# Contents

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Copyright</td>
<td>4</td>
</tr>
<tr>
<td>License</td>
<td>5</td>
</tr>
<tr>
<td>About this guide</td>
<td>9</td>
</tr>
<tr>
<td>Purpose of this guide</td>
<td>9</td>
</tr>
<tr>
<td>Audience</td>
<td>9</td>
</tr>
<tr>
<td>How this guide is organized</td>
<td>9</td>
</tr>
<tr>
<td>Part one: Derby Server Guide</td>
<td>11</td>
</tr>
<tr>
<td>Derby in a multi-user environment</td>
<td>11</td>
</tr>
<tr>
<td>About this guide and the Network Server documentation</td>
<td>14</td>
</tr>
<tr>
<td>Using the Network Server with preexisting Derby applications</td>
<td>14</td>
</tr>
<tr>
<td>The Network Server and JVMs</td>
<td>14</td>
</tr>
<tr>
<td>Installing required jar files and adding them to the classpath</td>
<td>14</td>
</tr>
<tr>
<td>Starting the Network Server</td>
<td>15</td>
</tr>
<tr>
<td>Shutting down the Network Server</td>
<td>17</td>
</tr>
<tr>
<td>Obtaining system information</td>
<td>18</td>
</tr>
<tr>
<td>Accessing the Network Server by using the network client driver</td>
<td>19</td>
</tr>
<tr>
<td>Accessing the Network Server by using a DataSource</td>
<td>28</td>
</tr>
<tr>
<td>XA and the Network Server</td>
<td>28</td>
</tr>
<tr>
<td>Using the Derby tools with the Network Server</td>
<td>29</td>
</tr>
<tr>
<td>Differences between running Derby in embedded mode and using the Network Server</td>
<td>30</td>
</tr>
<tr>
<td>Setting port numbers</td>
<td>33</td>
</tr>
<tr>
<td>Managing the Derby Network Server</td>
<td>33</td>
</tr>
<tr>
<td>Overview</td>
<td>33</td>
</tr>
<tr>
<td>Setting Network Server properties</td>
<td>34</td>
</tr>
<tr>
<td>Verifying Startup</td>
<td>39</td>
</tr>
<tr>
<td>Managing the Derby Network Server remotely by using the servlet interface</td>
<td>39</td>
</tr>
<tr>
<td>Start-up page</td>
<td>40</td>
</tr>
<tr>
<td>Running page</td>
<td>40</td>
</tr>
<tr>
<td>Trace session page</td>
<td>41</td>
</tr>
<tr>
<td>Trace directory page</td>
<td>41</td>
</tr>
<tr>
<td>Set Network Server parameters</td>
<td>41</td>
</tr>
<tr>
<td>Derby Network Server advanced topics</td>
<td>41</td>
</tr>
<tr>
<td>Network Server security</td>
<td>41</td>
</tr>
<tr>
<td>Running the Network Server under the security manager</td>
<td>42</td>
</tr>
<tr>
<td>Network encryption and authentication with SSL/TLS</td>
<td>45</td>
</tr>
<tr>
<td>Configuring the Network Server to handle connections</td>
<td>49</td>
</tr>
<tr>
<td>Controlling logging by using the log file</td>
<td>49</td>
</tr>
<tr>
<td>Controlling tracing by using the trace facility</td>
<td>50</td>
</tr>
<tr>
<td>Derby Network Server sample programs</td>
<td>50</td>
</tr>
<tr>
<td>The NsSample sample program</td>
<td>50</td>
</tr>
<tr>
<td>Network Server sample programs for embedded and client connections</td>
<td>53</td>
</tr>
<tr>
<td>Part two: Derby Administration Guide</td>
<td>56</td>
</tr>
<tr>
<td>Checking database consistency</td>
<td>56</td>
</tr>
<tr>
<td>The SYCS_CHECK_TABLE function</td>
<td>56</td>
</tr>
<tr>
<td>Sample SYCS_CHECK_TABLE error messages</td>
<td>56</td>
</tr>
<tr>
<td>Sample SYCS_CHECK_TABLE queries</td>
<td>57</td>
</tr>
<tr>
<td>Backing up and restoring databases</td>
<td>57</td>
</tr>
</tbody>
</table>
Backing up a database ................................................................................................. 57
Restoring a database from a backup copy................................................................. 60
Creating a database from a backup copy................................................................. 61
Roll-forward recovery .............................................................................................. 61

Logging on a separate device .................................................................................. 63
   Using the logDevice attribute ............................................................................. 63
   Example of creating a log in a non-default location ........................................... 64
   Example of moving a log manually ..................................................................... 64
   Issues for logging in a non-default location ....................................................... 64

Obtaining locking information .................................................................................. 64
   Monitoring deadlocks ......................................................................................... 64

Reclaiming unused space ...................................................................................... 65

Trademarks ............................................................................................................... 67
Copyright

Copyright 2004-2007 The Apache Software Foundation
Licensed under the Apache License, Version 2.0 (the "License"); you may not use this
file except in compliance with the License. You may obtain a copy of the License at
http://www.apache.org/licenses/LICENSE-2.0.

Related information
License
License

The Apache License, Version 2.0

Applies to Licenses, Version 2.0, January 2004
http://www.apache.org/licenses/

TERMS AND CONDITIONS FOR USE, REPRODUCTION, AND DISTRIBUTION

1. Definitions.

"License" shall mean the terms and conditions for use, reproduction, and distribution as defined by Sections 1 through 9 of this document.

"Licensor" shall mean the copyright owner or entity authorized by the copyright owner that is granting the License.

"Legal Entity" shall mean the union of the acting entity and all other entities that control, are controlled by, or are under common control with that entity. For the purposes of this definition, "control" means (i) the power, direct or indirect, to cause the direction or management of such entity, whether by contract or otherwise, or (ii) ownership of fifty percent (50%) or more of the outstanding shares, or (iii) beneficial ownership of such entity.

"You" (or "Your") shall mean an individual or Legal Entity exercising permissions granted by this License.

"Source" form shall mean the preferred form for making modifications, including but not limited to software source code, documentation source, and configuration files.

"Object" form shall mean any form resulting from mechanical transformation or translation of a Source form, including but not limited to compiled object code, generated documentation, and conversions to other media types.

"Work" shall mean the work of authorship, whether in Source or Object form, made available under the License, as indicated by a copyright notice that is included in or attached to the work (an example is provided in the Appendix below).

"Derivative Works" shall mean any work, whether in Source or Object form, that is based on (or derived from) the Work and for which the editorial revisions, annotations, elaborations, or other modifications represent, as a whole, an original work of authorship. For the purposes of this License, Derivative Works shall not include works that remain separable from, or merely link (or bind by name) to the interfaces of, the Work and Derivative Works thereof.

"Contribution" shall mean any work of authorship, including the original version of the Work and any modifications or additions to that Work or Derivative Works thereof, that is intentionally submitted to Licensor for inclusion in the Work by the copyright owner or by an individual or Legal Entity authorized to submit on behalf of the copyright owner. For the purposes of this definition, "submitted" means any form of electronic, verbal, or written communication sent to the Licensor or its representatives, including but not limited to communication on electronic mailing lists, source code control systems, and issue tracking systems.
that are managed by, or on behalf of, the Licensor for the purpose of discussing and improving the Work, but excluding communication that is conspicuously marked or otherwise designated in writing by the copyright owner as "Not a Contribution."

"Contributor" shall mean Licensor and any individual or Legal Entity on behalf of whom a Contribution has been received by Licensor and subsequently incorporated within the Work.

2. Grant of Copyright License. Subject to the terms and conditions of this License, each Contributor hereby grants to You a perpetual, worldwide, non-exclusive, no-charge, royalty-free, irrevocable copyright license to reproduce, prepare Derivative Works of, publicly display, publicly perform, sublicense, and distribute the Work and such Derivative Works in Source or Object form.

3. Grant of Patent License. Subject to the terms and conditions of this License, each Contributor hereby grants to You a perpetual, worldwide, non-exclusive, no-charge, royalty-free, irrevocable (except as stated in this section) patent license to make, have made, use, offer to sell, sell, import, and otherwise transfer the Work, where such license applies only to those patent claims licensable by such Contributor that are necessarily infringed by their Contribution(s) alone or by combination of their Contribution(s) with the Work to which such Contribution(s) was submitted. If You institute patent litigation against any entity (including a cross-claim or counterclaim in a lawsuit) alleging that the Work or a Contribution incorporated within the Work constitutes direct or contributory patent infringement, then any patent licenses granted to You under this License for that Work shall terminate as of the date such litigation is filed.

4. Redistribution. You may reproduce and distribute copies of the Work or Derivative Works thereof in any medium, with or without modifications, and in Source or Object form, provided that You meet the following conditions:

(a) You must give any other recipients of the Work or Derivative Works a copy of this License; and

(b) You must cause any modified files to carry prominent notices stating that You changed the files; and

(c) You must retain, in the Source form of any Derivative Works that You distribute, all copyright, patent, trademark, and attribution notices from the Source form of the Work, excluding those notices that do not pertain to any part of the Derivative Works; and

(d) If the Work includes a "NOTICE" text file as part of its distribution, then any Derivative Works that You distribute must include a readable copy of the attribution notices contained within such NOTICE file, excluding those notices that do not pertain to any part of the Derivative Works, in at least one of the following places: within a NOTICE text file distributed as part of the Derivative Works; within the Source form or documentation, if provided along with the Derivative Works; or, within a display generated by the Derivative Works, if and wherever such third-party notices normally appear. The contents of the NOTICE file are for informational purposes only and do not modify the License. You may add Your own attribution notices within Derivative Works that You distribute, alongside or as an addendum to the NOTICE text from the Work, provided that such additional attribution notices cannot be construed as modifying the License.

You may add Your own copyright statement to Your modifications
and may provide additional or different license terms and
conditions for use, reproduction, or distribution of Your
modifications, or for any such Derivative Works as a whole,
provided Your use, reproduction, and distribution of the Work
otherwise complies with the conditions stated in this License.

5. Submission of Contributions. Unless You explicitly state
otherwise, any Contribution intentionally submitted for
inclusion in the Work by You to the Licensor shall be under the
terms and conditions of this License, without any additional
terms or conditions. Notwithstanding the above, nothing herein
shall supersede or modify the terms of any separate license
agreement you may have executed with Licensor regarding such
Contributions.

6. Trademarks. This License does not grant permission to use the
trade names, trademarks, service marks, or product names of the
Licensor, except as required for reasonable and customary use
in describing the origin of the Work and reproducing the content
of the NOTICE file.

7. Disclaimer of Warranty. Unless required by applicable law or
agreed to in writing, Licensor provides the Work (and each
Contributor provides its Contributions) on an "AS IS" BASIS,
WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or
implied, including, without limitation, any warranties or
conditions of TITLE, NON-INFRINGEMENT, MERCHANTABILITY, or
FITNESS FOR A PARTICULAR PURPOSE. You are solely responsible for
determining the appropriateness of using or redistributing the
Work and assume any risks associated with Your exercise of
permissions under this License.

8. Limitation of Liability. In no event and under no legal theory,
whether in tort (including negligence), contract, or otherwise,
unless required by applicable law (such as deliberate and
grossly negligent acts) or agreed to in writing, shall any
Contributor be liable to You for damages, including any direct,
indirect, special, incidental, or consequential damages of any
character arising as a result of this License or out of the use
or inability to use the Work (including but not limited to
damages for loss of goodwill, work stoppage, computer failure or
malfunction, or any and all other commercial damages or losses),
even if such Contributor has been advised of the possibility of
such damages.

9. Accepting Warranty or Additional Liability. While redistributing
the Work or Derivative Works thereof, You may choose to offer,
and charge a fee for, acceptance of support, warranty, indemnity,
or other liability obligations and/or rights consistent with this
License. However, in accepting such obligations, You may act only
on Your own behalf and on Your sole responsibility, not on behalf
of any other Contributor, and only if You agree to indemnify,
defend, and hold each Contributor harmless for any liability
incurred by, or claims asserted against, such Contributor by
reason of your accepting any such warranty or additional
liability.

END OF TERMS AND CONDITIONS

APPENDIX: How to apply the Apache License to your work.

To apply the Apache License to your work, attach the following
boilerplate notice, with the fields enclosed by brackets "[]"
replaced with your own identifying information. (Don't include
the brackets!) The text should be enclosed in the appropriate
comment syntax for the file format. We also recommend that a
file or class name and description of purpose be included on the
same "printed page" as the copyright notice for easier
identification within third-party archives.
About this guide

This section describes who this guide is for as well as how to use it.

Purpose of this guide

This guide explains how to use Derby in a multiple-client environment. It also provides information that a server administrator might need to keep Derby running with a high level of performance and reliability in a server framework or in a multiple-client application server environment. (When running in embedded mode, Derby databases typically do not need any administration).

To connect multiple clients with Derby, you can embed Derby in a server framework that you choose, or you can use the Derby Network Server. This guide describes these options.

Audience

The first part of this guide is intended for developers of client/server and multiple-client applications. The second part of this guide is intended for administrators.

How this guide is organized

This guide includes the following two parts:

Part one: Derby Server Guide

- Derby in a multi-user environment
  Describes the different options for embedding Derby in a server framework and explains the Network Server option.
- Using the Network Server with preexisting Derby applications
  Describes how to change existing Derby applications to work with the Network Server.
- Managing the Derby Network Server
  Describes how to use shell scripts, the command line, and the Network Server API to manage the Network Server.
- Managing the Derby Network Server remotely by using the servlet interface
  Describes how to use the servlet interface to manage the Network Server.
- Derby Network Server advanced topics
  Describes advanced topics for Derby Network Server users.

Part two: Derby Administration Guide

- Checking database consistency
  Describes how to check the consistency of Derby databases.
- Backing up and restoring databases
  Describes how to back up a database when it is online.
- Logging on a separate device
  Describes how to put a database's log on a separate device, which can improve the performance of large databases.
- Obtaining locking information
Describes how to get detailed information about locking status.

- **Reclaiming unused space**

  Describes how to identify and reclaim unused space in tables and related indexes.
Part one: Derby Server Guide

This part of the guide explains the Derby Network Server and other server frameworks.

Derby in a multi-user environment

This section describes how to use Derby in a multi-user (or "server") environment.

Derby in a server framework

In a sense, Derby is always an embedded product. You can embed it in an application in which users access the database from a single JVM or you can embed it in a server framework (an application that allows users from different JVMs to connect to Derby simultaneously). When Derby is embedded in an application, the local JDBC driver calls the local Derby database. When Derby is embedded in a server framework, the server framework's connectivity software provides data to multiple client JDBC applications over a network or the Internet.

For local or remote multi-user connectivity (multiple users who access Derby from different JVMs), use the Derby Network Server. If you require features that are not included in the Network Server, you can embed the basic Derby product in another server framework.

Connectivity configurations

There are several ways to embed Derby in a server framework:

Use the Network Server
This is the easiest way to provide connectivity to multiple users who are accessing Derby databases from different JVMs. The Derby Network Server provides this kind of connectivity to Derby databases within a single system or over a network.

Purchase another server framework
You can use Derby within many server frameworks, such as IBM WebSphere Application Server.

Write your own framework
Derby's flexibility allows other configurations as well. For example, rather than embedding Derby in a server that communicates with a client that uses JDBC, you can embed Derby within a servlet in a web server that communicates with a browser using HTTP.

Multiple-client features available in Derby

Derby contains some features that are useful for developing multi-user applications.

Row-level locking:

To support multi-user access, Derby utilizes row-level locking. However, you can configure Derby to use table-level locking in environments that have few concurrent transactions (for example, a read-only database). Table-level locking is preferable if there are few or no writes to the server, while row-level locking is essential for good performance if many clients write to the server concurrently. The Derby optimizer tunes lock choice for queries automatically.

Multiple concurrency levels:

Derby supports SERIALIZABLE (RR), REPEATABLE (RS), READ COMMITTED (CS), and READ UNCOMMITTED (UR) isolation levels.
CS (the default isolation level) provides the best balance between concurrency and consistency in multiple-client environments.

RS
RS provides less consistency than RR but allows more concurrency.

RR
RR provides greatest consistency.

UR
UR provides maximum concurrency, if uncommitted values are allowed in the query. It is typically used if approximate results are acceptable.

See “Types and Scope of Locks in Derby Systems” in the *Derby Developer's Guide* for more information.

**Multi-connection and multi-threading:**

Derby allows multiple simultaneous connections to a database, even in embedded mode. Derby is also fully multi-threaded, and you can have multiple threads active at the same time. However, JDBC semantics impose some limitations on multi-threading. See the *Derby Developer's Guide* for more information.

**Administrative tools:**

Derby provides some tools and features to assist database administrators, including:

- Consistency checker
- Online backup
- The ability to put a database's log on a separate device

These tools and features are discussed in part two of this guide. See the sections in that part for more information.

**The Derby Network Server**

The Derby Network Server provides multi-user connectivity to Derby databases within a single system or over a network. The Network Server uses the standard Distributed Relational Database Architecture (DRDA) protocol to receive and reply to queries from clients. Databases are accessed through the Derby Network Server by using the Derby Network Client driver.

The Network Server is a solution for multiple JVMs that connect to the database, unlike the embedded scenario where only one JVM runs as part of the system. When Derby is embedded in a single-JVM application, the embedded JDBC driver calls the local Derby database. When Derby is embedded in a server framework, the server framework's connectivity software provides data to multiple client JDBC applications over a network or the Internet.

To run the Derby Network Server, you need to install the following files:

- On the server side, install `derby.jar` and `derbynet.jar`
- On the client side, install `derbyclient.jar`

There are several ways to manage the Derby Network Server, including:

- Through the command line
- By using .bat and .ksh scripts
- Through the servlet interface
- With your own Java program (written using the Network Server API)
- By setting Network Server properties

Using the Network Server with preexisting Derby applications explains how to change existing Java applications that currently run against Derby in embedded mode to run against the Derby Network Server.
Managing the Derby Network Server explains how to manage the Network Server by using the command line, including starting and stopping it.

Managing the Derby Network Server remotely by using the servlet interface explains how to use the servlet interface to manage the Network Server.

Derby Network Server advanced topics contains advanced topics for Derby Network Server users.

Because of the differences in JDBC drivers that are used, you might encounter differences in functionality when running Derby in the Network Server framework as opposed to running it embedded in a user application. Refer to Using the Network Server with preexisting Derby applications for a complete list of the differences between embedded and Network Server configurations.

Embedded servers

Because Derby is written in Java, you have great flexibility in how you choose to configure your deployment. For example, you can run Derby, the JDBC server framework, and another application in the same JVM as a single process.

How to start an embedded server from an application

In one thread, the embedding application starts the local JDBC driver for its own access.

```java
/*
   If you are running on JDK 1.6 or higher, then you do not
   need to invoke Class.forName(). In that environment, the
   EmbeddedDriver loads automatically.
*/
Class.forName("org.apache.derby.jdbc.EmbeddedDriver").newInstance();
Connection conn = DriverManager.getConnection("jdbc:derby:sample");
```

In another thread, the same application starts the server framework to allow remote access. Starting the server framework from within the application allows both the server and the application to run in the same JVM.

Embedded server example

You can start the Network Server in another thread automatically when Derby starts by setting the derby.drda.startNetworkServer property (see Setting Network Server properties), or you can start it by using a program. The following example shows how to start the Network Server by using a program:

```java
import org.apache.derby.drda.NetworkServerControl;
import java.net.InetAddress;
NetworkServerControl server = new NetworkServerControl
    (InetAddress.getByName("localhost"),1527);
server.start(null);
```

The program that starts the Network Server can access the database by using either the embedded driver or the Network Client driver. The server framework's attempt to boot the local JDBC driver is ignored because it has already been booted within the application's JVM. The server framework simply accesses the instance of Derby that is already booted. There is no conflict between the application and the server framework.

The remote client can then connect through the Derby client driver:

```java
String nsURL="jdbc:derby://localhost:1527/sample";
java.util.Properties props = new java.util.Properties();
props.put("user","usr");
props.put("password","pwd");
/*
```
If you are running on JDK 1.6 or higher, then you do not need to invoke Class.forName(). In that environment, the EmbeddedDriver loads automatically.

```
Class.forName("org.apache.derby.jdbc.ClientDriver").newInstance();
Connection conn = DriverManager.getConnection(nsURL, props);
```

/*interact with Derby*/
Statement s = conn.createStatement();
ResultSet rs = s.executeQuery("SELECT * FROM HotelBookings");

About this guide and the Network Server documentation

This guide assumes that you are familiar with Derby features and tuning. Before reading this guide, you should first learn about basic Derby functionality by reading the Derby Developer's Guide. Also, because multi-user environments typically have performance and tuning issues, you should read Tuning Derby.

Using the Network Server with preexisting Derby applications

You must modify Java applications that currently run against Derby in embedded mode so that they work with the Derby Network Server. The topics in this section discuss these changes.

The Network Server and JVMs

The Derby Network Server is compatible with Java(TM) 2 Platform, Standard Edition, v 1.4.2 (J2SE) and above.

Installing required jar files and adding them to the classpath

To use the Network Server and network client driver, add the following jar file to your server classpath:

- derbyrun.jar

Adding this file to your classpath has the effect of including all of the Derby classes in your classpath. These classes are in the following jar files, which you can also add to your classpath separately:

- derbynets.jar

  This jar file contains the Network Server code. It must be in your classpath to start the Network Server.

- derby.jar

  This jar file contains the Derby database engine code. It must be in the classpath in order for the Network Server to access Derby databases. derby.jar is included in the Class-Path attribute of derbynets.jar's manifest file. If you have derbynets.jar in the classpath and derby.jar is in the same directory as derbynets.jar, it is not necessary to include derby.jar explicitly.

- derbyclient.jar

  This jar file contains the Derby Network Client JDBC driver that is necessary for communication with the Network Server. It must be in the classpath of the application on the client side in order to access Derby databases over a network.

All of the jar files are in the $DERBY_HOME/lib directory.
Derby provides script files for setting the classpath to work with the Network Server. The scripts are located in the $DERBY_HOME/bin directory.

- `setNetworkClientCP.bat` (Windows)
- `setNetworkClientCP` (UNIX)
- `setNetworkServerCP.bat` (Windows)
- `setNetworkServerCP` (UNIX)

See Managing the Derby Network Server and Getting Started with Derby for more information on setting the classpath.

**Starting the Network Server**

To start the Network Server, you can invoke a script, a jar file, or a class.

> Important: Note that you should always properly shut down the Network Server after use, because failure to do so might result in unpredictable side-effects, such as blocked ports on the server.

You are strongly urged to enable user authentication when you run a Network Server. For details on how to configure user authentication, please consult the "Working with user authentication" section in the Developer's Guide. You are also urged to install a Java security manager with a customized security policy. For details on how to do this, see Customizing the Network Server’s security policy.

You can start the Network Server in any of the following ways:

- If you are relatively new to the Java programming language, follow the instructions in "Setting up your environment" in Getting Started with Derby to set the DERBY_HOME and JAVA_HOME environment variables and to add DERBY_HOME/bin to your path. Then use the `startNetworkServer.bat` script to start the Network Server on Windows machines and the `startNetworkServer` script to start the Network Server on UNIX systems. These scripts are located in $DERBY_HOME/bin, where $DERBY_HOME is the directory where you installed Derby.

You can run NetworkServerControl commands only from the host that started the Network Server.

<table>
<thead>
<tr>
<th>Operating System</th>
<th>Command</th>
</tr>
</thead>
</table>
| Windows          | `set DERBY_HOME=C:\derby`  
                  | `set JAVA_HOME=C:\Program Files\Java\jdk1.5.0_10`  
                  | `set PATH=%DERBY_HOME%\bin;%PATH%`  
                  | `startNetworkServer` |
| UNIX (Korn Shell)| `export DERBY_HOME=/opt/derby`  
                  | `export JAVA_HOME=/usr/j2se`  
                  | `export PATH="$DERBY_HOME/bin:$PATH"`  
                  | `startNetworkServer` |

- If you are a regular Java user but are new to Derby, set the DERBY_HOME environment variable, then use a java command to invoke the derbyrun.jar or derbynet.jar file:

<table>
<thead>
<tr>
<th>Operating System</th>
<th>Command</th>
</tr>
</thead>
</table>
| Windows          | `set DERBY_HOME=C:\derby`  
                  | `java -jar %DERBY_HOME%\lib\derbyrun.jar server`  
                  | `start`  
                  | `or`  

15
To see the command syntax, invoke `derbyrun.jar server` or `derbynet.jar` with no arguments.

* If you are familiar with both the Java programming language and Derby, you have already set `DERBY_HOME`. Set your classpath to include the Derby jar files. Then use a `java` command to invoke the `NetworkServerControl` class directly.

### Operating System | Command
--- | ---
Windows | `%DERBY_HOME%/bin/setNetworkServerCP
java org.apache.derby.drda.NetworkServerControl start`
UNIX (Korn Shell) | `$DERBY_HOME/bin/setNetworkServerCP
java org.apache.derby.drda.NetworkServerControl start`

The default system directory is the directory in which Derby was started. (See the *Derby Developer's Guide* for more information about the default system directory.)

You can specify a different host or port number when you start the Network Server by specifying an option to the command.

* Specify a port number other than the default (1527) by using the `-p portnumber` option, as shown in the following example:

```
java org.apache.derby.drda.NetworkServerControl start -p 1368
```

* Specify a specific interface (host name or IP address) to listen on other than the default (`localhost`) by using the `-h` option, as shown in the following example:

```
$DERBY_HOME/bin/startNetworkServer -h myhost -p 1368
```

where `myhost` is the host name or IP address.

**Remember:** Before using the `-h` option, you should run under the Java security manager with a customized security policy and you should enable user authentication.

By default, the Network Server will listen to requests only on the loopback address, which means that it will only accept connections from the local host.

**Starting the Network Server from a Java application**

Note that you should always properly shut down the Network Server after use, because failure to do so might result in unpredictable side-effects, such as blocked ports on the server.

There are two ways to start the Network Server from a Java application.

* You can include the following line in the `derby.properties` file:

  ```
derby.drda.startNetworkServer=true
  ```

  This starts the server on the default port, 1527, listening on `localhost` (all interfaces).
To specify a different port or a specific interface in the `derby.properties` file, include the following lines, respectively:

```java
derby.drda.portNumber=1110
derby.drda.host=myhost
```

You can also specify the `startNetworkServer` and `portNumber` properties by using a Java command:

```java
java -Dderby.drda.startNetworkServer=true
-Dderby.drda.portNumber=1110
-Dderby.drda.host=myhost yourApp
```

- You can use the `NetworkServerControl` API to start the Network Server from a separate thread within a Java application:

```java
NetworkServerControl server = new NetworkServerControl();
server.start (null);
```

Starting the Network Server on IPv6/IPv4 dual stack Windows machines

The following JVM properties need to be added to the command when starting the server on IPv6/IPv4 dual stack Windows machines:

```java
-Djava.net.preferIPv4Stack=false
-Djava.net.preferIPv6Addresses=true
```

### Shutting down the Network Server

If user authentication is disabled, a Derby database will shut down normally when the Network Server is shut down. If user authentication is enabled, you must explicitly shut down the database before shutting down the Network Server by specifying a valid Derby user name and password.

The database can be shut down either directly, or by the Derby server.

- To shut down the Network Server by using the scripts that are provided for Windows systems, use:

  ```
  stopNetworkServer.bat [-h hostname] [-p portnumber]
  ```

- To shut down the Network Server by using the scripts that are provided for UNIX systems, use:

  ```
  stopNetworkServer [-h hostname] [-p portnumber]
  ```

  These scripts are located in the `$DERBY_HOME/bin` directory.

### Shutting down by using the command line

From the command line, shut down the Network Server with the following command:

```java
java org.apache.derby.drda.NetworkServerControl
    shutdown [-h <hostname>] [-p <portnumber>]
```

### Shutting down by using the API

You can use the `NetworkServerControl` API to shut down the Network Server from within a Java application. The name of the method that you use to shutdown the Network Server is `shutdown()`.

For example, the following command shuts down the Network Server running on the current machine using the default port number (1527):

```java
NetworkServerControl server = new NetworkServerControl();
server.shutdown();
```
Obtaining system information

You can obtain information about the Network Server, such as version and current property values, Java information, and Derby database server information, by using the `sysinfo` utility. The `sysinfo` utility is available from scripts, the command line, the NetworkServerControl API, and through the servlet interface.

The following scripts are located in the `$DERBY_HOME/bin` directory. Before running these scripts, make sure that the Derby Network Server is started.

- Run the following script to obtain information about the Network Server on a Windows system:

  ```bash
  NetworkServerControl.bat sysinfo [-h hostname][-p portnumber]
  ```

- Run the following script to obtain information about the Network Server on a UNIX system:

  ```bash
  NetworkServerControl sysinfo [-h hostname] [-p portnumber]
  ```

For more information on the `sysinfo` utility, see the `Derby Tools and Utilities Guide`.

Obtaining system information by using the command line

To run `sysinfo` from the command line, use a command like one of the following while the Network Server is running:

```bash
java -jar $DERBY_HOME/lib/derbyrun.jar server sysinfo
  [-h hostname][-p portnumber]
```

```bash
java org.apache.derby.drda.NetworkServerControl sysinfo
  [-h hostname] [-p portnumber]
```

Administrative commands such as `sysinfo` can only execute on the host where the server was started, even if the server was started with the `-h` option.

Obtaining system information by using the API

The `sysinfo` method produces the same information as the `sysinfo` command. The signature for this method is:

```java
String getSysinfo();
```

For example:

```java
NetworkServerControl serverControl = new NetworkServerControl();
String myinfo = serverControl.getSysinfo();
```

The `getSysinfo()` method returns information about the Network Server that is running on the current machine on the default port number (1527).

Obtaining Network Server runtime information:

Use the `runtimeinfo` command or `getRuntimeInfo` method to get memory usage and current session information about the Network Server including user, database, and prepared statement information.

- To run `runtimeinfo` from the command line:

  ```bash
  java org.apache.derby.drda.NetworkServerControl runtimeinfo
    [-h <hostname>][-p <portnumber>]
  ```

- The `getRuntimeInfo` method returns the same information as the `runtimeinfo` command. The signature for the `getRuntimeInfo` method is `String getRuntimeInfo()`.
  For example:
NetworkServerControl serverControl = new NetworkServerControl();
String myinfo = serverControl.getRuntimeInfo();

**Obtaining Network Server properties by using the getCurrent Properties method:**

The getCurrentProperties method is a Java method that you can use to obtain information about the Network Server. It returns a Properties object with the value of all the Network Server properties as they are currently set.

The signature of this method is:

Properties getCurrentProperties();

For example:

NetworkServerControl server = new NetworkServerControl();
Properties p = server.getCurrentProperties();
p.list(System.out);
System.out.println(p.getProperty("derby.drda.host"));

As shown in the previous example, you can look up the current properties and then work with individual properties if needed by using various APIs on the Properties class. You can also print out all the properties by using the Properties.list() method.

See Managing the Derby Network Server remotely by using the servlet interface for information about obtaining system information using the servlet interface.

**Accessing the Network Server by using the network client driver**

When connecting to the Network Server, your application needs to load a driver and connection URL that is specific to the Network Server. In addition, you must specify a user name and password if you are using authentication.

The driver that you need to access the Network Server is:

org.apache.derby.jdbc.ClientDriver

The syntax of the URL that is required to access the Network Server is:

jdbc:derby://<server>[:<port>]/
<databaseName>[;<URL attribute>=<value> [;...]]

where the <URL attribute> is either a Derby embedded or network client attribute.

**Table 1. Standard JDBC DataSource properties**

<table>
<thead>
<tr>
<th>Property</th>
<th>Type</th>
<th>Description</th>
<th>URL attribute</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>databaseName</td>
<td>String</td>
<td>The name of the database. This property is required.</td>
<td>’</td>
<td>This property is also available using EmbeddedDataSource.</td>
</tr>
<tr>
<td>dataSourceName</td>
<td>String</td>
<td>The data source name.</td>
<td>’</td>
<td>This property is also available using EmbeddedDataSource.</td>
</tr>
<tr>
<td>description</td>
<td>String</td>
<td>A description of the data source.</td>
<td>’</td>
<td>This property is also available using EmbeddedDataSource.</td>
</tr>
<tr>
<td>Property</td>
<td>Type</td>
<td>Description</td>
<td>URL attribute</td>
<td>Notes</td>
</tr>
<tr>
<td>--------------</td>
<td>-----------</td>
<td>-----------------------------------------------------------------------------</td>
<td>---------------</td>
<td>--------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>user</td>
<td>String</td>
<td>The user's account name.</td>
<td>user</td>
<td>Default is APP. This property is also available using EmbeddedDataSource.</td>
</tr>
<tr>
<td>password</td>
<td>String</td>
<td>The user's database password.</td>
<td>password</td>
<td>This property is also available using EmbeddedDataSource.</td>
</tr>
<tr>
<td>serverName</td>
<td>String</td>
<td>The host name or TCP/IP address where the server is listening for requests.</td>
<td></td>
<td>Default is &quot;localhost&quot;.</td>
</tr>
<tr>
<td>portNumber</td>
<td>Integer</td>
<td>The port number where the server is listening for requests.</td>
<td></td>
<td>Default is “1527”.</td>
</tr>
</tbody>
</table>

Table 2. Client-specific DataSource properties

<table>
<thead>
<tr>
<th>Property</th>
<th>Type</th>
<th>Description</th>
<th>URL attribute</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>traceFile</td>
<td>String</td>
<td>The filename for tracing output. Setting this property turns on tracing.</td>
<td>traceFile</td>
<td>'</td>
</tr>
<tr>
<td>traceDirectory</td>
<td>String</td>
<td>The directory for the tracing output. Each connection will send output to a separate file. Setting this property turns on tracing. See Network client tracing.</td>
<td>traceDirectory</td>
<td>'</td>
</tr>
<tr>
<td>traceLevel</td>
<td>Integer</td>
<td>The level of client tracing if traceFile or traceDirectory are set.</td>
<td>traceLevel</td>
<td>The default is TRACE_ALL.</td>
</tr>
<tr>
<td>traceFileAppend</td>
<td>Boolean</td>
<td>Value is true if tracing output should append to the existing trace file.</td>
<td>traceFileAppend</td>
<td>The default is false.</td>
</tr>
<tr>
<td>Property</td>
<td>Type</td>
<td>Description</td>
<td>URL attribute</td>
<td>Notes</td>
</tr>
<tr>
<td>------------------</td>
<td>---------</td>
<td>------------------------------------------------------------------------------</td>
<td>---------------------</td>
<td>--------------------------------------------------------------</td>
</tr>
<tr>
<td>securityMechanism</td>
<td>Integer</td>
<td>The security mechanism. See Network client security.</td>
<td>securityMechanism</td>
<td>The default is USER_ONLY_SECURITY.</td>
</tr>
<tr>
<td>retrieveMessageText</td>
<td>Boolean</td>
<td>Retrieve message text from the server. A stored procedure is called to retrieve the message text with each SQLException and might start a new unit of work. Set this property to false if you do not want the performance impact or when starting new units of work.</td>
<td>retrieveMessageText</td>
<td>The default is true.</td>
</tr>
<tr>
<td>ssl</td>
<td>String</td>
<td>The SSL mode for the client connection. See Network encryption and authentication with SSL/TLS</td>
<td>ssl</td>
<td>The default is off.</td>
</tr>
</tbody>
</table>

**Table 3. Server-Specific DataSource properties**

<table>
<thead>
<tr>
<th>Property</th>
<th>Type</th>
<th>Description</th>
<th>URL attributes</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>connectionAttributes</td>
<td>String</td>
<td>Set to the list of Derby embedded connection attributes separated by semicolons.</td>
<td>Various</td>
<td>This property is also available using EmbeddedDataSource. See the Derby Reference Manual for more information about the various connection attributes.</td>
</tr>
<tr>
<td>createDatabase</td>
<td>String</td>
<td>If set to &quot;create&quot;, create the database specified with databaseName property.</td>
<td>create</td>
<td>This property is also available using EmbeddedDataSource. See the Derby Reference Manual for more information. Similar to setting connectionAttribute</td>
</tr>
</tbody>
</table>
### Notes

The result of conflicting settings of `createDatabase`, `shutdownDatabase`, and `connectionAttributes` is undefined. If authentication and `sqlAuthorization` are both enabled, database shutdown is restricted to the database owner.

### Network client security

The Derby Network Client allows you to select a security mechanism by specifying a value for the `securityMechanism` property.

You can set the `securityMechanism` property in one of the following ways:

- **When you are using the `DriverManager` interface**, set `securityMechanism` in a `java.util.Properties` object before you invoke the form of the `getConnection` method, which includes the `java.util.Properties` parameter.
- **When you are using the `DataSource` interface** to create and deploy your own `DataSource` objects, invoke the `DataSource.setSecurityMechanism` method after you create a `DataSource` object.

### Security mechanisms supported by the Derby Network Client

The Derby Network Client supports the following security mechanisms:

- **User id only** if no password is set.
- **User id and password**.
- **APP** if no other user is specified.

<table>
<thead>
<tr>
<th>Property</th>
<th>Type</th>
<th>Description</th>
<th>URL attributes</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>shutdownDatabase</code></td>
<td>String</td>
<td>If set to</td>
<td>shutdown</td>
<td>This property is also available using EmbeddedDataSource. See the Derby Reference Manual for more information. Similar to setting <code>connectionAttributes</code> to &quot;shutdown=true&quot;. Only &quot;shutdown&quot; is allowed, other values equate to null. The result of conflicting settings of <code>createDatabase</code>, <code>shutdownDatabase</code>, and <code>connectionAttributes</code> is undefined. If authentication and <code>sqlAuthorization</code> are both enabled, database shutdown is restricted to the database owner.</td>
</tr>
</tbody>
</table>

Note that `setAttributesAsPassword`, which is available for the embedded DataSource, is not available for the client DataSource.
### Security Mechanism

<table>
<thead>
<tr>
<th>Security mechanism</th>
<th>securityMechanism property value</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>User id and password</td>
<td>ClientDataSource.CLEAR_TEXT_PASSWORD_SECURITY (0x03)</td>
<td>Default if password is set</td>
</tr>
<tr>
<td>User id only</td>
<td>ClientDataSource.USER_ONLY_SECURITY (0x04)</td>
<td>Default if password is not set</td>
</tr>
<tr>
<td>Strong password substitution</td>
<td>ClientDataSource.STRONG_PASSWORD_SUBSTITUTE_SECURITY (0x08)</td>
<td>Strong password substitution cannot be used with external Derby authentication schemes (for example, LDAP). Also, this security mechanism uses the SHA1PRNG algorithm to generate a random number that gets exchanged between client and server. If you need to use this security mechanism, make sure that support for the SHA1PRNG algorithm is available in the JCE provider available with your JVM. For example, it is available with JVM version 1.4.1 and higher from Sun and with JVM version 1.4.2 and higher from IBM.</td>
</tr>
<tr>
<td>Encrypted user id and encrypted password</td>
<td>ClientDataSource.ENCRYPTED_USER_AND_PASSWORD_SECURITY (0x09)</td>
<td>Encryption requires a JCE implementation that supports the Diffie-Hellman algorithm with a public prime of 256 bits.</td>
</tr>
</tbody>
</table>

### Network Client Tracing

The Derby Network client provides a tracing facility to collect JDBC trace information and view protocol flows.

There are various ways to obtain trace output. However, the easiest way to obtain trace output is to use the `traceFile` attribute on the URL in `ij`. The following example shows all tracing going to the file `trace.out` from an `ij` session.

```
ij>connect 'jdbc:derby://localhost:1527/mydb;create=true;traceFile=trace.out;user=user1;password=secret4me';
```

### Implementing ClientDataSource Tracing

You can use one of three methods to collect tracing data while obtaining connections from the `ClientDataSource`:

- Use the `setLogWriter(java.io.PrintWriter)` method of `ClientDataSource` and set the `PrintWriter` to a non-null value.
• Use the `setTraceFile(String filename)` method of `ClientDataSource`.
• Use the `setTraceDirectory(String dirname)` method of `ClientDataSource` to trace each connection flow in its own file for programs that have multiple connections.

**Implementing DriverManager tracing**

Use one of the following two options to enable and collect tracing information while obtaining connections using the `DriverManager`:

• Use the `setLogWriter(java.io.PrintWriter)` method of `DriverManager` and set the `PrintWriter` to a non null-value.
• Use the `traceFile` or `traceDirectory` URL attributes to set these properties prior to creating the connection with the `DriverManager.getConnection()` method.

**Changing the default trace level**

The default trace level is `ClientDataSource.TRACE_ALL`. You can choose the tracing level by calling the `setTraceLevel(int level)` method or by setting the `traceLevel` URL attribute:

```java
String url = "jdbc:derby://samplehost.sampledomain.com:1528/mydb" + ";traceFile=/u/user1/trace.out" + ";traceLevel=" +
    org.apache.derby.jdbc.ClientDataSource.TRACE_PROTOCOL_FLOWS;
DriverManager.getConnection(url,"user1","secret4me");
```

Table 5. Available tracing levels and values

<table>
<thead>
<tr>
<th>Trace level</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>org.apache.derby.jdbc.ClientDataSource.TRACE_NONE</td>
<td>0x0</td>
</tr>
<tr>
<td>org.apache.derby.jdbc.ClientDataSource.TRACE_CONNECTION_CALLS</td>
<td>0x1</td>
</tr>
<tr>
<td>org.apache.derby.jdbc.ClientDataSource.TRACE_STATEMENT_CALLS</td>
<td>0x2</td>
</tr>
<tr>
<td>org.apache.derby.jdbc.ClientDataSource.TRACE_RESULT_SET_CALLS</td>
<td>0x4</td>
</tr>
<tr>
<td>org.apache.derby.jdbc.ClientDataSource.TRACE_DRIVER_CONFIGURATION</td>
<td>0x10</td>
</tr>
<tr>
<td>org.apache.derby.jdbc.ClientDataSource.TRACE_CONNECTS</td>
<td>0x20</td>
</tr>
<tr>
<td>org.apache.derby.jdbc.ClientDataSource.TRACE_PROTOCOL_FLOWS</td>
<td>0x40</td>
</tr>
<tr>
<td>org.apache.derby.jdbc.ClientDataSource.TRACE_RESULT_SET_META_DATA</td>
<td>0x80</td>
</tr>
<tr>
<td>org.apache.derby.jdbc.ClientDataSource.TRACE_PARAMETER_META_DATA</td>
<td>0x100</td>
</tr>
<tr>
<td>org.apache.derby.jdbc.ClientDataSource.TRACE_DIAGNOSTICS</td>
<td>0x200</td>
</tr>
<tr>
<td>org.apache.derby.jdbc.ClientDataSource.TRACE_XA_CALLS</td>
<td>0x800</td>
</tr>
<tr>
<td>org.apache.derby.jdbc.ClientDataSource.TRACE_ALL</td>
<td>0xFFFFFFFF;</td>
</tr>
</tbody>
</table>

To specify more than one trace level, use one of the following techniques:

• Use bitwise OR operators (|) with two or more trace values. For example, to trace PROTOCOL flows and connection calls, specify this value for `traceLevel`:

```java
TRACE_PROTOCOL_FLOWS | TRACE_CONNECTION_CALLS
```

• Use a bitwise complement operator (~) with a trace value to specify all except a certain trace. For example, to trace everything except PROTOCOL flows, specify this value for `traceLevel`:

```java
~TRACE_PROTOCOL_FLOWS
```
Network client driver examples

The following examples specify the user and password URL attributes. To enable user authentication, the property `derby.connection.requireAuthentication` must be set to true, otherwise, Derby does not require a user name and password. For a multi-user product, you would typically set it for the system in the `derby.properties` file for your server, since it is in a trusted environment. Below is a sample `derby.properties` file that conforms to these examples:

```
derby.connection.requireAuthentication=true
derby.authentication.provider=BUILTIN
derby.user.judy=no12see
```

**Example 1**

The following example connects to the default server name localhost on the default port, 1527, and to the database sample.

```
jdbc:derby://localhost:1527/sample;user=judy;password=no12see
```

**Example 2**

The following example specifies both Derby and Network Client driver attributes:

```
jdbc:derby://localhost:1527/sample;create=true;user=judy;password=no12see
```

**Example 3**

This example connects to the default server name localhost on the default port, 1527, and includes the path in the database name portion of the URL.

```
jdbc:derby://localhost:1527/c:/my-db-dir/my-db-name;user=judy;password=no12see
```

**Example 4**

The following example shows how to use the network client driver to connect the network client to the Network Server:

```
String databaseURL = "jdbc:derby://localhost:1527/sample";
// Load Derby Network Client driver class
// If you are running on JDK 1.6 or higher, then you do not need to invoke Class.forName(). In that environment, the network client driver loads automatically.
// Class.forName("org.apache.derby.jdbc.ClientDriver");
// Set user and password properties
Properties properties = new Properties();
properties.put("user", "judy");
properties.put("password", "no12see");
// Get a connection
Connection conn = DriverManager.getConnection(databaseURL, properties);
```

Accessing the Network Server by using the DB2(R) Driver for JDBC

You can use the IBM(R) DB2(R) Driver for JDBC instead of the Derby network client driver to connect to the Network Server.

You can use the DB2(R) Driver for JDBC instead of the Derby network client driver to connect to the Network Server. Your application needs to load the driver and connection URL that is specific to the Network Server. In addition, you specify a user name and
password. If you have not set up authentication, you can use any value for the user name and password. The driver that you use to access the Network Server is:

```
com.ibm.db2.jcc.DB2Driver
```

You must have the following two jar files present in your classpath in order to use the DB2® Driver for JDBC:
- `db2jcc.jar`
- `db2jcc_license_c.jar`

The syntax of the URL that is required to access the Network Server is:

```
jdbc:derby:net://<server>[:<port>]/<databaseName>[;<derby URL attribute>=<value> [;...]] [:<Universal Driver attribute>=<value>; [...;]]
```

After you specify the database name and attributes, you can include attributes for the DB2® Driver for JDBC. You must include a semicolon after the last DB2® Driver for JDBC attribute.

**server**
The name of the machine where the server is running. It can be the name of the machine (for example, *buffy*) or the IP address, for example, `158.58.62.225`.

**Note:** Unless the Network Server was started with the `-h` option or the `derby.drda.host` property set, this value must be `localhost`.

**port**
The port that the server is listening to. The default is 1527.

**database name**
The name of the database that you are connecting to. The database name can be a maximum of 18 characters. You must use quotation marks ("”) to include path information in the database name. Alternately, you can specify path information by setting the property `derby.system.home` in either the `derby.properties` file or in the Java® environment when you start the Network Server. See the *Derby Developer’s Guide* for more information about defining the system home.

**derby URL attribute=value**
Optional database connection URL attributes that are supported by Derby. See the *Derby Developer’s Guide* for more information.

**Universal Driver Attribute=value**
Optional database connection URL attributes that are supported by the DB2® Driver for JDBC.

The DB2® Driver for JDBC requires that you set the user and password attributes to non-null values.

The following DB2® Driver for JDBC attributes are available to you when running the Network Server:

**user**
User name (required by the DB2® Driver for JDBC).

**password**
User password (required by the DB2® Driver for JDBC).

**portNumber**
The TCP/IP port number where the Network Server listens for connection requests to this data source. The default is 1527.
Derby Server and Administration Guide

**retrieveMessagesFromServerOnGetMessage**
Displays error messages from the server.

**readOnly**
Creates a read-only connection. The default is false.

**logWriter**
A character output stream. All logging and tracing messages print to the logWriter property.

**traceLevel**
Specifies the granularity of tracing messages to the logWriter property.

**traceFile**
Provides an explicit file location for the trace output.

**securityMechanism**
Indicates what type of security mechanism is used.

**deferPrepares**
Controls when prepared statements are physically prepared in the database server. The default value is true.

**DB2(R) Driver for JDBC System information**
The Derby Network Server is compatible with the DB2(R) Driver for JDBC release 2.4 and higher.

**DB2(R) Driver for JDBC examples:**
The examples show how to specify the server name, database, and the URL attributes. You must specify the user name and password attributes when you use the IBM(R)DB2(R) Driver for JDBC.

**Example 1**
The following example connects to the default server name localhost on the default port, 1527, and to the database sample. It specifies the URL attributes user, password, and retrieveMessagesFromServerOnGetMessage.

```
jdbc:derby:net://localhost:1527/sample:user=judy;password=no12see;retrieveMessagesFromServerOnGetMessage=true;
```

**Example 2**
The following example specifies the attributes for both Derby and the DB2(R) Driver for JDBC:

```
jdbc:derby:net://localhost:1527/sample;create=true:user=judy;password=no12see;retrieveMessagesFromServerOnGetMessage=true;
```

**Example 3**
This example connects to the default server name localhost on the default port, 1527, and includes the path in the database name portion of the URL. The database name must be delimited by double quotation marks and you cannot specify Derby attributes on the URL.

```
jdbc:derby:net://localhost:1527/"c:/my-db-dir/my-db-name":user=judy;password=no12see;retrieveMessagesFromServerOnGetMessage=true;
```
Example 4
The following is a sample program fragment that connects to the Network Server using the DB2(R) Driver for JDBC:

```java
// Load DB2 Driver for JDBC class
Class.forName("com.ibm.db2.jcc.DB2Driver");
// Set user and password properties
Properties properties = new Properties();
properties.put("user", "APP");
properties.put("password", "APP");
properties.put("retrievemessagesfromserverongetmessage", "true");
// Get a connection
Connection conn = DriverManager.getConnection(databaseURL, properties);
```

Accessing the Network Server by using a DataSource


If your client runs on the Java SE 6 platform, and if you want to use DataSource methods specific to the JDBC 4 API, use the DataSources named `org.apache.derby.jdbc.ClientDataSource40` and `org.apache.derby.jdbc.ClientConnectionPoolDataSource40`.

If your client is running on the Java SE 6 platform, all connection objects returned from the DataSource will be JDBC 4 connection objects, whether or not you are using a DataSource whose name ends in "40".

DataSource access example

The following example uses `org.apache.derby.jdbc.ClientDataSource` to access the Network Server:

```java
org.apache.derby.jdbc.ClientDataSource ds =
        new org.apache.derby.jdbc.ClientDataSource();
ds.setDatabaseName("mydb");
ds.setCreateDatabase("create");
ds.setUser("user");
ds.setPassword("mypass");

// The host on which Network Server is running
ds.setServerName("localhost");

// The port on which Network Server is listening
ds.setPortNumber(1527);
Connection conn = ds.getConnection();
```

XA and the Network Server

Both the Derby embedded driver and the Network Server provide XA support. The Network Server provides DRDA level 7 support. DRDA clients that support XAMGR, such as the Derby network client, can send XA requests to the Network Server.

Using XA with the network client driver

You can access XA support for the Network Server by using the network client driver’s XA DataSource interface.
The interface `org.apache.derby.jdbc.ClientXADataSource` is available on all supported Java SE platforms. If your client runs on the Java SE 6 platform, and if you want to use XA DataSource methods specific to the JDBC 4 API, use the DataSource named `org.apache.derby.jdbc.ClientXADataSource40`.

If your client is running on the Java SE 6 platform, all connection objects returned from the DataSource will be JDBC 4 connection objects, whether or not you are using the DataSource whose name ends in "40".

The following example illustrates how to obtain an XA connection with the network client driver:

```java
import org.apache.derby.jdbc.ClientXADataSource;
import javax.sql.XAConnection;
...
XAConnection xaConnection = null;
Connection conn = null;
String driver = "org.apache.derby.jdbc.ClientDataSource";
ClientXADataSource ds = new ClientXADataSource();
ds.setDatabaseName("sample;create=true");
ds.setServerName("localhost");
ds.setPortNumber(1527);
Class.forName(driver);
xaConnection = ds.getXAConnection("auser", "shhhh");
conn = xaConnection.getConnection();
```

### Using the Derby tools with the Network Server

The Derby tools `ij` and `dblook` work in embedded mode and client/server mode.

#### Using the Derby `ij` tool with the Network Server

To use the `ij` tool with the network client driver:

1. Start `ij` in one of the following ways. For details, see "Starting `ij`" in the Derby Tools and Utilities Guide.
   a. Use a script.
      
      Run the `ij.bat` script on Windows systems and the `ij` script on UNIX systems. These scripts are located in the `$DERBY_HOME/bin` directory.
   b. Run the `ij` tool using the `$DERBY_HOME/lib/derbyrun.jar` file.
   
      ```
      java -jar derbyrun.jar ij
      ```
   c. Run the `ij` tool by specifying the class name.
      
      ```
      java org.apache.derby.tools.ij
      ```

2. Connect by specifying the URL:

   ```
   ij> CONNECT 'jdbc:derby://localhost:1527/sample'
   USER 'judy' PASSWORD 'no12see';
   ```

   See Network client driver examples for additional URL examples.

#### Using the Derby `dblook` tool with the Network Server

To use the `dblook` tool with the Network Client driver, make sure the Network Server is running (see Starting the Network Server), and then include the necessary Derby
and Network Client driver connection attributes as part of the database URL, as in the following example:

```
java org.apache.derby.tools.dblook -d
'db:derby://localhost:1527/sample;
user=user1;password=secret4me;'
```

For details on using the dblook tool, see the *Derby Tools and Utilities Guide*.

### Differences between running Derby in embedded mode and using the Network Server

This section describes the differences between running Derby in embedded mode and using the Network Server. Note that there may be undocumented differences that have not yet been identified.

**Differences between the embedded client and the network client driver**

The following are known differences that exist between the Derby embedded driver and the network client driver. Note that there may be undocumented differences that have not yet been identified. Some differences with the network client may be changed in future releases to match the embedded driver functionality.

- Error messages and SQLStates can differ between the network client and embedded driver. Some SQLStates may be null when using the network client, particularly for data conversion errors.
- Multiple SQL exceptions and warnings will only return the SQLState of the first exception when using the network client. The text of the additional exceptions will be appended to the text of the first exception. See *Error message differences*.
- Treatment of error situations encountered during batch processing with `java.sql.Statement`, `java.sql.PreparedStatement` and `java.sql.CallableStatement` is different. With the embedded driver processing stops when an error is encountered; with the network client driver processing continues, but an appropriate value as defined in the `java.sql.Statement` api is returned in the resulting update count array.
- To use an encrypted user id and password, you need to have the IBM's Java Cryptography Extension (JCE) Version 1.2.1 or later.

**Updatable Result Sets**

The functionality of updatable result sets in a server environment are similar to an embedded environment in Derby, with the exception of the following differences:

- The Network Client requires that there be at least one column in the select list from the target table. For example, the following statement will fail in a server environment:

  ```
  select 1, 2 from t1 for update of c11
  ```

  The Network Client driver looks at both of the columns in the select list and cannot determine the target table for update/delete by looking at the column metadata. This requirement is not necessary in an embedded environment.

- The embedded driver allows for statement name changes when there is an open result set on the statement object. This is not supported in a server environment.

Other differences between updatable result sets in a server or embedded environment can be found in the following table.

**Table 6. Comparison of updatable result sets features in server and embedded environments**

<table>
<thead>
<tr>
<th>Embedded environment</th>
<th>Server environment</th>
</tr>
</thead>
<tbody>
<tr>
<td>updateBytes on CHAR, VARCHAR, LONG VARCHAR datatypes supported.</td>
<td>Not supported</td>
</tr>
</tbody>
</table>
Embedded environment | Server environment
---|---
updateTime on TIMESTAMP datatypes supported. | Not supported
updateClob and updateBlob supported. | Not supported

**Error message differences**

The Network Server reports only the first error or warning message if multiple errors or warnings occur for a given statement. For example:

```sql
ij> create table ai (x int, y int generated always as identity (increment by 200000000));
ij> insert into ai (x) values (1),(2),(3),(4),(5),(6),(7),(8),(9),(10),(11),(12),(13),(14),(15),(16),(17),(18),(19);
```

The Network Server generates the following error message and appends the exception message to the error:

```
ERROR 42Z24: Overflow occurred in identity for column 'Y' in table 'AI':
SQLSTATE: 22003: The resulting value is outside the range for the data type INTEGER.
```

The Derby embedded driver, however, would generate two SQL exceptions:

```
ERROR 42Z24: Overflow occurred in identity for column 'Y' in table 'AI'.
ERROR 22003: The resulting value is outside the range for the data type INTEGER.
```

This is because the network client driver reports only one SQLException or one SQLWarning per statement.

**User authentication differences**

When running Derby in embedded mode or when using the Derby Network Server, you can enable or disable server-side user authentication. However, when using the Network Server, the default security mechanism (`CLEAR_TEXT_PASSWORD_SECURITY`) requires that you supply both the user name and password.

In addition to the default user name and password security mechanism, `org.apache.derby.jdbc.ClientDataSource.CLEAR_TEXT_PASSWORD_SECURITY`, Derby Network Server supports the following security properties:

- **UserID** (`org.apache.derby.jdbc.ClientDataSource.USER_ONLY_SECURITY`)
  
  When using this mechanism, you must specify only the user property. All other mechanisms require you to specify both the user name and the password.

- **Encrypted UserID and encrypted password** (`org.apache.derby.jdbc.ClientDataSource.ENCRYPTED_USER_AND_PASSWORD_SECURITY`)
  
  When using this mechanism, both password and user id are encrypted.

- **Strong password substitution** (`org.apache.derby.jdbc.ClientDataSource.STRONG_PASSWORD_SUBSTITUTE_SECURITY`)
  
  When using this mechanism, a strong password substitute is generated and used to authenticate the user with the network server. The original password is never sent in any form across the network.

The user's name that is specified upon connection is the default schema for the connection, if a schema with that name exists. See the *Derby Developer's Guide* for more information on schema and user names.

If you specify any other security mechanism, you will receive an exception.
To change the default, you can specify another security mechanism either as a property or on the URL (using the `securityMechanism=value` attribute) when making the connection. For details, see Network client security and "securityMechanism=value attribute" in the Derby Reference Manual.

Whether the security mechanism you specify for the client actually takes effect depends upon the setting of the `derby.drda.securityMechanism` property for the Network Server. If the `derby.drda.securityMechanism` property is set, the Network Server accepts only connections that use the security mechanism specified by the property setting. If the `derby.drda.securityMechanism` property is not set, clients can use any valid security mechanism. For details, see `derby.drda.securityMechanism` property.

Security mechanism options when user authentication is enabled on the Network Server:

When user authentication is enabled in Derby, you can use any of the following security mechanisms:

- Clear text user name and password security, the default
- Strong password substitute security
- Encrypted user name and password security

Security mechanism options when user authentication is disabled on the Network Server:

When user authentication is turned off in Derby, you can use any of the security mechanism options.

You must provide a user and password for all security mechanisms except `USER_ONLY_SECURITY`. However, because user authentication is disabled in the Derby server, the user name and password that you supply does not have to be one recognized as valid by Derby.

Enabling the encrypted user ID and password security mechanism:

To use the encrypted user ID and password security mechanism, you need a Java environment with an JCE (Java Cryptography Extension) which supports the Diffie-Hellman algorithm with a public prime of 256 bits. Sun's Java(TM) 2 Platform, Standard Edition, Version 1.4 (J2SE) and later requires a public prime of 512 bits or more. An alternative mechanism if the 256 bit public prime is not supported, is `STRONG_PASSWORD_SUBSTITUTE_SECURITY`.

To use the encrypted user id and password security mechanism during JDBC connection using the network client, specify the `securityMechanism` in the connection property. **Note:** If an encrypted database is booted in the Network Server, users can connect to the database without giving the `bootPassword`. The first connection to the database must provide the `bootPassword`, but all subsequent connections do not need to supply it. To remove access from the encrypted database, use the `shutdown=true` option to shut down the database.

Differences in JDBC 3.0 methods

The following JDBC 3.0 methods are supported only with the Derby embedded driver. Attempts to call these methods with the network client driver will result in a "not implemented" error.

```java
Connection.prepareStatement(String sql, String[] columnNames)
Connection.prepareStatement(String sql, int[] columnIndexes)
Statement.execute(String sql, String[] columnNames)
Statement.execute(String sql, int[] columnIndexes)
Statement.executeUpdate(String sql, String[] columnNames)
```
Statement.executeUpdate(String sql, int[] columnIndexes)

For more on the use of these methods, see the sections "java.sql.Connection interface: supported JDBC 3.0 methods", "java.sql.Statement interface: supported JDBC 3.0 methods", and "Autogenerated keys" in the Derby Reference Manual.

Differences using the Connection.setReadOnly method

In the embedded mode, when the Connection.setReadOnly method has true as the parameter, the connection is marked as a read-only connection. When using a Network Server, the Connection.setReadOnly(true) method is ignored and the connection is not marked as a read-only connection.

Setting port numbers

By default, Derby using the Network Server listens on TCP/IP port number 1527. If you want to use a different port number, you can specify it on the command line when starting the Network Server. For example:

```
java org.apache.derby.drda.NetworkServerControl start -p 1088
```

1. However, it is better to specify the port numbers by using any of the following methods
   • Change the startNetworkServer.bat or startNetworkServer.ksh scripts
   • Use the derby.drda.portNumber property in derby.properties

See Starting the Network Server for more information.

Managing the Derby Network Server

The Derby Network Server can run as a stand-alone server, with Derby as an embedded part of the application.

It can also be managed remotely from a web server by using a servlet interface. You can manage the Network Server by using shell scripts, the command line, or the Network Server API. See Managing the Derby Network Server remotely by using the servlet interface for information about starting and shutting down the Network Server using the servlet interface.

Overview

You start the Derby Network Server using the command line or using the Derby Server API. (Derby provides scripts for you to use to start the server from the command line.) Before starting the server, you will probably set certain Derby and Network Server properties.

Using the NetworkServerControl API

You need to create an instance of the NetworkServerControl class if you are using the API. There are two constructor methods for this class:

Note: Before enabling connections from other systems, ensure that you are running under security manager.

• NetworkServerControl()

This constructor method creates an instance that listens either on the default port (1527) or the port that is set by the derby.drda.portNumber property. It will also listen on the host set by the derby.drda.host property or the loopback address if the property is not set. This is the default constructor; it does not allow remote connections. It is equivalent to calling
NetworkServerControl(InetAddress.getByName("localhost"),1527) if no properties are set.

- NetworkServerControl (InetAddress address, int portNumber)

This constructor method creates an instance that listens on the specified portNumber on the specified address. The InetAddress will be passed to ServerSocket. NULL is an invalid address value. The following examples show how you might allow Network Server to accept connections from other hosts:

```java
//accepts connections from other hosts on an IPv4 system
NetworkServerControl serverControl = 
    new NetworkServerControl(InetAddress.getByName("0.0.0.0"),1527);

//accepts connections from other hosts on an IPV6 system
NetworkServerControl serverControl = 
    new NetworkServerControl(InetAddress.getByName("::"),1527);
```

Setting Network Server properties

You can specify Network Server properties in three ways:

- On the command line
- In the .bat or .ksh files (loading the properties by executing java -D)
- In the derby.properties file.

Properties in the command line or in the .bat or .ksh files take precedence over the properties in the derby.properties file. Arguments included on commands that are issued on the command line take precedence over property values.

**derby.drda.host property**

Causes the Network Server to listen on a specific network interface. This property allows multiple instances of Network Server to run on a single machine, each using its own unique host:port combination. The host needs to be set to enable remote connections. By default, the Network Server will listen only on the loopback address. If the property is set to 0.0.0.0, Network Server will listen on all interfaces. Ensure that you are running under the security manager and that user authorization is enabled before you enable remote connections with this property.

**Syntax**

derby.drda.host=hostname

**Default**

If no host name is specified, the Network Server listens on the loopback address of the current machine (localhost).

**Example**

derby.drda.host=myhost

**Static or dynamic**

Static. You must restart the Network Server for changes to take effect.

**derby.drda.keepAlive property**

Indicates whether SO_KEEPALIVE is enabled on sockets. The keepAlive mechanism is used to detect when clients disconnect unexpectedly. A keepalive probe is sent to the client if a long time (by default, more than two hours) passes with no other data being sent or received. The derby.drda.keepAlive property is used to detect and clean up connections for clients on powered-off machines or clients that have disconnected unexpectedly.
If the property is set to false, Derby will not attempt to clean up disconnected clients. The keepAlive mechanism might be disabled if clients need to resume work without reconnecting even after being disconnected from the network for some time. To disable keepAlive probes on Network Server connections, set this property to false.

Syntax

```
derby.drda.keepAlive=[true|false]
```

Default

True.

Example

```
derby.drda.keepAlive=false
```

Static or dynamic

Static. You must restart the Network Server for changes to take effect.

derby.drda.logConnections property

Indicates whether to log connections. Also controls the logging of the connection number. Connection number tracing, if enabled, goes to both the `derby.log` file and the network server console.

Syntax

```
derby.drda.logConnections=[true|false]
```

Default

False.

Example

```
derby.drda.logConnections=true
```

Static or dynamic

Dynamic. System values can be changed by using commands or the servlet interface after the Network Server has been started.

derby.drda.maxThreads property

Use the derby.drda.maxThreads property to set a maximum number of connection threads that Network Server will allocate. If all of the connection threads are currently being used and the Network Server has already allocated the maximum number of threads, the threads will be shared by using the derby.drda.timeSlice property to determine when sessions will be swapped.

Syntax

```
derby.drda.maxThreads=numthreads
```

Default

0

Example

```
derby.drda.maxThreads=50
```

Static or dynamic

Static. You must restart the Network Server for changes to take effect.
**derby.drda.minThreads property**

Use the derby.drda.minThreads property to set the minimum number of connection threads that Network Server will allocate. By default, connection threads are allocated as needed.

**Syntax**

```
derby.drda.minThreads=numthreads
```

**Default**

0

**Example**

```
derby.drda.minThreads=10
```

**Static or dynamic**

Static. You must restart the Network Server for changes to take effect.

**derby.drda.portNumber property**

Indicates the port number to use.

**Syntax**

```
derby.drda.portNumber=portnumber
```

**Default**

If no port number is specified, 1527 is the default.

**Example**

```
derby.drda.portNumber=1110
```

**Static or dynamic**

Static. You must restart the Network Server for changes to take effect.

**derby.drda.securityMechanism property**

The derby.drda.securityMechanism property restricts the client connections based on the security mechanism.

If the derby.drda.securityMechanism property is set to a valid mechanism, the Network Server accepts only connections which use that security mechanism. No other types of connections are accepted. If the derby.drda.securityMechanism property is not set, the Network Server accepts any connection which uses a valid security mechanism.

**Syntax**

```
derby.drda.securityMechanism = [ USER_ONLY_SECURITY | CLEAR_TEXT_PASSWORD_SECURITY | ENCRYPTED_USER_AND_PASSWORD_SECURITY | STRONG_PASSWORD_SUBSTITUTE_SECURITY ]
```

**Default**

None.

**Example**

```
derby.drda.securityMechanism=USER_ONLY_SECURITY
```
The server that runs with this setting accepts only client connections with the USER_ONLY_SECURITY value.

**Static or dynamic**
Static. You must restart the Network Server for the changes that are associated with this property to take effect.

**derby.drda.sslMode property**
The derby.drda.sslMode property indicates whether the client connection is encrypted or not, and whether certificate based peer authentication is enabled.

**Syntax**

``` derby.drda.sslMode = [ off | basic | peerAuthentication ] ```

**Default**
off

**Example**

derby.drda.sslMode=basic

The server that runs with this setting accepts client connections encrypted with SSL.

**Static or dynamic**
Static. You must restart the Network Server for the changes that are associated with this property to take effect.

**derby.drda.startNetworkServer property**
Use the derby.drda.startNetworkServer property to simplify embedding the Network Server in your Java application. When you set derby.drda.startNetworkServer to true, the Network Server will automatically start when you start Derby (in this context, Derby will start when the embedded driver is loaded). Only one Network Server can be started in a JVM.

NOTE: If you start the Network Server with this property set to true, the Network Server will stop when your application ends or when you stop it by other means (e.g. by using the Java API, the command line interface, or by shutting down the Derby system), whichever comes first.

**Syntax**

derby.drda.startNetworkServer=[true | false]

**Default**
False.

**Example**

derby.drda.startNetworkServer=true

**Static or dynamic**
Static. You must shut down the Network Server and restart Derby for this change to take effect.

**derby.drda.streamOutBufferSize property**
Configure size of buffer for streaming blob/clob from server to client. If the configured size is 0 or less, the buffer is not placed.

**Note:**
This configuration is used when optimizing streaming blob/clob from server to client. If there were found many small packets, of which sizes are much lower than maximum size of packet possible in the network, it will improve performance of streaming to setting this configuration. Recommended value of this configuration is maximum packet size possible in the network minus appropriate size for header.

**Syntax**

```
derby.drda.streamOutBufferSize=size of buffer
```

**Default**

0

**Example**

```
derby.drda.streamOutBufferSize=1024
```

**Static or dynamic**

Dynamic. System values can be changed by using commands or the servlet interface after the Network Server has been started.

**derby.drda.timeSlice property**

Use the `derby.drda.timeSlice` property to set the number of milliseconds that each connection will use before yielding to another connection. This property is relevant only if the `derby.drda.maxThreads` property is set to a value greater than zero.

**Syntax**

```
derby.drda.timeSlice=milliseconds
```

**Default**

0

**Example**

```
derby.drda.timeSlice=2000
```

**Static or dynamic**

Static. You must restart the Network Server for changes to take effect.

**derby.drda.traceAll property**

Turns tracing on for all sessions.

**Syntax**

```
derby.drda.traceAll=[true|false]
```

**Default**

False

**Example**

```
derby.drda.traceAll=true
```

**Static or dynamic**

Dynamic. System values can be changed by using commands or the servlet interface after the Network Server has been started.
**derby.drda.traceDirectory property**

Indicates the location of tracing files.

**Syntax**

```
derby.drda.traceDirectory=tracefiledirectory
```

**Default**

If the `derby.system.home` property has been set, it is the default. Otherwise, the default is the current directory.

**Example**

```
derby.drda.traceDirectory=c:/Derby/trace
```

**Static or dynamic**

Dynamic. System values can be changed by using commands or the servlet interface after the Network Server has been started.

**Verifying Startup**

To verify that the Derby Network Server is currently running, use the ping command.

You can use the ping command in the following ways:

- You can use the scripts `NetworkServerControl.bat` for Windows systems or `NetworkServerControl.ksh` for UNIX systems with the `ping` command. For example:
  ```
  NetworkServerControl ping [-h <hostname>;] [-p <portnumber>]
  ```

- You can use the `NetworkServerControl` command:
  ```
  java org.apache.derby.drda.NetworkServerControl
      ping [-h <hostname>] [-p <portnumber>]
  ```

- You can use the `NetworkServerControl` API to verify startup from within a Java application:
  ```
  ping();
  ```

The following example uses a method to verify startup. It will try to verify for the specified number of seconds:

```java
private static boolean isServerStarted(NetworkServerControl server, int ntries)
{
    for (int i = 1; i <= ntries; i++)
    {
        try {
            Thread.sleep(500);
            server.ping();
            return true;
        } catch (Exception e) {
            if (i == ntries)
                return false;
        }
    }
    return false;
}
```

**Managing the Derby Network Server remotely by using the servlet interface**
You can use the servlet interface to manage the Network Server remotely. To use
the servlet interface, the servlet must be registered with an Application Server, and
derby.system.home must be known to the Application Server.

A Web application archive (WAR) file, derby.war, for the Derby Network Server is
available in $DERBY_HOME/lib. This file registers the Network Server's servlet at the
relative path /derbynet. See the documentation for your Application Server for instructions
on how to install it.

For example, if derby.war is installed in WebSphere Application Server with a context
root of derby, the URL of the server is:

http://<server>[:port]/derby/derbynet

Notes:
• A servlet engine is not part of the Network Server.
• When the Network Server is started by the servlet interface, shutting down the
Application Server also shuts the Network Server down, since both run in the same
JVM.

The servlet takes the following optional configuration parameters:

host
  Specifies the host name to be used by the Network Server. See the Security
  Considerations section below.

portNumber
  Specifies the port number to be used by the Network Server.

startNetworkServerOnInit
  Specifies that the Network Server is to be started when the servlet is initialized.

tracingDirectory
  Specifies the location for trace files. If the tracing directory is not specified, the traces
  are placed in derby.system.home.

Security Considerations

For general security considerations for the Network Server, see Network Server security.

The "host" parameter allows configuration of the host name that will be used for the
listening socket for network connections. By default, the Network Server will listen to
requests only on the loopback address, which means that it will only accept connections
from the local host. Changing this value could expose the server to external connections,
which raises security concerns, so before using the "host" parameter, you should run
under the Java security manager and enable user authentication.

This section describes the servlet pages.

Start-up page

Use the start-up page to start the server.

In addition to starting the Network Server, you can use the startup page to perform the
following actions:

• Turn logging on when the server is started.
• Turn tracing on for all sessions when the server is started.

Running page

If the Network Server is running (whether it was started by initializing the servlet or in
some other manner), the running page is displayed. The running page indicates whether
logging is on or off, whether tracing is on or off, and if tracing is on, indicates for which session.

You can use the running page to stop the server and turn logging and tracing on or off. The following options are available from the running page:

- Start or stop logging.
- Start or stop tracing all sessions.
- Specify session to trace. (If you choose this option, the Trace session page is displayed.)
- Change tracing directory (If you choose this option, the Trace directory page is displayed.)
- Specify threading parameters for Network Server. (If you choose this option, the Thread parameters page is displayed.)
- Stop the Network Server.

**Trace session page**

If on the running page you choose to specify a session to trace, this page is displayed. You must enter the Session ID.

You are given the option to turn tracing on or off or return to the previous menu. When you push the Trace On/Off button, information indicating the current tracing state is displayed.

**Trace directory page**

This page is displayed if the you choose to change the tracing directory on the Running page. You must enter the Trace Directory.

You can either set a tracing directory, or you can return to the previous menu. Additional information is displayed that indicates the current tracing directory when you push the Set Directory button.

**Set Network Server parameters**

The first page is displayed if the thread parameter button is pressed. Use this page to set the new parameters. Enter the following information:

- New maximum number of threads
- New thread time slice

If either the maximum threads or time slice parameters are left blank, that value is left unchanged from the current setting.

Click Set Network Server parameters to display the updated values for the maximum threads and the time slice parameters.

**Derby Network Server advanced topics**

This section discusses several advanced topics for users of the Derby Network Server.

**Network Server security**

By default, the Derby Network Server will only listen on the localhost. Clients must use the localhost host name to connect. By default, clients cannot access the Network Server from another host. To enable connections from other hosts, set the derby.drda.host property, or start the Network Server with the `-h` option in the `java org.apache.derby.drda.NetworkServerControl start` command.
In the following example the server will listen only on localhost and clients cannot access the server from another host.

```java
org.apache.derby.drda.NetworkServerControl start
```

In the following example, the server runs on host machine `sampleserver.sampledomain.com` and also listens for clients from other hosts. Clients must specify the server in the URL or DataSource as `sampleserver.sampledomain.com`:

```java
org.apache.derby.drda.NetworkServerControl start -h sampleserver.sampledomain.com
```

To start the Network Server so that it will listen on all interfaces, start with an IP address of `0.0.0.0`, shown in the following example:

```java
org.apache.derby.drda.NetworkServerControl start -h 0.0.0.0
```

A server that is started with the `-h 0.0.0.0` option will listen to client requests that originate from both `localhost` and from other machines on the network.

However, administrative commands (for example, `org.apache.derby.drda.NetworkServerControl shutdown`) can run only on the host where the server was started, even if the server was started with the `--h` option.

### Running the Network Server under the security manager

By default, the Network Server boots with a Basic security policy. You are encouraged to customize this policy to fit the security needs of your application and its runtime environment. You may also run the Network Server without a security manager, although this is not recommended.

#### Basic Network Server security policy

If you boot the Network Server without specifying a security manager, the Network Server will install a default Java security manager enforcing a Basic policy. This happens if you boot the Network Server as your VM's entry point, e.g.:

```java
org.apache.derby.drda.NetworkServerControl start ...
```

Note that you should run your Network Server with user authentication enabled. For details on how to enable user authentication, please see "Working with user authentication" in the *Derby Developer's Guide*.

Some of your application code may run as procedures and functions which you have declared using the `CREATE PROCEDURE` and `CREATE FUNCTION` statements. You will need to add privileged blocks to your declared procedures and functions if they perform sensitive operations such as file and network i/o, classloading, system property reading, etc.

If for some reason you do not want to run your client/server application under a security manager, you may override the Network Server's impulse to install a default policy. For details, see *Running the Network Server without a security policy*.

Note that the Network Server attempts to install a security manager only if you boot the server as the entry point of your VM. The Network Server will not attempt to install a security manager if you start the server from your application using the programmatic API described in the following section: *Starting the Network Server from a Java application*.

You will find a template security policy in the Derby distribution at `demo/templates/server.policy`. Most likely, you will want to customize this policy. For
example, probably you will want to restrict the server's liberal file i/o permissions which let
the server backup/restore to/from any location in the local file system. For details on how
to customize the Template policy, please see Customizing the Network Server's security
policy. The following example is a copy of the Basic policy:

grant codeBase "${derby.install.url}derby.jar"
{
  // These permissions are needed for everyday, embedded Derby usage.
  permission java.lang.RuntimePermission "createClassLoader";
  permission java.util.PropertyPermission "derby.*", "read";
  permission java.util.PropertyPermission "user.dir", "read";
  permission java.util.PropertyPermission "derby.storage.jvmInstanceId",
    "write";
  permission java.io.FilePermission "${derby.system.home}"", "read";
  permission java.io.FilePermission "${derby.system.home}/${/}-", "read,write,delete";

  // This permission lets you backup and restore databases
  // to and from arbitrary locations in your file system.
  // This permission also lets you import/export data to and from
  // arbitrary locations in your file system.
  // You may want to restrict this access to specific directories.
  permission java.io.FilePermission "<ALL FILES>"", "read,write,delete";
};

grant codeBase "${derby.install.url}derbynet.jar"
{
  // This permission lets the Network Server manage connections from
  // clients.
  // Accept connections from any host. Derby is listening to the host
  // interface specified via the -h option to "NetworkServerControl
  // start" on the command line, via the address parameter to the
  // org.apache.derby.drda.NetworkServerControl constructor in the API
  // or via the property derby.drda.host; the default is localhost.
  // You may want to restrict allowed hosts, e.g. to hosts in a specific
  // subdomain, e.g. "*.acme.com".

   permission java.net.SocketPermission "*", "accept";
};

Customizing the Network Server's security policy

The Network Server's Basic security policy is documented in the section Basic Network
Server security policy. Most likely, you will want to customize your own security policy.
For example, you might want to restrict the server's liberal file i/o permissions which let
the server backup to and restore from any location in the local file system. Customizing
the security policy is simple:

- A template policy lives in the Derby distribution at demo/templates/server.policy.
  Copy the from this location to your own file, say myCustomized.policy. All of the
  following edits take place in your custom file.
- Replace the ${derby.install.url} variable with the location of the Derby jars in your
  local file system.
- Replace the ${derby.system.home} variable with the location of your Derby system
directory. Alternatively, rather than replacing this variable, you can simply set the
  value of the derby.system.home system property when you boot the server.
• Replace the \${derby.security.host} variable with the address of the network interface on which the server is listening. Note that the special wildcard address "0.0.0.0" is not understood by SocketPermission, even though Derby accepts this wildcard as a valid value for the -h option of the Network Server startup command. If you must grant blanket permission to this wildcard address, specify "*" in the policy file.

• Refine the file permissions needed by backup/restore, import/export, and the loading of application jars.

The following example is a copy of a sample, customized policy file:

```java
grant codeBase "file:/usr/local/share/sw/derby/lib/derby.jar"
{
    // These permissions are needed for everyday, embedded Derby usage.
    //
    // These permissions are needed for everyday, embedded Derby usage.
    //
    permission java.lang.RuntimePermission "createClassLoader";
    permission java.util.PropertyPermission "derby.*", "read";
    permission java.util.PropertyPermission "user.dir", "read";
    permission java.io.FilePermission
        "/usr/local/shoppingCartApp/databases","read";
    permission java.io.FilePermission
        "/usr/local/shoppingCartApp/databases/-", "read,write,delete";
    permission java.util.PropertyPermission
        "derby.storage.jvmInstanceId", "write";
    //
    // This permission lets the Network Server manage connections from
    // This permission lets the Network Server manage connections from
    // clients.
    //
    permission java.net.SocketPermission "localhost:0-", "accept";
}
```
After customizing the Basic policy, you may bring up the Network Server as follows:

```java
java -Djava.security.manager -Djava.security.policy=/usr/local/shoppingCartApp/lib/myCustomized.policy org.apache.derby.drda.NetworkServerControl start -h localhost
```

**Running the Network Server without a security policy**

You may override the Network Server's impulse to install a security manager if, for some reason, you need to run your application outside Java's security protections.

**CAUTION:** You incur a severe security risk by opening up the server to all clients without limiting access via user authentication and a security policy.

Use the `-noSecurityManager` option to force the Network Server to come up without a security manager. E.g.:

```java
java org.apache.derby.drda.NetworkServerControl start -h localhost -noSecurityManager
```

**Network encryption and authentication with SSL/TLS**

By default, all Derby network traffic is unencrypted, with the exception of user names and user passwords which may be encrypted separately (See Network client security). There is also no network layer access control mechanism. For deployment scenarios where these are possible security issues, Derby Network Server supports network security with Secure Socket Layer/Transport Layer Security (SSL/TLS).

With SSL/TLS, the client/server communication protocol is encrypted and both the client and the server may independently of each other require certificate based authentication of the other part.

It is assumed that the reader is somewhat familiar with SSL, key pairs and certificates. This documentation is also based on the Sun JDK and its `keytool` application.

For the remainder of this section, the term SSL is used for SSL/TLS and the term peer is used for the other part of the communication (The server’s peer is the client and vice versa).

SSL for Derby (both for client and for server) operates in three possible modes:

- **off**
  - The default, no SSL encryption
- **basic**
  - SSL encryption, no peer authentication
- **peerAuthentication**
  - SSL encryption and peer authentication

Peer authentication may be set either on the server or on the client or on both. Peer authentication means that the other side of the SSL connection is authenticated based on a trusted certificate installed locally.

Alternatively, a Certification Authority (CA) certificate may be installed locally and the peer has a certificate signed by that authority. How to achieve this is not described in this document. Consult your Java environment documentation for details on this.

**Attention:** If a plaintext client tries to communicate with an SSL server or an SSL client tries to communicate with a plaintext server, the plaintext side of the communication will see the SSL communication as noise and report protocol errors.
Key and certificate handling

For SSL operation, the server always needs a key pair. If the server runs in peer authentication mode (the server authenticates the clients), then each client needs its own key pair. In general, if one end of the communication wants to authenticate its partner, then the first end needs to install a certificate generated by the partner.

The key pair is located in a file which is called a key store and the JDK's SSL provider needs the system properties `javax.net.ssl.keyStore` and `javax.net.ssl.keyStorePassword` to access the key store.

The certificates of trusted parties are installed in a file called a trust store. The JDK's SSL provider needs the system properties `javax.net.ssl.trustStore` and `javax.net.ssl.trustStorePassword` to access the trust store.

Key pair generation

Key pairs are generated with `keytool -genkey`. The simplest way to generate a key pair is to do

```
keytool -genkey <alias> -keystore <keystore>
```

`keytool` will prompt for needed information like identity details and passwords.

Consult the JDK documentation for more information on `keytool`.

Certificate generation

Certificates are generated with `keytool -export` like this:

```
keytool -export -alias <alias> -keystore <keystore> \
    -rfc -file <certificate file>
```

The certificate file may then be distributed to the relevant parties.

Certificate installation

Installation of a certificate in a trust store is done with `keytool -import` like this:

```
keytool -import -alias <alias> -file <certificate file> \
    -keystore <trust store>
```

Examples

Generate the server key pair:

```
keytool -genkey -alias myDerbyServer -keystore serverKeyStore.key
```

Generate a server certificate:

```
keytool -export -alias myDerbyServer -keystore serverKeyStore.key \
    -rfc -file myServer.cert
```

Generate a client key pair:

```
keytool -genkey -alias aDerbyClient -keystore clientKeyStore.key
```

Generate a client certificate:

```
keytool -export -alias aDerbyClient -keystore clientKeyStore.key \
    -rfc -file aClient.cert
```

Install a client certificate in the server's trust store:
keytool -import -alias aDerbyClient -file aClient.cert
    -keystore serverTrustStore.key

Install the server certificate in a client's trust store:

keytool -import -alias myDerbyServer -file myServer.cert
    -keystore clientTrustStore.key

Starting the server with SSL/TLS

For server SSL/TLS, a server key pair needs to be generated. If the server is going to do
client authentication, the client certificates need to be installed in the trust store. These
operations are described in Key and certificate handling.

SSL at the server side is activated with the property derby.drda.sslMode (default off)
or the -ssl option for the server start command.

Starting the server with basic SSL encryption

When the SSL mode is set to basic, the server will only accept SSL encrypted
connections.

The properties javax.net.ssl.keyStore and
javax.net.ssl.keyStorePassword need to be set with the proper values.

Example:

java -Djavax.net.ssl.keyStore=serverKeyStore.key \
    -Djavax.net.ssl.keyStorePassword=qwerty \
    -jar derbyrun.jar server start -ssl basic

Starting a server which authenticates clients

When the server's SSL mode is set to peerAuthentication, then the server
authenticates its clients' identity in addition to encrypting network traffic. In this situation,
the server's trust store must contain a certificate for each client which will connect.

The javax.net.ssl.trustStore and javax.net.ssl.trustStorePassword
need to be set in addition to the properties above.

See Running the client with SSL/TLS for client settings when the server does client
authentication

Example:

java -Djavax.net.ssl.keyStore=serverKeyStore.key \
    -Djavax.net.ssl.keyStorePassword=qwerty \
    -Djavax.net.ssl.trustStore=serverTrustStore.key \
    -Djavax.net.ssl.trustStorePassword=qwerty \
    -jar derbyrun.jar server start -ssl peerAuthentication

Running the client with SSL/TLS

Basic SSL encryption on the client is enabled either by the URL attribute ssl, the
property ssl or the datasource attribute ssl set to basic.

Example:

Connection c =
    getConnection("jdbc:derby://myhost:1527/db;ssl=basic");

Running a client which authenticates the server
If the client wants to authenticate the server, then the client's trust store must contain the server's certificate. See Key and certificate handling.

Client SSL with server authentication is enabled by the URL attribute ssl or the property ssl set to peerAuthentication. In addition, the system properties javax.net.ssl.trustStore and javax.net.ssl.trustStorePassword need to be set.

Example:

```java
System.setProperty("javax.net.ssl.trustStore","clientTrustStore.key");
System.setProperty("javax.net.ssl.trustStorePassword","qwerty");
Connection c = getConnection("jdbc:derby://myhost:1527/db;ssl=peerAuthentication");
```

Running the client when the server does client authentication

If the server does client authentication, the client will need a key pair and a client certificate which is installed in the server's trust store, See Key and certificate handling.

The client needs to set javax.net.ssl.keyStore and javax.net.ssl.keyStorePassword.

Example:

```java
System.setProperty("javax.net.ssl.keyStore","clientKeyStore.key");
System.setProperty("javax.net.ssl.keyStorePassword","qwerty");
Connection c =
    getConnection("jdbc:derby://myhost:1527/db;ssl=basic");
```

Running the client when both parties do peer authentication

This is a combination of the two last variants.

Example:

```java
System.setProperty("javax.net.ssl.keyStore","clientKeyStore.key");
System.setProperty("javax.net.ssl.keyStorePassword","qwerty");
System.setProperty("javax.net.ssl.trustStore","clientTrustStore.key");
System.setProperty("javax.net.ssl.trustStorePassword","qwerty");
Connection c =
    getConnection("jdbc:derby://myhost:1527/db;ssl=peerAuthentication");
```

Other server commands

The other server commands (shutdown, ping, sysinfo, runtimeinfo, logconnections, maxthreads, timeslice, trace, tracedirectory) are implemented as clients, and they behave exactly as clients with regards to SSL. The SSL mode is set with the property derby.drda.sslMode or the server command option -ssl.

Example:

```java
java -jar derbyrun.jar server shutdown -ssl basic
```

will shutdown an SSL-enabled server.

Example:

Similarly, if you have peerAuthentication on both sides, use the following command:
Configuring the Network Server to handle connections

You can configure the Network Server to use a specific number of threads to handle connections. You can change the configuration on the command line or by using the servlet interface.

The minimum number of threads is the number of threads that are started when the Network Server is booted. This value is specified as a property, derby.drda.minThreads = <min>. The maximum number of threads is the maximum number of threads that will be used for connections. If more connections are active than there are threads available, the extra connections must wait until the next thread becomes available. Threads can become available after a specified time, which is checked only when a thread has finished processing a communication.

- You can change the maximum number of threads by using the following command:

  java org.apache.derby.drda.NetworkServerControl maxthreads <max> [-h <hostname>] [-p <portnumber>]

  You can also use the derby.drda.maxThreads property to assign the maximum value. A <max> value of 0 means that there is no maximum and a new thread will be generated for a connection if there are no current threads available. This is the default. The <max> and <min> values are stored as integers, so the theoretical maximum is 2147483647 (the maximum size of an integer). But the practical maximum is determined by the machine configuration.

- To change the time that a thread should work on one session's request and check if there are waiting sessions, use the following command:

  java org.apache.derby.drda.NetworkServerControl
timeslice <milliseconds> [-h <hostname>] [-p <portnumber>]

  You can also use the derby.drda.timeSlice property to set this value. A value of 0 milliseconds indicates that the thread will not give up working on the session until the session ends. A value of -1 milliseconds indicates to use the default. The default value is 0. The maximum number of milliseconds that can be specified is 2147483647 (the maximum size of an integer).

Controlling logging by using the log file

The Network Server uses the derby.log file to log problems that it encounters. It also logs connections when the property derby.drda.logConnections is set to true. The derby.log file is created when the Derby server is started. The Network Server then records the time and version. If a log file exists, it is overwritten, unless the property derby.infolog.append is set to true.

When the Network Server is logging connections, it also logs the Connection Number; this log message is written both to the derby.log file and to the Network Server console.

- To turn on connection logging, you can use the servlet interface or you can issue the following command:

  java org.apache.derby.drda.NetworkServerControl
Controlling tracing by using the trace facility

Use the trace facility only if you are working with technical support and they require tracing information.

See Managing the Derby Network Server remotely by using the servlet interface for information about managing the trace facility using the servlet interface.

Turning on the trace facility

1. Turn on tracing for all sessions by specifying the following property:

   
   ```
   derby.drda.traceAll=true
   ```

   Alternatively, while the Network Server is running, you can use the following command to turn on the trace facility:

   ```
   java org.apache.derby.drda.NetworkServerControl
   trace on [-s <connection number>] [-h <hostname>] [-p <portnumber>]
   ```

   If you specify a `<connection number>`, tracing will be turned on only for that connection.

2. Set the location of the tracing files by specifying the following property:

   ```
   derby.drda.traceDirectory=<directory for tracing files>
   ```

   You need to specify only the directory where the tracing files will reside. The names of the tracing files are determined by the system. If you do not set a trace directory, the tracing files will be placed in derby.system.home.

3. While the Network Server is running, enter the following command to set the trace directory:

   ```
   java org.apache.derby.drda.NetworkServerControl traceDirectory
   <directory for tracing files> [-h <hostname>] [-p <portnumber>]
   ```

Turning off the trace facility

Enter the following command to turn off tracing:

```
java org.apache.derby.drda.NetworkServerControl trace off [-s <connection number>]
[-h <hostname>] [-p <portnumber>]
```}

The tracing files are named ServerX.trace, where X is a connection number.

Derby Network Server sample programs

This section describes several Derby Network Server sample programs for Network Server users.

The NsSample sample program

The `NsSample` demonstration program is a simple JDBC application that interacts with the Network Server.
The **NsSample** program performs the following tasks:

- Starts the Network Server.
- Checks that the Network Server is running.
- Loads the Network Client driver. (Note that this step is not necessary if you are running the client on JDK 1.6 or higher. In that environment, the network client driver loads automatically.)
- Creates the **NsSampledb** database if not already created.
- Checks to see if the schema is already created, and if not, creates the schema which includes the SAMPLETBL table and corresponding indexes.
- Connects to the database.
- Loads the schema by inserting data.
- Starts client threads to perform database related operations.
- Has each of the clients perform DML operations (select, insert, delete, update) using JDBC calls. For example, one client thread establishes an embedded connection to perform database operations, while another client thread establishes a client connection to the Network Server to perform database operations.
- Waits for the client threads to finish the tasks.
- Shuts down the Network Server at the end of the demonstration.

You must install the following files in the `%DERBY_HOME%/demo/nserverdemo/` directory before you can run the sample program:

- **NsSample.java**
  
  This is the entry point into the sample program. The program starts up two client threads. The first client establishes an embedded connection to perform database operations, and the second client establishes a client connection to the Network Server to perform database operations.

  You can change the following constants to modify the sample program:

  **NUM_ROWS**
  The number of rows that must be initially loaded into the schema.

  **ITERATIONS**
  The number of iterations for which each client thread does database related work.

  **NUM_CLIENT_THREADS**
  The number of clients that you want to run the program against.

  **NETWORKSERVER_PORT**
  The port on which the Network Server is running.

- **NsSampleClientThread.java**
  
  This file contains two Java classes:
  
  - The **NsSampleClientThread** class extends **Thread** and instantiates a **NsSampleWork** instance.
  - The **NsSampleWork** class contains everything that is required to perform DML operations using JDBC calls. The **doWork** method in the **NsSampleWork** class represents all the work done as part of this sample program.

- **NetworkServerUtil.java**
  
  This file contains helper methods to start the Network Server and to shutdown the server.

The compiled class files for the **NsSample** program are:

- **NsSample.class**
- **NsSampleClientThread.class**
- **NsSampleWork.class**
- **NetworkServerUtil.class**

**Running the NsSample sample program**

To run the **NsSample** program:
1. Open a command prompt and change directories to the `%DERBY_HOME%\demo\` directory, where `%DERBY_HOME%` is the directory where you installed Derby.

2. Set the CLASSPATH to the current directory ("." ) and also include the following jar files in order to use the Network Server and the network client driver:

   **derbynet.jar**
   The Network Server jar file. It must be in your CLASSPATH to use any of the Network Server functions.

   **derbyclient.jar**
   This jar file must be in your CLASSPATH to use the Network Client driver.

   **derby.jar**
   The Derby database engine jar file.

   **derbytools.jar**
   The Derby tools jar file.

3. Test the CLASSPATH settings by running the following Java command:

   ```
   java org.apache.derby.tools.sysinfo
   ```

   This command shows the Derby jar files that are in the classpath as well as their respective versions.

4. After you set up your environment correctly, run the NsSample program from the same directory:

   ```
   java nserverdemo.NsSample
   ```

   If the program runs successfully, you will receive output similar to that shown in the following table:
Running the \textit{NsSample} program also creates the following new directories and files:

\textbf{NSSampledb}
- This directory makes up the \textit{NSSampledb} database.

\textbf{derby.log}
- This log file contains Derby progress and error messages.

\textbf{Network Server sample programs for embedded and client connections}

This Derby Network Server sample program demonstrates how to obtain an embedded connection and client connections to the same database by using the Network Server. This program shows how to use either the DriverManager or a DataSource to obtain client connections.

For a database to be consistent, only one JVM can access it at a time. The embedded driver is loaded when the Network Server is started. The JVM that starts the Network Server can obtain an embedded connection to the same database that the Network Server is accessing to serve clients from other JVMs. This solution provides the performance benefits of the embedded driver and also allows client connections from other JVMs to connect to the same database.

\textbf{Overview of the SimpleNetworkServerSample program}

The SimpleNetworkServerSample program starts the Derby Network Server, as well as the embedded driver, and waits for clients to connect. The program performs the following tasks.

- Starts the Derby Network Server by using a property and also loads the embedded driver
- Determines if the Network Server is running
- Creates the NSSimpleDB database if it is not already created
- Obtains an embedded database connection
- Tests the database connection by executing a sample query
- Allows client connections to connect to the server until you decide to stop the server and exit the program
- Closes the connection
- Shuts down the Network Server before exiting the program

To run the sample program, install the following files in the \texttt{%DERBY_HOME%}/demo\nserverdemo\ directory:

- The source file: SimpleNetworkServerSample.java
- The compiled class file: SimpleNetworkServerSample.class

\textbf{Running the SimpleNetworkServerSample program}

To run the Derby Network Server sample program:
1. Open a command prompt and change directories to the
%DERBY_HOME%\demo\serverdemo directory, where %DERBY_HOME% is the
directory where you installed Derby.
2. Set the classpath to include the current directory ("."), and the following jar files:

   **derbynet.jar**
   The Network Server jar file. It must be in your CLASSPATH because you start the
   Network Server in this program.

   **derby.jar**
   The database engine jar file.

   **derbytools.jar**
   The Derby tools jar file.

3. Test the CLASSPATH settings by running the following Java command:

   ```java
   java org.apache.derby.tools.sysinfo
   ```
   This command displays the Derby jar files that are in the classpath.

4. After you set up your environment correctly, run the SimpleNetworkServerSample
program from the same directory:

   ```java
   java SimpleNetworkServerSample
   ```
   If the program runs successfully, you will receive output that is similar to that shown
in the following exampleS:

   ```
   Starting Network Server
   Testing if Network Server is up and running!
   Derby Network Server now running
   Got an embedded connection.
   Testing embedded connection by executing a sample query
   number of rows in sys.systables = 16
   While my app is busy with embedded work, ij might connect like this:
   $ java -Dij.user=me -Dij.password=pw -Dij.protocol=
   jdbc:derby:\localhost:1527\ org.apache.derby.tools.ij
   iJ> connect 'NSSimpleDB';
   Clients can continue to connect:
   Press [Enter] to stop Server
   ```

Running the SimpleNetworkServerSample program also creates the following new
directories and files:

   **NSSimpleDB**
   This directory makes up the NSSimpleDB database.

   **derby.log**
   This log file contains Derby progress and error messages.

Connecting a client to the Network Server with the SimpleNetworkClientSample program
The SimpleNetworkClientSample program is a client program that interacts with the
Derby Network Server from another JVM. The program performs the following tasks:

- Loads the network client driver. (Note that this step is not necessary if you are
running the client on JDK 1.6 or higher. In that environment, the network client
driver loads automatically.)
- Obtains a client connection by using the DriverManager.
- Obtains a client connection by using a DataSource.
- Tests the database connections by running a sample query.
- Closes the connections and then exits the program.

You must install the following files in the %DERBY_HOME%\demo\serverdemo directory before you can run the sample program:

- The source file: SimpleNetworkClientSample.java.
- The compiled class file: SimpleNetworkClientSample.class.
Running the SimpleNetworkClientSample program

To connect to the Network Server that has been started with the SimpleNetworkServerSample program:

1. Open a command prompt and change directories to the %DERBY_HOME%\demo\serverdemo directory, where %DERBY_HOME% is the directory where you installed Derby.
2. Set the classpath to include the following jar files:
   - The current directory (".")
   - derbyclient.jar
3. After you set up your environment correctly, run the SimpleNetworkClientSample program from the same directory:

   java SimpleNetworkClientSample

If the program runs successfully, you will receive output similar to that shown in the following example:

```
Starting Sample client program
Got a client connection via the DriverManager.
connection from datasource;
Got a client connection via a DataSource.
Testing the connection obtained via DriverManager by executing a sample query
number of rows in sys.systables = 16
Testing the connection obtained via a DataSource by executing a sample query
number of rows in sys.systables = 16
Goodbye!
```
Part two: Derby Administration Guide

This section of the guide is divided into several administrative tasks.

Checking database consistency

If you experience hardware or operating system failure, you can use the SYSCS_UTIL.SYSCS_CHECK_TABLE function to verify that the database is still consistent.

Check consistency only if there are indications that such a check is needed because a consistency check can take a long time on a large database.

The SYCS_Check_TABLE function

The SYCS_UTIL.SYSCS_CHECK_TABLE() function checks the consistency of a Derby table. In particular, the SYCS_UTIL.SYSCS_CHECK_TABLE function verifies the following conditions:

- Base tables are internally consistent
- Base tables and all associated indexes contain the same number of rows
- The values and row locations in each index match those of the base table
- All BTREE indexes are internally consistent

You run this function in an SQL statement, as follows:

```
VALUES SYCS_UTIL.SYSCS_CHECK_TABLE(
    SchemaName, TableName)
```

where `SchemaName` and `TableName` are expressions that evaluate to a string data type.

If you created a schema or table name as a non-delimited identifier, you must present their names in all upper case. For example:

```
VALUES SYCS_UTIL.SYSCS_CHECK_TABLE('APP', 'CITIES')
```

The SYCS_UTIL.SYSCS_CHECK_TABLE function returns a smallint. If the table is consistent (or if you run SYCS_UTIL.SYSCS_CHECK_TABLE on a view), SYCS_UTIL.SYSCS_CHECK_TABLE returns a non-zero value. Otherwise, the function throws an exception on the first inconsistency that it finds.

For a consistent table, the following result is displayed:

```
1
-----
1
1 row selected
```

Sample SYCS_Check_TABLE error messages

This section provides examples of error messages that the SYCS_UTIL.SYSCS_CHECK_TABLE() function can return.

If the row counts of the base table and an index differ, error message X0Y55 is issued:

```
ERROR X0Y55: The number of rows in the base table does not match the number of rows in at least 1 of the indexes on the table. Index 'T1_I' on table 'APP.T1' has 4 rows, but the base table has 5 rows. The suggested corrective action is to recreate the index.
```
If the index refers to a row that does not exist in the base table, error message X0X62 is issued:

ERROR X0X62: Inconsistency found between table 'APP.T1' and index 'T1_I'. Error when trying to retrieve row location '(1,6)' from the table. The full index key, including the row location, is '{ 1, (1,6) }'. The suggested corrective action is to recreate the index.

If a key column value differs between the base table and the index, error message X0X61 is issued:

ERROR X0X61: The values for column 'C10' in index 'T1_C10' and table 'APP.T1' do not match for row location (1,7). The value in the index is '2 2   ', while the value in the base table is 'NULL'. The full index key, including the row location, is '{ 2 2       , (1,7) }'. The suggested corrective action is to recreate the index.

Sample SYCS_CHECK_TABLE queries

This section provides examples that illustrate how to use the SYCS_UTIL.SYSCS_CHECK_TABLE function in queries.

To check the consistency of a single table, run a query that is similar to the one shown in the following example:

VALUES SYCS_UTIL.SYSCS_CHECK_TABLE('APP', 'FLIGHTS')

To check the consistency of all of the tables in a schema, stopping at the first failure, run a query that is similar to the one shown in the following example:

SELECT tablename, SYCS_UTIL.SYSCS_CHECK_TABLE('SAMP', tablename)
FROM sys.sysschemas s, sys.systables t
WHERE s.schemaname = 'SAMP' AND s.schemaid = t.schemaid

To check the consistency of an entire database, stopping at the first failure, run a query that is similar to the one shown in the following example:

SELECT schemaname, tablename,
SYCS_UTIL.SYSCS_CHECK_TABLE(schemaname, tablename)
FROM sys.sysschemas s, sys.systables t
WHERE s.schemaname = t.schemaname

Backing up and restoring databases

Derby provides a way to back up a database while it is online. You can also restore a full backup from a specified location.

Backing up a database

The topics in this section describe how to back up a database.

Offline backups

To perform an offline backup of a database, use operating system commands to copy the database directory. You must shut down the database prior to performing an offline backup.

For example, on Windows systems, the following operating system command backs up a (closed) database that is named sample and that is located in d:\mydatabases by copying it to the directory c:\mybackups\2005-06-01:
xcopy d:\mydatabases\sample c:\mybackups\2005-06-01\sample /s /i

If you are not using Windows, substitute the appropriate operating system command for copying a directory and all contents to a new location.

**Note:** On Windows systems, do not attempt to update a database while it is being backed up in this way. Attempting to update a database during an offline backup will generate a java.io.IOException. Using online backups prevents this from occurring.

For large systems, shutting down the database might not be convenient. To back up a database without having to shut it down, you can use an online backup.

**Online backups**

Use online backups to back up a database while it is running.

You can perform online backups by using several types of backup procedures or by using operating systems commands with the freeze and unfreeze system procedures.

**Using the backup procedure to perform an online backup:**

Use the SYSCS_UTIL.SYSCS_BACKUP_DATABASE procedure to perform an online backup of a database to a specified location.

The SYSCS_UTIL.SYSCS_BACKUP_DATABASE procedure takes a string argument that represents the location in which to back up the database. Typically, you provide the full path to the backup directory. (Relative paths are interpreted as relative to the current directory, not to the derby.system.home directory.)

For example, to specify a backup location of `c:/mybackups/2005-06-01` for a database that is currently open, use the following statement (forward slashes are used as path separators in SQL commands):

```
CALL SYSCS_UTIL.SYSCS_BACKUP_DATABASE('c:/mybackups/2005-06-01')
```

The `SYSCS_UTIL.SYSCS_BACKUP_DATABASE()` procedure puts the database into a state in which it can be safely copied, then copies the entire original database directory (including data files, online transaction log files, and jar files) to the specified backup directory. Files that are not within the original database directory (for example, `derby.properties`) are not copied.

The following example shows how to back up a database to a directory with a name that reflects the current date:

```java
public static void backUpDatabase(Connection conn) throws SQLException {
    // Get today's date as a string:
    java.text.SimpleDateFormat todaysDate = new java.text.SimpleDateFormat("yyyy-MM-dd");
    String backupdirectory = "c:/mybackup/" + todaysDate.format((java.util.Calendar.getInstance()).getTime());

    CallableStatement cs = conn.prepareCall("CALL SYSCS_UTIL.SYSCS_BACKUP_DATABASE(?)");
    cs.setString(1, backupdirectory);
    cs.executeUpdate();
    cs.close();
    System.out.println("backed up database to " + backupdirectory);
}
```

For a database that was backed up on 2005-06-01, the previous commands copy the current database to a directory of the same name in `c:/mybackups/2005-06-01`.

Uncommitted transactions do not appear in the backed-up database.
Note: Do not back up different databases with the same name to the same backup directory. If a database of the same name already exists in the backup directory, it is assumed to be an older version and is overwritten.

Unlogged Operations

For some operations, Derby does not log because it can keep the database consistent without logging the data.

The SYSCS_UTIL.SYSCS_BACKUP_DATABASE procedure will issue an error if there are any unlogged operations in the same transaction as the backup procedure.

If any unlogged operations are in progress in other transactions in the system when the backup starts, this procedure will block until those transactions are complete before performing the backup.

Derby automatically converts unlogged operations to logged mode if they are started while the backup is in progress (except operations that maintain application jar files in the database). Procedures to install, replace, and remove jar files in a database are blocked while the backup is in progress.

If you do not want backup to block until unlogged operations in other transactions are complete, use the SYSCS_UTIL.SYSCS_BACKUP_DATABASE_NOWAIT procedure. This procedure issues an error immediately at the start of the backup if there are any transactions in progress with unlogged operations, instead of waiting for those transactions to complete.

Unlogged operations include:

1. Index creation.

   Only CREATE INDEX is logged, not all the data inserts into the index. The reason inserts into the index are not logged is: if there is a failure, it will just drop the index.

   If you create an index when the backup is in progress, it will be slower because it has to be logged.

   Foreign Keys, Primary Keys create backing indexes. Adding those keys to an existing table with data will also run slower.

2. Import to an empty table or replacing all the data in a table.

   In this case also, data inserts into table are not logged. Internally, Derby creates a new table for the import and changes the catalogs to point to the new table and drops the original table when import completes.

   If you perform such an import operation when backup is in progress, it will be slower because data is logged.

Using operating system commands with the freeze and unfreeze system procedures to perform an online backup:

Typically, these procedures are used to speed up the copy operation involved in an online backup. In this scenario, Derby does not perform the copy operation for you. You use the SYSCS_UTIL.SYSCS_FREEZE_DATABASE procedure to lock the database, and then you explicitly copy the database directory by using operating system commands.

For example, because the UNIX tar command uses operating system file-copying routines, and the SYSCS_UTIL.SYSCS_BACKUP_DATABASE procedure uses java I/O calls with additional internal synchronization that allow updates during the backup, the tar command might provide faster backups than the SYSCS_UTIL.SYSCS_BACKUP_DATABASE procedure.
To use operating system commands for online database backups, call the SYSCS_UTIL.SYSCS_FREEZE_DATABASE system procedure. The SYCS_UTIL.SYSCS_FREEZE_DATABASE system procedure puts the database into a state in which it can be safely copied. After the database has been copied, use the SYCS_UTIL.SYSCS_UNFREEZE_DATABASE system procedure to continue working with the database. Only after SYCS_UTIL.SYSCS_UNFREEZE_DATABASE has been specified can transactions once again write to the database. Read operations can proceed while the database is "frozen."

**Note:** To ensure a consistent backup of the database, Derby might block applications that attempt to write to a frozen database until the backup is completed and the SYCS_UTIL.SYSCS_UNFREEZE_DATABASE system procedure is called.

The following example demonstrates how the freeze and unfreeze procedures are used to surround an operating system copy command:

```java
public static void backUpDatabaseWithFreeze(Connection conn) throws SQLException
{
    Statement s = conn.createStatement();
    s.executeUpdate(  
    "CALL SYSCS_UTIL.SYSCS_FREEZE_DATABASE();";
    //copy the database directory during this interval
    s.executeUpdate(  
    "CALL SYSCS_UTIL.SYSCS_UNFREEZE_DATABASE();";
    s.close();
}
```

**When the log is in a non-default location**

**Note:** Read [Logging on a separate device](#) to find out about the default location of the database log.

If you put the database log in a non-default location prior to backing up the database, be aware of the following requirements:

- If you are using an operating system command to back up the database, you must explicitly copy the log file as well, as shown in the following example:

  xcopy d:\mydatabases\sample c:\mybackups\2005-06-01\sample /s /i
  xcopy h:\janet\tourslog\log c:\mybackups\2005-06-01\sample\log /s /i

If you are not using Windows, substitute the appropriate operating system command for copying a directory and all of its contents to a new location.

- Edit the `logDevice` entry in `service.properties` of the database backup so that it points to the correct location for the log. In the previous example, the log was moved to the default location for a log, so you can remove the `logDevice` entry entirely, or leave the `logDevice` entry as is and wait until the database is restored to edit the entry.

See [Logging on a separate device](#) for information about putting the log in a non-default location.

**Backing up encrypted databases**

When you back up an encrypted database, both the backup and the log files remain encrypted.

To restore an encrypted database, you must know the boot password.

### Restoring a database from a backup copy

To restore a database by using a full backup from a specified location, specify the `restoreFrom=Path` attribute in the boot time connection URL.
If a database with the same name exists in the derby.system.home location, the system will delete the database, copy it from the backup location, and then restart it.

The log files are copied to the same location they were in when the backup was taken. You can use the logDevice attribute in conjunction with the restoreFrom=Path attribute to store logs in a different location.

For example, to restore the sample database by using a backup copy in c:\mybackups\sample, the connection URL should be:

```
jdbc:derby:sample;restoreFrom=c:\mybackups\sample
```

**Creating a database from a backup copy**

To create a database from a full backup copy at a specified location, specify the createFrom=Path attribute in the boot time connection URL.

If there is already a database with the same name in derby.system.home, an error will occur and the existing database will be left intact. If there is not an existing database with the same name in the current derby.system.home location, the system will copy the whole database from the backup location to derby.system.home and start it.

The log files are also copied to the default location. You can use the logDevice attribute in conjunction with the createFrom=Path attribute to store logs in a different location. With the createFrom=Path attribute, you do not need to copy the individual log files to the log directory.

For example, to create the sample database from a backup copy in c:\mybackups\sample, the connection URL should be:

```
jdbc:derby:sample;createFrom=c:\mybackups\sample
```

**Roll-forward recovery**

Derby supports roll-forward recovery to restore a damaged database to the most recent state before a failure occurred.

Derby restores a database from full backup and replays all the transactions after the backup. All the log files after a backup are required to replay the transactions after the backup. By default, the database keeps only logs that are required for crash-recovery. For roll-forward recovery to be successful, all log files must be archived after a backup. Log files can be archived using the backup function calls that enable log archiving.

In roll-forward recovery the log archival mode ensures that all old log files are available. The log files are available only from the time that the log archival mode is enabled.

Derby uses the following information to restore the database:
- The backup copy of the database
- The set of archived logs
- The current online active log

You cannot use roll-forward recovery to restore individual tables. Roll-forward recovery recovers the entire database.

To restore a database by using roll-forward recovery, you must already have a backup copy of the database, all the archived logs since the backup was created, and the active log files. All the log files should be in the database log directory.

There are two types of log files in Derby: active logs and online archived logs.

**Active logs**
Active logs are used during crash recovery to prevent a failure that might leave a
database in an inconsistent state. Roll-forward recovery can also use the active logs
to recover to the end of the log files. Active logs are located in the database log path
directory.

**Online archived logs**

Log files that are stored for roll-forward recovery use when they are no longer needed
for crash recovery. Online archived logs are also kept in the database log path
directory.

**Enabling log archival mode**

Online archive logs are available only if the database is enabled for log archival mode. You can use the following system procedure to enable the database for log archival mode:

```sql
SYSCS_UTIL.SYSCS_BACKUP_DATABASE_AND_ENABLE_LOG_ARCHIVE_MODE
(IN BACKUPDIR VARCHAR(32672), IN SMALLINT DELETE_ARCHIVED_LOG_FILES)
```

The input parameters for the calls in the previous example specify the location
where the backup should be stored and specify whether or not the database should
keep online archived logs for the backup. Existing online archived log files that
were created before this backup will be deleted if the input parameter value for the
deleteOnlineArchivedLogFiles parameter is non-zero. The log files are deleted only after
a successful backup.

**Note:** Make sure to store the backup database in a safe place when you choose the log
file removal option.

The
SYCS_UTIL.SYSCS_BACKUP_DATABASE_AND_ENABLE_LOG_ARCHIVE_MODE
procedure will issue an error if there are any unlogged operations in the same transaction
as backup procedure.

If any unlogged operations are in progress in other transactions in the system when
the backup starts, this procedure will block until those transactions are complete before
performing the backup. Derby automatically converts unlogged operations to logged
mode if they are started while the backup is in progress (except operations that maintain
application jar files in the database). Procedures to install, replace, and remove jar files in
a database are blocked while the backup is in progress.

If you do not want backup to block until unlogged operations
in other transactions are complete, use the
SYCS_UTIL.SYCS_BACKUP_DATABASE_AND_ENABLE_LOG_ARCHIVE_MODE_NOWAIT
procedure. This procedure issues an error immediately at the start of the backup if there
are any transactions in progress with unlogged operations, instead of waiting for those
transactions to complete.

**Disabling log archival mode:**

After you enable log archival mode, the database will always have the log archival mode
enabled even if it is subsequently booted or backed up. The only way to disable the log
archive mode is to run the following procedure:

```sql
SYCS_UTIL.SYCS_DISABLE_LOG_ARCHIVE_MODE(IN SMALLINT
DELETE_ARCHIVED_LOG_FILES)
```

This system procedure disables the log archive mode and deletes any existing online
archived log files if the input parameter `DELETE_ARCHIVED_LOG_FILES` is non-zero.

**Performing roll-forward recovery:**

By using the full backup copy, archived logs, and active logs, you can restore
a database to its most recent state by performing roll-forward recovery. You
perform a roll-forward recovery by specifying a connection URL attribute
`rollForwardRecoveryFrom=<BackupPath>` at boot time. This brings the database to its
most recent state by using full backup copy, archived logs, and active logs. All the log
files should be in the database log path directory.

**Backing up a database:**

In the following example, a database named wombat is backed up to the d:/backup
directory with log archive mode enabled:

```java
connect 'jdbc:derby:wombat;create=true';
create table t1(a int not null primary key);
------------------DML/DDL Operations
CALL SYSCS_UTIL.SYSCS_BACKUP_DATABASE_AND_ENABLE_LOG_ARCHIVE_MODE
('d:/backup', 0);
insert into t1 values(19);
create table t2(a int);
-----------------DML/DDL Operations
-----------------Database Crashed (Media Corruption on data disks)
```

**Restoring a database using roll-forward recovery:**

In the following example, the database is restored using roll-forward recovery after a
media failure:

```java
connect 'jdbc:derby:wombat;rollForwardRecoveryFrom=d:/backup/wombat';
select * from t1;
------------------DML/DDL Operations
```

The following attribute can be specified in the JDBC boot time connection URL:
`rollForwardRecoveryFrom=<Path>`

For more information, see the `rollForwardRecoveryFrom=<Path>` section in the *Derby
Reference Manual*.

After a database is restored from full backup, transactions from the online archived logs
and active logs are replayed.

**Logging on a separate device**

You can improve the performance of update-intensive, large databases by putting a
database's log on a separate device, which reduces I/O contention.

By default, the transaction log is in the `log` subdirectory of the database directory. Use
either of the following methods to store this `log` subdirectory in another location:

- Specify the non-default location by using the `logDevice` attribute on the database
  connection URL when you create the database.
- If the database is already created, move the log manually and update the
  `service.properties` file.

**Using the logDevice attribute**

To specify a non-default location for the log directory, set the logDevice attribute on the
database connection URL when you create the database.

This attribute is meaningful only when you are creating a database. You can specify
`logDevice` as either an absolute path or as a path that is relative to the directory where
the JVM is executed.

Setting logDevice on the database connection URL adds an entry to the
`service.properties` file. If you ever move the log manually, you will need to alter the

---

63
entry in service.properties. If you move the log back to the default location, remove the logDevice entry from the service.properties file.

To check the log location for an existing database, you can retrieve the logDevice attribute as a database property by using the following statement:

```
VALUES SYCS_UTIL.SYCS_GET_DATABASE_PROPERTY('logDevice')
```

### Example of creating a log in a non-default location

The following database connection URL creates a database in the directory d:/mydatabases, but puts the database log directory in h:/janets/tourslog:

```
jdbc:derby:d:/mydatabases/toursDB;create=true;logDevice=h:/janets/tourslog
```

### Example of moving a log manually

If you want to move the log to g:/bigdisk/tourslog, move the log with operating system commands:

```
moved:\janets\tourslog\log\*.* g:\bigdisk\tourslog\log
```

Then, alter the logDevice entry in service.properties to read as follows:

```
logDevice=g:\bigdisk\toursLog
```

**Note:** You can use either a single forward slash or double backslashes for a path separator.

If you later want to move the log back to its default location (in this case, d:\mydatabases\toursDB\log), move the log manually as follows:

```
moved:\bigdisk\tourslog\log\*.* d:\mydatabases\toursDB\log
```

Then, delete the logDevice entry from service.properties.

**Note:** This example uses commands that are specific to the Windows NT operating system. Use commands appropriate to your operating system to copy a directory and all of its contents to a new location.

### Issues for logging in a non-default location

When the log is not in the default location, backing up and restoring a database can require extra steps. See [Backing up and restoring databases](#) for details.

### Obtaining locking information

Derby provides a tool to monitor and display locking information. This tool can help you create applications that minimize deadlock. It can also help you locate the cause of deadlock when it does occur.

To diagnose locking problems, constantly monitor locking traffic by logging all deadlocks by using the derby.locks.monitor property.

### Monitoring deadlocks

The derby.stream.error.logSeverityLevel property determines the level of error that you are informed about.
By default, derby.stream.error.logSeverityLevel is set to 40000. If derby.stream.error.logSeverityLevel is set to display transaction-level errors (that is, if it is set to a value less than 40000), deadlock errors are logged to the derby.log file. If it is set to a value of 40000 or higher, deadlock errors are not logged to the derby.log file.

The derby.locks.monitor property ensures that deadlock errors are logged regardless of the value of derby.stream.error.logSeverityLevel. When derby.locks.monitor is set to true, all locks that are involved in deadlocks are written to derby.log along with a unique number that identifies the lock.

To see a thread's stack trace when a lock is requested, set derby.locks.deadlockTrace to true. This property is ignored if derby.locks.monitor is set to false.

Note: Use derby.locks.deadlockTrace with care. Setting this property can alter the timing of the application, severely affect performance, and produce a very large derby.log file.

For information about how to set properties, and information about the specific properties that are mentioned in this topic, see Tuning Derby.

Here is an example of an error message when Derby aborts a transaction because of a deadlock:

```
--SQLException Caught--
SQLState: 40001 =
Error Code: 30000
Message: A lock could not be obtained due to a deadlock, cycle of locks and waiters is: Lock : ROW, DEPARTMENT, (1,14)
  Waiting XID : {752, X} , APP, update department set location='Boise'
    where deptno='E21'
  Granted XID : {758, X} Lock : ROW, EMPLOYEE, (2,8)
    Waiting XID : {758, U} , APP, update employee set bonus=150 where
      salary=23840
  Granted XID : {752, X} The selected victim is XID : 752
```

Note: You can use the derby.locks.waitTimeout and derby.locks.deadlockTimeout properties to configure how long Derby waits for a lock to be released, or when to begin deadlock checking. For more information about these properties, see the section that discusses controlling Derby application behavior in the Derby Developer's Guide.

Reclaiming unused space

A Derby table or index (sometimes called a conglomerate) can contain unused space after large amounts of data have been deleted or updated.

This happens because, by default, Derby does not return unused space to the operating system. After a page has been allocated to a table or index, Derby does not automatically return the page to the operating system until the table or index is dropped, even if the space is no longer needed. However, Derby does provide a way to reclaim unused space in tables and associated indexes.

If you determine that a table and its indexes have a significant amount of unused space, use either the SYSCS_UTIL.SYSCS_COMPRESS_TABLE or SYSCS_UTIL.SYSCS_INPLACE_COMPRESS_TABLE procedure to reclaim that space. SYSCS_COMPRESS_TABLE is guaranteed to recover the maximum amount of free space, at the cost of temporarily creating new tables and indexes before the statement is committed. SYSCS_INPLACE_COMPRESS attempts to reclaim space within the same table, but cannot guarantee it will recover all available space. The difference between the two procedures is that unlike SYSCS_COMPRESS_TABLE, the SYSCS_INPLACE_COMPRESS procedure uses no temporary files and moves rows around within the same conglomerate.
As an example, after you have determined that the FlightAvailability table and its related indexes have too much unused space, you could reclaim that space with the following command:

call SYCS_UTIL.SYSCS_COMPRESS_TABLE('APP', 'FLIGHTAVAILABILITY', 0);

The third parameter in the SYCS_UTIL.SYSCS_COMPRESS_TABLE() procedure determines whether the operation will run in sequential or non-sequential mode. If you specify 0 for the third argument in the procedure, the operation will run in non-sequential mode. In sequential mode, Derby compresses the table and indexes sequentially, one at a time. Sequential compression uses less memory and disk space but is slower. To force the operation to run in sequential mode, substitute a non-zero SMALLINT value for the third argument. The following example shows how to force the procedure to run in sequential mode:

call SYCS_UTIL.SYSCS_COMPRESS_TABLE('APP', 'FLIGHTAVAILABILITY', 1);

For more information about this command, see the Derby Reference Manual.
Trademarks

The following terms are trademarks or registered trademarks of other companies and have been used in at least one of the documents in the Apache Derby documentation library:

Cloudscape, DB2, DB2 Universal Database, DRDA, and IBM are trademarks of International Business Machines Corporation in the United States, other countries, or both.

Microsoft, Windows, Windows NT, and the Windows logo are trademarks of Microsoft Corporation in the United States, other countries, or both.

Java and all Java-based trademarks are trademarks of Sun Microsystems, Inc. in the United States, other countries, or both.

UNIX is a registered trademark of The Open Group in the United States and other countries.

Other company, product, or service names may be trademarks or service marks of others.