CHAPTER 11

Administering Database Connectivity

This chapter provides procedures for performing database connectivity tasks in the Oracle GlassFish Server 3.1 environment by using the `asadmin` command-line utility.

The following topics are addressed here:

- "About Database Connectivity" on page 215
- "Setting Up the Database" on page 216
- "Configuring Access to the Database" on page 219
- "Configuration Specifics for JDBC Drivers" on page 232

Instructions for accomplishing these tasks by using the Administration Console are contained in the Administration Console online help.

About Database Connectivity

A database management system (DBMS) provides facilities for storing, organizing, and retrieving data. The information in databases is often described as persistent data because it is saved on disk and exists after the application process ends. Most business applications store data in relational databases. Applications can access database information by using the Java Database Connectivity (JDBC) API.

The key elements of database connectivity are the following:

- **Database.** The repository where data is stored for an enterprise. Java EE applications access relational databases through the JDBC API. For administration procedures, see "Setting Up the Database" on page 216.
- **JDBC Connection Pool.** A JDBC connection pool is a group of reusable connections for a particular database. For administration procedures, see "Administering JDBC Connection Pools" on page 220.

- **JDBC Resource.** A JDBC resource (data source) provides applications with a means of connecting to a database. To create a JDBC resource, specify the connection pool with which it is associated. Multiple JDBC resources can specify a single connection pool. A JDBC resource is identified by its Java Naming and Directory Interface (JNDI) name. For administration procedures, see "Administering JDBC Resources" on page 228.
- **JDBC Driver.** A database driver is a software component that enables a Java application to interact with a database connectivity API . Each database requires its own driver. For administration procedures, see "Integrating the JDBC Driver" on page 231.

At runtime, the following sequence occurs when an application connects to a database:

1. The application gets the JDBC resource associated with the database by making a call through the JNDI API.

   Using the JNDI name of the resource, the naming and directory service locates the JDBC resource. Each JDBC resource specifies a connection pool.
2. Using the JDBC resource, the application gets a database connection.

   GlassFish Server retrieves a physical connection from the connection pool that corresponds to the database. The pool defines connection attributes such as the database name (URL), user name, and password.
3. After the database connection is established, the application can read, modify, and add data to the database.

   The application accesses the database by making calls to the JDBC API. The JDBC driver translates the application's JDBC calls into the protocol of the database server.
4. When the application is finished accessing the database, the application closes the connection and returns the connection to the connection pool.

Setting Up the Database

Most applications use relational databases to store, organize, and retrieve data. Applications access relational databases through the Java Database Connectivity (JDBC) API.

The following topics are addressed here:

- "To Install the Database and Database Driver" on page 217
- "To Start the Database" on page 217
- "To Stop the Database" on page 218
- "Java DB Utility Scripts" on page 218

▼ To Install the Database and Database Driver

1 **Install a supported database product.**

To see the current list of database products supported by GlassFish Server, refer to the *GlassFish Server Open Source Edition 3.1 Release Notes*.

2 **Install a supported JDBC driver for the database product.**

For a list of drivers supported by GlassFish Server, see "Configuration Specifics for JDBC Drivers" on page 232.

3 **Make the JDBC driver JAR file accessible to the domain administration server (DAS).**

See "Integrating the JDBC Driver" on page 231.

4 **Create the database.**

The application provider usually delivers scripts for creating and populating the database.

Next Steps You are now ready to create a connection pool for the database, and a JDBC resource that points to the connection pool. See "To Create a JDBC Connection Pool" on page 220 and "To Create a JDBC Resource" on page 229. The final step is to integrate the JDBC driver into an administrative domain as described in "Integrating the JDBC Driver" on page 231.

▼ To Start the Database

GlassFish Server includes an implementation of Java DB (formerly known as Derby), however, you can use any JDBC-compliant database. The database is not started automatically when you start GlassFish Server, so if you have applications that require a database, you need to start Java DB manually by using the local `start-database` subcommand.

● **Start the database by using the `start-database`(1) subcommand.**

When the database server starts, or a client connects to it successfully, the following files are created at the location that is specified by the `--dbhome` option:

- The `derby.log` file contains the database server process log along with its standard output and standard error information.
- The database files contain your schema (for example, database tables).

Example 11–1 Starting a Database

This example starts Derby on the host host1 and port 5001.

```
asadmin> start-database --dbhost host1 --dbport 5001 --terse=true
Starting database in the background.
Log redirected to /opt/SUNWappserver/databases/javadb.log.
Command start-database executed successfully.
```

See Also You can also view the full syntax and options of the subcommand by typing `asadmin help start-database` at the command line.

▼ To Stop the Database

Use the local `stop-database` subcommand to stop Java DB on a specified port. A single host can have multiple database server processes running on different ports.

1 **If necessary, notify users that the database is being stopped.**

2 **Stop the database by using the `stop-database`(1) subcommand.**

Example 11–2 Stopping a Database

This example stops Java DB on port 5001 of `localhost`.

```
asadmin> stop-database --dbhost=localhost --dbport=5001
onnection obtained for host: localhost, port number 5001.
Apache Derby Network Server - 10.2.2.1 - (538595) shutdown
at 2008-10-17 23:34:2 7.218 GMT
Command stop-database executed successfully.
```

Troubleshooting For a laptop that roams between networks, you might have trouble shutting down the database. If you start Java DB and then change your IP address, you will not be able to stop Java DB unless you add a specific `--dbhost` argument. For example, if you run `asadmin start-database --dbhost = 0.0.0.0`, and then disconnect Ethernet and switch to wifi, you should run a command similar to the following to stop the database:

```
asadmin stop-database --dbhost localhost
```

See Also You can also view the full syntax and options of the subcommand by typing `asadmin help stop-database` at the command line.

Java DB Utility Scripts

The Java DB configuration that is available for use with GlassFish Server includes scripts that can help you use Java DB. The following scripts are available in the *as-install*/`javadb/frameworks/NetworkServer/bin` directory:

`startNetworkServer`,`startNetworkServer.bat`
: Script to start the network server

`stopNetworkServer`,`stopNetworkServer.bat`
: Script to stop the network server

`ij`,`ij.bat`
: Interactive JDBC scripting tool

`dblook`,`dblook.bat`
: Script to view all or part of the DDL for a database

`sysinfo`, `sysinfo.bat`
: Script to display versioning information about the Java DB environment

`NetworkServerControl`,`NetworkServerControl.bat`
: Script to execute commands on the `NetworkServerControl` API

▼ To Configure Your Environment to Run Java DB Utility Scripts

1. **Ensure that the `JAVA_HOME` environment variable specifies the directory where the JDK is installed.**

2. **Set the `JAVADB_HOME` environment variable to point to the *as-install*`/derby` directory.**

See Also For more information about these utilities, see the following documentation:

- *Derby Tools and Utilities Guide* (`http://db.apache.org/derby/docs/10.6/tools/`)
- (*Derby Server and Administration Guide* (`http://db.apache.org/derby/docs/10.6/adminguide/`))

Configuring Access to the Database

After establishing the database, you are ready to set up access for GlassFish Server applications. The high-level steps include creating a JDBC connection pool, creating a JDBC resource for the connection pool, and integrating a JDBC driver into an administrative domain.

Instructions for performing these steps are contained in the following sections:

- "Administering JDBC Connection Pools" on page 220
- "Administering JDBC Resources" on page 228
- "Integrating the JDBC Driver" on page 231

Administering JDBC Connection Pools

A *JDBC connection pool* is a group of reusable connections for a particular database. Because creating each new physical connection is time consuming, GlassFish Server maintains a pool of available connections. When an application requests a connection, it obtains one from the pool. When an application closes a connection, the connection is returned to the pool. JDBC connection pools can be globally accessible or be scoped to an enterprise application, web module, EJB module, connector module or application client module, as described in "Application-Scoped Resources" in *GlassFish Server Open Source Edition 3.1 Application Deployment Guide*.

A JDBC resource is created by specifying the connection pool with which the resource is associated. Multiple JDBC resources can specify a single connection pool. The properties of connection pools can vary with different database vendors. Some common properties are the database name (URL), the user name, and the password.

The following tasks and information are used to administer JDBC connection pools:

- "To Create a JDBC Connection Pool" on page 220
- "To List JDBC Connection Pools" on page 221
- "To Contact (Ping) a Connection Pool" on page 222
- "To Reset (Flush) a Connection Pool" on page 222
- "To Update a JDBC Connection Pool" on page 223
- "To Delete a JDBC Connection Pool" on page 224
- "Configuring Specific JDBC Connection Pool Features" on page 224

▼ To Create a JDBC Connection Pool

Use the `create-jdbc-connection-pool` subcommand in remote mode to register a new JDBC connection pool with the specified JDBC connection pool name. A JDBC connection pool or a connector connection pool can be created with authentication. You can either use a subcommand option to specify user, password, or other connection information using the `asadmin` utility, or specify the connection information in the XML descriptor file.

One connection pool is needed for each database, possibly more depending on the application. When you are building the connection pool, certain data specific to the JDBC driver and the database vendor is required. You can find some of the following specifics in"Configuration Specifics for JDBC Drivers" on page 232:

- Database vendor name
- Resource type, such as `javax.sql.DataSource` (local transactions only) `javax.sql.XADataSource` (global transactions)
- Data source class name
- Required properties, such as the database name (URL), user name, and password

Creating a JDBC connection pool is a dynamic event and does not require server restart. However, there are some parameters that do require server restart. See "Configuration Changes That Require Server Restart" on page 35.

Before You Begin Before creating the connection pool, you must first install and integrate the database and its associated JDBC driver. For instructions, see "Setting Up the Database" on page 216.

1 **Ensure that the server is running.**

Remote subcommands require a running server.

2 **Create the JDBC connection pool by using the `create-jdbc-connection-pool`(1) subcommand.**

3 **(Optional) If needed, restart the server.**

Some parameters require server restart. See "Configuration Changes That Require Server Restart" on page 35.

Example 11–3 Creating a JDBC Connection Pool

This example creates a JDBC connection pool named `sample_derby_pool` on `localhost`.

```
asadmin> create-jdbc-connection-pool
--datasourceclassname org.apache.derby.jdbc.ClientDataSource
--restype javax.sql.XADataSource
--property portNumber=1527:password=APP:user=APP:serverName=
localhost:databaseName=sun-appserv-samples:connectionAttribut
es=\;create\\=true sample_derby_pool
Command create-jdbc-connection-pool executed successfully.
```

See Also You can also view the full syntax and options of the subcommand by typing `asadmin help create-jdbc-connection-pool` at the command line.

▼ To List JDBC Connection Pools

Use the `list-jdbc-connection-pools` subcommand in remote mode to list all existing JDBC connection pools.

1 **Ensure that the server is running.**

Remote subcommands require a running server.

2 **List the JDBC connection pools by using the `list-jdbc-connection-pools`(1) subcommand.**

Example 11–4 Listing JDBC Connection Pools

This example lists the JDBC connection pools that are on `localhost`.

```
asadmin> list-jdbc-connection-pools
sample_derby_pool2
poolA
__TimerPool
DerbyPool
sample_derby_pool
Command list-jdbc-connection-pools executed successfully.
```

See Also You can also view the full syntax and options of the subcommand by typing `asadmin help list-jdbc-connection-pools` at the command line.

▼ To Contact (Ping) a Connection Pool

Use the `ping-connection-pool` subcommand in remote mode to test if a connection pool is usable. For example, if you create a new JDBC connection pool for an application that is expected to be deployed later, you can test the JDBC pool with this subcommand before the application is deployed. Running a ping will force the creation of the pool if it hasn't already been created.

Before You Begin Before you can contact a connection pool, the connection pool must be created with authentication, and the server or database must be running.

1 Ensure that the server is running.

Remote subcommands require a running server.

2 Ping a connection pool by using the `ping-connection-pool`(1) subcommand.

Example 11–5 Contacting a Connection Pool

This example tests to see if the `DerbyPool` connection pool is usable.

```
asadmin> ping-connection-pool DerbyPool
Command ping-connection-pool executed successfully
```

See Also You can also view the full syntax and options of the subcommand by typing `asadmin help ping-connection-pool` at the command line.

You can also specify that a JDBC connection pool is automatically tested when created or reconfigured by setting its `--ping` option to `true` (the default is `false`). See "To Create a JDBC Connection Pool" on page 220 or "To Update a JDBC Connection Pool" on page 223.

▼ To Reset (Flush) a Connection Pool

Use the `flush-connection-pool` in remote mode to reinitialize all connections established in the specified connection pool without the need for reconfiguring the pool. Connection pool reconfiguration can result in application redeployment, which is a time-consuming operation.

The JDBC connection pool or connector connection pool is reset to its initial state. Any existing live connections are destroyed, which means that the transactions associated with these connections are lost and must be retried. The subcommand then recreates the initial connections for the pool, and restores the pool to its steady pool size.

1. **Ensure that the server is running.**

   Remote subcommands require a running server.

2. **Reset a connection pool by using the`flush-connection-pool`(1) subcommand.**

Example 11–6 Resetting (Flushing) a Connection Pool

This example resets the JDBC connection pool named `__TimerPool` to its steady pool size.

```
asadmin> flush-connection-pool __TimerPool
Command flush-connection-pool executed successfully.
```

See Also You can also view the full syntax and options of the subcommand by typing `asadmin help flush-connection-pool` at the command line.

▼ To Update a JDBC Connection Pool

You can change all of the settings for an existing pool except its name. Use the `get` and `set` subcommands to view and change the values of the JDBC connection pool properties.

1. **List the JDBC connection pools by using the `list-jdbc-connection-pools`(1) subcommand.**

2. **View the attributes of the JDBC connection pool by using the get subcommand.**

   For example:

   ```
   asadmin get resources.jdbc-connection-pool.DerbyPool.property
   ```

3. **Set the attribute of the JDBC connection pool by using the set subcommand.**

   For example:

   ```
   asadmin set resources.jdbc-connection-pool.DerbyPool.steady-pool-size=9
   ```

4. **(Optional) If needed, restart the server.**

   Some parameters require server restart. See “Configuration Changes That Require Server Restart” on page 35.

See Also For information about how to tune a connection pool, see the *Sun Java System Application Server 9.1 Performance Tuning Guide*.

▼ To Delete a JDBC Connection Pool

Use the `delete-jdbc-connection-pool` subcommand in remote mode to delete an existing JDBC connection pool. Deleting a JDBC connection pool is a dynamic event and does not require server restart.

Before You Begin Before deleting a JDBC connection pool, all associations to the resource must be removed.

1. **Ensure that the server is running.**

   Remote subcommands require a running server.

2. **List the JDBC connection pools by using the `list-jdbc-connection-pools`(1) subcommand.**

3. **If necessary, notify users that the JDBC connection pool is being deleted.**

4. **Delete the connection pool by using the `delete-jdbc-connection-pool`(1) subcommand.**

Example 11–7 Deleting a JDBC Connection Pool

This example deletes the JDBC connection pool named `DerbyPool`.

```
asadmin> delete-jdbc-connection-pool jdbc/DerbyPool
Command delete-jdbc-connection-pool executed successfully.
```

See Also You can also view the full syntax and options of the subcommand by typing `asadmin help delete-jdbc-connection-pool` at the command line.

Configuring Specific JDBC Connection Pool Features

In GlassFish Server, JDBC Connection Pools support a variety of features to simplify administration, monitoring and performance tuning. The following topics address several of these features:

- "Transparent Pool Reconfiguration" on page 224
- "Using an Initialization Statement" on page 225
- "Setting a Statement Timeout" on page 225
- "Statement Leak Detection and Leaked Statement Reclamation" on page 226
- "Statement Caching" on page 227
- "Statement Tracing" on page 227

Transparent Pool Reconfiguration

When the properties or attributes of a JDBC connection pool are changed, the connection pool is destroyed and re-created. Normally, applications using the connection pool must be redeployed as a consequence. This restriction can be avoided by enabling transparent JDBC connection pool reconfiguration. When this feature is enabled, applications do not need to be redeployed. Instead, requests for new connections are blocked until the reconfiguration operation completes. Connection requests from any in-flight transactions are served using the old pool configuration so as to complete the transaction. Then, connections are created using the pool's new configuration, and any blocked connection requests are served with connections from the re-created pool..

To enable transparent JDBC connection pool reconfiguration, set the `dynamic-reconfiguration-wait-timeout-in-seconds` property of the JDBC connection pool to a positive, nonzero value in one of the following ways:

- Add it as a property in the Edit JDBC Connection Pool Properties page in the Administration Console. For more information, click the Help button in the Administration Console.
- Specify it using the `--property` option in the `create-jdbc-connection-pool` subcommand. For more information, see `create-jdbc-connection-pool`(1).
- Set it using the `set` subcommand. For example:

```
asadmin set resources.jdbc-connection-pool.pool-name.property.dynamic-reconfiguration-wait-timeout-in-seconds=15
```

This property specifies the time in seconds to wait for in-use connections to close and in-flight transactions to complete. Any connections in use or transaction in flight past this time must be retried.

Using an Initialization Statement

You can specify a statement that executes each time a physical connection to the database is created (not reused) from a JDBC connection pool. This is useful for setting request or session specific properties and is suited for homogeneous requests in a single application. Set the Init SQL attribute of the JDBC connection pool to the SQL string to be executed in one of the following ways:

- Enter an Init SQL value in the Edit Connection Pool Advanced Attributes page in the Administration Console. For more information, click the Help button in the Administration Console.
- Specify the `--initsql` option in the `asadmin` `create-jdbc-connection-pool` command. For more information, see `create-jdbc-connection-pool`(1).
- Specify the `init-sql` option in the `asadmin` `set` command. For example:

  ```
  asadmin set domain1.resources.jdbc-connection-pool.DerbyPool.init-sql="sql-string"
  ```

Setting a Statement Timeout

An abnormally long running JDBC query executed by an application may leave it in a hanging state unless a timeout is explicitly set on the statement. Setting a statement timeout guarantees that all queries automatically time out if not completed within the specified period. When statements are created, the queryTimeout is set according to the statement timeout setting. This works only when the underlying JDBC driver supports queryTimeout for Statement, PreparedStatement, CallableStatement, and ResultSet.

You can specify a statement timeout in the following ways:

- Enter a Statement Timeout value in the Edit Connection Pool Advanced Attributes page in the Administration Console. For more information, click the Help button in the Administration Console.
- Specify the --statementtimeout option in the asadmin create-jdbc-connection-pool command. For more information, see create-jdbc-connection-pool(1).

Statement Leak Detection and Leaked Statement Reclamation

If statements are not closed by an application after use, it is possible for the application to run out of cursors. Enabling statement leak detection causes statements to be considered as leaked if they are not closed within a specified period. Additionally, leaked statements can reclaimed automatically.

To enable statement leak detection, set Statement Leak Timeout In Seconds for the JDBC connection pool to a positive, nonzero value in one of the following ways:

- Specify the --statementleaktimeout option in the create-jdbc-connection-pool subcommand. For more information, see create-jdbc-connection-pool(1).
- Specify the statement-leak-timeout-in-seconds option in the set subcommand. For example:

```
asadmin set resources.jdbc-connection-pool.pool-name.statement-leak-timeout-in-seconds=300
```

When selecting a value for Statement Leak Timeout In Seconds, make sure that:

- It is less than the Connection Leak Timeout; otherwise, the connection could be closed before the statement leak is recognized.
- It is greater than the Statement Timeout; otherwise, a long running query could be mistaken as a statement leak.

After enabling statement leak detection, enable leaked statement reclamation by setting Reclaim Leaked Statements for the JDBC connection pool to a true value in one of the following ways:

- Specify the --statementleakreclaim=true option in the create-jdbc-connection-pool subcommand. For more information, see create-jdbc-connection-pool(1).
- Specify the statement-leak-reclaim option in the set subcommand. For example:

  ```
  asadmin set resources.jdbc-connection-pool.pool-name.statement-leak-reclaim=true
  ```

Statement Caching

Statement caching stores statements, prepared statements, and callable statements that are executed repeatedly by applications in a cache, thereby improving performance. Instead of the statement being prepared each time, the cache is searched for a match. The overhead of parsing and creating new statements each time is eliminated.

Statement caching is usually a feature of the JDBC driver. The GlassFish Server provides caching for drivers that do not support caching. To enable this feature, set the Statement Cache Size for the JDBC connection pool in one of the following ways:

- Enter a Statement Cache Size value in the Edit Connection Pool Advanced Attributes page in the Administration Console. For more information, click the Help button in the Administration Console.
- Specify the `--statementcachesize` option in the `asadmin` `create-jdbc-connection-pool` command. For more information, see `create-jdbc-connection-pool`(1).
- Specify the `statement-cache-size` option in the `asadmin` `set` command. For example:

  ```
  asadmin set domain1.resources.jdbc-connection-pool.DerbyPool.statement-cache-size=10
  ```

By default, this attribute is set to zero and the statement caching is turned off. To enable statement caching, you can set any positive nonzero value. The built-in cache eviction strategy is LRU-based (Least Recently Used). When a connection pool is flushed, the connections in the statement cache are recreated.

Statement Tracing

You can trace the SQL statements executed by applications that use a JDBC connection pool. Set the SQL Trace Listeners attribute to a comma-separated list of trace listener implementation classes in one of the following ways:

- Enter an SQL Trace Listeners value in the Edit Connection Pool Advanced Attributes page in the Administration Console. For more information, click the Help button in the Administration Console.
- Specify the `--sqltracelisteners` option in the `asadmin` `create-jdbc-connection-pool` command. For more information, see `create-jdbc-connection-pool`(1).
- Specify the `sql-trace-listeners` option in the `asadmin` `set` command. For example:

```
asadmin set domain1.resources.jdbc-connection-pool.DerbyPool.sql-trace-listeners=listeners
```

The GlassFish Server provides a public interface, org.glassfish.api.jdbc.SQLTraceListener, that implements a means of recording `SQLTraceRecord` objects. To make custom implementations of this interface available to the GlassFish Server, place the implementation classes in *as-install*`/lib`.

The GlassFish Server provides an SQL tracing logger to log the SQL operations in the form of `SQLTraceRecord` objects in the `server.log` file. The module name under which the SQL operation is logged is `javax.enterprise.resource.sqltrace`. SQL traces are logged as FINE messages along with the module name to enable easy filtering of the SQL logs. A sample SQL trace record looks like this:

```
[#|2009-11-27T15:46:52.202+0530|FINE|glassfishv3.0|javax.enterprise.resource.sqltrace.com.sun.gjc.util
|_ThreadID=29;_ThreadName=Thread-1;ClassName=com.sun.gjc.util.SQLTraceLogger;MethodName=sqlTrace;
|ThreadID=77 | ThreadName=p: thread-pool-1; w: 6 | TimeStamp=1259317012202
| ClassName=com.sun.gjc.spi.jdbc40.PreparedStatementWrapper40 | MethodName=executeUpdate
| arg[0]=insert into table1(colName) values(100) | arg[1]=columnNames | |#]
```

This trace shows that an `executeUpdate(String sql, String columnNames)` operation is being done.

When SQL statement tracing is enabled and JDBC connection pool monitoring is enabled, GlassFish Server maintains a tracing cache of recent queries and their frequency of use. The following JDBC connection pool properties can be configured to control this cache and the monitoring statistics available from it:

`time-to-keep-queries-in-minutes`
: Specifies how long in minutes to keep a query in the tracing cache, tracking its frequency of use. The default value is 5 minutes.

`number-of-top-queries-to-report`
: Specifies how many of the most used queries, in frequency order, are listed the monitoring report. The default value is 10 queries.

Set these parameters in one of the following ways:

- Add them as properties in the Edit JDBC Connection Pool Properties page in the Administration Console. For more information, click the Help button in the Administration Console.
- Specify them using the `--property` option in the `create-jdbc-connection-pool` subcommand. For more information, see `create-jdbc-connection-pool`(1).
- Set them using the `set` subcommand. For example:

```
asadmin set resources.jdbc-connection-pool.pool-name.property.time-to-keep-queries-in-minutes=10
```

Administering JDBC Resources

A *JDBC resource*, also known as a data source, provides an application with a means of connecting to a database. Typically, you create a JDBC resource for each database that is accessed by the applications deployed in a domain. Multiple JDBC resources can be specified for a database. JDBC resources can be globally accessible or be scoped to an enterprise application, web module, EJB module, connector module or application client module, as described in "Application-Scoped Resources" in *GlassFish Server Open Source Edition 3.1 Application Deployment Guide*.

A JDBC resource is created by specifying the connection pool with which the resource will be associated . Use a unique Java Naming and Directory Interface (JNDI) name to identify the resource. For example, the JNDI name for the resource of a payroll database might be `java:comp/env/jdbc/payrolldb`.

The following tasks and information are used to administer JDBC resources:

- "To Create a JDBC Resource" on page 229
- "To List JDBC Resources" on page 230
- "To Update a JDBC Resource" on page 230
- "To Delete a JDBC Resource" on page 230

▼ To Create a JDBC Resource

Use the `create-jdbc-resource` subcommand in remote mode to create a JDBC resource. Creating a JDBC resource is a dynamic event and does not require server restart.

Because all JNDI names are in the `java:comp/env` subcontext, when specifying the JNDI name of a JDBC resource in the Administration Console, use only the `jdbc/`*name* format. For example, a payroll database might be specified as `jdbc/payrolldb`.

Before You Begin Before creating a JDBC resource, you must first create a JDBC connection pool. For instructions, see "To Create a JDBC Connection Pool" on page 220.

1 **Ensure that the server is running.**

Remote subcommands require a running server.

2 **Create a JDBC resource by using the `create-jdbc-resource`(1) subcommand.**

Information about properties for the subcommand is included in this help page.

3 **If necessary, notify users that the new resource has been created.**

Example 11–8 Creating a JDBC Resource

This example creates a JDBC resource named `DerbyPool`.

```
asadmin> create-jdbc-resource --connectionpoolid DerbyPool jdbc/DerbyPool
Command create-jdbc-resource executed successfully.
```

See Also You can also view the full syntax and options of the subcommand by typing `asadmin help create-jdbc-resource` at the command line.

▼ To List JDBC Resources

Use the `list-jdbc-resources` subcommand in remote mode to list the existing JDBC resources.

1 **Ensure that the server is running.**

Remote subcommands require a running server.

2 **List JDBC resources by using the `list-jdbc-resources`(1) subcommand.**

Example 11–9 Listing JDBC Resources

This example lists JDBC resources for `localhost`.

```
asadmin> list-jdbc-resources
jdbc/__TimerPool
jdbc/DerbyPool
jdbc/__default
jdbc1
Command list-jdbc-resources executed successfully.
```

See Also You can also view the full syntax and options of the subcommand by typing `asadmin help list-jdbc-resources` at the command line.

▼ To Update a JDBC Resource

You can enable or disable a JDBC resource by using the `set` subcommand. The JDBC resource is identified by its dotted name.

1 **List JDBC resources by using the `list-jdbc-resources`(1) subcommand.**

2 **Modify the values for the specified JDBC resource by using the `set`(1) subcommand.**

For example:

Example 11–10 Updating a JDBC Resource

This example changes the `res1` enabled setting to false.

```
asadmin>set resources.jdbc-resource.res1.enabled=false
```

▼ To Delete a JDBC Resource

Use the `delete-jdbc-resource` subcommand in remote mode to delete an existing JDBC resource. Deleting a JDBC resource is a dynamic event and does not require server restart.

Before You Begin Before deleting a JDBC resource, all associations with this resource must be removed.

1 **Ensure that the server is running.**

Remote subcommands require a running server.

2 **List JDBC resources by using the `list-jdbc-resources`(1) subcommand.**

3 **If necessary, notify users that the JDBC resource is being deleted.**

4 **Delete a JDBC resource by using the `delete-jdbc-resource`(1) subcommand.**

Example 11–11 Deleting a JDBC Resource

This example deletes a JDBC resource named `DerbyPool`.

```
asadmin> delete-jdbc-resource jdbc/DerbyPool
Command delete-jdbc-resource executed successfully.
```

See Also You can also view the full syntax and options of the subcommand by typing `asadmin help delete-jdbc-resource` at the command line.

Integrating the JDBC Driver

To use JDBC features, you must choose a JDBC driver to work with the GlassFish Server, then you must set up the driver. This section covers these topics:

- "Supported Database Drivers" on page 231
- "Making the JDBC Driver JAR Files Accessible" on page 231
- "Automatic Detection of Installed Drivers" on page 232

Supported Database Drivers

Supported JDBC drivers are those that have been fully tested by Oracle. For a list of the JDBC drivers currently supported by the GlassFish Server, see the *GlassFish Server Open Source Edition 3.1 Release Notes*. For configurations of supported and other drivers, see "Configuration Specifics for JDBC Drivers" on page 232.

Note – Because the drivers and databases supported by the GlassFish Server are constantly being updated, and because database vendors continue to upgrade their products, always check with Oracle technical support for the latest database support information.

Making the JDBC Driver JAR Files Accessible

To integrate the JDBC driver into a GlassFish Server domain, copy the JAR files into the *domain-dir*/`lib` directory, then restart the server. This makes classes accessible to all applications or modules deployed on servers that share the same configuration. For more information about GlassFish Server class loaders, see Chapter 2, "Class Loaders," in *GlassFish Server Open Source Edition 3.1 Application Development Guide*.

If you are using an Oracle database with EclipseLink extensions, copy the JAR files into the *domain-dir*`/lib/ext` directory, then restart the server. For details, see "Oracle Database Enhancements" in *GlassFish Server Open Source Edition 3.1 Application Development Guide*.

Automatic Detection of Installed Drivers

The Administration Console detects installed JDBC Drivers automatically when you create a JDBC connection pool. To create a JDBC connection pool using the Administration Console, open the Resources component, open the JDBC component, select Connection Pools, and click on the New button. This displays the New JDBC Connection Pool page.

Based on the Resource Type and Database Vendor you select on the New JDBC Connection Pool page, data source or driver implementation class names are listed in the Datasource Classname or Driver Classname field when you click on the Next button. When you choose a specific implementation class name on the next page, additional properties relevant to the installed JDBC driver are displayed in the Additional Properties section.

Configuration Specifics for JDBC Drivers

GlassFish Server is designed to support connectivity to any database management system by using a corresponding JDBC driver.

- "JDBC Drivers, Full Support" on page 232
- "JDBC Drivers, Limited Support" on page 238

JDBC Drivers, Full Support

The following JDBC driver and database combinations have been tested and are supported for container-managed persistence:

- "IBM DB2 Database Type 2 DataDirect JDBC Driver" on page 233
- "IBM DB2 Database Type 2 JDBC Driver" on page 233
- "Java DB/Derby Type 4 JDBC Driver" on page 234
- "Microsoft SQL Server Database Type 4 DataDirect JDBC Driver" on page 234
- "MySQL Server Database Type 4 DataDirect JDBC Driver" on page 235
- "MySQL Server Database Type 4 JDBC Driver" on page 235
- "Oracle 11 Database DataDirect JDBC Driver" on page 235
- "Oracle OCI Type 2 Driver for Oracle Databases" on page 236
- "Oracle 11 Database Thin Type 4 JDBC Driver" on page 236
- "PostgreSQL Type 4 JDBC Driver" on page 237

- "Sybase Database Type 4 DataDirect JDBC Driver" on page 238

To see the most current list of supported JDBC drivers, refer to the *GlassFish Server Open Source Edition 3.1 Release Notes*.

IBM DB2 Database Type 2 DataDirect JDBC Driver

The JAR file for DataDirect driver is `db2.jar`. Configure the connection pool using the following settings:

- **Name:** Use this name when you configure the JDBC resource later.
- **Resource Type:** Specify the appropriate value.
- **Database Vendor:** DB2
- **DataSource Classname:** `com.ddtek.jdbcx.db2.DB2DataSource`
- **Properties:**
  - **`serverName`** – Specify the host name or IP address of the database server.
  - **`portNumber`** – Specify the port number of the database server.
  - **`databaseName`** – Set as appropriate.
  - **`user`** – Set as appropriate.
  - **`password`** – Set as appropriate.

IBM DB2 Database Type 2 JDBC Driver

The JAR files for the DB2 driver are `db2jcc.jar`, `db2jcc_license_cu.jar`, and `db2java.zip`. Set your environment variables . For example:

```
LD_LIBRARY_PATH=/usr/db2user/sqllib/lib:${Java EE.home}/lib
DB2DIR=/opt/IBM/db2/V8.2
DB2INSTANCE=db2user
INSTHOME=/usr/db2user
VWSPATH=/usr/db2user/sqllib
THREADS_FLAG=native
```

Configure the connection pool using the following settings:

- **Name:** Use this name when you configure the JDBC resource later.
- **Resource Type:** Specify the appropriate value.
- **Database Vendor:** DB2
- **DataSource Classname:** `com.ibm.db2.jcc.DB2SimpleDataSource`

  DataDirect DataSource Classname: `com.ddtek.jdbcx.db2.DB2DataSource`
- **Properties:**
  - **`databaseName`** - Set as appropriate.
  - **`user`** – Set as appropriate.
  - **`password`** – Set as appropriate.

- **`driverType`** – Set to 2.
- **`deferPrepares`** – Set to `false`.

Java DB/Derby Type 4 JDBC Driver

The JAR file for the Java DB driver is `derbyclient.jar`. (Java DB is based upon Apache Derby.) Configure the connection pool using the following settings:

- **Name:** Use this name when you configure the JDBC resource later.
- **Resource Type:** Specify the appropriate value.
- **Database Vendor:** Java DB
- **DataSource Classname:** Specify one of the following:

  ```
  org.apache.derby.jdbc.ClientDataSource
  org.apache.derby.jdbc.ClientXADataSource
  ```
- **Properties:**
  - **`serverName`** – Specify the host name or IP address of the database server.
  - **`portNumber`** – Specify the port number of the database server if it is different from the default.
  - **`databaseName`** – Specify the name of the database.
  - **`user`** - Specify the database user.

    This is only necessary if Java DB is configured to use authentication. Java DB does *not* use authentication by default. When the user is provided, it is the name of the schema where the tables reside.
  - **`password`** – Specify the database password.

    This is only necessary if Java DB is configured to use authentication.

Microsoft SQL Server Database Type 4 DataDirect JDBC Driver

The JAR file for the DataDirect driver is `sqlserver.jar`. Configure the connection pool using the following settings:

- **Name:** Use this name when you configure the JDBC resource later.
- **Resource Type:** Specify the appropriate value.
- **Database Vendor:** Microsoft SQL Server
- **DataSource Classname:** `com.ddtek.jdbcx.sqlserver.SQLServerDataSource`
- **Properties:**
  - **`serverName`** – Specify the host name or IP address and the port of the database server.
  - **`portNumber`** – Specify the port number of the database server.
  - **`user`** – Set as appropriate.
  - **`password`** – Set as appropriate.

- **`selectMethod`** – Set to `cursor`.

MySQL Server Database Type 4 DataDirect JDBC Driver

The JAR file for the DataDirect driver is `mysql.jar`. Configure the connection pool using the following settings:

- **Name:** Use this name when you configure the JDBC resource later.
- **Resource Type:** Specify the appropriate value.
- **Database Vendor:** MySQL Server
- **DataSource:** `com.ddtek.jdbcx.mysql.MySQLDataSource`
- **Properties:**
  - **`serverName`** – Specify the host name or IP address and the port of the database server.
  - **`portNumber`** – Specify the port number of the database server.
  - **`user`** – Set as appropriate.
  - **`password`** – Set as appropriate.
  - **`selectMethod`** – Set to `cursor`.

MySQL Server Database Type 4 JDBC Driver

The JAR file for the MySQL driver is `mysql-connector-java-5.1.7-bin.jar`. Configure the connection pool using the following settings:

- **Name:** Use this name when you configure the JDBC resource later.
- **Resource Type:** Specify the appropriate value.
- **Database Vendor:** Microsoft SQL Server
- **DataSource Classname:**

  ```
  com.mysql.jdbc.jdbc2.optional.MysqlDataSource
  com.mysql.jdbc.jdbc2.optional.MysqlXADataSource
  ```
- **Properties:**
  - **`serverName`** – Specify the host name or IP address of the database server.
  - **`portNumber`** – Specify the port number of the database server.
  - **`databaseName`** – Set as appropriate.
  - **`user`** – Set as appropriate.
  - **`password`** – Set as appropriate.

Oracle 11 Database DataDirect JDBC Driver

The JAR file for the DataDirect driver is `oracle.jar`.

Note – To make the Oracle driver behave in a Java EE-compliant manner, you must set this system property as `true`: `oracle.jdbc.J2EE13Compliant=true`.

Configure the connection pool using the following settings:

- **Name:** Use this name when you configure the JDBC resource later.
- **Resource Type:** Specify the appropriate value.
- **Database Vendor:** Oracle
- **DataSource Classname:** `com.ddtek.jdbcx.oracle.OracleDataSource`
- **Properties:**
  - **`serverName`** – Specify the host name or IP address of the database server.
  - **`portNumber`** – Specify the port number of the database server.
  - **`user`** – Set as appropriate.
  - **`password`** – Set as appropriate.

Oracle OCI Type 2 Driver for Oracle Databases

The JAR file for the OCI Oracle driver is `ojdbc14.jar`. Make sure that the shared library is available through `LD_LIBRARY_PATH` and that the `ORACLE_HOME` property is set. Configure the connection pool using the following settings:

- **Name:** Use this name when you configure the JDBC resource later.
- **Resource Type:** Specify the appropriate value.
- **Database Vendor:** Oracle
- **DataSource Classname:** Specify one of the following:

  ```
  oracle.jdbc.pool.OracleDataSource
  oracle.jdbc.xa.client.OracleXADataSource
  ```
- **Properties:**
  - **`user`** – Set as appropriate.
  - **`password`** – Set as appropriate.

Oracle 11 Database Thin Type 4 JDBC Driver

The JAR file for the Oracle driver is `ojdbc6.jar`.

Note – When using this driver, keep in mind that you cannot insert more than 2000 bytes of data into a column. To circumvent this problem, use the OCI driver (JDBC type 2).

Note – To make the Oracle driver behave in a Java EE-compliant manner, you must set this system property as `true`: `oracle.jdbc.J2EE13Compliant=true`.

Configure the connection pool using the following settings:

- **Name:** Use this name when you configure the JDBC resource later.
- **Resource Type:** Specify the appropriate value.
- **Database Vendor:** Oracle
- **DataSource Classname:** Specify one of the following:

  ```
  oracle.jdbc.pool.OracleDataSource
  oracle.jdbc.xa.client.OracleXADataSource
  ```

  DataDirect DataSource Classname: `com.ddtek.jdbcx.oracle.OracleDataSource`
- **Properties:**
  - **`user`** – Set as appropriate.
  - **`password`** – Set as appropriate.

Note – For the Oracle thin driver, the `XAResource.recover` method repeatedly returns the same set of in-doubt Xids regardless of the input flag. According to the XA specifications, the Transaction Manager initially calls this method with `TMSTARTSCAN` and then with `TMNOFLAGS` repeatedly until no Xids are returned. The `XAResource.commit` method also has some issues.

To disable this GlassFish Server workaround, the `oracle-xa-recovery-workaround` property value must be set to `false`.

PostgreSQL Type 4 JDBC Driver

The JAR file for the PostgreSQL driver is `postgresql-8.4-701.jdbc4.jar`. Configure the connection pool using the following settings:

- **Name:** Use this name when you configure the JDBC resource later.
- **Resource Type:** Specify the appropriate value.
- **Database Vendor:** PostgreSQL Server
- **DataSource Classname:** `org.postgresql.ds.PGSimpleDataSource`
- **Properties:**
  - **`serverName`** – Specify the host name or IP address of the database server.
  - **`portNumber`** – Specify the port number of the database server.
  - **`databaseName`** – Set as appropriate.
  - **`user`** – Set as appropriate.

- `password` – Set as appropriate.

Sybase Database Type 4 DataDirect JDBC Driver

The JAR file for the DataDirect driver is `sybase.jar`. Configure the connection pool using the following settings:

- **Name:** Use this name when you configure the JDBC resource later.
- **Resource Type:** Specify the appropriate value.
- **Database Vendor:** Sybase
- **DataSource Classname:** `com.ddtek.jdbcx.sybase.SybaseDataSource`
- **Properties:**
  - `serverName` – Specify the host name or IP address of the database server.
  - `portNumber` – Specify the port number of the database server.
  - `databaseName` – Set as appropriate. This is optional.
  - `user` – Set as appropriate.
  - `password` – Set as appropriate.

JDBC Drivers, Limited Support

The following JDBC drivers can also be used with GlassFish Server, but have not been fully tested. Although Oracle offers no product support for these drivers, Oracle does offer limited support for the use of these drivers with GlassFish Server:

- "IBM Informix Type 4 Driver for DataDirect" on page 238
- "Inet Oraxo JDBC Driver for Oracle Databases" on page 239
- "Inet Merlia JDBC Driver for Microsoft SQL Server Databases" on page 239
- "Inet Sybelux JDBC Driver for Sybase Databases" on page 240
- "JConnect Type 4 Driver for Sybase ASE 12.5 Databases" on page 240

IBM Informix Type 4 Driver for DataDirect

Configure the connection pool using the following settings:

- **Name:** Use this name when you configure the JDBC resource later.
- **Resource Type:** Specify the appropriate value.
- **Database Vendor:** Informix
- **DataSource Classname:** Specify one of the following:

  ```
  com.informix.jdbcx.IfxDataSource
  com.informix.jdbcx.IfxXADataSource
  ```

  DataDirect DataSource Classname: `com.ddtek.jdbcx.informix.InformixDataSourcee`
- **Properties:**

- **`serverName`** – Specify the Informix database server name.
- **`portNumber`** – Specify the port number of the database server.
- **`databaseName`** – Set as appropriate. This is optional.
- **`user`** – Set as appropriate.
- **`password`** – Set as appropriate.
- **`IfxIFXHost`** – Specify the host name or IP address of the database server.

Inet Oraxo JDBC Driver for Oracle Databases

The JAR file for the Inet Oracle driver is `Oranxo.jar`. Configure the connection pool using the following settings:

- **Name:** Use this name when you configure the JDBC resource later.
- **Resource Type:** Specify the appropriate value.
- **Database Vendor:** Oracle
- **DataSource Classname:** `com.inet.ora.OraDataSource`
- **Properties:**
  - **`serverName`** – Specify the host name or IP address of the database server.
  - **`portNumber`** – Specify the port number of the database server.
  - **`user`** – Specify the database user.
  - **`password`** – Specify the database password.
  - **`serviceName`** – Specify the URL of the database. The syntax is as follows:

    `jdbc:inetora:`*server*`:`*port*`:`*dbname*

    For example:

    ```
    jdbc:inetora:localhost:1521:payrolldb
    ```

    In this example,`localhost` is the name of the host running the Oracle server, `1521` is the Oracle server's port number, and `payrolldb` is the SID of the database. For more information about the syntax of the database URL, see the Oracle documentation.
  - **`streamstolob`** - If the size of BLOB or CLOB data types exceeds 4 KB and this driver is used for CMP, this property must be set to `true`.

Inet Merlia JDBC Driver for Microsoft SQL Server Databases

The JAR file for the Inet Microsoft SQL Server driver is `Merlia.jar`. Configure the connection pool using the following settings:

- **Name:** Use this name when you configure the JDBC resource later.
- **Resource Type:** Specify the appropriate value.
- **Database Vendor:** Microsoft SQL Server
- **DataSource Classname:** `com.inet.tds.TdsDataSource`

- **Properties:**
  - `serverName` – Specify the host name or IP address and the port of the database server.
  - `portNumber` – Specify the port number of the database server.
  - `user` – Set as appropriate.
  - `password` – Set as appropriate.

Inet Sybelux JDBC Driver for Sybase Databases

The JAR file for the Inet Sybase driver is `Sybelux.jar`. Configure the connection pool using the following settings:

- **Name:** Use this name when you configure the JDBC resource later.
- **Resource Type:** Specify the appropriate value.
- **Database Vendor:** Sybase
- **DataSource Classname:** `com.inet.syb.SybDataSource`
- **Properties:**
  - `serverName` – Specify the host name or IP address of the database server.
  - `portNumber` – Specify the port number of the database server.
  - `databaseName` – Set as appropriate. Do not specify the complete URL, only the database name.
  - `user` – Set as appropriate.
  - `password` – Set as appropriate.

JConnect Type 4 Driver for Sybase ASE 12.5 Databases

The JAR file for the Sybase driver is `jconn4.jar`. Configure the connection pool using the following settings:

- **Name:** Use this name when you configure the JDBC resource later.
- **Resource Type:** Specify the appropriate value.
- **Database Vendor:** Sybase
- **DataSource Classname:** Specify one of the following:

  ```
  com.sybase.jdbc4.jdbc.SybDataSource
  com.sybase.jdbc4.jdbc.SybXADataSource
  ```
- **Properties:**
  - `serverName` – Specify the host name or IP address of the database server.
  - `portNumber` – Specify the port number of the database server.
  - `databaseName` – Set as appropriate. Do not specify the complete URL, only the database name.

- **`user`** – Set as appropriate.
- **`password`** – Set as appropriate.
- **`BE_AS_JDBC_COMPLIANT_AS_POSSIBLE`** – Set to `true`.
- **`FAKE_METADATA`** – Set to `true`.