

You can set the `CLASSPATH` environment variable under Unix, Linux or Mac OS X either locally for a user within their `.profile`, `.login` or other login file. You can also set it globally by editing the global `/etc/profile` file.

For example, add the Connector/J driver to your `CLASSPATH` using one of the following forms, depending on your command shell:

```
# Bourne-compatible shell (sh, ksh, bash, zsh):
shell> export CLASSPATH=/path/mysql-connector-java-ver-bin.jar:$CLASSPATH

# C shell (csh, tcsh):
shell> setenv CLASSPATH /path/mysql-connector-java-ver-bin.jar:$CLASSPATH
```

Within Windows 2000, Windows XP, Windows Server 2003 and Windows Vista, you set the environment variable through the System Control Panel.

To use MySQL Connector/J with an application server such as GlassFish, Tomcat or JBoss, read your vendor's documentation for more information on how to configure third-party class libraries, as most application servers ignore the `CLASSPATH` environment variable. For configuration examples for some J2EE application servers, see [Section 21.3.7, “Connection Pooling with Connector/J”](#), [Section 21.3.8, “Load Balancing with Connector/J”](#), and [Section 21.3.9, “Failover with Connector/J”](#). However, the authoritative source for JDBC connection pool configuration information for your particular application server is the documentation for that application server.

If you are developing servlets or JSPs, and your application server is J2EE-compliant, you can put the driver's `.jar` file in the `WEB-INF/lib` subdirectory of your webapp, as this is a standard location for third party class libraries in J2EE web applications.

You can also use the `MysqlDataSource` or `MysqlConnectionPoolDataSource` classes in the `com.mysql.jdbc.jdbc2.optional` package, if your J2EE application server supports or requires them. Starting with Connector/J 5.0.0, the `javax.sql.XADataSource` interface is implemented using the `com.mysql.jdbc.jdbc2.optional.MysqlXADataSource` class, which supports XA distributed transactions when used in combination with MySQL server version 5.0.

The various `MysqlDataSource` classes support the following parameters (through standard set mutators):

- `user`
- `password`
- `serverName` (see the previous section about fail-over hosts)
- `databaseName`
- `port`

### 21.3.3.3. Upgrading from an Older Version

This section has information for users who are upgrading from one version of Connector/J to another, or to a new version of the MySQL server that supports a more recent level of JDBC. A newer version of Connector/J might include changes to support new features, improve existing functionality, or comply with new standards.

#### 21.3.3.3.1. Upgrading to MySQL Connector/J 5.1.x

- In Connector/J 5.0.x and earlier, the alias for a table in a `SELECT` statement is returned when accessing the result set metadata using `ResultSetMetaData.getColumnName()`. This behavior however is not JDBC compliant, and in Connector/J 5.1 this behavior was changed so that the original table name, rather than the alias, is returned.

The JDBC-compliant behavior is designed to let API users reconstruct the DML statement based on the metadata within `ResultSet` and `ResultSetMetaData`.

You can get the alias for a column in a result set by calling `ResultSetMetaData.getColumnLabel()`. To use the old noncompliant behavior with `ResultSetMetaData.getColumnName()`, use the `useOldAliasMetadataBehavior` option and set the value to `true`.

In Connector/J 5.0.x, the default value of `useOldAliasMetadataBehavior` was `true`, but in Connector/J 5.1 this was changed to a default value of `false`.

#### 21.3.3.3.2. JDBC-Specific Issues When Upgrading to MySQL Server 4.1 or Newer

- *Using the UTF-8 Character Encoding* - Prior to MySQL server version 4.1, the UTF-8 character encoding was not supported by the server, however the JDBC driver could use it, allowing storage of multiple character sets in `latin1` tables on the server.

Starting with MySQL-4.1, this functionality is deprecated. If you have applications that rely on this functionality, and can not upgrade them to use the official Unicode character support in MySQL server version 4.1 or newer, add the following property to your connection URL:

```
useOldUTF8Behavior=true
```

- *Server-side Prepared Statements* - Connector/J 3.1 will automatically detect and use server-side prepared statements when they are available (MySQL server version 4.1.0 and newer). If your application encounters issues with server-side prepared statements, you can revert to the older client-side emulated prepared statement code that is still presently used for MySQL servers older than 4.1.0 with the following connection property:

```
useServerPrepStmts=false
```

#### 21.3.3.3.3. Upgrading from MySQL Connector/J 3.0 to 3.1

Connector/J 3.1 is designed to be backward-compatible with Connector/J 3.0 as much as possible. Major changes are isolated to new functionality exposed in MySQL-4.1 and newer, which includes Unicode character sets, server-side prepared statements, `SQLState` codes returned in error messages by the server and various performance enhancements that can be enabled or disabled using configuration properties.

- **Unicode Character Sets:** See the next section, as well as [Section 10.1, “Character Set Support”](#), for information on this MySQL feature. If you have something misconfigured, it will usually show up as an error with a message similar to `Illegal mix of collations`.
- **Server-side Prepared Statements:** Connector/J 3.1 will automatically detect and use server-side prepared statements when they are available (MySQL server version 4.1.0 and newer).

Starting with version 3.1.7, the driver scans SQL you are preparing using all variants of `Connection.prepareStatement()` to determine if it is a supported type of statement to prepare on the server side, and if it is not supported by the server, it instead prepares it as a client-side emulated prepared statement. You can disable this feature by passing `emulateUnsupportedPstmts=false` in your JDBC URL.

If your application encounters issues with server-side prepared statements, you can revert to the older client-side emulated prepared statement code that is still presently used for MySQL servers older than 4.1.0 with the connection property `useServerPrepStmts=false`.

- **Datetimes** with all-zero components (`0000-00-00 ...`): These values cannot be represented reliably in Java. Connector/J 3.0.x always converted them to `NULL` when being read from a `ResultSet`.

Connector/J 3.1 throws an exception by default when these values are encountered, as this is the most correct behavior according to the JDBC and SQL standards. This behavior can be modified using the `zeroDateTimeBehavior` configuration property. The permissible values are:

- `exception` (the default), which throws an `SQLException` with an `SQLState` of `S1009`.
- `convertToNull`, which returns `NULL` instead of the date.
- `round`, which rounds the date to the nearest closest value which is `0001-01-01`.

Starting with Connector/J 3.1.7, `ResultSet.getString()` can be decoupled from this behavior using `noDatetimeStringSync=true` (the default value is `false`) so that you can retrieve the unaltered all-zero value as a `String`. Note that this also precludes using any time zone conversions, therefore the driver will not allow you to enable `noDatetimeStringSync` and `useTimezone` at the same time.

- **New SQLState Codes:** Connector/J 3.1 uses SQL:1999 `SQLState` codes returned by the MySQL server (if supported), which are different from the legacy X/Open state codes that Connector/J 3.0 uses. If connected to a MySQL server older than MySQL-4.1.0 (the oldest version to return `SQLStates` as part of the error code), the driver will use a built-in mapping. You can revert to the old mapping by using the configuration property `useSqlStateCodes=false`.
- **`ResultSet.getString()`:** Calling `ResultSet.getString()` on a `BLOB` column will now return the address of the `byte[]` array that represents it, instead of a `String` representation of the `BLOB`. `BLOB` values have no character set, so they cannot be converted to `java.lang.Strings` without data loss or corruption.

To store strings in MySQL with LOB behavior, use one of the `TEXT` types, which the driver will treat as a `java.sql.Clob`.

- **Debug builds:** Starting with Connector/J 3.1.8 a debug build of the driver in a file named `mysql-connector-java-version-bin-g.jar` is shipped alongside the normal binary jar file that is named `mysql-connector-java-version-bin.jar`.

Starting with Connector/J 3.1.9, we do not ship the `.class` files unbundled, they are only available in the JAR archives that ship with the driver.

Do not use the debug build of the driver unless instructed to do so when reporting a problem or bug, as it is not designed to be run in production environments, and will have adverse performance impact when used. The debug binary also depends on the Aspect/J runtime library, which is located in the `src/lib/aspectjrt.jar` file that comes with the Connector/J distribution.

#### 21.3.3.4. Installing from the Development Source Tree



##### Caution

Read this section only if you are interested in helping us test our new code. To just get MySQL Connector/J up and running on your system, use a standard binary release distribution.

To install MySQL Connector/J from the development source tree, make sure that you have the following prerequisites:

- A Bazaar client, to check out the sources from our Launchpad repository (available from <http://bazaar-vcs.org/>).
- Apache Ant version 1.7 or newer (available from <http://ant.apache.org/>).
- JDK 1.4.2 or later. Although MySQL Connector/J can be used with older JDKs, compiling it from source requires at least JDK 1.4.2. To build Connector/J 5.1 requires JDK 1.6.x and an older JDK such as JDK 1.5.x; point your `JAVA_HOME` environment variable at the older installation.

To check out and compile a specific branch of MySQL Connector/J, follow these steps:

1. Check out the latest code from the branch that you want with one of the following commands.