# Table of Contents

Purpose of This Document  ix
Audience  x
How This Document Is Organized  x

## PART ONE
Cloudscape Server Guide

1 Using Cloudscape in a Multi-User Environment  1-1
Cloudscape in a Server Framework  1-1
   Connectivity Configurations  1-2
   Differences Between Embedded and Server Configurations  1-2
   Embedding Servers  1-3
   Using a Server Framework with Cloudsync  1-3
Multi-User Features Available in All Cloudscape Products  1-3
   RmiJdbc  1-4
   Row-Level Locking  1-4
   Multiple Concurrency Levels  1-4
   Multi-Connection and Multi-Threading  1-4
   Administrative Tools  1-5
Cloudconnector  1-5
   Features of Cloudconnector  1-5
   Development  1-6
Integrated Development Environments (IDEs) 1-6
Web Development in Java 1-6
Workspaces 1-6
Integration 1-6
Deployment 1-6
Transactions 1-7
Firewall Support 1-7
Data Caching 1-7
Shared Database Connections 1-7
Server Scalability 1-7
Standard Internet Protocols 1-7
Management 1-8
Graphical Management Console 1-8
Integrated Logging 1-8
The WebLogic JDBC Services Integration 1-8

About this Book and the Cloudconnector Documentation 1-9

2 Installing Cloudscape in a Server Framework 2-1
Installation Directory 2-1
JDK 2-3
Libraries 2-4
When to Use client.jar 2-4
Setting PATH for a Server 2-5
Class Path for a Server 2-5
Library Path for Cloudconnector on UNIX 2-5

3 Starting and Shutting Down the Cloudconnector Server 3-1
Overview 3-1
Starting Cloudconnector on the Server 3-2
Startup Tasks 3-2
Step One: Open a Command Window for Starting the Server 3-3
Step Two: Include Cloudconnector in the PATH Variable 3-3
Step Three: Modify the class path Script (setServerCloudscapeCP) 3-3
Step Four: Modify the Startup Script (startCS) 3-4
### Step Five: Optionally Modify `weblogic.properties` 3-6

- Setting a Password 3-7
- Modifying the Port Number 3-7

### Step Six: Run Both Scripts 3-7

- Verifying Startup 3-8

- Shutting Down Cloudconnector 3-8
  
  - Shutting Down Cloudconnector when User Authentication Is Enabled 3-10
  
  - Using `:shutdown=true` with Concurrent Embedded and Remote Access 3-11

### Accessing Cloudconnector from a Client 4-1

- Preparing Clients 4-1
  
  - JDK Version 4-1
  
  - Class Path 4-2
  
  - Cloudconnector WebLogic Version 4-2

- Using Cloudconnector Clients 4-2
  
  - Specifying the Client JDBC Driver 4-3
    
    - Programmatically 4-3
    
    - From `ij` or Cloudview 4-3
  
  - Specifying the Database Connection URL 4-3
    
    - Example URLs 4-5

- Setting Session-Level WebLogic Properties on the URL 4-5
  
  - Turning off Prefetching and Caching 4-6
  
  - Turning off the Parameter Cache 4-6

- Using `ij` and Cloudview with Cloudconnector 4-7

### Working with Java Data Types in a Server 5-1

- Deploying Java Classes 5-1

- Using the Cloudscape Types on a Client 5-2

- Working with Non-Serializable Types 5-2

### User Authentication Using Cloudconnector 6-1

- Why Use Connection Pools? 6-2
<table>
<thead>
<tr>
<th>Chapter</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>6</td>
<td>Creating a Connection Pool</td>
<td>6-2</td>
</tr>
<tr>
<td></td>
<td>Setting Up Access Control Lists</td>
<td>6-3</td>
</tr>
<tr>
<td></td>
<td>Using a Connection from a Connection Pool</td>
<td>6-3</td>
</tr>
<tr>
<td></td>
<td>Refreshing or Resetting a Connection Pool</td>
<td>6-4</td>
</tr>
<tr>
<td>7</td>
<td>Configuring a Server Framework for Cloudscape</td>
<td>7-1</td>
</tr>
<tr>
<td></td>
<td>Setting Cloudscape Properties</td>
<td>7-1</td>
</tr>
<tr>
<td></td>
<td>System Properties</td>
<td>7-1</td>
</tr>
<tr>
<td></td>
<td>Conglomerate-Specific and Database-Wide Properties</td>
<td>7-2</td>
</tr>
<tr>
<td></td>
<td>Other Properties</td>
<td>7-2</td>
</tr>
<tr>
<td></td>
<td>Setting Cloudconnector Properties</td>
<td>7-2</td>
</tr>
<tr>
<td></td>
<td>Configuring Multi-Threading</td>
<td>7-3</td>
</tr>
<tr>
<td></td>
<td>Disabling HTTP Services</td>
<td>7-4</td>
</tr>
<tr>
<td></td>
<td>Cloudconnector Administration</td>
<td>7-4</td>
</tr>
<tr>
<td>8</td>
<td>SSL Security in Cloudconnector</td>
<td>8-1</td>
</tr>
<tr>
<td></td>
<td>Setting Properties to Enable SSL</td>
<td>8-1</td>
</tr>
<tr>
<td></td>
<td>URLs for SSL</td>
<td>8-2</td>
</tr>
<tr>
<td>9</td>
<td>Cloudscape Access Using RmiJdbc</td>
<td>9-1</td>
</tr>
<tr>
<td></td>
<td>How RmiJdbc Works</td>
<td>9-1</td>
</tr>
<tr>
<td></td>
<td>Configuring Cloudscape for RmiJdbc</td>
<td>9-2</td>
</tr>
<tr>
<td></td>
<td>Setting Class Path</td>
<td>9-2</td>
</tr>
<tr>
<td></td>
<td>RmiJdbc JVM Information</td>
<td>9-2</td>
</tr>
<tr>
<td></td>
<td>Setting Port Numbers</td>
<td>9-3</td>
</tr>
<tr>
<td></td>
<td>Starting and Stopping the RmiJdbc Server</td>
<td>9-3</td>
</tr>
<tr>
<td></td>
<td>Starting the Server</td>
<td>9-3</td>
</tr>
<tr>
<td></td>
<td>Altering the StartCS Script</td>
<td>9-4</td>
</tr>
<tr>
<td></td>
<td>Shutting Down the Server</td>
<td>9-4</td>
</tr>
<tr>
<td></td>
<td>Accessing RmiJdbc Server from a Client</td>
<td>9-5</td>
</tr>
<tr>
<td></td>
<td>Client Class Path</td>
<td>9-5</td>
</tr>
<tr>
<td></td>
<td>JDBC Driver and Database Connection URL for an RmiJdbc Client</td>
<td>9-5</td>
</tr>
<tr>
<td></td>
<td>Client JDK</td>
<td>9-6</td>
</tr>
<tr>
<td></td>
<td>Database Connection URL for RmiJdbc</td>
<td>9-6</td>
</tr>
</tbody>
</table>
Using the Cloudscape Tools with RmiJdbc | 9-7
The RJPing Utility | 9-7

10 Embedding Servers | 10-1
Embedded Server Topology | 10-2
How to Start an Embedded Server From an Application | 10-4
  RmiJdbc Server Example | 10-4
  Cloudconnector Example | 10-5

PART TWO
Cloudscape Administration Guide

11 Checking Database Consistency | 11-1
The checkTable Method | 11-1
Sample Error Messages | 11-2
Sample Queries | 11-3

12 Backing Up and Restoring Databases | 12-1
Backing Up a Database | 12-1
  Offline Backups | 12-1
  On-line Backups | 12-2
    Using the backup Method | 12-2
    Using Operating System Commands with the freeze and unfreeze Methods | 12-3
  When the Log is in a non-Default Location | 12-4
  Backing Up Encrypted Databases | 12-5
Backup History | 12-5
  Failure Recovery | 12-5
Restoring from a Backup | 12-6
  Restoring a Database When the Log Is in a Non-Default Location | 12-6
    When You Backed up with Cloudscape’s Built-In Backup Command | 12-6
    When You Backed up with Operating System Commands | 12-7
13 Logging on a Separate Device
   Using the logDevice Property  13-1
   Example of Creating a Log in a Non-Default Location  13-2
   Example of Moving a Log Manually  13-2
   Issues for Logging in a Non-Default Location  13-2

14 Getting Locking Information
   Displaying Current Locking Information  14-1
      LockTable  14-1
      TransactionTable  14-2
   Example Using the Lock Manager VTIs  14-3
   Lock Monitoring  14-4

Index  IN-1
About This Document

- “Purpose of This Document” on page ix
- “Audience” on page x
- “How This Document Is Organized” on page x

Purpose of This Document

This book explains how to use Cloudscape in a multi-user environment. It also provides information that a server administrator might need to keep Cloudscape running with high performance and reliability in a server framework or a multi-user application server. (When running as single-user databases in embedded mode, Cloudscape databases typically do not need any administration.)

To connect multiple users with Cloudscape, you can embed Cloudscape in a server framework of your own choosing, or you can use the Cloudconnector option. This manual describes these options.

The Cloudsync option provides the ability to develop and deploy source and target databases in a synchronized system. Cloudsync can be very useful in a multi-user framework. See the Cloudscape Synchronization Guide for more information about Cloudsync.
About This Document

Audience

This book contains two parts. The first part is for developers of client/server and multi-user applications. The second part is for administrators.

How This Document Is Organized

This document includes the following chapters:

Part One: (Cloudscape Server Guide):

- **Chapter 1, “Using Cloudscape in a Multi-User Environment”**
  Describes the different options for embedding Cloudscape in a server framework; explains the Cloudconnector option.

- **Chapter 2, “Installing Cloudscape in a Server Framework”**
  Describes the installation directory and Cloudscape deployment options.

- **Chapter 3, “Starting and Shutting Down the Cloudconnector Server”**
  Tells how to start, run, and shut down a Cloudscape server.

- **Chapter 4, “Accessing Cloudconnector from a Client”**
  Tells how to develop and deploy applications that access a Cloudscape server.

- **Chapter 5, “Working with Java Data Types in a Server”**
  Provides special deployment information for working with Java data types.

- **Chapter 6, “User Authentication Using Cloudconnector”**
  Explains Cloudconnector’s native user support and how to use it in conjunction with Cloudscape’s user support.

- **Chapter 7, “Configuring a Server Framework for Cloudscape”**
  Tells how to administer and configure Cloudscape in a server framework.

- **Chapter 8, “SSL Security in Cloudconnector”**
  Shows how to use Security Socket Layer security when using Cloudconnector.

- **Chapter 9, “Cloudscape Access Using RmiJdbc”**
  Tells how to use Cloudscape with RmiJdbc, a lightweight freeware server framework.

- **Chapter 10, “Embedding Servers”**
  Tells how to run a server framework in the same JVM as an embedding application to allow concurrent local and remote access.
Part Two (Cloudscape Administration Guide):

- *Chapter 11, “Checking Database Consistency”*
  Shows how to check the consistency of Cloudscape databases.

- *Chapter 12, “Backing Up and Restoring Databases”*
  Shows how to back up a database while it is online.

- *Chapter 13, “Logging on a Separate Device”*
  Shows how to put a database’s log on a separate device, which is helpful for the performance of large databases.

- *Chapter 14, “Getting Locking Information”*
  Shows how to get detailed information about locking status.
Using Cloudscape in a Multi-User Environment

This chapter contains the following sections:

- “Cloudscape in a Server Framework” on page 1-1
- “Multi-User Features Available in All Cloudscape Products” on page 1-3
- “Cloudconnector” on page 1-5
- “About this Book and the Cloudconnector Documentation” on page 1-9

Cloudscape in a Server Framework

In a sense, Cloudscape is always an embedded product. You can embed it in a single-user application or in a server framework—an application that allows multiple users to connect to Cloudscape simultaneously. When Cloudscape is embedded in a single-user application, the local JDBC driver calls the local Cloudscape. When Cloudscape 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.

Cloudscape provides the Cloudconnector option if you prefer a pre-packaged server framework specially configured to be used with Cloudscape. Cloudconnector also provides many other features useful in a multi-user environment. If you do not require the features included in Cloudconnector, you can embed the basic Cloudscape product in another server framework.
Using Cloudscape in a Multi-User Environment

Connectivity Configurations

There are many ways you can embed Cloudscape in a server framework:

- **Use Cloudconnector.**
  The easiest way to provide multi-user connectivity. Cloudconnector provides a complete server framework as well as many other features. Cloudconnector provides the functionality of WebLogic/JDBC, as well as Cloudscape customizations for its use.

- **Use RmiJdbc.**
  A simple freeware server framework, suitable for prototyping, included in the basic Cloudscape package.

- **Purchase another server framework.**
  You can use Cloudscape within many server frameworks, such as Novera and Apache.

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

The simple Cloudscape demo can be run using Cloudconnector, using RmiJdbc, or embedded. This demo can be found in

```
$CLOUDSCAPE_INSTALL/demo/programs/simple
```

See the Cloudscape Developer’s Guide for diagrams of standard client/server configurations. Also see the Cloudscape Web site (www.cloudscape.com) for white papers showing examples of server configurations that are possible with Cloudscape.

Differences Between Embedded and Server Configurations

There are very few differences between Cloudscape running embedded in a user application and embedded in a server framework. These minor differences arise because a server implements an intermediate JDBC protocol. The differences are as follows:
Multi-User Features Available in All Cloudscape Products

- Different server frameworks handle type conversions in different ways. This can affect a Cloudscape application that is expecting a certain data type.
- Date/time formats resulting from invoking the `toString` method on `java.sql.Date`, `java.sql.Time`, and `java.sql.Timestamp` may be different. (This depends on the server framework; it is not an issue for Cloudconnector.)

Apart from these differences, Cloudscape functions identically regardless of the way it is embedded.

Embedding Servers

Since Cloudscape is written in Java, you have great flexibility in configuring your deployment. For example, you can run Cloudscape, the JDBC server framework, and another application in the same JVM as a single process. For more information, see Chapter 10, “Embedding Servers”.

Using a Server Framework with Cloudsync

If you choose the Cloudsync option, you will probably run Cloudscape within a server framework. Cloudsync allows you to create and operate source databases. A source database—the central, master database with which multiple targets synchronize—typically runs as a server. Target databases often run embedded in single-user applications that connect intermittently to the source database. When run in a server framework, Cloudsync can synchronize over a network with clients, over the Internet with occasionally connected targets, or both. The server embedding Cloudscape can even itself be embedded in another application, for example as a servlet running in an application server. See the Cloudscape Synchronization Guide for more details about Cloudsync.

Multi-User Features Available in All Cloudscape Products

The basic Cloudscape product contains some features that are useful for developing multi-user applications. These features include:
Using Cloudscape in a Multi-User Environment

- “RmiJdbc” on page 1-4
- “Row-Level Locking” on page 1-4
- “Multiple Concurrency Levels” on page 1-4
- “Multi-Connection and Multi-Threading” on page 1-4
- “Administrative Tools” on page 1-5

RmiJdbc

Cloudscape includes RmiJdbc, a simple connectivity framework, as well as a few Cloudscape customizations of RmiJdbc. RmiJdbc is freeware suitable for prototyping simple server scenarios. RmiJdbc provides JDBC connectivity, but not the many additional features of Cloudconnector.

Row-Level Locking

To support multiple users, Cloudscape provides the option of row-level or table-level locking. 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. Cloudscape’s optimizer tunes lock choice for queries automatically, or it can be controlled by the application.

Multiple Concurrency Levels

Cloudscape supports both SERIALIZABLE and READ COMMITTED isolation levels. READ COMMITTED provides greater concurrency for multiple users, and SERIALIZABLE provides greater consistency. See the Cloudscape Developer’s Guide for more information.

Multi-Connection and Multi-Threading

Cloudscape allows multiple simultaneous connections to a database, even in embedded mode. Cloudscape 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 Cloudscape Developer’s Guide for more information.
**Administrative Tools**

Cloudscape provides some tools and features to assist database administrators. These tools are:

- Lock debugging VT1 classes and deadlock monitoring
- Consistency checker
- On-line backup
- The ability to put a database’s log on a separate device

These tools and features are discussed in part two of this book. See the chapters in that section for more information.

**Cloudconnector**

Cloudconnector provides a complete server framework that is customized for Cloudscape.

Cloudconnector consists of a JDBC server with compatible client software. This software (WebLogic Version 4.5) is provided by WebLogic and has been customized for use with Cloudscape.

**Features of Cloudconnector**

Cloudconnector provides most of the functionality of WebLogic server (formerly known as Tengah/JDBC and then WebLogic). WebLogic, like Cloudscape, is a Java application. Therefore, Cloudscape with Cloudconnector provides all the flexibility of Java: You can develop and deploy database applications as standard client or server applications, browser applets, or servlets (HTML clients). Cloudconnector enables any Java application to access and update remote enterprise databases with security and transactional integrity.

Cloudconnector provides HTTP-based administration through servlets. It also provides easy-to-use, HTTP-based administrative tools.

The following sections list some of the features of Cloudconnector.
Development
Cloudconnector makes it easy to develop and deploy Java database and Web applications. Cloudconnector applications are simple and portable. Cloudconnector insulates programmers from disparities between Java platforms and databases, and handles lower-level programming complexities such as sockets and threads.

Integrated Development Environments (IDEs)
Cloudconnector extends leading Java IDEs to support the development and debugging of multitier database applications. IDE GUI builders, HTML generation facilities, and compatible data-aware controls can all be used with Cloudconnector.

Web Development in Java
Cloudconnector supports standard HTTP servlets for invoking Java business logic in HTTP and provides Java HTML generation for incorporating dynamic content into Web pages. For example, Cloudconnector can automatically generate HTML tables from query results. You can embed Java code in an HTML page. Servlet session management gives Cloudconnector the ability to manage client sessions across many browser-server interactions as scalable, reliable conversations.

Workspaces
Cloudconnector includes WebLogic’s support for a set of hierarchical workspaces that maintain client state. Workspaces can also be used to store private objects or accommodate any level of sharing.

Integration
Using the Java-standard Java Naming and Directory Interface (JNDI), Cloudconnector can access existing directory services such as Novell NDS, Sun NIS+, Microsoft Active Directory, or others supporting the Internet-standard Lightweight Directory Access Protocol (LDAP).

Deployment
Cloudconnector protects the execution of Java applications by ensuring scalability, security, and transactional integrity. Since both Cloudscape and WebLogic are
written in Java, no client-side library installation is required. Instead, client libraries are dynamically loaded directly from Cloudconnector (or any Web server).

**Transactions**

Cloudconnector support for transaction processing safeguards business-critical applications and corporate data so that integrity cannot be compromised.

**Firewall Support**

All of Cloudconnector’s services are securely available through firewalls via HTTP tunneling.

**Data Caching**

With Cloudconnector, JDBC result sets can be cached in server-side workspaces so that appropriately authorized clients do not need to repeat common queries. Sharing query results reduces the load on both the database and the network.

**Shared Database Connections**

Cloudconnector scales to support large numbers of simultaneous users by maintaining configurable pools of database connections that can be shared among many clients. Connection pools accommodate intermittent users without the performance overhead of opening and closing database connections. See “User Authentication Using Cloudconnector” on page 6-1 for more information on connection pools.

**Server Scalability**

Cloudconnector scales to support many clients by carefully managing threads and connections. A single client/server connection is shared across all bidirectional communications.

**Standard Internet Protocols**

Web browsers can access Cloudconnector via normal HTTP requests. Forwarding capabilities, such as HTTP proxying, enable dispatching to servers other than the original Web server. For higher performance, HTTP connections are maintained across requests.
Using Cloudscape in a Multi-User Environment

Management
Cloudconnector provides centralized management for a potentially large distributed configuration of connected clients and servlets through a single cohesive view of the overall system.

Graphical Management Console
Cloudconnector offers a comprehensive pure-Java console for remotely monitoring and updating the state of Cloudconnector applications and servers. Multiple clients and servers can be securely and easily managed from a single remote console.

Integrated Logging
Cloudconnector automatically logs diagnostic and security audit information and provides interfaces for applications to log their own exception conditions. Optionally, JDBC use can be logged in database-specific logs. Logs can be viewed remotely from a Web browser or from the WebLogic Console.

The WebLogic JDBC Services Integration
Cloudconnector has customized WebLogic JDBC in the following ways:

- **CloudscapeServer class**

- **License file**
  Cloudconnector comes with a special license file (`/license/WebLogicLicense.xml`). A standard (non-Cloudconnector) WebLogic server requires you to request keys from WebLogic. When using Cloudconnector, you do not need keys for JDBC features.

- **Customized server directory**
  In a generic WebLogic installation, the server runs in a directory called `myserver`. In Cloudconnector’s integrated installation, the server runs in a directory called `CloudscapeServer`. Cloudscape provides this directory for you in `/frameworks/cloudconnect`. 
• **Customized properties**
  Cloudconnector provides a `weblogic.properties` file that is customized for use with Cloudscape. Cloudscape provides this directory for you in `/frameworks/cloudconnect`.

• **Customized JDBC Driver**
  Cloudscape provides a specially integrated client JDBC driver (`COM.cloudscape.core.WebLogicDriver`) that accepts a simpler URL for Cloudconnector access.

• **Customized startup and shutdown utilities**
  Cloudscape provides utilities for starting and shutting down Cloudconnector.

---

### About this Book and the Cloudconnector Documentation

This guide assumes you are familiar with Cloudscape features, administration, and tuning. Before reading this guide, you should first learn the basic Cloudscape product by reading the *Cloudscape Developer’s Guide*. The *Cloudscape Developer’s Guide* also includes a discussion of deployment options and how embedding Cloudscape into a server fits into these options. Also, since multi-user environments typically have performance and tuning issues, you should read *Tuning Cloudscape*.

If you have installed Cloudconnector, your installation directory includes some of the WebLogic documentation. These documents are:

- *The WebLogic Developer’s Guide*
- *The WebLogic Administrators and Deployment Guide*

These books are provided in the PDF format under `/frameworks/cloudconnect/doc`. Use the WebLogic documentation included in this distribution only if you need additional information on the following issues:

- **performance and tuning**
  See *Tuning the Server* in the *WebLogic Administrators and Deployment Guide*. 
Using Cloudscape in a Multi-User Environment

- security
  For using ACLs (e.g., to define groups, credentials, and certificates), see
  Using Tengah ACLs in the The WebLogic Developer’s Guide.
  For using SSLs (e.g., in servlets), see Using SSL in the The WebLogic
  Developer’s Guide.

- advanced administration
  For example, to disable HTTP services or check memory usage, see the The
  WebLogic Administrators and Deployment Guide

You should also check the accompanying WebLogic release notes for information
on any of these topics. You will find a link to the Release Notes in the
Cloudconnector Release Notes.

Read this guide (the Cloudscape Server and Administration Guide) first. Consult
the WebLogic guides only for the topics listed above, or when this guide
specifically refers you to a WebLogic guide about a given topic. There are two
reasons for this:

- This guide describes Cloudscape customizations not described in the BEA
documentation. For example, WebLogic JDBC’s documentation refers to
  a TengahClient, which uses a more complicated URL than Cloudconnector
does.

- The WebLogic documents may refer to features not included in
  Cloudconnector. You will need additional licenses from WebLogic to use
  these features.
Installation Directory

The installation program installs the Cloudscape software in a directory of your choice, called in this document the *cloudscape base directory*. (It is recommended that you name this directory *cloudscape*.) Figure 2-1 shows the layout of this directory. The server framework is installed in the *frameworks* directory under the *cloudscape base directory*. 
The demonstration in `demo/programs/simple` shows how to connect a client with a server.

For more information about the installation directory structure, see the *Cloudscape Developer’s Guide*.

If you install Cloudconnector, the directory structure shown in Figure 2-2 on page 2-3 appears under `frameworks/cloudconnect`.
JDK

CloudConnector currently works both JDK 1.1.x and JDK 1.2.x. See the Release Notes for more details about JVMs.

For information on JDK and RmiJdbc, see Chapter 9, “Cloudscape Access Using RmiJdbc”.

Cloudscape has been certified 100% Pure Java. See our support Web site for a list of JVMs with which it has been used. (Go to www.cloudscape.com/Engineering/vm.html for info on currently supported JVMs.)
Installing Cloudscape in a Server Framework

Libraries

The Cloudscape installation programs automatically install the Cloudscape (and optionally, Cloudconnector) libraries. If you want to deploy Cloudscape through a means other than the installation program, this section describes the Cloudscape product’s directory structure.

Cloudscape libraries are installed in the /lib subdirectory of the cloudscape base directory. /lib includes the following files:

- cloudscape.jar
- client.jar
- tools.jar
- cloudsync.jar (only if you have installed the Cloudscape synchronization option)

If you have installed the Cloudconnector option, frameworks/cloudconnect/lib includes weblogicaux.jar. Cloudconnector also uses the files in frameworks/cloudconnect/classes and frameworks/cloudconnect/license.

The server framework-specific libraries are installed in the frameworks subdirectory of the cloudscape base directory. The frameworks subdirectory always includes an RmiJdbc subdirectory. If you install the Cloudconnector option, frameworks also includes a cloudconnect subdirectory. If you install another server framework, you may want to install it in a directory under frameworks.

When to Use client.jar

The client.jar file is provided with all Cloudscape products.

When an application accesses a Cloudscape database by using Cloudconnector or the RmiJdbc server, the client application needs client.jar on its class path along with the Cloudconnector or RmiJdbc server client classes.

For other server frameworks, client applications may not need client.jar. Typically, JDBC client applications need client.jar and non-JDBC client applications do not. In general, server frameworks that completely hide Cloudscape from their client applications can safely omit client.jar from their client application libraries.

Specifically, a client application needs client.jar in its class path if either of the following is true:

- The application can execute arbitrary SQL statements and get back data or exceptions from the database.
• The application can make requests that return any COM.cloudscape.* objects, such as COM.cloudscape.types.TypeDescriptor. COM.cloudscape.* types are primarily used in the Cloudscape system catalogs and built-in system functions such as RuntimeStatistics.

**Setting PATH for a Server**

Cloudscape provides a script called startCS to start both the RmiJdbc server and Cloudconnector. Add the /bin subdirectory from the appropriate frameworks directory to your path in order to use the start-up and shutdown utilities.

**Class Path for a Server**

Set the class path for a Cloudscape server as a runtime option in a startup script. Instructions on working with startup scripts are presented in Chapter 3, “Starting and Shutting Down the Cloudconnector Server”.

For JDK 1.1, the class path must include the path to the classes.zip file of the JDK in addition to the libraries shown for the server machine hosting the Cloudscape server in “Libraries” on page 2-4. (For JDK 1.2, the file is called rt.jar and is not needed in the class path.)

**Library Path for Cloudconnector on UNIX**

On UNIX platforms, add /framework/cloudconnect/lib to the library path of the server.

• On Solaris for ksh/sh:

```
LD_LIBRARY_PATH=$CLOUDSCAPE_INSTALL/framework/cloudconnect/lib:$LD_LIBRARY_PATH
export LD_LIBRARY_PATH
```
Installing Cloudscape in a Server Framework

- **On Solaris for csh:**
  
  ```
  setenv LD_LIBRARY_PATH $CLOUDSCAPE_INSTALL/framework/cloudconnect/lib:$(LD_LIBRARY_PATH)
  ```

- **On HP:**

  ```
  setenv SHLIB_PATH $CLOUDSCAPE_INSTALL/framework/cloudconnect/lib:$(SHLIB_PATH)
  ```

- **On AIX:**

  ```
  setenv LIBPATH $CLOUDSCAPE_INSTALL/framework/cloudconnect/lib:$(LIBPATH)
  ```
Cloudscape provides scripts for starting and stopping the Cloudconnector server. Use of these scripts is optional. See “Starting Cloudconnector on the Server” on page 3-2 for information on how these scripts are used, as well as other configuration issues for Cloudconnector.

- “Overview” on page 3-1
- “Starting Cloudconnector on the Server” on page 3-2
- “Shutting Down Cloudconnector” on page 3-8
- “The Cloudconnector Scripts” on page 3-11

Overview

The Cloudscape server runs “inside” the Cloudconnector framework, which consists of WebLogic’s services integrated with Cloudscape. The name of the class that starts the server is \texttt{COM.cloudscape.core.CloudscapeServer}. The \texttt{CloudscapeServer} class starts both parts of Cloudconnector:

- the JDBC server framework
- the Cloudscape engine

The command line in the startup script (\texttt{startCS}) uses this class to start the server. See “Starting Cloudconnector on the Server” on page 3-2 for details on the startup script. Shutting down \texttt{CloudscapeServer} stops both parts of Cloudconnector.
Starting and Shutting Down the Cloudconnector Server

Prior to starting the server, you typically will set certain WebLogic and Cloudscape properties. This chapter includes information about setting these properties.

Information about the WebLogic properties set by a client application can be found in Chapter 4, “Accessing Cloudconnector from a Client”.

This chapter includes these sections:

- “Starting Cloudconnector on the Server” on page 3-2
- “Shutting Down Cloudconnector” on page 3-8
- “The Cloudconnector Scripts” on page 3-11

Starting Cloudconnector on the Server

Start Cloudconnector by running the included scripts and setting some basic properties. You need to modify these scripts for your environment.

To successfully complete the steps in this chapter, you will need to know the following:

- The location of your Cloudscape and Cloudconnector installation
- The location of your `weblogic.properties` file and `CloudscapeServer` directory
- The location of your databases
- Any customizations you may need to make to your `weblogic.properties` file, including modifying the default password or the listening port number

This chapter is divided into two major sections:

- “Startup Tasks” on page 3-2
- “Verifying Startup” on page 3-8

Startup Tasks

- “Step One: Open a Command Window for Starting the Server” on page 3-3
- “Step Two: Include Cloudconnector in the PATH Variable” on page 3-3
- “Step Three: Modify the class path Script (setServerCloudscapeCP)” on page 3-3
- “Step Four: Modify the Startup Script (startCS)” on page 3-4
Starting Cloudconnector on the Server

• “Step Five: Optionally Modify weblogic.properties” on page 3-6
• “Step Six: Run Both Scripts” on page 3-7

Step One: Open a Command Window for Starting the Server

Open a command window. You will set up the server environment in this window using commands in the provided scripts.

Step Two: Include Cloudconnector in the PATH Variable

For the scripts to work properly, the PATH variable must include the Cloudconnector installation directory: $CLOUDSCAPE_INSTALL/frameworks/cloudconnect/bin (where $CLOUDSCAPE_INSTALL is the directory in which you installed Cloudscape). This directory should appear in the path before any other Cloudscape directories.

Step Three: Modify the class path Script (setServerCloudscapeCP)

In this step, modify the script that sets the class path on the server. While you can set the class path directly from the command line, you may find it easier to use the included scripts. They are located in $CLOUDSCAPE_INSTALL/frameworks/cloudconnect/bin. Modify the appropriate script for your platform:

- setServerCloudscapeCP.bat (Windows)
- setServerCloudscapeCP.ksh (UNIX)

NOTE: If your platform does not support scripts, use these files as examples.

The setServerCloudscapeCP script sets the CLOUDSCAPE_INSTALL variable to point to the Cloudscape installation directory, and then modifies the CLASSPATH variable to include that directory.

1. In setServerCloudscapeCP.bat (or ksh), modify the value of CLOUDSCAPE_INSTALL to point to the Cloudscape installation directory on your machine.

2. Typically, you do not need to modify the second line in this file, which sets the class path. However, to add application-specific libraries (i.e., for Java data types), modify the CLASSPATH line to include any .zip, .jar, or class files which contain Java types referred to in SQL-J statements, work units, views, or tables in your database.

At a minimum, the Cloudconnector class path must include:
Starting and Shutting Down the Cloudconnector Server

- frameworks/cloudconnect/classes
- frameworks/cloudconnect/lib/weblogicaux.jar
- frameworks/cloudconnect/license
- lib/cloudscape.jar or lib/cloudsync.jar

**NEW:** The requirement for adding frameworks/cloudconnect/license to the class path is new in Cloudconnector Version 3.0.

**Step Four: Modify the Startup Script (startCS)**

In this step, modify the script that you will use to start up the server. While you can start up the server directly from the command line, you may find it easier to use the provided scripts.

- startCS.bat (Windows)
- startCS.ksh (UNIX)

They are located in $CLOUDSCAPE_INSTALL/frameworks/cloudconnect/bin. To view a printout of startCS, see “The Cloudconnector Scripts” on page 3-11.

1. Modify the startCS script to reflect the correct settings for two properties in the java command line:
   - weblogic.system.home
   - cloudscape.system.home

If you do not set these system properties, Cloudscape assumes you intend the current directory for both (see “Default File Configuration” on page 3-5).

- **weblogic.system.home**
  Set this property to the directory holding the weblogic.properties file and the Cloudscape-Cloudconnector server directory. By default, the properties file and CloudServer directory are located in cloudconnect/frameworks, beneath the Cloudscape installation directory. The directory to which weblogic.system.home points must exist at the time the server starts up, because the server looks in it for the weblogic.properties file.

- **cloudscape.system.home**
  Set this property to the directory containing your Cloudscape databases and the optional cloudscape.properties file. This directory can be anywhere on your system, including the directory to which the weblogic.system.home property points.
Starting Cloudconnector on the Server

**NOTE:** At the time the server starts, the directory to which `weblogic.system.home` points must contain a directory called `CloudscapeServer` and the `weblogic.properties` file.

To specify these properties, modify the JVM -D command-line options in the `startCS` script. For example:

- `Dweblogic.system.home=c:\srvStartDir`
- `Dcloudscape.system.home=c:\myDBDir`

The following figure illustrates how you would modify these system properties.

![Figure 3-1](image)

**Figure 3-1** The `cloudscape.system.home` and `weblogic.system.home` directories

When you have finished altering the `startCS` script, it should look similar to the following:

```
java -ms16m -mx32m -noasyncgc -
Dweblogic.system.home=c:\myweblogichome
-Dcloudscape.system.home=d:\mydatabases
COM.cloudscape.core.CloudscapeServer
```

**Default File Configuration**

If you run the `startCS` script without modifying it, the directory from which you run it effectively becomes the location for the `cloudscape.system.home` and the `weblogic.system.home` properties. It must therefore contain the following files and directories, as illustrated in Figure 3-2 on page 3-6:

- the Cloudscape database(s) that you want to access
Starting and Shutting Down the Cloudconnector Server

- the `weblogic.properties` file
- the directory for Cloudconnector
  Typically, this directory is named `CloudscapeServer`. You can specify a different name by resetting the `weblogic.system.name` property (see “Setting Cloudconnector Properties” on page 7-2).
- the `cloudscape.properties` file (optional)

If `startCS` is run from the `frameworks/cloudconnect` subdirectory, the `weblogic.properties` file provided with Cloudscape is used, no `cloudscape.properties` file is used, and the current directory becomes the system directory—that is, it holds Cloudscape database directories and the `CloudscapeServer` directory.

Figure 3-2  The default layout of directories and properties files.

**Step Five: Optionally Modify `weblogic.properties`**

You may need to modify the `weblogic.properties` file that will be used by Cloudconnector upon startup.

Two common customizations are:

- Setting a Password
- Modifying the Port Number

There are numerous other customizations you can make, described in “Setting Cloudconnector Properties” on page 7-2.

**NOTE:** If you will be using the `weblogic.httpd.documentRoot` property, you will need to uncomment and customize the default entry for the directory path
Starting Cloudconnector on the Server

in the `weblogic.properties` file. This property affects the way the administrative servlet displays HTML pages.

**Setting a Password**

A password is required for Cloudconnector. You can locate and use the default password provided in the `weblogic.password.system` property of the `weblogic.properties` file, or you can modify this property to reflect a different password.

**Modifying the Port Number**

By default, Cloudconnector listens for client requests on port 7001. You can change this number by changing the value of `weblogic.system.listenPort` in the `weblogic.properties` file.

**NOTE:** Do not use the same port number for both client requests and incoming synchronization refresh messages (determined by the `cloudscape.DataComm.port` variable).

**Step Six: Run Both Scripts**

After you have modified both scripts to reflect your environment, run them in the following order to start up the Cloudconnector server:

1. Run the `setServerCloudscapeCP` script in the command window to set the class path:

   ```
c:\> setServerCloudscapeCP
   ```

2. Run the `startCS` script to start Cloudconnector:

   ```
c:\> startCS
   ```

Typically, you should see messages showing that Cloudconnector is starting. Once Cloudconnector starts, you can access it from a client application or applet, using the WebLogic client tools or simply the client JDBC driver and database connection URL.

Following is a sample excerpt of messages you might see:

```
Fri Sep 10 13:28:22 PDT 1999:Opened log file .\CloudscapeServer\weblogic.log
Fri Sep 10 13:28:22 PDT 1999:<I> <WebLogicServer> Read global properties .\weblogic.properties
```
Verifying Startup

You can use the WebLogic utility class `utils.t3dbping` to verify that Cloudconnector is running and that database services are available through it. For example, if you have a database `pingDB` available at the `cloudscape.system.home` location for Cloudconnector running on your machine on port 7001, the following should succeed:

```
java utils.t3dbping t3://hostname:portnumber "" "" "" 
COM.cloudscape.core.JDBCDriver jdbc:cloudscape:pingDB
```

Enter this command in a single line. The "" arguments are needed to satisfy the `utils.t3dbping` command line.

To test only the availability of the WebLogic services:

```
java weblogic.T3Admin t3://hostname:portnumber Connect 1
```

Shutting Down Cloudconnector

The Cloudconnector installation includes two shutdown scripts, located in `$CLOUDSCAPE_INSTALL/frameworks/cloudconnect/bin`. (Make sure your PATH includes this directory before you run the shutdown script.)

- `stopCS.bat` (Windows)
- `stopCS.ksh` (UNIX)
To shut down Cloudconnector, run the appropriate script from any directory on the client. Property shutting down Cloudconnector causes a checkpoint command to be issued, and minimizes the recovery procedures required upon the next startup (thereby reducing startup time). If you simply interrupt the running server with a ^C command, recovery will be run on the databases when they are rebooted.

For example, on Windows:

```
C:\> stopCS
```

In the client Window, a message similar to the following displays:

```
T3://127.0.0.1:7001 successfully shutdown
Shutdown sequence initiated
```

An example of the output you will see at the server console:

```
Fri Sep 10 13:34:54 PDT 1999:<I> <RJVM> Closing connection to localhost/127.0.0.1 -5754524976542505027
Fri Sep 10 13:34:54 PDT 1999:<I> <CliCon#|CloudscapeServer|2.936995306288> Connection to client for [CliCon: #|CloudscapeServer|2.936995306288] has been unexpectedly lost because t3 socket to - 5754524976542505027C10.10.11.67 failed and could not be reconnected.
Initiating hard disconnect.
Fri Sep 10 13:35:06 PDT 1999:<I> <WebLogicServer> Invoking garbage collection
Fri Sep 10 13:35:06 PDT 1999:<I> <GC> GC: Before free/total=15071048/16777208 (89%)
Fri Sep 10 13:35:06 PDT 1999:<I> <GC> GC: After free/total=15685880/16777208 (93%)
Fri Sep 10 13:35:06 PDT 1999:<I> <WebLogicServer> Invoking garbage collection
Fri Sep 10 13:35:06 PDT 1999:<I> <GC> GC: Before free/total=15674504/16777208 (93%)
Fri Sep 10 13:35:06 PDT 1999:<I> <GC> GC: After free/total=15685464/16777208 (93%)
```

If Cloudscape has already shut down (for example, because an application used the `shutdown=true` form of the database connection URL), a warning message is issued:

```
The WebLogic Server at T3://127.0.0.1:7001 is not responding
```

Cloudscape Version 3.0
because of [java.io.IOException: Bootstrap unable to get a t3 connection to localhost/127.0.0.1].
Are you sure it’s running?
Ignoring request to disconnect client that is already disconnected.
java.io.IOException: Bootstrap unable to get a t3 connection to localhost/127.0.0.1.

Shutting Down Cloudconnector when User Authentication Is Enabled

When user authentication has been turned on in Cloudscape with the cloudscape.connection.requireAuthentication property, user credentials are required to shut down the server. To provide user credentials for the shutdown, edit the weblogic.properties file to provide valid arguments, as follows:

weblogic.system.shutdownArgs.CloudscapeShutdown=user=name,password=pwd
Using `;shutdown=true` with Concurrent Embedded and Remote Access

If you attempt to shut down Cloudconnector via the `jdbc:cloudscape:weblogic:` URL and also have an embedded system booted in the client JVM, the shutdown will fail with the following message:

"Database name must not be specified"

To avoid this error, use a longer-form WebLogic database connection URL:

```
jdbc:weblogic:t3?weblogic.t3.driverURL=t3://hostname:portnumber
&weblogic.t3.serverClassName=COM.cloudscape.core.JDBCDriver
&weblogic.t3.serverURL=jdbc:cloudscape;;shutdown=true
```

Specify the URL as a single string with no white space, or represent line breaks as the parameter value in a `DriverManager.getConnection` request.

**The Cloudconnector Scripts**

The following table shows the pre-modified contents of the supplied Cloudconnector scripts for starting and stopping the Cloudscape server:

<table>
<thead>
<tr>
<th>Script</th>
<th>Purpose</th>
<th>Pre-modified Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>startCS</code></td>
<td>Starts Cloudscape as a server in a Cloudconnector framework</td>
<td><code>java -ms16m -mx32m -noasyncgc -Dweblogic.system.home=.COM.cloudscape.core.CloudscapeServer</code></td>
</tr>
<tr>
<td><code>stopCS</code></td>
<td>Stops the Cloudscape server</td>
<td><code>java weblogic.T3Admin t3://localhost:7001 SHUTDOWN systempaceesalute</code></td>
</tr>
<tr>
<td><code>setServerCloudscapeCP</code></td>
<td>Sets the class path on the server</td>
<td><code>&lt;see file in cloudscape/frameworks/cloudconnect/bin&gt;</code></td>
</tr>
<tr>
<td><code>setClientCloudscapeCP</code></td>
<td>Sets the class path on the client</td>
<td><code>&lt;see file in cloudscape/frameworks/cloudconnect/bin&gt;</code></td>
</tr>
</tbody>
</table>

For instructions on modifying these scripts, see “Startup Tasks” on page 3-2. If you want to run these scripts without modifying them, you must run them from `cloudscapedir/frameworks/cloudconnect/bin` (where `cloudscapedir` is the...
installation directory for Cloudscape and Cloudconnector). If run uncustomized, Cloudscape assumes default values for certain properties. For more information, see “Default File Configuration” on page 3-5.
4 Accessing Cloudconnector from a Client

Once Cloudconnector has been started on a server, you can access it from a client. For information about starting the Cloudconnector server, see “Starting Cloudconnector on the Server” on page 3-2.

This chapter includes the following topics:

- “Preparing Clients” on page 4-1
- “Using Cloudconnector Clients” on page 4-2
  - Specifying the Client JDBC Driver
  - Specifying the Database Connection URL
- “Using ij and Cloudview with Cloudconnector” on page 4-7

Preparing Clients

Before attempting to connect to the Cloudconnector server, verify your JDK version, your classpath, and WebLogic version.

JDK Version

Cloudconnector client applications can use JDK version 1.1 or 1.2.
Class Path

Prior to connecting to Cloudconnector, verify that the client class path has been set to include the following files and directories (paths are shown as relative to the Cloudscape installation directory):

- /lib/client.jar
- /frameworks/cloudconnect/classes/
- /frameworks/cloudconnect/lib/weblogicaux.jar
- /frameworks/cloudconnect/license/
- /lib/tools.jar
  (if using the Cloudscape tools ij or Cloudview on the client.)

NOTE: Cloudscape provides a script named SetClientCloudscapeCP (.bat and .ksh) which you can modify and run, or use as a guideline for setting your class path. The SetClientCloudscapeCP file is located in the frameworks/cloudconnect/bin subdirectory of the Cloudconnector installation.

Cloudconnector WebLogic Version

The Cloudconnector client(s) and server must both use the same version of the WebLogic software. For example, BEA WebLogic 3.1.5 clients cannot access the 4.5 server. Version information is displayed at startup.

See the Cloudconnector release notes for information on the version of the WebLogic software provided with Cloudconnector Version 3.0

Using Cloudconnector Clients

Cloudconnector supplies a tailored JDBC driver and database connection URL for use by clients. To access the Cloudconnector server, the client must specify the JDBC Driver and the database connection URL.

- “Specifying the Client JDBC Driver” on page 4-3
- “Specifying the Database Connection URL” on page 4-3
- “Setting Session-Level WebLogic Properties on the URL” on page 4-5

NOTE: The Cloudscape Developer’s Guide documents the database connection URL for using Cloudscape as an embedded database.
Specifying the Client JDBC Driver

Depending on whether you are connecting programmatically or from a Cloudscape tool, use one of the following to specify the JDBC driver:

- a call to `Class.forName.newInstance()`
- the `drivers` system property

Programmatically

In client code, specify the client JDBC driver for connecting to a Cloudscape server:

```java
Class.forName("COM.cloudscape.core.WebLogicDriver").newInstance();
```

From ij or Cloudview

If using ij or Cloudview, specify the `drivers` system property when starting up the tool. For example:

```
java -Djdbc.drivers=COM.cloudscape.core.WebLogicDriver
COM.cloudscape.tools.cvview
```

or

```
java -Djdbc.drivers=COM.cloudscape.core.WebLogicDriver
COM.cloudscape.tools.ij
```

Specifying the Database Connection URL

After you have specified the JDBC driver, you connect to the database using a database connection URL.

A Cloudscape client to Cloudconnector uses the following database connection URL:

```
jdbc:cloudscape:weblogic[-ssl]://[hostname:portnum/]
[subsubprotocol:][databaseName];[cloudscapeAttributes]
[&webLogicAttributes]
```

(All of this is one string; the new lines above just make the URL readable.)

- `ssl`
  Use `-ssl` after the `weblogic` parameter if you are using Cloudconnector with the Secure Socket Layer (SSL) turned on. For more information, see “SSL Security in Cloudconnector” on page 8-1.
Accessing Cloudconnector from a Client

- //hostname:portnum/
  If the host and client are running on different machines, replace hostname with the name of the host running Cloudconnector. If not specified, hostname defaults to localhost. For single-system configurations, you can explicitly specify localhost, or you can leave this blank.

  Replace portnum with the port number for the host hostname. The port number may be specified in the weblogic.properties file (as the weblogic.system.listenPort property) or on the Cloudconnector JVM command line. If not specified, portnum defaults to 7001 (or 7002 if SSL is turned on).

  If you do not specify values for either hostname or portnum, the default value is //localhost:7001/ (or //localhost:7002/ for weblogic-ssl).

- subsubprotocol:
  subsubprotocol is typically not specified. It indicates whether Cloudscape should look for a database in a directory, class path, or jar file. It is used only in rare instances, typically for read-only databases. The value of subsubprotocol can be one of the following:

  - directory
  - classpath
    Databases are treated as read-only databases, and all databaseNames must begin with at least a slash, because you specify them “relative” to the class path directory or archive.

  - jar
    Databases are treated as read-only databases.

  If jar is specified, an additional element is required before the databaseName:

    (pathToArchive)

  where pathToArchive is the full path (including filename) of the jar or zip file that holds the database.

- [databaseName];[cloudscapeAttributes]
  Specify the database name and any database connection URL attributes as specified in Cloudscape Developer’s Guide. For example, to create and connect to a new database called newDB, include the following in the database connection URL:

    newDB;create=true
Using Cloudconnector Clients

• **webLogicAttributes**
  Optionally, use this argument to set WebLogic properties for Cloudconnector such as prefetching and caching. (Prefetching always works with caching.) Properties set on the database connection URL are good for the current session only (as opposed to properties set in the `weblogic.properties` file on the server). (See “Setting Session-Level WebLogic Properties on the URL” on page 4-5.)

For example, to turn off prefetching and caching, specify:

```text
&weblogic.t3.cacheRows=0
```

Separate each property with an ampersand (&).

**Example URLs**

Here are some example database connection URLs for Cloudscape clients:

• `jdbc:cloudscape:weblogic:newDB;create=true;autocommit=false &weblogic.t3.cacheRows=0`
  Connects to Cloudconnector installed on the local machine at the default port, creates and connects to a database called newDB, and turns off autocommit (a Cloudscape property) as well as prefetching and caching (WebLogic properties).

• `jdbc:cloudscape:weblogic://Jeeves:5001/accounting/accounts`
  Connects to Cloudconnector running on host Jeeves and listening on port 5001, and connects to an existing accounts database in the accounting directory.

• `jdbc:cloudscape:weblogic:newDB;user=fred;password=secret`
  Specifies user credentials for Fred.

**NOTE:** Typically, client applications should not use the `;shutdown=true` form of the database connection URL in client/server environments. The system administrator should use the provided utility to shut down Cloudconnector. See “Shutting Down Cloudconnector” on page 3-8.

**Setting Session-Level WebLogic Properties on the URL**

WebLogic properties can be set durably for a system using the `weblogic.properties` file. The `weblogic.properties` file is discussed in the “Setting Cloudconnector Properties” on page 7-2 section of chapter 7. This section discusses temporary,
Accessing CloudConnector from a Client

session-level WebLogic properties, which are specified by including them in the client database connection URL.

Setting WebLogic properties on a connection URL makes them valid only for the duration of the current connection.

The following sections describe two properties you might want to set for a client session:

- “Turning off Prefetching and Caching” on page 4-6
- “Turning off the Parameter Cache” on page 4-6

Turning off Prefetching and Caching

To improve performance, CloudConnector provides server-side prefetching and client-side caching of rows. However, prefetching and caching can cause the following behavior:

- They cause slightly different behavior in some result set manipulation situations. For example, with prefetching and caching turned on, getting a result set fails if there are any errors (such as division by zero) when the system prefetches and caches the first set of rows.
- They prevent you from using updatable cursors.
  
  Updatable cursors are described in the Cloudscape Reference Manual.

If you need more control over result set behavior or want to use the positioned update and delete statements (cursors), set the following property value, which disables both prefetching and caching:

```
weblogic.t3.cacheRows=0
```

Turning off the Parameter Cache

If you issue PreparedStatement clearParameters requests and need to ensure that statements are not executed until all parameters are reset, set the following property value:

```
weblogic.t3.enableSetParamCache=false
```
Using ij and Cloudview with Cloudconnector

The Cloudscape tools ij and Cloudview work in both embedded mode and client/server mode. To use Cloudconnector from these tools, specify the client driver and client database connection URL in the form required by the tool.

For example, in Cloudview, specify driver and URL information on the Cloudview System properties tab. In ij, you can specify the driver and URL in the java command line that invokes ij, or in the connection statement once ij is started.

The book Cloudscape Tools and Utilities Guide provides numerous examples of how to do this.
5 Working with Java Data Types in a Server

For you to use Java data types in a server, they must be deployed on the server. This chapter discusses:

- “Deploying Java Classes” on page 5-1
- “Using the Cloudscape Types on a Client” on page 5-2
- “Working with Non-Serializable Types” on page 5-2

Deploying Java Classes

Using Java types in a database means one or more of the following:

- storing Java objects in the database
- referring to a class in SQL-J statements or views
- referring to a class in work units or database-side methods

For you to use Java types in a database, the .class files for those types must be available to both the server and the client. This means the .class file must be installed on both machines and included in the class paths for both the client and the server. If the client and the server are on different machines, copy the .class, .zip, or .jar files onto both machines. The classes must be exactly the same on both machines. For the client, use a class path to the class files on the client machine, and for the server, use a class path to the class files on the server machine.

**NOTE:** For the server, the classes can be stored in the database itself. See the Cloudscape Tools and Utilities Guide for details.

For more information on Java data types, see the Cloudscape Developer’s Guide.
NOTE: Cloudscape synchronization provides a way to distribute Java classes. If you have installed Cloudsync, see the Cloudscape Synchronization Guide for details.

Using the Cloudscape Types on a Client

Cloudscape types, which are Java types returned by the system or stored in system catalogs, are found in /lib/client.jar, and so are available on both the client and the server.

Working with Non-Serializable Types

For you to instantiate a class within an SQL-J statement, the class must be serializable. In an embedded environment, this requirement is not enforced. However, when a query such as a VALUES statement “returns” an object that is not serializable, in the client/server environment you will get a ResultNotSerializable exception because the server framework uses serialization to return Java objects to the client.

Frameworks that embed Cloudscape more deeply, such as Web servers that provide client/server access via servlets, do not return Java objects to clients.
6 User Authentication Using Cloudconnector

This chapter is for Cloudconnector users only.

You can support users directly in Cloudscape (see the Cloudscape Developer’s Guide for details), or you can use Cloudconnector’s native (BEA WebLogic) user support. This chapter explains how to set up and administer users in BEA WebLogic’s native framework.

Cloudconnector allows you to set up a connection pool. Creating a pool of JDBC connections gives a Cloudscape client fast and easy access to connections that are already open. The connection pool avoids the overhead of opening a new connection for each Cloudscape user.

For additional information on how to set up a JDBC connection pool and define users in the BEA WebLogic server, please consult the BEA WebLogic Tengah Administrators Guide in frameworks/cloudconnect/doc.

This chapter contains the following sections:

- “Why Use Connection Pools?” on page 6-2
- “Creating a Connection Pool” on page 6-2
- “Setting Up Access Control Lists” on page 6-3
- “Using a Connection from a Connection Pool” on page 6-3
- “Refreshing or Resetting a Connection Pool” on page 6-4
Why Use Connection Pools?

Since you can create users directly in Cloudscape, it may not be clear why you would want to create users in the BEA WebLogic server as well.

One reason for doing this is that for a Web-based application, you may prefer not to create a new user in Cloudscape for each user connecting over the Web. It is more efficient to create a small, fixed number of users in Cloudscape and to allow the BEA WebLogic server to maintain a pool of users that connects to the users you have created in Cloudscape. For example, Cloudscape may define five sales users and six supplier users, and the application at the server can classify users connecting from the Web, assign each of them a connection from the pool, and assign the connection to the appropriate Cloudscape user.

Creating a Connection Pool

To create a connection pool, edit the `weblogic.properties` file. The default entries for a Cloudscape connection pool definition are in the `weblogic.properties` file shipped with Cloudconnector.

The following example sets up a connection pool called `sales` to the `myDB` database:

```properties
weblogic.jdbc.connectionPool.sales=
    url=jdbc:cloudscape:myDB;create=true,\n    driver=COM.cloudscape.core.JDBCDriver,\n    loginDelaySecs=1,\n    initialCapacity=4,\n    capacityIncrement=2,\n    maxCapacity=10,\n    props=user=francois;password=echecs
```

Once you have set up the connection pool, add users and their passwords to the `weblogic.properties` file as follows:

```properties
weblogic.password.john=johnPasswd
weblogic.password.janet=janetPasswd
```
To view a list of open JDBC connections, use the BEA WebLogic Admin servlet (T3AdminMain servlet).

**Setting Up Access Control Lists**

BEA WebLogic controls access to internal resources such as JDBC connection pools through Access Control Lists (ACLs) set up in the BEA WebLogic realm. Entries for ACLs in the WebLogic realm are listed as properties in the `weblogic.properties` file.

To set the “reserve” and “reset” permissions for JDBC connections in a connection pool, enter the appropriate property in the `weblogic.properties` file. Setting a permission on ACL `weblogic.jdbc.connectionPool` limits access to all connection pools. See the BEA WebLogic ACL documentation for more information on the “reserve” and “reset” permissions.

**NOTE:** Only the system administrator can reset a connection pool.

To add permissions for other users, add an entry for the ACL name `weblogic.jdbc.connectionPool.<poolID>`, which controls access to the connection pool ID. The special user system always grants the permissions “reserve” and “reset” for every ACL, no matter what other permissions have been set.

Examples of setting up “reserve” and “reset” permissions are:

```plaintext
weblogic.allow.reserve.weblogic.jdbc.connectionPool.sales=margaret,joe,mary

weblogic.allow.reset.weblogic.jdbc.connectionPool.sales=sysMonitor
```

The following example adds users to a previously created connection pool:

```plaintext
weblogic.allow.reserve.weblogic.jdbc.connectionPool.eng=margaret,joe
```

**Using a Connection from a Connection Pool**

To use a connection from a connection pool in your client application:
User Authentication Using CloudConnector

1. Create a `java.util.Properties` object.
2. Set the `weblogic.t3.connectionPoolID` property to the name of the connection pool you created in the Tengah Server’s `weblogic.properties` file.

The following simple example opens a connection to BEA WebLogic, sets up a `Properties` object, and then opens a series of connections from the “eng” connection pool created in a previous example.

```java
Class.forName("COM.cloudscape.core.WebLogicDriver").newInstance();

// Set the weblogic.t3.connectionPoolID to specify
// the connection pool to use the connection from;
// specify userName and passwd for this connection.

Properties tengahProps = new Properties();
tengahProps.put("weblogic.t3.connectionPoolID", "eng");
tengahProps.put("user", "fred");
tengahProps.put("password", "fredPasswd");

Connection conn = DriverManager.getConnection(
    "jdbc:cloudscape:weblogic://localhost:7001+
    &weblogic.t3.cacheRows=0",
    tengahProps);

// Do some database work

// Release the connection
conn.close();

Note that the client releases the JDBC connection and returns it to the pool before disconnecting. When the JDBC connection is returned to the connection pool, all outstanding JDBC transactions are rolled back and closed.

Refreshing or Resetting a Connection Pool

You can refresh a single connection from a connection pool, or reset the entire connection pool, if one or more connections in the pool go stale. This can happen
if Cloudscape is taken down while the BEA WebLogic server is actively supporting a pool of connections.

**NOTE:** Never reset a connection pool as a routine part of a user program. Usually a connection pool needs resetting only when Cloudscape has become unavailable and the connections in the pool are no longer viable. Attempting to reset the pool before you are certain that Cloudscape is up and available causes an exception to be thrown. Only a user with administration privileges should reset a pool.

An `SQLException` is raised if a connection cannot be established.

See the accompanying *Tengah Administrators Guide* for more details about resetting connection pools.
This chapter explains how to configure and administer a server.

You have the option of setting properties before you start a server. Those that absolutely must be set are described in “Starting and Shutting Down the Cloudconnector Server” on page 3-1. This section provides additional property and configuration information.

- “Setting Cloudscape Properties” on page 7-1
- “Setting Cloudconnector Properties” on page 7-2
- “Cloudconnector Administration” on page 7-4

## Setting Cloudscape Properties

This section covers:

- “System Properties” on page 7-1
- “Conglomerate-Specific and Database-Wide Properties” on page 7-2
- “Other Properties” on page 7-2

### System Properties

When using Cloudscape as a database server, you specify Cloudscape system properties at the time the server is started. Setting system properties on the client
application has no effect on the server, because they are not transmitted to the server.

Set Cloudscape system properties in the `cloudscape.properties` file in the `cloudscape.system.home` directory. This is the same way you set properties in an embedded environment. Be sure to set the properties in the `cloudscape.properties` file on the server machine before starting up the server.

It is a good idea to set `cloudscape.system.bootAll` to `true` in a server, so that client connection time is reduced when the server boots. It is also a good idea to set the server to boot databases that are not under `cloudscape.system.home`.

### Conglomerate-Specific and Database-Wide Properties

You can set properties for specific databases. Some properties that apply to specific tables or indexes may be set in a client application within an SQL-J statement. See Tuning Cloudscape for information about conglomerate-specific and database-wide properties.

### Other Properties

To learn about core Cloudscape properties, see Tuning Cloudscape. To learn about synchronization-related properties, see the Cloudscape Synchronization Guide.

### Setting Cloudconnector Properties

**NOTE:** The rest of this chapter applies only to Cloudconnector users.

You can set system-wide (persistent) Cloudconnector properties using the `weblogic.properties` file, or you can set temporary Cloudconnector properties by specifying them in the database connection URL. For information about temporary Cloudconnector properties, see “Setting Session-Level WebLogic Properties on the URL” on page 4-5 in chapter 4. This section discusses using the `weblogic.properties` file.

To set persistent Cloudconnector properties, modify the `weblogic.properties` file. The only property that you must modify before starting the server is the password property, as discussed in “Setting a Password” on page 3-7. The `weblogic.properties` file requires no additional modification prior to running.
Cloudconnector, however there are several optional Cloudconnector properties you may want to specify or modify.

You can set optional Cloudconnector properties in this file or in the Cloudscape client database connection URL. Cloudconnector properties specified on a connection URL are valid only for that connection.

To set `cloudscape.system.home` when using Cloudconnector, set it when starting Cloudconnector in the `startCS` script. You add it as a parameter to the JVM command in the script:

```
-Dcloudscape.system.home=c:\cloudscape\serverDIR
```

Alternatively, you can set `cloudscape.system.home` by uncommenting the setting `java.system.property.cloudscape.system.home` in the weblogic.properties file.

**Configuring Multi-Threading**

Cloudconnector automatically multi-threads operations. When the value of the `weblogic.system.executeThreadCount` property is larger than 1, Cloudconnector uses multiple threads to handle incoming JDBC requests. The default value of the Cloudconnector configuration is 15.

**NOTE:** Execute threads are used by client JDBC requests, but not by embedded requests in servlets.

The higher the number of threads, the greater the possible concurrency.

Users running under a JVM with poor threading capabilities may want to disable multi-threading. To disable multi-threading, modify the property in the `weblogic.properties` file:

```
weblogic.system.executeThreadCount=1
```

However, if you disable multi-threading, concurrency is drastically reduced and deadlock becomes more likely. For example, consider the following scenario:

Client transaction A inserts into the `People` table and thus obtains a lock on that table. Cloudconnector assigns one thread to that action. When the action completes, Cloudconnector releases the execute thread while still holding the lock. A few seconds later, client transaction B tries to select from the `People` table and must wait for A to release the lock. Cloudconnector assigns the one available thread to that action. This thread waits along with the waiting SELECT statement. A few seconds later, transaction A commits. There is no other Cloudconnector thread to assign to that thread, however, because the one available thread is tied to the waiting SELECT statement. Finally, transaction B aborts because of a deadlock timeout.
The thread in Cloudconnector becomes available, and transaction A can commit. Transaction B must resubmit the SELECT. (See the Cloudscape Developer’s Guide for more information on deadlocks.)

You cannot set the executeThreadCount property at the session level on the database connection URL. You must set it in the weblogic.properties file before starting up Cloudconnector.

You can see the values of all Cloudconnector properties for the current server through the administrative servlet; see “Cloudconnector Administration” on page 7-4 for details.

Disabling HTTP Services

By default, HTTP services are enabled in Cloudconnector. To disable HTTP services, set:

\[ \text{weblogic.httpd.enable=false} \]

in the weblogic.properties file.

**NOTE:** The WebLogic default setting of weblogic.httpd.enable is false, but the Cloudconnector customization of WebLogic sets it to true.

You can also control the number of HTTP threads by setting the weblogic.system.httpdThreadCount property. The default value is 5.

Cloudconnector Administration

Cloudconnector provides a servlet-based interface for administering the WebLogic Server.

**NOTE:** Cloudscape has its own mechanism for defining users. See the Cloudscape Developer’s Guide for details. This section addresses Cloudconnector users only.

When Cloudconnector is running, the administrative servlet is available at the URL http://hostname:7001/T3AdminMain. When you enter this URL, you will be prompted for a username and password. Use the default user “system” with a password that you defined in the weblogic.system.password property of the weblogic.properties file. You can also define additional users and passwords in the weblogic.properties file by adding entries of the form:
The interface allows you to look up information about topics such as the following:

- **Event topics**
- **Event registrations**
- **JDBC info**
  - WebLogic/JDBC Connection Pool capability.
- **Licenses**
- **Properties**
  - Shows the settings made in your `weblogic.properties` file.
- **Client workspaces**
- **Connections**
  - The administration interface shows the WebLogic Connections, which map to the JDBC connections made to Cloudconnector.
- **Threads**
  - Shows WebLogic activity occurring within Cloudconnector to satisfy JDBC requests.
- **Versions**
- **Help**

For information on administering Access Control Lists (ACLs), see the *WebLogic Administrators and Deployment Guide*. 
Cloudconnector supports the Secure Socket Layer (SSL) protocol for data security between client and server. SSL allows client and server to authenticate themselves to each other using certificates, and allows both machines to establish a connection that transports encrypted data. SSL is particularly useful for sending encrypted passwords.

For more details on the SSL protocol, see the section *Using WebLogic SSL* in the *WebLogic Administrators Guide* included with this release. This guide is in frameworks/cloudconnect/doc.

### Setting Properties to Enable SSL

To use SSL in Cloudconnector, set the following properties in the `weblogic.properties` file:

- `weblogic.security.ssl.enable=true`  
  Default is `false`.
- `weblogic.system.SSLListenPort= <SSL listen port number>`  
  Default is 7002.
- `weblogic.security.certificate.server=democert.pem`
- `weblogic.security.key.server=demokey.pem`
- `weblogic.security.certificate.authority=ca.pem`

See the BEA WebLogic *Using LogicWorks SSL* document for more information on these properties.
SSL Security in Cloudconnector

The `weblogic/CloudscapeServer` directory contains three sample `.pem` files. These example files show how the certificate works, but they are not adequate for standard client-to-server communication, because the certificate is not signed by a well-known or trusted certificate authority. For information on generating a signed certificate, see the “Using WebLogic SSL” section of the BEA WebLogic Developers Guide.

URLs for SSL

As shown in “Using Cloudconnector Clients” on page 4-2, the standard Cloudscape database connection URL is:

```
jdbc:cloudscape:weblogic:[//hostname:portnum/]
dbname[cloudscape_attributes]
```

When you are using SSL, the database connection URL is:

```
jdbc:cloudscape:weblogic-ssl:[//hostname:portnum/]
dbname[cloudscape_attributes]
```

where `portnum` is the value specified in the `weblogic.system.SSLListenPort` property. (Do not enter line breaks.)

For example, the following URL connects to a database using SSL:

```
jdbc:cloudscape:weblogic-ssl:myDB;user=joe;password=sEcReT
```
Cloudscape Access
Using RmiJdbc

RmiJdbc is free software that provides remote access to JDBC drivers. It is included with all Cloudscape products. RmiJdbc provides some server framework functionality, but it does not provide the same level of scalability as a commercial server framework. It is a client/server JDBC driver that uses JavaSoft’s Java RMI to transport a client’s JDBC requests to a JDBC server bound to a database JDBC driver. Cloudscape can run embedded inside the RmiJdbc server.

This chapter contains the following sections:

- “How RmiJdbc Works” on page 9-1
- “Configuring Cloudscape for RmiJdbc” on page 9-2
- “RmiJdbc JVM Information” on page 9-2
- “Setting Port Numbers” on page 9-3
- “Starting and Stopping the RmiJdbc Server” on page 9-3
- “Accessing RmiJdbc Server from a Client” on page 9-5
- “The RJPing Utility” on page 9-7

How RmiJdbc Works

A daemon (RJJdbcServer) runs on the server side. The daemon offers an RMI server object that implements the whole interface of a JDBC driver object. The server object can give access to any JDBC driver registered in the server’s JDBC DriverManager. The object acts as a proxy gateway to registered JDBC drivers that it contains.
The RmiJdbc JDBC driver itself runs on the client side. The JDBC classes it implements contact the RmiJdbc server to access databases.

The only requests supported by an RmiJdbc server are JDBC requests.

To allow an applet to connect to databases on hosts other than the Web server, you can use the RmiJdbc server as a proxy. This prevents your getting security exceptions from the “sandbox” restrictions of the original Java security model, and allows you to connect to hosts other than the Web server. An RmiJdbc server can connect to another RmiJdbc server using its own transport, the RmiJdbc client driver.

Cloudscape provides a customized distribution of the RmiJdbc software. You do not need to follow the RmiJdbc server installation instructions.

Configuring Cloudscape for RmiJdbc

Setting Class Path

Before starting the server, add the class paths of the RmiJdbc and Cloudscape libraries to your class path. For instance, if you have installed Cloudscape under C:\cloudscape, add C:\cloudscape\frameworks\RmiJdbc\classes\RmiJdbc.jar and C:\cloudscape\lib\cloudscape.jar to the class path. Also add the libraries for any Java classes used in your databases.

Cloudconnector also provides script files for setting the class path, located in $CLOUDSCAPE_INSTALL\frameworks\RmiJdbc\bin.

See Getting Started with Cloudscape for more information on setting class path.

RmiJdbc JVM Information

RmiJdbc is compatible with JDK 1.1.x. and JDK 1.2.

By default, Cloudscape does not install the JDK security manager. If you want to use the security manager, use the -sm flag when booting an RmiJdbc server.

NOTE: The RmiJdbc server uses the 1.1 JDK package to look up the server host name (java.net.InetAddress.getLocalHost). On Solaris, this method is sometimes unreliable and may return “localhost” instead of the server host.
name, which means that clients will not be able to connect. To get around this problem, set the property `java.rmi.server.hostname` to the actual server host name when starting the RmiJdbc server. For example:

```
java -ms16m -mx32m -Djava.rmi.server.hostname=godfrey
    RmiJdbc.RJJjdbcServer COM.cloudscape.core.JDBCDriver
```

For information on permissions and security with JDK1.2, see the JDK Web site:

`//java.sun.com/products/jdk/1.2/docs/index.html`

### Setting Port Numbers

By default, Cloudscape using RmiJdbc listens on TCP/IP port number 1099. If you want to use a different port number, you can specify it on the command line when starting the RmiJdbc server. For example:

```
C:\> java -ms16m -mx32m -noasyncgc RmiJdbc.RJJjdbcServer -port 1088 COM.cloudscape.core.JDBCDriver
```

It is better, however, to specify these changes in the `startCS.bat` or `startCS.ksh` scripts. See “Starting and Stopping the RmiJdbc Server” on page 9-3 for more information on these scripts.

### Starting and Stopping the RmiJdbc Server

#### Starting the Server

There is no need to start a separate RMI registry daemon. The RMI Registry service for the RmiJdbc server is embedded and runs inside the server itself.

The `startCS.bat` (Windows) and `startCS.ksh` (UNIX) scripts start the server, and the `stopCS.bat` (Windows) and `stopCS.ksh` (UNIX) scripts stop it. These scripts are located in the `$CLOUDSCAPE_INSTALL/frameworks/RmiJdbc/bin` directory. The RmiJdbc server package itself resides in the `$CLOUDSCAPE_INSTALL/frameworks/RmiJdbc/classes` directory.

For example, if you have installed Cloudscape in the default directory on the C drive (in Windows), go to the directory where your Cloudscape databases will reside and type the following command:
**Cloudscape Access Using RmiJdbc**

```bash
start C:\cloudscape\frameworks\RmiJdbc\bin\startCS.bat
```

You can set `C:\cloudscape\frameworks\RmiJdbc\bin` in your path to shorten the command.

**Altering the StartCS Script**

You can modify the `StartCS` script to do any of the following:

- **disable RMI registry creation from inside an RmiJdbc server**
  
  Add `-noreg` after `RJJdbcServer`.

- **specify a port number other than the default (1099) for the RMI registry**
  
  Add `-port <portnumber>` after `RJJdbcServer`.

- **add additional JDBC driver classes to register**
  
  List the classes after `RJJdbcServer`.

- **prevent dumping of informational messages to the console**
  
  Add `-DRmiJdbc.verbose=false` after the `jre` command.

- **cause the application not to exit from the JVM on shutdown**
  
  Add `-DRmiJdbc.exitOnShutdown=false` after the `jre` command.

- **load the RMI security manager (the default behavior is not to load the security manager)**
  
  Add `-sm` after `RJJdbcServer`.

**Shutting Down the Server**

To shut down the RmiJdbc server, use the `stopCS.bat` on Windows or `stopCS.ksh` on UNIX. `StopCS` shuts down Cloudscape as well as the RmiJdbc server. If you have enabled user authentication with the `cloudscape.connection.requireAuthentication` property, you must provide user credentials for shutdown. To do this, add a user name and password to the first line of the `stopCS.bat` or `stopCS.ksh` file. For example:

```bash
java RJPing
jdbc:rmi:jdbc:cloudscape;;shutdown=true;user=YOURUSERNAME;
password=PWD connect
```

(This should all be written on one line.)
Accessing RmiJdbc Server from a Client

- “Client Class Path” on page 9-5
- “JDBC Driver and Database Connection URL for an RmiJdbc Client” on page 9-5
- “Client JDK” on page 9-6
- “Database Connection URL for RmiJdbc” on page 9-6
- “Using the Cloudscape Tools with RmiJdbc” on page 9-7

Client Class Path

Clients need to install and set the class path to point to the libraries indicated in “Libraries” on page 2-4. To summarize, clients need the following files and directories, shown in their paths relative to the cloudscape base directory:

- /lib/client.jar
- server framework classes
  For RmiJdbc, the classes are in frameworks/RmiJdbc/classes/RmiJdbc.jar.
- /lib/tools.jar
  If using the Cloudscape tools on the client.

JDBC Driver and Database Connection URL for an RmiJdbc Client

The Cloudscape Developer’s Guide documents the database connection URL for using Cloudscape as an embedded database.

RmiJdbc supplies a tailored JDBC driver and database connection URL for use with a client.

In client code, specify the client JDBC driver for connecting to a Cloudscape server:

```java
Class.forName("COM.cloudscape.core.RmiJdbc").newInstance();
```

You can specify the driver in a system property when starting a client tool. For example (using supplied Cloudscape tools):

```java
java -Djdbc.drivers=COM.cloudscape.core.RmiJdbcDriver
COM.cloudscape.tools.cview
```
java -Djdbc.drivers=COM.cloudscape.core.RmiJdbcDriver
COM.cloudscape.tools.ij

Client JDK

RmiJdbc client applications can use JDK of 1.1 or 1.2.

Database Connection URL for RmiJdbc

A Cloudscape client to RmiJdbc uses the following database connection URL:

```
jdbc:cloudscape:rmi://hostname:portnum/
[databaseName];[cloudscapeAttributes]
```

(All of this is one string; the new lines above just make the URL readable.)

**NOTE:** If you are using the Secure Socket Layer (SSL), there is a special extended URL. See Chapter 8, “SSL Security in Cloudconnector”, for details.

- `//hostname:portnum/`
  Replace the name `hostname` with the name of the host running RmiJdbc if it is not running on the same machine as the client. `hostname` can be `localhost`. Replace `portnum` with the port number specified for the `weblogic.system.listenPort` property in the `weblogic.properties` file or on the RmiJdbc `java` command line. 7001 is the default value.

- `[databaseName];[cloudscapeAttributes]`
  Specify the database name and any database connection URL attributes as specified in Cloudscape Developer’s Guide. For example, to create and connect to a new database called `newDB`, specify:

  ```
  newDB;create=true
  ```

Here are some example Cloudscape client database connection URLs:

- `jdbc:cloudscape:rmi://localhost:1099/newDB;create=true;autocommit=false`
  Connects to RmiJdbc server installed on the local machine at the default port, creates and connects to a new database called `newDB`, turning off auto-commit (Cloudscape property).

  Connects to RmiJdbc server running on host jeeves and listening on port 5001, connects to an existing `accounts` database in the accounting directory.
• jdbc:cloudscape:rmi://localhost:1099/newDB;user=fred;password=secret
  Specifies user credentials for Fred.

NOTE: Client applications generally should not use the ;shutdown=true form of
the database connection URL in client/server environments. The system
administrator should use the provided utility to shut down a server.

For a simple example of an application that connects to RmiJdbc server as a client,
see the program CloudscapeSimpleApp in the demo/programs/simple directory.

Using the Cloudscape Tools with RmiJdbc

The Cloudscape tools ij and Cloudview work in both embedded mode and client/
server mode.

To access RmiJdbc server using these tools, simply specify the client driver and
client database connection URL for the client/server environment (see “JDBC
Driver and Database Connection URL for an RmiJdbc Client” on page 9-5) where
appropriate.

The book Cloudscape Tools and Utilities Guide provides numerous examples of
how to do this.

The RJPing Utility

RJPing is an RmiJdbc utility class that allows various operations against a running
RmiJdbc server. RJPing lets you:

• connect to the server
• get driver information
• shut down the server
• list all current drivers registered in the server

The RJPing.bat and RJPing.ksh scripts are located under the
C:\cloudscape\JBMSServer\RmiJdbc\bin directory.

The syntax of RJPing is:

RJPing [://hostname:port#/]dbURL [connect, driverInfo, shutdown]
If neither `connect`, `driverInfo`, nor `shutdown` is specified, `RJPing` returns a list of all current JDBC drivers in the server.

For help with `RJPing` syntax, type:

```
RjPing -?
```

or

```
RjPing help
```

`RJPing` returns 0 if it succeeds, -1 otherwise.

Examples using the `RJPing` utility are:

```
java RJPing
java RJPing //anotherHost:1099/
java RJPing //localhost:1099/ jdbc:cloudscape: driverInfo
java RJPing jdbc:cloudscape:myDB  connect
java RJPing //localhost:1099/jdbc:cloudscape:myDB connect
java RJPing jdbc:rmi://localhost:1099/jdbc:cloudscape:myDB connect
java RJPing //anotherHost:1099/ shutdown
java RJPing jdbc:rmi: shutdown
```
Since Cloudscape is written in Java, you have great flexibility in configuring your deployment. For example, you can run Cloudscape, a server framework, and another application in the same JVM as a single process. In this scenario, the server framework, like Cloudscape, is embedded in the application. Multiple client threads in the same JVM as Cloudscape can access a Cloudscape database through the local JDBC driver, and multiple external clients can access the same Cloudscape database by connecting to the embedded server framework.

The following are two typical scenarios that use embedded servers:

- **Occasional Remote Access or Occasional JDBC/ODBC Access**
  Most work is done locally, but administrators want occasional remote access. You can embed a small footprint, lightweight server framework (like the provided RmiJdbc) in the same JVM as Cloudscape and the embedding application to handle the occasional access from outside the JVM. Another reason for providing occasional access through the lightweight server framework is that the main application may not provide JDBC access (for example, servlets provide only http access). Loading the server framework allows the administrator to use JDBC or ODBC tools.

- **Frequent Remote Access**
  Many remote connections are needed, but local administration is also occasionally necessary. You can embed a robust server framework (like Cloudconnector) in the same application as Cloudscape and the embedding application. External connectivity goes through Cloudconnector, and local administration is handled by the embedding application through the local JDBC driver.

This chapter includes the following sections:

- “Embedded Server Topology” on page 10-2
- “How to Start an Embedded Server From an Application” on page 10-4
Embedded Server Topology

Figure 10-1, Figure 10-2, and Figure 10-3 show some typical Cloudscape topologies.

**Figure 10-1** The typical Cloudscape deployment. Cloudscape is embedded in a Java application; they both run in the same JVM. The embedding application uses the local JDBC driver to connect to Cloudscape. Other applications are not allowed to connect to Cloudscape.
Figure 10-2  A client/server Cloudscape deployment. Cloudscape is embedded in a server framework. End-user applications connect Cloudscape via the server frameworks. No application other than the server framework runs in the same JVM as Cloudscape or uses the local JDBC driver.

Figure 10-3  An “embedded server.” The server framework is started up by the embedding application, and so the server framework is embedded in the application. The embedding application, the server framework, and Cloudscape all run in the same JVM.
The server framework allows external JDBC applications to connect to Cloudscape. The embedding application and the server framework use the local JDBC driver to connect to Cloudscape; external JDBC applications use a client JDBC driver to connect.

**How to Start an Embedded Server From an Application**

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

```java
/* loading the client driver boots the client driver only*/
Class.forName("COM.cloudscape.core.JDBCDriver").newInstance();
Connection conn = DriverManager.getConnection(
    "jdbc:cloudscape:toursDB;autocommit=true");
```

In another thread the same application boots the server framework to allow remote access. Booting the server framework from within the application allows it to run in the same JVM as the application.

**RmiJdbc Server Example**

Instead of booting the server framework on the command line, the application boots the server framework programmatically:

```java
/* you could set any needed RmiJdbc properties as system properties here*/
myargs = new String[1];
myargs[0] = "COM.cloudscape.core.JDBCDriver";
System.out.println(
    RmiJdbc.RJjdbcServer.main(myargs));
```

The server framework’s attempt to boot the local JDBC driver is redundant and ignored, because it has already been booted within the framework’s JVM. The server framework simply accesses the instance of Cloudscape already booted; there is no conflict between the application and the server framework.

The remote client can then connect via the client driver:
Connection conn = DriverManager.getConnection(
    "jdbc:cloudscape:rmi://localhost:1099/toursDB");
/* interact with Cloudscape */
Statement s = conn.createStatement();
ResultSet rs = s.executeUpdate(
    "DELETE FROM HotelBookings WHERE booking_date < current_date");

Cloudconnector Example

Instead of booting the server framework on the command line, the application boots
the server framework programmatically:

// set any needed properties before booting Cloudconnector:
// you may also want to set weblogic.system.home
Properties p = System.getProperties();
p.put("cloudscape.system.home", "C:\serverdirectory");
String[] args = new String[1];
args[0] = "";
COM.cloudscape.core.CloudscapeServer.main(args);

Remote Cloudconnector clients connect using the Cloudconnector client driver and
database connection URL:

/* remote WebLogic clients connect using the WebLogic driver and URL*/
Connection conn = DriverManager.getConnection(
    "jdbc:cloudscape:weblogic:toursDB");

Because Cloudconnector is also an HTTP server, no additional configuration is
needed to allow servlet access. Servlets are also embedded in Cloudconnector,
which is embedded in the same JVM as Cloudscape, and so servlet access to
Cloudscape can be through the local JDBC driver, which is already booted.
PART TWO

Cloudscape Administration Guide
Checking Database Consistency

Cloudscape is a robust product containing many safeguards to ensure that the database remains consistent.

If, however, you have experienced hardware or operating system failure and you want to verify that the database is still consistent, Cloudscape provides a way to do this. 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 checkTable Method” on page 11-1
• “Sample Error Messages” on page 11-2
• “Sample Queries” on page 11-3

The checkTable Method

The method checkTable in the class

```
COM.cloudscape.database.ConsistencyChecker
```

checks the consistency of a Cloudscape table. (The class is aliased as ConsistencyChecker.) You run this method in an SQL-J statement, like this:

```
VALUES ConsistencyChecker.checkTable(
    SchemaName, TableName)
```

checkTable returns a boolean. If the table is consistent (or if checkTable is run on a view), checkTable returns true. Otherwise, checkTable throws an exception on the first inconsistency it finds.

checkTable verifies the following:
Checking Database Consistency

- Heaps are internally consistent.
- Heaps and all associated indexes contain the same number of rows.
- The values and row locations in each index match those of the heap.
- All BTREE indexes are internally consistent.

**NOTE:** Both `SchemaName` and `TableName` must be any expression that evaluates 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:

```sql
VALUES ConsistencyChecker.checkTable('APP', 'CITIES')
```

For a consistent table, the following result is displayed:

```
SQLC6
-----
true

1 row selected
```

## Sample Error Messages

This section provides examples of error messages that `checkTable` can return.

The row counts of the base table and an index differ:

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.

The index refers to a row that does not exist in the base table:

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.

A key column value differs between the base table and the index:

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 Queries

This section provides examples that illustrate using checkTable in queries.

Check the consistency of a single table:

VALUES ConsistencyChecker.checkTable('APP', 'FLIGHTS')

Check the consistency of all of the tables in a schema, stopping at the first failure:

SELECT tablename, ConsistencyChecker.checkTable('APP', tablename)
FROM sys.sysschemas s, sys.systables t
WHERE s.schemaname = 'APP' AND s.schemaid = t.schemaid

Check the consistency of an entire database, stopping at the first failure:

SELECT schemaname, tablename,
ConsistencyChecker.checkTable(schemaname, tablename)
FROM sys.sysschemas s, sys.systables t
WHERE s.schemaid = t.schemaid

NOTE: See the sample program JBMSTours.AdminHelper for an example of
running the consistency checker.
CloudeScape provides a way to back up a database while it is on-line. While the backup is in progress, update operations are temporarily blocked but read operations can still proceed.

- “Backing Up a Database” on page 12-1
- “Backup History” on page 12-5
- “Restoring from a Backup” on page 12-6

NEW: The online backup utility is new in Version 3.0.

### Backing Up a Database

- “Offline Backups” on page 12-1
- “On-line Backups” on page 12-2
- “When the Log is in a non-Default Location” on page 12-4
- “Backing Up Encrypted Databases” on page 12-5
- “Failure Recovery” on page 12-5

### Offline Backups

To perform an offline backup of a database, simply use operating system commands to copy the database directory. The database must be shut down prior to performing an offline backup.
For example, on Windows NT, the following operating system command would back up a (closed) database named *toursDB* located in *d:\mydatabases* by copying it to the directory *d:\mybackups\1999-01-01*:

```
xcopy d:\mydatabases\toursDB c:\mybackups\1999-01-01\toursDB /s /i
```

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

For large systems, shutting down the database may not be convenient. To back up a database without having to take it off line, system administrators should use on-line backups, described next.

### On-line Backups

Use online backups to back up a database while it is booted. During the interval the backup is running, the database can be read, but writes to the database are blocked.

There are two ways to perform on-line backups:

- “Using the backup Method” on page 12-2
- “Using Operating System Commands with the freeze and unfreeze Methods” on page 12-3

### Using the backup Method

The *backup* method locks the database and performs the copy operation.

The *backup* method is located in the *COM.cloudscape.database.Database* class. When connected to the database you want to back up, use the static method *getDatabaseOfConnection* in *COM.cloudscape.database.Factory* (aliased as *Factory*), to get a *Database* object for calling this method. The *backup* method takes a string argument representing 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 *cloudscape.system.home* directory.)

For example, to specify a backup location of *c:/mybackups/1999-01-01* for the currently-open database, you would use the following statement:

```
CALL Factory.getDatabaseOfConnection().backup('c:/mybackups/1999-01-01')
```
NOTE: Use forward slashes as the path separator in SQL-J commands.

The backup method puts the database into a state in which it can be safely copied, then copies the entire original database directory (including data files, on-line transaction log files, and jar files) to the specified backup directory. Files not within the original database directory (for example, cloudscape.properties) are not copied.

Here is an excerpt from JBMSTours.AdminHelper, which backs up a database to a directory with a name that reflects the current date:

```java
public static void backUpDatabase(Connection conn) throws SQLException {
    String backupdirectory = "c:/mybackups/" + JCalendar.getToday();
    PreparedStatement ps = conn.prepareStatement(
        "CALL Factory.getDatabaseOfConnection().backup(CAST (? AS java.lang.String))");
    ps.setString(1, backupdirectory);
    ps.executeUpdate();
    ps.close();
    System.out.println("backed up database to " + backupdirectory);
}
```

For a database backed up on 1999-01-01, the above commands copy the current database to a directory of the same name in c:/mybackups/1999-01-01.

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.

Using Operating System Commands with the freeze and unfreeze Methods

Typically, this procedure is used to speed up the copy operation involved in the on-line backup. In this scenario, Cloudscape does not perform the copy operation for you. You use the freeze method to lock the database, and then explicitly copy the database directory using operating system commands.

For example, because the UNIX tar command uses the JDK file-copying routines, it may provide faster backups than the backup method.

To use operating system commands for on-line database backups, first use the freeze method located in the COM.cloudscape.database.Database class. The freeze method puts the database into a state in which it can be safely copied. Once the copy

Cloudscape Version 3.0
is complete, use the \textit{unfreeze} method to continue working with the database. Only after \textit{unfreeze} has been specified can transactions once again write to the database. Read operations can proceed while the database is “frozen”.

\textbf{NOTE:} Since Cloudscape does not use the lock manager for freeze operations, it is imperative that no applications attempt to write to a ‘frozen’ database. If writes are attempted, the requesting application may lock up.

As with the backup method, use the static method \texttt{getDatabaseOfConnection} in \texttt{COM.cloudscape.database.Factory} (aliased as \texttt{Factory}), to get a \texttt{Database} object for calling these methods.

The following example demonstrates using \texttt{freeze} and \texttt{unfreeze} surrounding an operating system copy command:

\begin{verbatim}
String backupdirectory = "c:/mybackups/" + JCalendar.getToday();
Statement s = conn.createStatement();
s.executeUpdate("CALL Factory.getDatabaseOfConnection().freeze()");
//copy the database directory during this interval
s.executeUpdate("CALL Factory.getDatabaseOfConnection().unfreeze()");
s.close();
\end{verbatim}

\section*{When the Log is in a non-Default Location}

\textbf{NOTE:} Read “Logging on a Separate Device” on page 13-1 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, note the following:

\begin{itemize}
\item If backing up the database using an operating system command, you must explicitly copy the log file as well.
\begin{verbatim}
xcopy d:\mydatabases\toursDB c:\mybackups\1999-01-03\toursDB /s /i
xcopy h:\janet\tourslog\log c:\mybackups\1999-01-03\toursDB\log /s /i
\end{verbatim}
\end{itemize}

\textbf{NOTE:} If you are not using Windows NT, substitute the appropriate operating system command for copying a directory and all contents to a new location.
• Edit the \textit{logDevice} entry in \texttt{service.properties} of the database backup so that it points to the correct location for the log. In the above example, the log was moved to the default location for a log, so you could remove the logDevice entry entirely, or leave the logDevice entry as is and wait until the database is restored to edit the entry.

See Chapter 13, “Logging on a Separate Device” for information about putting the log in a non-default location.

\section*{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.

\textbf{NOTE:} See the sample program \texttt{JBMS\textbackslash{}Tours\textbackslash{}AdminHelper} for an example of backing up a database.

\section*{Backup History}

Backup history is stored in an ASCII file (\texttt{BACKUP.HISTORY}) in the database directory of the original database (not its backup). This file records the time and location of the backup. If the file does not already exist, it is created during the backup. If it does already exist, new information is appended to it.

\begin{verbatim}
Thu Sep 09 09:18:35 PDT 1999: Backup started for database located at D:\mydatabases\toursDB
Thu Sep 09 09:18:44 PDT 1999: copied database directory from D:\mydatabases\toursDB to C:\mybackups\1999-01-01\toursDB
Thu Sep 09 09:18:44 PDT 1999: copied log directory from D:\mydatabases\toursDB\log to C:\mybackups\1999-01-01\toursDB\log
Thu Sep 09 09:18:44 PDT 1999: Backup completed, log instant at \(10,15855\)
\end{verbatim}

\section*{Failure Recovery}

Regardless of which backup option you choose, if a backup does not complete, the newly created copy of the database is unusable. There is no automatic cleanup for backup failure.
If the database crashes during a backup or freeze, the original database is “normal” after it is recovered, and the copy is unusable and should be discarded.

**Restoring from a Backup**

To restore a database for which both of the following are true:

- The log is in the default location
- There is no logDevice entry in the service.properties file governing the location to which the database is being restored

simply copy the backed-up database directory to the appropriate location and boot the database normally. Once the database has been booted, you may want to run a consistency check if time permits. (See Chapter 11, “Checking Database Consistency”.)

**NOTE:** When copying a backed-up database to its new location, make sure there is not already a database of the same name there.

Here is an example that uses Windows NT operating system commands to restore a backup of toursDB to the directory d:\mydatabases:

```
rdir   d:\mydatabases\toursDB /s
xcopy c:\mybackups\1999-01-01\toursDB d:\mydatabases\toursDB /s /i
```

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

**Restoring a Database When the Log Is in a Non-Default Location**

**When You Backed up with Cloudscape's Built-In Backup Command**

When you back up a database whose log is in a non-default location, in the backup itself Cloudscape puts the log in the default location. When you restore the database from such a backup, if you wish to put the log back in a non-default location, you
will have to move the log by hand. Follow the instructions in Chapter 13, “Logging on a Separate Device” for moving a log by hand.

**When You Backed up with Operating System Commands**

If the database log is not in the default location, or if a logDevice entry is specified in the service.properties file governing the location to which the database is being restored, make sure the logDevice entry and the new log location correspond. Once they correspond, simply copy the database to the desired location and boot it normally.

To make the logDevice entry and the log location correspond, do one of the following:

- Move the log to the location indicated by the logDevice property.
- Alter the logDevice entry in service.properties to correspond to the log location.
- Change both the new log location and logDevice property to represent the same location.
- Move the log to the default location and delete the logDevice entry in service.properties.

Here is an example:

1. Move the backup to the desired location for the database.
   
   ```
   xcopy c:\mybackups\1999-01-03\toursDB d:\mydatabases\toursDB /s /i
   ```

2. Copy the backup of the log to the desired location.
   
   ```
   xcopy c:\mybackups\1999-01-03\toursDB\log h:\janets\tourslog\log /s /i
   ```

3. Edit the service.properties file to reflect the correct location for the log:
   
   ```
   logDevice=H:\janets/tourslog
   ```
Backing Up and Restoring Databases
13 Logging on a Separate Device

System administrators can improve the performance for 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. If you want to store this log subdirectory in another location, do either of the following:

- Specify the non-default location 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.

NEW: The ability to put the log on a separate device is new in Version 3.0.

Using the logDevice Property

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 a path relative to the directory where the JVM is being executed.

Setting logDevice on the database connection URL adds an entry to the service.properties file. If you ever move the log manually, alter the entry in service.properties. If you move the log back to the default location, remove the logDevice entry from service.properties.
To check the log location for an existing database, you can retrieve the logDevice attribute as a database property. Issue the following statement:

```
VALUES PropertyInfo.getDatabaseProperty('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:cloudscape:d:/mydatabases/
toursDB;create=true;logDevice=h:/janets/tourslog'
```

**Example of Moving a Log Manually**

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

```
move h:\janets\tourslog\log\*.* g:\bigdisk\tourslog\log
```

and alter the `logDevice` entry in `service.properties` to read as follows:

```
logDevice=g:/bigdisk/toursLog
```

**NOTE:** You can use a single forward slash 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:

```
move g:\bigdisk\tourslog\log\*.* d:/mydatabases/toursDB/log
```

and delete the `logDevice` entry from `service.properties`.

**NOTE:** This example uses commands specific to the Windows NT operating system. Use commands appropriate to your operating system to copy a directory and all 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 may require extra steps. See Chapter 12, “Backing Up and Restoring Databases” for details.
Cloudscape provides tools to monitor and display locking information. These tools can help you create applications that minimize deadlock. They can also help you locate the cause of deadlock when it does occur.

To diagnose locking problems, you can do one or both of the following:

- display current lock manager information by querying built-in VTIs (LockTable and TransactionTable)
- constantly monitor locking traffic by logging all deadlocks (using the cloudscape.locks.monitor property)

The following topics are covered in this chapter:
- “Displaying Current Locking Information” on page 14-1
- “Lock Monitoring” on page 14-4

Displaying Current Locking Information

Locking information is provided by querying the LockTable and TransactionTable VTIs. Typically, you will derive the locking information you need by writing queries that join these two virtual tables on the XID column.

LockTable

Each row of the COM.cloudscape.vti.LockTable virtual table corresponds to a lock and its associated objects.
The columns of \textit{LockTable} are:

<table>
<thead>
<tr>
<th>Name</th>
<th>Data Type</th>
<th>Width</th>
<th>Nullable</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>XID</td>
<td>VARCHAR</td>
<td>10</td>
<td>no</td>
<td>The transaction ID</td>
</tr>
<tr>
<td>TYPE</td>
<td>VARCHAR</td>
<td>5</td>
<td>no</td>
<td>The type of lock:</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>\begin{itemize}</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>\item ROW (row lock)</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>\item TABLE (table lock)</td>
</tr>
<tr>
<td>MODE</td>
<td>VARCHAR</td>
<td>4</td>
<td>no</td>
<td>Lock mode:</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>\begin{itemize}</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>\item S (shared lock)</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>\item U (update lock)</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>\item X (exclusive lock)</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>\item IS (intent shared lock, N/A to Row lock)</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>\item IX (intent exclusive lock, N/A to Row lock)</td>
</tr>
<tr>
<td>TABLENAME</td>
<td>VARCHAR</td>
<td>30</td>
<td>no</td>
<td>Table name</td>
</tr>
<tr>
<td>LOCKNAME</td>
<td>VARCHAR</td>
<td>20</td>
<td>no</td>
<td>ID of locked object (for example, page and row number)</td>
</tr>
<tr>
<td>STATE</td>
<td>VARCHAR</td>
<td>5</td>
<td>no</td>
<td>GRANT or WAIT</td>
</tr>
<tr>
<td>TABLETYPE</td>
<td>VARCHAR</td>
<td>9</td>
<td>no</td>
<td>Type of table:</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>\begin{itemize}</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>\item S (system table)</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>\item T (user table)</td>
</tr>
<tr>
<td>LOCKCOUNT</td>
<td>VARCHAR</td>
<td>5</td>
<td>no</td>
<td>An internal count</td>
</tr>
<tr>
<td>INDEXNAME</td>
<td>VARCHAR</td>
<td>30</td>
<td>yes</td>
<td>Index name, if applicable</td>
</tr>
</tbody>
</table>

\textit{LockTable} is a built-in class alias, so you can invoke \textit{LockTable} without using the class name. For example:

\begin{verbatim}
SELECT * FROM NEW LockTable() AS LT
\end{verbatim}

\textbf{TransactionTable}

Each row of the \textit{COM.cloudscape.vti.TransactionTable} VTI (aliased as \textit{TransactionTable}) corresponds to a transaction and its associated objects.

The columns of \textit{TransactionTable} are:
Displaying Current Locking Information

NEW: These VTIs are new in Version 3.0.

Example Using the Lock Manager VTIs

The following example illustrates a use of the lock diagnostics VTIs. Statements that create locks on the HotelAvailability and FlightAvailability tables are run, and then a query of LockTable and TransactionTable shows the status of the locks.

```
SELECT Locks.xid, Locks.Type, TableName, Mode, Lockname, State
FROM NEW LockTable() AS Locks, NEW TransactionTable() AS Trans
WHERE Locks.xid = Trans.xid AND TABLENAME IN (VALUES 'HOTELAVAILABILITY', 'FLIGHTAVAILABILITY')
```

<table>
<thead>
<tr>
<th>Name</th>
<th>Data Type</th>
<th>Width</th>
<th>Nullable</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>XID</td>
<td>VARCHAR</td>
<td>10</td>
<td>no</td>
<td>The transaction ID</td>
</tr>
<tr>
<td>GLOBAL_XID</td>
<td>VARCHAR</td>
<td>140</td>
<td>yes</td>
<td>The global transaction id if it is a participant of a JTA-related global transaction.</td>
</tr>
<tr>
<td>USERNAME</td>
<td>VARCHAR</td>
<td>30</td>
<td>no</td>
<td>The user name (APP by default).</td>
</tr>
<tr>
<td>TYPE</td>
<td>VARCHAR</td>
<td>30</td>
<td>no</td>
<td>The transaction type. Types other than UserTransaction are typically short-lived and internal.</td>
</tr>
<tr>
<td>STATUS</td>
<td>VARCHAR</td>
<td>8</td>
<td>no</td>
<td>IDLE or ACTIVE</td>
</tr>
<tr>
<td>FIRST_INSTANT</td>
<td>VARCHAR</td>
<td>20</td>
<td>yes</td>
<td>The instant of the first log record (if this is an update transaction).</td>
</tr>
<tr>
<td>SQL_TEXT</td>
<td>LONGVARCHAR</td>
<td>1024</td>
<td>yes</td>
<td>The text of any currently-executing SQL-J statements. For all transactions not currently executing a statement, this column displays NULL.</td>
</tr>
</tbody>
</table>
Note in this example that transactions 995 and 1000 are competing for the same
locks. For the row lock on (1,6) in HotelAvailability, Transaction 995 has the lock
and transaction 1000 is waiting. For the rowlock on (1,6) in FlightAvailability,
transaction 1000 has the lock and Transaction 995 is waiting. (These two
transactions are deadlocked).

At this point, you may want to see the text of the SQL statements associated
with each of these transactions. You could query the TransactionTable like this:

```sql
SELECT SQL_TEXT FROM NEW TransactionTable() AS Trans
WHERE XID = 995 OR XID = 1000
```

**NOTE:** See the sample program JBMSTours.AdminHelper for this example of
forcing a deadlock and working with the lock diagnostics VTIs.

### Lock Monitoring

The cloudscape.stream.error.logSeverityLevel property determines the level of
error you are informed about. By default, cloudscape.stream.error.logSeverityLevel
is set at 40000. If cloudscape.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 cloudscape.LOG file. If it is set to a value of 40000 or
higher, deadlock errors are not logged to the cloudscape.LOG file.

The cloudscape.locks.monitor property ensures that deadlock errors are logged
regardless of the value of cloudscape.stream.error.logSeverityLevel. When
cloudscape.locks.monitor is set to true, all locks involved in deadlocks are written
to cloudscape.LOG along with a unique number identifying the lock.
To see a thread’s stack trace when a lock is requested, set cloudscape.locks.deadlockTrace to true. This property is ignored if cloudscape.locks.monitor is set to false.

**NOTE:** Use cloudscape.locks.deadlockTrace with care. Setting this property may alter the timing of the application, severely affect performance, and produce a very large cloudscape.LOG file.

For information on how to set properties, see Tuning Cloudscape. For specific information about the properties mentioned in this section, see the properties chapter in that book.

Here is an example of a victim’s error message when Cloudscape 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 & waiters is:
  Lock : ROW, X, FLIGHTAVAILABILITY, (1,6)
    Victim XID : 938, APP, UPDATE FlightAvailability SET economy_seats_taken = 0 WHERE flight_id = 'AA111'
    Granted XID : 939
  Lock : ROW, X, HOTELAVAILABILITY, (1,6)
    Waiting XID : 939, APP, UPDATE HotelAvailability SET rooms_taken = 0 WHERE hotel_id = 9
    Granted XID : 938
```

**NEW:** Beginning in Version 3.0, you can use the cloudscape.locks.waitTimeout and cloudscape.locks.deadlockTimeout properties to configure how long Cloudscape waits for a lock to be released, or when to begin deadlock checking. For more information on these properties, see the chapter in the Cloudscape Developer’s Guide that discusses controlling Cloudscape application behavior.
BACKUP.HISTORY file 12-5
BEA WebLogic
Cloudscape's customization of 1-8

C
Caching
turning off for server 4-6
Class path
for a server framework 2-5
setting for clients 9-5
Clients
setting class paths for 9-5
client.jar file
in class path 4-2, 9-5
when needed 2-4
Cloudconnector
description 1-5
cloudconnector
using with ij and Cloudview 4-7
Cloudscape base directory 2-1
cloudscape.connection.requireAuthentication 3-10, 9-4
cloudscape.locks.deadlockTrace 14-5
cloudscape.locks.monitor 14-4
cloudscape.properties file 3-6
cloudscape.stream.error.logSeverityLevel 14-4
cloudscape.system.bootAll 7-2
cloudscape.system.home 3-4
cloudscape.system.home 3-4
Cloudview
using with cloudconnector 4-7

D
Database connection URL
for an embedded database 4-2, 9-5
for client connection to server 4-3, 9-6
for SSL 8-2
setting WebLogic properties on 4-5
Database object
getting 12-2
DATABASE.backup 12-2
Deadlock
tools to help minimize 14-1

COM.cloudscape.core.CloudscapeServer class
1-8, 3-1
COM.cloudscape.database.ConsistencyChecker.checkTable 11-1
COM.cloudscape.database.Database.backup 12-2
COM.cloudscape.database.Database.freeze 12-3
COM.cloudscape.database.Database.unfreeze 12-4
COM.cloudscape.database.Factory.getDatabaseOfConnection 12-2, 12-4
COM.cloudscape.vti.LockTable 14-1
COM.cloudscape.vti.TransactionTable 14-2
Connection pools
definition 6-1
reasons for using 6-2
refreshing 6-4
resetting 6-4
setting up 6-2
using a connection from 6-3
Consistency
checking database 11-1
Encrypted databases
backing up 12-5

FACTORY 12-4
freeze method 12-3

HTTP services 7-4

ij
using with cloudconnector 4-7
Installation directory 2-1
Isolation levels 1-4

Java data types
in a client 5-1
in a server 5-1

Java.util.Properties 6-4
JDBC driver 1-9, 4-3, 9-5

Libraries
for server frameworks 2-4
License file 1-8
Locking
row-level 1-4
table-level 1-4
Locks
displaying current 14-1
monitoring 14-1
LockTable virtual table 14-1, 14-2
Log
default location for 13-1
logDevice 13-1
Logging transactions 13-1

Multiple connections
from single application running in embedded mode 1-4
Multi-threading
and JDBC 1-4
configuring for server 7-3

Parameter cache 4-6
Password 3-7

Port numbers
for RmiJdbc 9-3
modifying for server listening for client requests 3-7
Prefetching
turning off for server 4-6

Remote access
combining with local access 10-1
Restoring a database 12-6
RJPing utility 9-7
RmiJdbc
configuring 9-2
definition 9-1
driver 9-1
protocol 9-6
setting port numbers for 9-3
starting 9-3
stopping 9-3
Row-level locking 1-4

Secure Socket Layer (SSL)
database connection URL for 8-2
definition 8-1
setting Cloudconnector properties for 8-1
Serializable
working with types that are not 5-2
Servers
embedding 10-1
setting properties for 7-1
setting system properties for 7-1
shutting down 3-8–3-10
starting 3-1–3-8
service.properties file
setting logDevice in 13-1
SSL
database connection URL for 8-2
definition 8-1
setting Cloudconnector properties for 8-1
Start scripts 3-1
startCS script
for RmiJdbc 9-3
with Cloudconnector 3-4
Stop scripts 3-1
stopCS script 3-8
stopCS script
for RmiJdbc 9-3
System properties
for servers 7-1
T
Table-level locking 1-4
tools.jar file 4-2, 9-5
Transaction log
putting on separate device 13-1
TransactionTable virtual table 14-1

U
unfreeze method 12-4
Users
Cloudscape server support for 6-1
WebLogic server support for 6-1

W
WebLogic
Cloudscape’s customization of 1-8
weblogicaux.jar file 4-2
WebLogicLicense file 1-8
weblogic.httpd.enable property 7-4
weblogic.properties file
editing for shutdown 3-10
purpose 3-6
weblogic.security.certificate.authority property 8-1
weblogic.security.certificate.server property 8-1
weblogic.security.key.server property 8-1
weblogic.system.executeThreadCount property 7-3
weblogic.system.home 3-4
weblogic.system.home property 3-4
weblogic.system.httpdThreadCount property 7-4
weblogic.system.name property 3-6
weblogic.system.SSLListenPort property 8-1