Turns on logging of connection information such as the time of the connection, the client IP address, and the name of the database.
`IN`	Turns on logging of input information coming from the client.
`INHEX`	Turns on logging of the hexadecimal dump of the input information coming from the client.
`OUT`	Turns on logging of output information being sent back to the client.
`OUTHEX`	Turns on logging of the hexadecimal dump of the output information being sent back to the client.
`INFO`	Turns on logging of miscellaneous information about the processing of the server. This has the potential to log a fair amount of data, which may affect performance.
`TIMESTAMP`	This causes each log entry to be prefaced by a timestamp of the form `HH:MM:SS.mmm`.
`WARNING`	Turns on logging of warning messages generated by the Server.
`ERROR`	Turns on logging of error messages generated by the Server.
`FATAL`	This is the same as specifying `INFO`, `WARNING`, and `ERROR`.
`DEBUG`	Turns on logging of a lot of debugging messages. This should only be done at the request of support personnel. This generates a large amount of logging information and can severely affect performance.

For a sample of a JDBC Server log file, refer to Appendix C , "HP JDBC Server."

A Data Types

This appendix gives a detailed description of Data Types supported by ALLBASE/SQL and HP Driver for JDBC. Data type names, size and format are discussed.

ALLBASE/SQL Data Types

SMALLINT	16-bit signed integer.
INTEGER	32-bit signed integer.
NUMERIC	Fixed-point, packed, decimal number with a specified precision of p and scale of s, where p is in the range 1 to 27, and s is in the range 0 to p. The default value of p is 27 and s is 0.
DECIMAL	Identical to NUMERIC.
REAL	32-bit floating-point number.
FLOAT	A floating-point number with a precision of p where p is in the range 1 to 53. The default value of p is 53. If the value of p is from 1 to 24, the value is treated as a 32-bit floating-point number. If the value of p is from 25 to 53, the value is treated as a 64-bit floating-point number.
DOUBLE	64-bit floating-point number.
CHAR	String of characters of fixed-length n where n is in the range 1 to 3996 bytes. Space characters are used to pad any strings of length less than n to length n.
VARCHAR	String of characters of variable-length up to length n where n is in the range 1 to 3996 bytes.
DATE	String of characters in the format "YYYY-MM-DD" where YYYY represents the calendar year, MM is the month, and DD is the day of month. DATE is in the range from "0000-01-01" to "9999 12-31".
TIME	String of characters in the format "HH:MI:SS" where HH represents hours, MI is minutes, and SS is seconds. TIME is in the range "00:00:00" to "23:59:59"

DATETIME	String of characters in the format "YYYY-MM-DD HH:MI:SS.FFF" where YYYY represents the calendar year, MM is the month, DD is the day of the month, HH is the hour, MI is the minute, SS is the second, and FFF is the thousandth of a second. DATETIME is in the range "0000-01-01 00:00:00.000" to "9999-12-31 23:59:59.999".
INTERVAL	String of characters in the format "DDDDDDD HH:MI:SS.FFF" where DDDDDDD is a number of days, HH is a number of hours, MI is a number of minutes, SS is a number of seconds, and FFF is a number of milliseconds. INTERVAL is in the range "0 00:00:00.000" to "3652436 23:59:59.999".
BINARY	Binary string of fixed-length n where n is in the range of 1 to 3996 bytes. Each byte stores 2 hexadecimal digits.
VARBINARY	Binary string of variable-length up to length n where n is in the range of 1 to 3996 bytes. Each byte stores 2 hexadecimal digits.
LONG BINARY	Binary string of fixed-length up to length n where n is in the range of 1 to $(2^{31}-1)$ bytes. This data type is currently not supported by the HP Driver for JDBC.
LONG VARBINARY	Binary string of variable-length up to length n where n is in the range of 1 to $(2^{31}-1)$ bytes. This data type is currently not supported by the HP Driver for JDBC.

JDBC Data Types

CHAR	Small, fixed-length character string.
VARCHAR	Small, variable-length character string.
LONGVARCHAR	Large, variable-length character string.
NUMERIC	Fixed-precision decimal value represented with a precision that can be greater than the original precision.
DECIMAL	Fixed-precision decimal value represented with a precision that is no greater than the original precision.
BIT	A single bit value that can be either zero or one.
TINYINT	8-bit unsigned integer.
SMALLINT	16-bit signed integer.
INTEGER	32-bit signed integer.
BIGINT	64-bit signed integer.
REAL	“Single-precision” floating-point number that supports seven digits of mantissa.
FLOAT	“Double-precision” floating-point number that supports 15 digits of mantissa.
DOUBLE	“Double-precision” floating-point number that supports 15 digits of mantissa.
BINARY	Small, fixed-length binary value.
VARBINARY	Small, variable-length binary value.
LONGVARBINARY	Large, variable-length binary value.
DATE	A value representing a date consisting of day, month, and year.
TIME	A value representing a time consisting of hours, minutes, and seconds.
TIMESTAMP	A DATE plus a TIME plus a nanosecond field.

B JDBC Monitor

This appendix describes one of the HP JDBC Server components, the JDBC Monitor. JDBC Monitor configuration, logging, startup and shutdown are discussed.

HP-UX Monitor

On the HP-UX platform, a separate server-process services each client connection. These processes are dissociated from the monitor process, so if the monitor terminates or dies, the existing server-processes servicing client requests will continue to function, until the client closes the connection, or the server times out.

Normally, the monitor is automatically started up during the host boot process via a startup script. This startup script is installed when the HP JDBC product is installed on the server. Normally, the monitor is never brought down unless the machine is brought down, in which case it will be automatically stopped via a stop script that is installed along with the startup script.

If there is a need to start or shut down the monitor when the host machine is still up and running, the tool `monctrl` (also installed with the HP JDBC product) can be used to perform the startup and shut down. The command to use `monctrl` is:

`monctrl {start|reset|kill} [`*`portnumber`*`]`

The “start” argument will start up the monitor on either the default port number or on the specified port number. The “kill” argument will first terminate each of the monitor’s active child processes, and then terminate the monitor process itself. Doing this gives a better chance of being able to restart the monitor once it has been shut down. The *`portnumber`* argument is optional, if it is not specified the default port number will be used.

NOTE

`monctrl` should only be executed by “root” because the monitor must run as “root” to allow it to spawn processes and set user ids. If it is not started by “root,” other users will not be able to connect to the monitor.

The “reset” argument is discussed in the section dealing with the monitor configuration file.

You cannot start a monitor on the same port number as a currently running monitor. You also can not restart a monitor on its original port number until all child processes that were spawned by the previous monitor process are terminated, and the port released. If the monitor is intentionally terminated or accidentally terminates, all of its child processes must be terminated before it can be restarted. For this reason, the monitor *must not* be terminated by using the UNIX `kill` command. Always use the `monctrl` tool to kill the monitor. The tool will search out all the child processes and kill them first, before killing the monitor.

MPE/iX Monitor

On the MPE/iX platform, each client connection is serviced by a separate process that is in the same session as the monitor. Thus, if the monitor is terminated or dies, the existing server process already serving the client requests will also die. This could result in client applications experiencing dropped connections.

HP-UX Monitor Configuration File

This represents the standard HP-UX JDBC monitor configuration file. This is a text file that resides on the server host in the same directory as the monitor executable file and can be used to alter some of the monitor's behavior. The name of the monitor configuration file is `moncfg`, and it is located in `/opt/allbase/jdbc/bin`.

```
MLOGPATH      /opt/allbase/jdbc/logs
MAXLOGSIZE    500000
MAXSESSIONS   128


[JDBCSERV.JDBC.SYS]
SERVICE /opt/allbase/jdbc/bin/jdbcserv
```

The first configuration option `MLOGPATH` refers to the path for the monitor log file. Normally this is set to `/opt/allbase/jdbc/logs`. The log file name for the monitor is called `monlog`.

The second configuration option `MAXLOGSIZE` specifies the maximum size of the monitor log file, before the file is backed up to the name `monlog.old`, and the log file reset to zero length. This prevents the log file from growing without bounds. The maximum amount of disk space that will be used for the monitor log is twice the amount specified by this option (total size of the log file and the backup log file).

The third configuration option `MAXSESSIONS` sets the limit on the maximum number of concurrent sessions that is allowed by the monitor. Limiting the number of active connections can increase performance of the server host and prevent overload of the system. Client connections are rejected with a `java.sql.SQLException` if the number of sessions are exceeded. It means that the server is busy, and that they should try again later. If this number is not set or set to `0`, then the number of sessions is unlimited.

The last two lines of the configuration file above provide the mapping to the JDBC Server executable. The service name of the JDBC Server is "`JDBCSERV.JDBC.SYS`". The `SERVICE` line after the service name gives the full path and filename of the JDBC Server.

Normally, there is no reason to edit the monitor configuration file. If for some reason it is edited, the monitor must be reset using the `monctrl` command to trigger the monitor to re-read the configuration file. The syntax for this command is:

```
monctrl reset [portnumber]
```

where *portnumber* is the port number on which the monitor to be reset is running. If the port number is not specified, the default port number is assumed. The reset of the monitor occurs “on the fly,” without affecting any of the current established client connections.

HP-UX Monitor Log

This is a sample of an HP-UX JDBC monitor log file. The beginning of the file shows when the monitor was started, as well as the monitor configuration settings. The left-hand column shows the UNIX process id number (PID) of the process that is logging the information.

```
27152   ============================================================
27152   JDBC Monitor starting up at Wed Oct  7 09:25:00 1998
27152   Configuration file:       moncfg
27152   Socket has port:          31700
27152   JDBC Monitor PID:         27152
27152   Max logfile size (bytes): 100000
27152   Max number of sessions:   128
27152   alias set: [JDBCSERV.JDBC.SYS] /opt/allbase/jdbc/bin/jdbcserv
  (0 PUTENVs)
27152
27152   [15.0.121.172 Wed Oct  7 09:25:47 1998
27426   PID: 27426 assigned.
27426   Login successful, user: subo
27426   Service 'JDBCSERV.JDBC.SYS'.
27426   Matched to 'JDBCSERV.JDBC.SYS'.
27426   Service to exec: /opt/allbase/jdbc/bin/jdbcserv
27152   PID: 27426 died. 0 left.
27152   [15.0.121.172 Wed Oct  7 09:26:20 1998
27545   PID: 27545 assigned.
27545 E Invalid userid, user: joebob1
27545   Exiting.]
27152   PID: 27545 died. 0 left.
27152   [15.0.121.172 Wed Oct  7 09:26:39 1998
27578   PID: 27578 assigned.
27578 E Invalid password, user: subo
27578   Exiting.]
27152   PID: 27578 died. 0 left.
```

The sample log file shows three client connections. For each connection, the client IP address is logged, along with the time of the connection. If there the user validation fails, the cause of the failure is logged (note the “E” in the second column denoting an error condition). If the user validation succeeds, the connection to the JDBC Server is logged.

MPE/iX Monitor Log

This is a sample of a standard list spool file for the MPE/iX HP JDBC. The beginning of the file shows when the monitor was started.

```
JOB JDBCMON,MGR.JDBC,JDBC.

Priority = DS; Inpri = 8; Time = UNLIMITED seconds.

Job number = #j586.

FRI, OCT  9, 1998, 10:52 AM.

HP3000  Release: C.50.00   User Version: C.50.00

MPE/iX  HP31900 B.79.06  Copyright Hewlett-Packard 1987.

All rights reserved.

STREAMED BY STREAMER,MANAGER.SYS (#J471) ON LDEV# 10

   STREAM DATE:   FRI, OCT  9, 1998, 10:52 AM

***************

** SYSTEM 39 **

***************

*****************************************************************

* "This is a private system operated for Hewlett-Packard company *

* business.  Authorization from HP management is required to use *

* this system. Use by unauthorized persons is prohibited."       *

*****************************************************************

* WARNING - This computer is accessed by companies outside HP.   *

* All security and control procedures must be strictly followed. *

*****************************************************************


END OF PROGRAM

:run jdbcmon.jdbc.sys;info="31700";pri=cs



 Waiting for a client connection...


 Connection established from 15.0.121.172 on FRI, OCT  9, 1998, 10:54 AM

 Unable to validate ID 'mgr.jdbc' - AIFCHANGELOGON() rc=-2520: Invalid
account
```

```
 password specified.

 Waiting for a client connection...

 Connection established from 15.0.121.172 on FRI, OCT  9, 1998, 10:55 AM
 Unable to validate ID 'mgr.jdbc' - AIFCHANGELOGON() rc=-2521:
  Invalid user pas sword specified.

 Waiting for a client connection...

 Connection established from 15.0.121.172 on FRI, OCT  9, 1998, 11:00 AM
 Server 'servprog;info=""' created for userid 'mgr.jdbc' PIN=1914

 Waiting for a client connection...

 Connection established from 15.0.121.172 on FRI, OCT  9, 1998, 11:09 AM
 *** Monitor shutdown requested on FRI, OCT  9, 1998, 11:09 AM ***
 Waiting for 2 children to terminate...
 Waiting for 2 children to terminate...

 Waiting for 1 child to terminate...

*** JDBC Monitor Ending on FRI, OCT  9, 1998, 11:12 AM



END OF PROGRAM
 :eoj
CPU sec. = 3.  elapsed min. = 20.  FRI, OCT  9, 1998, 11:12 AM.
```

The sample spool file shows three client connections. For each connection, the client IP address is logged, along with the time of the connection. If the user validation fails, the cause of the failure is logged. If the user validation succeeds, the connection to the JDBC Server is logged.

HP-UX Monitor Startup and Shutdown

The startup and shutdown of the HP-UX JDBC Monitor is normally controlled by the system startup and shutdown scripts. Thus as long as the machine is up and running, the JDBC Monitor will also be up.

The only time the monitor should ever be brought down is to install a newer version. In this case, use the `monctrl` command to kill the monitor process. Do not use the UNIX `kill` command, as this could render the server unable to start a new monitor process. The `monctrl` command to shut down the monitor is:

`monctrl kill [`*`portnumber`*`]`

The *`portnumber`* parameter is only necessary if the monitor you wish to shut down is not running on the default port number `31700`.

To restart the monitor after it has accidentally terminated or was shut down, use the `monctrl` command to start it up:

`monctrl start [`*`portnumber`*`]`

Again, the *`portnumber`* parameter is only necessary if you wish to start the monitor on a port number other than the default port number `31700`.

MPE/iX Monitor Startup and Shutdown

The startup and shutdown of the MPE/iX HP JDBC Monitor is normally done by the startup and shutdown stream jobs, JSTRTMON and JSTOPMON, which are included in system startup and shutdown scripts. Thus as long as the machine is up and running, the JDBC Monitor will also be up.

The only time the monitor should ever be brought down is to install a newer version. In this case, use the JSTOPMON stream job or ABORTJOB command to kill the monitor process. The ABORTJOB command to shut down the monitor is:

```
:ABORTJOB J#XX
```

To restart the monitor after it has accidentally terminated or was shut down, use the JSTRTMON stream job to start it up.

```
:stream JSTRTMON.JDBC.SYS
```

C HP JDBC Server

This appendix discusses the HP JDBC Server components, the JDBC Monitor. Unlike the HP JDBC Monitor, the HP JDBC Server components behave the same on both the HP-UX and MPE/iX platforms. Thus both the configuration file and log file are the same (except for filenames).

HP JDBC Server Configuration File

The following example shows the standard HP JDBC Default Server configuration file. This is a text file that resides on the server host in the same directory as the HP JDBC Server executable files and can be used to alter some of the server's behavior. On HP-UX the name of the default server configuration file is `servcfg`, and it is located in `/opt/allbase/jdbc/bin`. On MPE/iX the name of the default server configuration file is `SERVCFG` and it is located in `JDBC.SYS`.

```
LOGFILE /opt/allbase/jdbc/logs/servlog
TIMEOUT 7200


#LOGGING ERROR
#LOGGING CONNECTION
#LOGGING INFO
#LOGGING WARNING
#LOGGING FATAL
#LOGGING IN
#LOGGING OUT
#LOGGING INHEX
#LOGGING OUTHEX
#LOGGING DEBUG
#LOGGING TIMESTAMP
LOGGING NONE
# If LOGGING NONE is not commented out, it must be the
# last line for it to have the desired effect.
```

The first configuration option `LOGFILE` refers to the path and filename for the HP JDBC Server log file. On HP-UX this is normally set to `/opt/allbase/jdbc/logs/servlog`. On MPE/iX this is normally set to `SERVLOG.JDBC.SYS`.

The second configuration option `TIMEOUT` specifies the number of seconds that the HP JDBC Server can remain idle before terminating. This idle time is the time between the returning of a result to the client and the submission of the next client request. It does not include database processing time. If this value is not provided, or is set to 0, the timeout will be set to an infinite value.

The next set of lines are the LOGGING levels used to determine the type of information to be logged to the server log file. The meanings of the various logging levels is discussed in the Troubleshooting section, under Server Logging. The logging levels can be turned on and off by either commenting the line out, or uncommenting the line. If the last line is LOGGING NONE, and the line is not commented out, all logging is turned off.

The server configuration file is read in anew each time a new server process is started. Thus once the file is edited and saved, the changes will take place beginning with the next server process. Existing server processes cannot be made to reread the configuration file.

NOTE This is the default server configuration file, meaning that all server processes use the same configuration file. Thus, if logging is turned on in this file, it will be in effect for all subsequent server processes.

JDBC Server Logs

This is a sample of a JDBC Server log file. This log file was generated using the LOGGING CONNECTION, LOGGING IN, and LOGGING TIMESTAMP logging levels set in the server configuration file.

```
17039 [12:26:14.152] Connected to SOLID DM on Wed Oct  7 12:26:14
17039 [12:26:14.156] Connected from '15.0.121.172' by '<unavailable>'
17039 [12:26:14.243] TID: 'SET_LIB_OPTIONS'
17039 [12:26:14.443] TID: 'CONNECT'
17039 [12:26:14.445] [connect_to_db] dbname: '/tmp/db/mydb'
17039 [12:26:14.638] [connect_to_db] Connected.
17039 [12:26:14.843] TID: 'SET_OPTIONS'
17039 [12:26:14.844] [TID_SET_OPTIONS] datalen: 8 data: 0x0000010000000100
17039 [12:26:15. 43] TID: 'SET_OPTIONS'
17039 [12:26:15. 44] [TID_SET_OPTIONS] datalen: 8 data: 0x0001000000010000
17039 [12:26:15. 55] TID: 'GET_DBMS_TYPE'
17039 [12:26:15.261] TID: 'GET_INFO'
17039 [12:26:15.644] TID: 'GET_METADATA'
17039 [12:26:15.645] [get_metadata] metadatatypestr: 'INFO'
17039 [12:26:15.844] TID: 'GET_METADATA'
17039 [12:26:15.845] [get_metadata] metadatatypestr: 'max_values'
17039 [12:26:19.643] TID: 'PREPARE_SQL'
17039 [12:26:19.645] [TID_PREPARE] inbuf: 'select * from mytest'
17039 [12:26:19.655] [After Parse] stmt_buf2: 'SELECT * FROM mytest '
17039 [12:26:19.736] TID: 'EXECUTE_SQL'
17039 [12:26:45.391] [TID_DISCONNECT] Disconnected
17039 [12:26:45.392] Client exited.
17039 [12:26:45.392] Connection ended on Wed Oct  7 12:26:45
```

This sample shows a single client connection connecting to the /tmp/db/mydb database, issuing the SQL statement SELECT * FROM MYTEST, and disconnecting.

The beginning of the file shows when the server was started and the IP address of the client. The first column shows the UNIX process id number (PID) of the process that is logging the information. The second column shows the timestamp.

D Simple Client Source Code

The Java source listing given in this appendix is a program demonstrating the use of HP Driver for JDBC to access an ALLBASE/SQL or IMAGE/SQL database. It can be run from any platform which has JDK 1.1 (or above) and HP Driver for JDBC installed. This program is part of the HP Driver for JDBC distribution and it is named SimpleClient.java.

```
/**
 * JDBC Simple Client
 *
 * This simple Java application loads a JDBC Driver, prompts the
 * user for connection information, creates the Driver URL, makes
 * the connection to the database, and sends user provided SQL
 * statements to the database.
 *
 * If the result of the statement is a result set, it is printed
 * out.
 *
 * This code and information is provided "as is" without warranty
 * of any kind, either expressed or implied.
 *
 */
import java.io.*;
import java.net.URL;
import java.sql.*;

class SimpleClient
   {
   private static Connection con = null;
   private static Statement stmt = null;

   /**
     * Command:  java SimpleClient
     */
   public static void main(String args[])
```

```
{
/* User input fields */

String hostName       = getString("Host Name: ");
String userName       = getString("User Name: ");
String userPassword   = getString("User Password: ");
String dbName         = getString("Database Name: ");

/* Create the URL based on the user input */

String url = "jdbc:allbase://" +
   hostName + "/" + dbName + "?";

/* Try to make the connection to the database */

try
   {
   /* Load the jdbc driver */

   Class.forName("com.hp.jdbc.allbase.JdbcDriver");

   /* Attempt to get a connection */

   con = DriverManager.getConnection(url, userName, userPassword);
   }
catch (SQLException ex)
   {
   System.out.println("SQLException caught during connection:");
   printException(ex);
   System.exit(0);
   }
catch (Exception e)
   {
   System.out.println("Exception caught during connection:");
```

```
   e.printStackTrace();
   System.exit(0);
   }

/* Main loop */
String sql = null;
ResultSet rs = null;
while (true)
   {
   /* Ask user for an sql statement */

   System.out.println("");
   sql = getString("SQL -> ");
   System.out.println("");
   /* If empty string, then exit */

   if (sql == null || sql.length() == 0)
               break;
   /* Submit the sql statement */

   try
      {
      stmt = con.createStatement();
      /* If execute returns true, there is a result set */

      if (stmt.execute(sql))
         {
         rs = stmt.getResultSet();
         displayResultSet(rs);
         }
      /* Otherwise, there is an update count */

      else
         System.out.println(stmt.getUpdateCount() + " rows changed.");
```

```
            }
         catch (SQLException ex)
            { printException(ex); }
         catch (java.lang.Exception ex)
            { ex.printStackTrace(); }
         try
            {
            rs.close();
            stmt.close();
            rs = null;
            stmt = null;
            }
         catch (Exception e)
            {}

         } /* End of main loop */
      try
         { con.close(); }
      catch (Exception e)
          {}

      System.exit(0);
      }
/**
 * Print out SQLException information.
 */
private static void printException(SQLException e)
      {
      SQLException ex = e;

      System.out.println("SQLException:");
      while (ex != null)
         {
         System.out.println("   SQLState   : " + ex.getSQLState());
```

```
      System.out.println("   Vendor code: " + ex.getErrorCode());
      System.out.println("   Message    : " + ex.getMessage());
      ex = ex.getNextException();
      System.out.println();
      }
   return;
   }

/**
 * Displays all columns and rows in the given result set
 */
private static void displayResultSet (ResultSet rs)
   throws SQLException
   {
   int i;
   /* Get the ResultSetMetaData.  This will be used for
   ** the column headings.
   */

   ResultSetMetaData rsmd = rs.getMetaData();
   /* Get the number of columns in the result set */

   int numCols = rsmd.getColumnCount();
   /* For each column, print out the column names, separated by
   ** commas
   */

   for (i=1; i <= numCols; i++)
      {
      System.out.print(rsmd.getColumnName(i));
      /* Print separator */

      if (i < numCols)
                  System.out.print(", ");
```

```
      }
   System.out.println("\n");

   /* Print out the data, fetching until end of the result set */
   String colValue;

   while (rs.next())
      {
      /* For each column, print out the data as String */

      for (i=1; i<=numCols; i++)
         {
         colValue = rs.getString(i);
         /* If data was null, use the null string instead */

         if (rs.wasNull())
            colValue = "<NULL>";
         System.out.print(colValue);

         /* Print separator */
         if (i < numCols)
                     System.out.print(", ");
        }

      System.out.println();
      /* Fetch the next result set row */

      }
   System.out.println();
   }

/**
 * This captures console input until a CR is hit and
 * then returns a Java String with the input.
```

```
 */
private static String getString(String prompt)
   {
   BufferedReader in =
      new BufferedReader(new InputStreamReader(System.in));
   if (prompt != null)
      System.out.print(prompt);

   String rString = "";
   try
      { rString = in.readLine(); }
   catch (Exception e)
      { rString = ""; }
   return rString;
   }
}
```

E HP ALLBASE/SQL JDBC File Lists

This appendix lists each file that is provided in the HP JDBC product and gives a brief description of the purpose of the file.

Java Client Files

In the com/hp/jdbc/allbase directory:

`Jdbc.class`

Parent JDBC Driver class

`JdbcCallableStatement.class`

JDBC callable statement implementation class

`JdbcConnection.class`

JDBC connection implementation class

`JdbcDatabaseMetaData.class`

JDBC database metadata implementation class

`JdbcDBMD.class`

JDBC database metadata utility class

`JdbcDriver.class`

JDBC Driver implementation class

`JdbcPreparedStatement.class`

JDBC prepared statement implementation class

`JdbcResultSet.class`

JDBC result set implementation class

`JdbcResultSetMetaData.class`

JDBC result set metadata implementation class

`JdbcStatement.class`

JDBC statement implementation class

`JdbcVersion.class`

JDBC driver version information class

`LogonDialog.class`

JDBC driver logon dialog box utility class

`ThreadInfo.class`

JDBC driver thread information class

`Tracing.class`

JDBC driver tracing utility class

In the com/hp/jdbc/allbase/samples directory:

README

JDBC sample client readme file

SampleClient.java

JDBC sample client source code

SimpleClient.java

JDBC simple client source code (very simple)

HP-UX Server Files

In the /opt/allbase/jdbc directory:

bin/jdbcmon	JDBC Monitor executable program
bin/moncfg	JDBC Monitor text configuration file
bin/monctrl	JDBC Monitor control file
bin/jdbcserv	JDBC Server executable program
bin/servcfg	JDBC Server text configuration file
logs/monlog	JDBC Monitor log file
logs/servlog	JDBC Server default log file
driver/driver.tar	JDBC Driver files in UNIX tape archive format
driver/driver.jar	JDBC Driver files in Java archive format
driver/driver.zip	JDBC Driver files in Windows ZIP format

Startup scripts:

/sbin/init.d/Jdbcmonitor

JDBC Monitor startup/shutdown script

/etc/rc.config.d/Jdbcmonitor

JDBC Monitor startup/shutdown script configuration variables

/sbin/rc1.d/K910Jdbcmonitor

System shutdown link to the startup/shutdown script

/sbin/rc2.d/S090Jdbcmonitor

System startup link to the startup/shutdown script

MPE/iX Server Files

In JDBC.SYS group:

JDBCMON	JDBC Monitor startup executable program
JDBCSTOP	JDBC Monitor shutdown executable program
JSTRTMON	JDBC Monitor startup job stream
JSTOPMON	JDBC Monitor shutdown job stream
JDBCSERV	JDBC Server executable program
SERVCFG	JDBC Server text configuration file
SERVLOG	JDBC Server default log file
DRIVERT	JDBC Driver files in UNIX tape archive format
DRIVERJ	JDBC Driver files in Java archive format
DRIVERZ	JDBC Driver files in Windows ZIP format
HFSFILES	JDBC Distribution archive
I00IJDBC	JDBC Installation job
HPREADME	JDBC README file

Index