# Table of Contents

**About This Book**
- Purpose of This Document vii
- Audience vii
- How This Document Is Organized viii

## 1 What Are the Cloudscape Tools and Utilities? 1-1

- Overview 1-1
  - JDK and Class Path for Cloudscape Tools 1-2
    - JDK 1-2
    - Class Path 1-3
  - JDBC Basics 1-3
    - Drivers 1-3
    - Database Connection URLs 1-4
    - Protocol 1-5
  - About Cloudscape Databases 1-6

## 2 Using Cloudview 2-1

- What Is Cloudview? 2-1
- Starting Cloudview 2-2
  - Cloudview Log 2-2
  - Cloudview Properties 2-2
  - Cloudview On-Line Help 2-3
  - Cloudview’s Stored Prepared Statements 2-3
  - Cloudview and Cloudscape synchronization Features 2-3
Connecting to a Server 2-3

3 Using ij 3-1
Starting ij 3-1
ij Properties 3-2
Getting Started with ij 3-3
    Connecting to a Cloudscape Database 3-3
    ij and Servers 3-6
Using ij Commands 3-7
    Other Uses for ij 3-7
Running ij Scripts 3-7

4 ij Properties Reference 4-1
    ij.connection.connectionName 4-2
    ij.database 4-3
    ij.driver 4-4
    ij.maximumDisplayWidth 4-5
    ij.outfile 4-6
    ij.protocol 4-7
    ij.protocol.protocolName 4-8
    ij.showErrorCode 4-9

5 ij Commands and Errors Reference 5-1
    ij Commands 5-1
        Conventions for ij Examples 5-2
    ij SQL Command Behavior 5-2
        Absolute 5-4
        After Last 5-5
        Async 5-6
        Autocommit 5-7
        Before First 5-8
        Close 5-9
        Commit 5-10
        Connect 5-11
        Disconnect 5-13
        Driver 5-15
        Elapsedtime 5-16
        Execute 5-17
        Exit 5-19
        First 5-20
        Get Cursor 5-21

Cloudscape Tools and Utilities Guide
6 Using the Bulk Import and Export Utilities

Introduction

Overview

Preparation, Requirements, and Use

Requirements for Fast Mode

How to Use the Utilities

Bulk Import and Export with Cloudview

Bulk Import and Bulk Export with the Stand-Alone Utilities

Arguments to Import and Export Utility Commands

Bulk Import and Export Utility Examples

Bulk Import and Export Within an SQL-J Statement

To Bulk-Import

Arguments to Import Command

Parameter Sets for COM.cloudscape.tools.FileImport Constructor
Parameter Sets for Static Method Import in
*COM.cloudscape.tools.FileImport* 6-8
To Bulk-Export 6-9
*COM.cloudscape.tools.FileExport.Export*
Parameter Sets 6-10
Import Properties 6-10
Bulk Import and Export Within SQL-J Statement Examples 6-12
Default Input or Output File Format 6-13
About the Control File 6-14
Treatment of NULLS 6-14

7 **Using the Database Class Utilities** 7-1
Storing Jar Files in a Database Class Loading 7-1
Working with the Utility 7-2
Complete Examples 7-3
Forcing Database Class Loading for an Application 7-3

8 **sysinfo** 8-1

**Index** IN-1
About This Book

• “Purpose of This Document” on page vii
• “Audience” on page vii
• “How This Document Is Organized” on page viii

For general information about the Cloudscape documentation, such as a complete list of books, conventions, and further reading, see Using the Cloudscape Documentation.

Purpose of This Document

This book describes how to use the Cloudscape tools and utilities. The tools and utilities covered in this book include:

• Cloudview
• ij
• the import and export utilities
• the database class loading utilities

Audience

This book is for:

• developers, who might use the tools when developing applications
• system administrators, who might use the tools to run backup scripts or to import large amounts of data
• end-users, who might use one of the tools to run ad-hoc queries against a database

How This Document Is Organized

This document includes the following chapters:

• Chapter 1, “What Are the Cloudscape Tools and Utilities?”
  Overview of the tools and utilities, and Cloudscape and JDBC basics for new or infrequent users.
• Chapter 2, “Using Cloudview”
  How to get started with Cloudview, a visual database tool.
• Chapter 3, “Using ij”
  How to get started with ij, a JDBC and SQL scripting tool.
• Chapter 4, “ij Properties Reference”
  Reference for ij properties.
• Chapter 5, “ij Commands and Errors Reference”
  Reference for ij commands and errors.
• Chapter 6, “Using the Bulk Import and Export Utilities”
  Reference and how-to instructions for using bulk import and export.
• Chapter 7, “Using the Database Class Utilities”
  Reference information on the utilities that allow you to store classes in and load classes from a database.
1 What Are the Cloudscape Tools and Utilities?

This chapter covers some Java, Cloudscape, and JDBC basics for new users or infrequent users such as those who only use the Cloudscape tools and are not developing applications. For more complete information on these topics, see the Cloudscape Developer’s Guide.

- “Overview” on page 1-1
- “JDBC Basics” on page 1-3
- “About Cloudscape Databases” on page 1-6

Overview

Cloudscape is a DBMS, a database management system, accessed by applications through the JDBC API.

Included with the product you purchased are some standalone Java tools and utilities that make it easier to use Cloudscape and to develop Cloudscape applications.

These tools and utilities include:

- *ij*
  
  *ij* is Cloudscape’s interactive JDBC scripting tool. It is a simple utility for running scripts against a Cloudscape database. You can also use it
What Are the Cloudscape Tools and Utilities?

interactively to run ad hoc queries. ij provides several commands for ease in accessing a variety of JDBC features.

ij can be used in an embedded or a client/server environment.

- **Cloudview**
  Cloudview is a graphical user interface for creating and managing Cloudscape databases. It can be used in an embedded or a client/server environment.

- **The import and export utilities**
  These server-side utilities allow you to import data directly from files into tables and to export data from tables into files. (You can use the utilities in a client/server environment.)

- **Database class utilities**
  These utilities allow you to store application logic in a database and to boot an application using the stored classes.

- **sysinfo**
  sysinfo provides information about your version of Cloudscape and your environment.

**JDK and Class Path for Cloudscape Tools**

ij and Cloudview are tools that can be used in an embedded or a client/server environment. The import and export utilities and database class utilities are database-side utilities, which means that they run in the same JVM as Cloudscape, although the import and export utilities can also be used in a client/server environment. sysinfo can be run in either environment.

**JDK**

- ij is a JDK 1.1 or higher application.
- Cloudview is a JDK 1.2 or higher application.

It is possible to use Cloudview with a 1.1 JDK (1.1.6 or higher). To do so, you must download an extra library (swingall.jar) from the Cloudscape Web site and place it in the class path. You can download this library from http://www.cloudscape.com/support/Downloads/.

For complete information about Cloudview and platforms, see the release notes.

- The import and export utilities are a JDK 1.1 or higher application.
- The class storing and loading utilities are a JDK 1.1.1 or higher application.

*Cloudscape Tools and Utilities Guide*
• sysinfo is a JDK 1.1.1 or higher application.

Class Path

ij and Cloudview use the libraries in tools.jar. To use any of these tools, you need to add tools.jar to your class path. The import and export utilities and the database class utilities are in the primary Cloudscape library for your product, cloudscape.jar or cloudsync.jar. sysinfo is in both tools.jar and the main Cloudscape library.

JDBC Basics

Most of the Cloudscape tools are JDBC applications. A JDBC application is one that uses the classes in the java.sql package to interact with a DBMS. When you work with JDBC applications, you need to know about the following two JDBC concepts:

• Drivers
• Database Connection URLs

Drivers

Before a JDBC application interacts with a database, it must cause the appropriate JDBC driver to be loaded in the Java session. Cloudscape provides three JDBC drivers for use with the Cloudscape database engine. When you use the Cloudscape tools that are JDBC applications, you will need to know which driver to load.

• COM.cloudscape.core.JDBCDriver
  For embedded environments, when Cloudscape runs in the same JVM as the application
• COM.cloudscape.core.WebLogicDriver
  For client/server environments in which Cloudscape runs in the Cloudconnector framework and applications connect via the network
• COM.cloudscape.core.RmiJdbcDriver
  For client/server environments in which Cloudscape runs in the RmiJdbc Server framework and applications connect via the network
**What Are the Cloudscape Tools and Utilities?**

ij is a JDBC-neutral application, which means that you can use it to connect to any database that supplies a JDBC driver. For those databases, you would need to load the supplied JDBC driver.

**NEW:** The `COM.cloudscape.core.RmiJdbcDriver` driver is new in Version 3.0.

Cloudscape also provides an ODBC driver. Look for information on Cloudscape’s Web site.

**Database Connection URLs**

A JDBC URL provides a way of identifying a database so that the appropriate driver recognizes it and connects to it. In the Cloudscape documents, a JDBC URL is referred to as a database connection URL.

After the driver is loaded, an application must specify the correct database connection URL to connect to a specific database. The Cloudscape database connection URL allows you to accomplish more tasks than simply connecting. For more information about the Cloudscape database connection URLs, see the *Cloudscape Developer’s Guide*.

A JDBC URL always starts with `jdbc:`. After that, the format for the database connection URL depends on the JDBC driver.

For the Cloudscape-provided drivers listed above, here are the formats for the database connection URLs for connecting to an existing database:

- `jdbc:cloudscape:databaseName;CloudscapeURLAttributes`
  For embedded environments
- `jdbc:cloudscape:weblogic:databaseName;CloudscapeURLAttributes`
  For connecting to Cloudscape running inside the Cloudconnector framework, when the server is running on the default host `localhost` and is using the default port number 7001
- `jdbc:cloudscape:weblogic-ssl:databaseName;CloudscapeURLAttributes`
  For connecting to Cloudscape running inside the Cloudconnector framework with SSL turned on, when the server is running on the default host `localhost` and is using the default SSL port number 7002
- `jdbc:cloudscape:weblogic://hostname:portnumber/databaseName;CloudscapeURLAttributes`
  For connecting to Cloudscape running inside Cloudconnector when it is not running on `localhost` or the default port number
• jdbc:cloudscape:weblogic-ssl://hostname:portnumber/databaseName;CloudscapeURLAttributes
  For connecting to Cloudscape running inside Cloudconnector with SSL turned on when it is not running on localhost or the default port number

• jdbc:cloudscape:rmi:databaseName;CloudscapeURLAttributes
  for connecting to Cloudscape running inside the RmiJdbc framework when it is running on the default host localhost and issuing the default port number 1099

• jdbc:cloudscape:rmi://hostname:portnumber/databaseName;CloudscapeURLAttributes
  for connecting to Cloudscape running inside the RmiJdbc framework when it is not running on localhost or the default port number

NEW: This format of the database connection URL for RmiJdbc is new in Version 3.0. (The old format also works.)

Here are explanations for the italicized items, which stand for something the user fills in:

• databaseName
  The name of the database you want to connect to

• CloudscapeURLAttributes
  One or more of the supported attributes of the database connection URL
  (for more information, see the Cloudscape Developer’s Guide)

• hostname
  The name of the machine on which the server framework is running

• portnumber
  The port number of the server framework

Protocol

Officially, the portion of the database connection URL called the protocol is jdbc:, just as http:// is a protocol in Web URLs. However, the second portion of the database connection URL (everything between jdbc: and databaseName), which is called the subprotocol, is informally considered part of the protocol. Later in this book you may see references to protocol. Consider protocol to be everything that comes before databaseName.

For complete information about the database connection URL, see the Cloudscape Developer’s Guide.
What Are the Cloudscape Tools and Utilities?

About Cloudscape Databases

A Cloudscape database consists of platform-independent files stored in a directory that has the same name as the database.
This section covers Cloudview. Cloudview is a graphical user interface for creating and managing Cloudscape databases.

This section includes the following topics:

- “What Is Cloudview?” on page 2-1
- “Starting Cloudview” on page 2-2
- “Cloudview Log” on page 2-2

**What Is Cloudview?**

Cloudview is a graphical tool that allows you to perform the following tasks (and more):

- create and access databases
- browse system tables
- define and browse tables
- browse indexes
- browse data (SQL-92 data types and Java data types)
- edit and insert SQL-92 data types
- execute SQL-J statements interactively or from files
- browse the data of an object, and execute methods
- browse synchronization publications
Using Cloudview

Starting Cloudview

Before starting Cloudview, set the class path as described in Chapter 1, “What Are the Cloudscape Tools and Utilities?” If you are using Cloudscape as a database server, start the server before connecting, and specify the JDBC driver as documented in the Cloudscape Server and Administration Guide in the system protocol panel.

To start the Cloudview application, use this command line:

```
java [options] COM.cloudscape.tools.cview [databaseName]*
```

Cloudview uses the protocol information from its last run, or `jdbc:cloudscape:` on its first use, as the default protocol. `java` is the JVM you wish to use to run the application; `options` are options applicable to that JVM; `databaseName` is a list of databases (short names, not full database connection URLs) for Cloudview to connect to.

```
java -Dcloudscape.system.home=d:\todays_build
    COM.cloudscape.tools.cview toursDB
```

If you are using Cloudscape as an embedded database, the application in which it is embedded must not be running. Only one application at a time can access a Cloudscape database. On platforms where Cloudscape cannot prevent this, database corruption can result if you don’t follow that rule.

When you start Cloudview, the Main Screen appears. It lets you specify database connections for a Cloudview session.

Cloudview Log

Cloudview creates a file called `SysVisual.LOG` in the directory from which Cloudview is started. This file contains messages about the Cloudview session that you should provide to Technical Support when reporting any problems you may encounter when using Cloudview.

Cloudview Properties

Cloudview creates a file called `SysVisual.properties` in the directory from which Cloudview is started. It contains state information about which driver to load, what databases were connected, and so on. You can remove this file to restore the default settings. Do not edit this file.
Cloudview On-Line Help

Use the Cloudview on-line help for instructions on using Cloudview.

If Cloudview cannot find your on-line help files, you can add a property specifying the path to the directory in which you installed Cloudscape in the SysVisual.properties file created by Cloudview in the directory in which you started it.

SysVisual.helpLocation=c:/yourlocation

(Use a forward slash).

Cloudview’s Stored Prepared Statements

Cloudview creates and stores stored prepared statements in any databases that you examine with the tool. Cloudview stores these in a schema called SYSVISUAL.

Cloudview and Cloudscape synchronization Features

In order to work with features that are specific to Cloudscape synchronization, you must have cloudsync.jar in your class path before starting the tool.

Connecting to a Server

You can use Cloudview as a client application that interacts with Cloudscape running in a server framework. To do that, simply load the appropriate driver and use the appropriate protocol for the server framework, either RmiJdbc or Cloudconnector.

To see what driver you need for the server framework, see “Drivers” on page 1-3.
To see what protocol you need for the server framework, see “Drivers” on page 1-3.

Here’s an example using the Cloudconnector framework:

1. To use the appropriate protocol, select system icon then select the Connection tab.
2. Select jdbc:cloudscape:weblogic: from the pull-down menu.
3. Fill in the correct host name and port number.
4 Cloudview automatically knows to load the correct driver, as shown in the picture.
Starting ij

If you are using Cloudscape as a database server, start the server before connecting to the Cloudscape database.

To start ij, use this command:

```bash
java [options] COM.cloudscape.tools.ij [-p propertyFile] [inputFile]
```

The command line items are:

- `java`
  The JVM you want to run (`java` is the name of the JVM program in the JDK).
Using ij

- **options**
  Options used by the JVM. You can specify -D options to set ij properties (see “ij Properties” on page 3-2) or system properties such as Cloudscape properties.

- **propertyFile**
  You can set ij properties in a file. The property file should be in the format created by the java.tools.Properties.save methods, which is the same format as that of the cloudscape.properties file. A sample cloudscape.properties file is provided in demo/programs/tours/scripts. A sample ij properties file is not provided.

- **inputFile**
  A file from which to read commands. ij exits at the end of the file or an Exit command. Using an input file causes ij to print out the commands as it runs them. If you reroute standard input, ij does not print out the commands. If you do not supply an input file, ij reads from standard input.

For detailed reference about ij commands, see Chapter 5, “ij Commands and Errors Reference”.

**NOTE:** Cloudscape provides batch and shell scripts for users in Windows and UNIX environments. If you put the appropriate customizable script in your path, you can start ij with a simple command, “ij”. These scripts use the ij.protocol property, which automatically loads a driver and simplifies connecting to a database. These scripts can be found in /bin subdirectories under %CLOUDSCAPE_INSTALL%/frameworks/. You can customize the ij script to suit your environment.

**ij Properties**

You set ij properties using the -D command on the command line or in a properties file that you specify using the -p propertyfile option on the command line.

ij property names are case-sensitive, unlike the ij commands, which are case-insensitive.

Here are some examples of using ij properties:

To start ij using the properties file ij.properties:

```
java COM.cloudscape.tools.ij -p ij.properties
```

To start ij with a maximumDisplayWidth of 1000:
To start ij with an `ij.protocol` of `jdbc:cloudscape:` and an `ij.database` of `toursDB`:

```
java -Dij.protocol=jdbc:cloudscape: -Dij.database=toursDB
COM.cloudscape.tools.ij
```

To start ij with two named connections, using the `ij.connection.connectionName` property:

```
java -Dij.connection.toursDB=jdbc:cloudscape:toursDB
-Dij.connection.History=jdbc:cloudscape:History
-Dcloudscape.system.home=c:/cloudscape/demo/databases
COM.cloudscape.tools.ij
```

ij version 2.0 (c) 1997-1999 Cloudscape, Inc.
ij(HISTORY)> show connections;
HISTORY* - jdbc:cloudscape:History
TOURSDB - jdbc:cloudscape:toursDB
* = current connection
ij(HISTORY)>

---

**Getting Started with ij**

- “Connecting to a Cloudscape Database” on page 3-3
- “ij and Servers” on page 3-6
- “Using ij Commands” on page 3-7
- “Running ij Scripts” on page 3-7

---

**Connecting to a Cloudscape Database**

To connect to a Cloudscape database, you need to perform two steps:

- Load the appropriate driver.
- Provide a database connection URL for the database.

In ij, there are three basic ways of accomplishing these steps:

- **Full database connection URL**
  
  ij can work with any JDBC driver. For drivers supplied by other vendors, you need to load the driver separately. For drivers supplied by Cloudscape, you can load the driver implicitly, by specifying the full database URL.
connection URL in the connection. You do not need to load the driver explicitly in a second step.

To connect, specify the full database connection URL in a Connect command, `ij.connection.connectionName` property, or `ij.database` property.

The database connection URL’s protocol must correspond to one of the three drivers provided by Cloudscape (see “Database Connection URLs” on page 1-4).

For drivers supplied by Cloudscape, specifying a protocol automatically loads the appropriate driver. You do not need to load the driver explicitly in a separate step. You specify a protocol with a property (`ij.protocol` or `ij.protocol.protocolName`) or command (`Protocol`).

To connect, specify the "short form" of the database connection URL in a Connect command, `ij.connection.connectionName` property, or `ij.database` property. A short form of the database connection URL eliminates the protocol (see “Protocol” on page 1-5).
• **Driver and full database connection URL**

You can also load the driver explicitly with an ij property (ij.driver), a system property (jdbc.drivers), or a command (Driver).

If you are using one of the Cloudscape-supplied drivers, use the driver names listed in “Drivers” on page 1-3.

When using ij with another driver, use the appropriate driver.

To connect, specify the full database connection URL in a `Connect` command, `ij.connection.connectionName` property, or `ij.database` property.
**Using ij**

`ij` version 2.0 (c) 1997-1999 Cloudscape, Inc.

```
ij> driver 'sun.jdbc.odbc.JdbcOdbcDriver';
ij> connect 'myOdbcDataSource';
ij>
```

Table 3-1, “Specifying the Driver Name and database connection URL”, summarizes the different ways to specify the driver name and database connection URL.

<table>
<thead>
<tr>
<th>Action</th>
<th>System Property</th>
<th><code>ij</code> Property</th>
<th><code>ij</code> Command</th>
</tr>
</thead>
<tbody>
<tr>
<td>loading the driver implicitly</td>
<td></td>
<td><code>ij.connection.connectionName</code> (plus full URL)</td>
<td><code>ij.database</code> (plus full URL)</td>
</tr>
<tr>
<td></td>
<td></td>
<td><code>ij.database</code> (plus full URL)</td>
<td>Connect</td>
</tr>
<tr>
<td></td>
<td></td>
<td><code>ij.protocol</code></td>
<td>(plus full URL)</td>
</tr>
<tr>
<td></td>
<td></td>
<td><code>ij.protocol.protocolName</code></td>
<td>Connect command</td>
</tr>
<tr>
<td></td>
<td><code>jdbc.drivers</code></td>
<td>-Dij.driver</td>
<td>Driver</td>
</tr>
<tr>
<td>specifying the database connection URL</td>
<td></td>
<td><code>ij.connection.connectionName</code></td>
<td>Connect</td>
</tr>
<tr>
<td></td>
<td></td>
<td><code>ij.database</code></td>
<td></td>
</tr>
</tbody>
</table>

**ij and Servers**

You can use `ij` as a client application that interacts with Cloudscape running in a server framework. To do that, simply load the appropriate driver and use the appropriate protocol for the server framework, either RmiJdbc or Cloudconnector.

To see what driver you need for the server framework, see “Drivers” on page 1-3.

To see what protocol you need for the server framework, see “Database Connection URLs” on page 1-4.

For examples of doing this with either framework, see the previous section, “Connecting to a Cloudscape Database” on page 3-3. This section contains numerous examples of doing that.
Using ij Commands

ij accepts a number of different commands that let you execute SQL statements or run scripts. Each ij statement must end with a semicolon.

For complete information about ij commands, see “ij Commands and Errors Reference” on page 5-1.

Other Uses for ij

ij is a JDBC-neutral scripting tool with a small command set. It can be used to access any JDBC driver and database accessible through that driver.

The main benefit of a tool such as ij is that it is easy to run scripts for creating a database schema and automating other repetitive database tasks.

In addition, ij accepts and processes SQL commands interactively for ad hoc database access.

Running ij Scripts

You can run scripts in ij in a few different ways:

- You name an input file as a command-line argument. For example:
  ```
  java -Djdbc.drivers=COM.cloudscape.core.JDBCDriver
  COM.cloudscape.tools.ij myscript.sql
  ```

- You can redirect standard input to come from a file. For example:
  ```
  java -Djdbc.drivers=COM.cloudscape.core.JDBCDriver
  COM.cloudscape.tools.ij < myscript.sql
  ```

- You can use the Run command from the ij command line. For example:
  ```
  ij> run 'myscript.sql';
  ```

ij echoes input from a file if

- you name an input file as a command-line argument
- you use the Run command

If you redirect standard input to come from a file, ij does not echo commands.

You can save output by redirecting output to a file or by setting the $ij.outfile$ property. (See “$ij.outfile$” on page 4-6.) For example:
Using ij

```
java -Djdbc.drivers=COM.cloudscape.core.JDBCDriver
    COM.cloudscape.tools.ij < myscript.sql > myoutput.txt
java -Dij.outfile=myoutput.txt COM.cloudscape.tools.ij < myscript.sql
```

ij automatically shuts down an embedded Cloudscape system (issues a `connect 'jdbc:cloudscape::;shutdown=true'` request) when the Exit command is issued. It does not shut down Cloudscape running in a server framework.

ij exits when the Exit command is entered or if given a command file on the Java invocation line, when the end of the command file is reached.
When starting up ij, you can specify properties on the command line or in a properties file, as described in “ij Properties” on page 3-2.

- “ij.connection.connectionName” on page 4-2
- “ij.database” on page 4-3
- “ij.driver” on page 4-4
- “ij.maximumDisplayWidth” on page 4-5
- “ij.outfile” on page 4-6
- “ij.protocol” on page 4-7
- “ij.protocol.protocolName” on page 4-8
- “ij.showErrorCode” on page 4-9
ij Properties Reference

ij.connection.connectionName

**Function**

Creates a named connection to the given database connection URL when ij starts up; it is equivalent to the Connect AS Identifier command. The database connection URL can be of the short form if an *ij.protocol* is specified. This property can be specified more than once per session, creating multiple connections. When ij starts, it displays the names of all the connections created in this way. It also displays the name of the current connection, if there is more than one, in the ij prompt.

**Syntax**

```
ij.connection. connectionName = databaseConnectionURL
```

The *databaseConnectionURL* is not a string; do not enclose it in quotation marks.

**Examples**

```
D:> java -Dij.connection.tours1=jdbc:cloudscape:toursDB
-Dij.connection.anotherConn=jdbc:cloudscape:anotherDB;create=true

COM.cloudscape.tools.ij
ij version 2.0 (c) 1997-1999 Cloudscape, Inc.
ANOTHERCONN* = jdbc:cloudscape:anotherDB;create=true
TOURS1 = jdbc:cloudscape:toursDB
* = current connection
ij(ANOTHERCONN)>
```

**See Also**

- “Connect” on page 5-11
**ij.database**

**Function**
Creates a connection with a generated name to the given database connection URL when ij starts up, thus creating an initial connection (with a generated name) for the ij session. (If you have specified an `ij.protocol`, you can use a shortened form of the URL.) After it boots, ij displays the generated name of the connection made with this property.

**Syntax**

```java
ij.database=databaseConnectionURL
```

The `databaseConnectionURL` is not a string; do not enclose it in quotation marks.

**Synonyms**
In releases prior to 1.5, this property was called `database`. The `database` property is still supported and is a synonym.

**Example**

```java
java -Dij.protocol=jdbc:cloudscape: -Dij.connection.tours1=toursDB
    -Dij.connection.anotherConn=anotherDB
    -Dij.database=wombat;create=true COM.cloudscape.tools.ij
```

ij version 2.0 (c) 1997-1999 Cloudscape, Inc.

```
CONNECTION2* -  jdbc:cloudscape:wombat;create=true
ANOTHERCONN -   jdbc:cloudscape:anotherDB
TOURS1 -        jdbc:cloudscape:toursDB
* = current connection
ij(CONNECTION2)>
```
**ij Properties Reference**

**ij.driver**

**Function**

Loads the JDBC driver that the class specifies.

**Syntax**

```
ij.driver=JDBCDriverClassName
```

**Synonyms**

In releases prior to 1.5, this property was called `driver`. The driver property is still supported and is a synonym.

**Examples**

```
D:> java -Dij.driver=sun.jdbc.odbc.JdbcOdbcDriver
COM.cloudscape.tools.ij
ij version 2.0 (c) 1997-1999 Cloudscape, Inc.
ij> Connect ‘jdbc:odbc:MyODBCDataSource’;
ij>
```

**See Also**

- “Driver” on page 5-15
**ij.maximumDisplayWidth**

**Function**
Specifies the maximum number of characters used to display any column. The default value is 128. Values with display widths longer than the maximum are truncated and terminated with an & character.

**Syntax**

```
ij.maximumDisplayWidth=numberOfCharacters
```

**Synonyms**
In releases prior to 1.5, this property was called `maximumDisplayWidth`. The `maximumDisplayWidth` property is still supported and is a synonym.

**Examples**

```
java -Dij.maximumDisplayWidth=1000 COM.cloudscape.tools.ij
```

**See Also**

- “MaximumDisplayWidth” on page 5-27
ij.outfile

Function

Specifies a file to which the system should direct output for a session. Specify the file name relative to the current directory, or specify the absolute path.

Syntax

ij.outfile=fileName

Examples

java -Dij.outfile=out.txt COM.cloudscape.tools.ij myscript.sql
**ij.protocol**

**Function**

Specifies the protocol and subprotocol portions of the database connection URL for connections. Automatically loads the appropriate driver for recognized subprotocols. The recognized protocols are:

- `jdbc:cloudscape:`
- `jdbc:cloudscape:weblogic://hostname:portnum/`
- `jdbc:cloudscape:weblogic-ssl://hostname:portnum/`
- `jdbc:cloudscape:rmi://hostname:portnumber/`
- `jdbc:cloudscape:rmi://hostname:portnumber/`

Allows you to use a short form of a database name.

**Syntax**

```
ij.protocol=protocolForEnvironment
```

**Examples**


ij version 2.0 (c) 1997-1999 Cloudscape, Inc.

ij> Connect 'newDB;create=true';
ij>

**See Also**

- “Protocol” on page 5-31
**ij.protocol.protocolName**

**Function**

This property is similar to the `ij.protocol` property. The only difference is that it associates a name with the value, thus allowing you to define and use more than one protocol. (See “Connect” on page 5-11.)

**Syntax**

```
ij.protocol.protocolName=protocolForEnvironment
```

**Examples**

```
D:> java -Dij.protocol.wlp=jdbc:cloudscape:weblogic:
-Dij.protocol.emp=jdbc:cloudscape:COM.cloudscape.tools.ij
ij version 2.0 (c) 1997-1999 Cloudscape, Inc.
ij> Connect 'newDB' protocol wlp as new;
ij(NEW)>
```

**See Also**

- “Protocol” on page 5-31
**ij.showErrorCode**

**Function**

Set this property to `true` to have ij display the `SQLException Error Code` value with error messages. The default is `false`.

Error codes denote the severity of the error. For more information, see the *Cloudscape Reference Manual*.

**Syntax**

```
ij.showErrorCode=trueOrFalse
```

**Example**

```
java -Dij.showErrorCode=true -Dij.protocol=jdbc:cloudscape:
    COM.cloudscape.tools.ij
ij version 2.0 (c) 1997-1999 Cloudscape, Inc.
ij> Connect 'toursDB';
ij> VLUES 1;
ERROR 42X01: Syntax error: Encountered "VLUES" at line 1, column 1.
  (errorCode = 20000)
ij>
```
ij Commands

ij accepts several commands to control its use of JDBC. It recognizes a semicolon as the end of an ij or SQL command; it treats semicolons within SQL comments, strings, and delimited identifiers as part of those constructs, not as the end of the command. Semicolons are required at the end of an ij or SQL statement.

All ij commands, identifiers, and keywords are case-insensitive.

Commands can span multiple lines without any special escaping for the ends of lines. This means that if a string spans a line, the newlines will show up in the value in the string.

ij treats any command that it does not recognize as an SQL command to be passed to the underlying connection, so syntactic errors in ij commands will cause them to be handed to the SQL engine and will probably result in SQL parsing errors.

The ij commands are:

<table>
<thead>
<tr>
<th>Absolute</th>
<th>After Last</th>
<th>Autocommit</th>
<th>Before First</th>
<th>Close</th>
</tr>
</thead>
<tbody>
<tr>
<td>Commit</td>
<td>Connect</td>
<td>Disconnect</td>
<td>Driver</td>
<td>Elapsedtime</td>
</tr>
<tr>
<td>Execute</td>
<td>Exit</td>
<td>First</td>
<td>Get Cursor</td>
<td>Get Scroll Insensitive Cursor</td>
</tr>
<tr>
<td>Help</td>
<td>Last</td>
<td>MaximumDisplayWidth</td>
<td>Next</td>
<td>Prepare</td>
</tr>
</tbody>
</table>
The ij common elements (used in several commands) are:

- Comment
- Identifier
- String

**Conventions for ij Examples**

Examples in this document show input from the keyboard or a file in bold text and console output from the DOS prompt or the ij application in regular text.

```
C:\> REM This example is from a DOS prompt using JDK1.1.x java:
C:\> java -Dij.protocol=jdbc:cloudscape:COM.cloudscape.tools.ij
ij version 3.0 (c) 1997-1999 Cloudscape, Inc.
ij> connect 'menuDB;create=true';
ij> create table menu(course char(10), item char(20), price integer);
0 rows inserted/updated/deleted
ij> disconnect;
ij> exit;
C:\>
```

**ij SQL Command Behavior**

Any command other than those documented in “ij Command Reference” are handed to the current connection to execute directly. The statement’s closing semicolon, used by ij to determine that it has ended, is not passed to the underlying connection. Only one statement at a time is passed to the connection. If the underlying connection itself accepts semicolon-separated statements (which Cloudscape does not), they can be passed to the connection using ij’s Execute command to pass in a command string containing semicolon-separated commands.

ij uses the result of the JDBC execute request to determine whether it should print a number-of-rows message or display a result set.

If a JDBC execute request causes an exception, it displays the SQLState, if any, and error message.
Setting the ij property `ij.showErrorCode` to `true` displays the `SQLException`'s error code (see Chapter 4, “ij Properties Reference”).

The number-of-rows message for inserts, updates, and deletes conforms to the JDBC specification for any SQL statement that does not have a result set. DDL (data definition language) commands typically report “0 rows inserted/updated/deleted” when they successfully complete.

To display a result set, ij formats a banner based on the JDBC `ResultSetMetaData` information returned from `getColumnLabel` and `getColumnWidth`. Long columns wrap the screen width, using multiple lines. An `&` character denotes truncation (ij limits displayed width of a column to 128 characters by default; see “MaximumDisplayWidth” on page 5-27).

ij displays rows as it fetches them. If the underlying DBMS materializes rows only as they are requested, ij displays a partial result followed by an error message if there is a error in fetching a row partway through the result set.

ij verifies that a connection exists before issuing statements against it and does not execute SQL when no connection has yet been made.

There is no support in ij for the JDBC feature multiple result sets.

**Example**

```
ij> INSERT INTO menu VALUES ('appetizer','baby greens',7);
1 row inserted/updated/deleted
ij> SELECT * FROM menu;
COURSE |ITEM  |PRICE
-----------------------------------------------
etree  |lamb chop |14
dessert |creme brulee |6
appetizer |baby greens |7
3 rows selected
ij>
```
Absolute

Syntax

ABSOLUTE int Identifier

Description

Moves the cursor to the row specified by the int, then fetches the row. The cursor must have been created with the Get Scroll Insensitive Cursor command. It displays a banner and the values of the row.

NOTE: This command works only in a JDK 1.2 environment.

NEW: This command is new in Version 3.0

Example

ij> get scroll insensitive cursor scrollCursor as 'SELECT * FROM menu FOR UPDATE OF price';
ij> absolute 5 scrollCursor;

COURSE | ITEM                |PRICE
-----------------------------------------------
entree | lamb chop           |14
After Last

Syntax

\texttt{AFTER LAST Identifier}

Description

Moves the cursor to after the last row, then fetches the row. (Since there is no such row, it returns the message “no such row.”)

The cursor must have been created with the Get Scroll Insensitive Cursor command.

\textbf{NOTE:} This command works only in a JDK 1.2 environment.

\textbf{NEW:} The After Last command is new in Version 3.0

Example

\begin{verbatim}
ij> get scroll insensitive cursor scrollCursor as
   'SELECT * FROM menu FOR UPDATE OF price';
ij> after last scrollcursor;
No current row
\end{verbatim}
Async

Syntax

ASYNC Identifier String

Description

The Async command lets you execute an SQL statement in a separate thread. It is used in conjunction with the Wait For command to get the results.

You supply the SQL statement, which is any valid SQL statement, as a String. The Identifier you must supply for the async SQL statement is used in the Wait For command and is a case-insensitive ij identifier; it must not be the same as any other identifier for an async statement on the current connection. You cannot reference a statement previously prepared and named by the ij Prepare command in this command.

ij creates a new thread in the current connection to issue the SQL statement. The separate thread is closed once the statement completes.

Example

ij> async aInsert 'INSERT INTO t VALUES 1,2,3,4';
ij> INSERT INTO t VALUES 5,6,7,8;
4 rows inserted/updated/deleted.
ij> wait for aInsert;
4 rows inserted/updated/deleted.
-- the result of the asynchronous insert
**Autocommit**

**Syntax**

```
AUTOCOMMIT { ON | OFF }
```

**Description**

Turns the connection’s auto-commit mode on or off. JDBC specifies that the default auto-commit mode is ON. Certain types of processing require that auto-commit mode be OFF. For information about auto-commit, see the Cloudscape Developer’s Guide.

If auto-commit mode is turned on when there is a transaction outstanding, that work is committed when the current transaction commits, not at the time auto-commit is turned on. Use Commit or Rollback before turning on auto-commit when there is a transaction outstanding, so that all prior work is completed before the return to auto-commit mode.

**Example**

```
ij> autocommit off;
ij> DROP TABLE menu;
0 rows inserted/updated/deleted
ij> CREATE TABLE menu (course CHAR(10), item CHAR(20), price INT);
0 rows inserted/updated/deleted
ij> INSERT INTO menu VALUES ('entree', 'lamb chop', 14),
    ('dessert', 'creme brulee', 6),
    ('appetizer', 'baby greens', 7);
3 rows inserted/updated/deleted
ij> commit;
ij> autocommit on;
ij>
```
Before First

**Syntax**

BEFORE FIRST int Identifier

**Description**

Moves the cursor to before the first row, then fetches the row. (Since there is no such row, it returns the message “no such row.”)

The cursor must have been created with the Get Scroll Insensitive Cursor command.

**NOTE:** This command works only in a JDK 1.2 environment.

**NEW:** The Before First command is new in Version 3.0

**Example**

```
ij> get scroll insensitive cursor scrollCursor as
'SELECT * FROM menu FOR UPDATE OF price';
ij> before first scrollcursor;
No current row
```
Close

Syntax

CLOSE Identifier

Description

Closes the named cursor. The cursor must have previously been successfully created with the ij Get Cursor or Get Scroll Insensitive Cursor commands.

Example

ij> get cursor menuCursor as 'SELECT * FROM menu';
ij> next menuCursor;
COURSE | ITEM | PRICE
-----------------------------------------------
entree | lamb chop | 14
ij> next menuCursor;
COURSE | ITEM | PRICE
-----------------------------------------------
dessert | creme brulee | 6
ij> close menuCursor;
ij>
Commit

Syntax

COMMIT

Description

Issues a java.sql.Connection.commit request. Use this command only if auto-commit is off. A java.sql.Connection.commit request commits the currently active transaction and initiates a new transaction.

Example

ij> commit;
ij>
Connect

Syntax

CONNECT String  [ PROTOCOL Identifier ] [ AS Identifier ] [ USER String  PASSWORD String ]

Description

Takes the value of the string database connection URL and issues a java.sql.DriverManager.getConnection request to set the current connection to that database connection URL.

You have the option of specifying a name for your connection. Use the Set Connection command to switch between connections. If you do not name a connection, the system generates a name automatically.

You also have the option of specifying a named protocol previously created with the Protocol command or the ij.protocol.protocolName property.

If the connection requires a user name and password, supply those with the optional user and password parameters.

NEW: The ability to supply a user name and password as parameters to the Connect command is new in Version 3.0.

If the connect succeeds, the connection becomes the current one and ij displays a new prompt for the Next command. If you have more than one open connection, the name of the connection appears in the prompt.

All further commands are processed against the new connection.

Example

ij> connect 'jdbc:cloudscape:menuDB;create=true';
ij> -- we create a new table in menuDB:
CREATE TABLE menu(course CHAR(10), item CHAR(20), price INTEGER);
0 rows inserted/updated/deleted
ij> protocol 'jdbc:cloudscape:';
ij> connect 'toursDB;autocommit=false' as tours1;
ij> connect 'newDB;create=true' as newDB;
NEWDB> show connections;
TOURS1 -    jdbc:cloudscape:toursDB;autocommit=false
NEWDB* -    jdbc:cloudscape:newDB;create=true
* = current connection
ij(NEWDB)\textgreater{} set connection tours1;
ij(TOURS1)\textgreater{} disconnect all;
ij>
ij> connect 'jdbc:cloudscape:toursDB' user 'sa' password 'cloud3x9';
ij>
Disconnect

Syntax

DISCONNECT [ ALL | CURRENT | Identifier ]

Description

Issues a java.sql.Connection.close request against the current connection. There must be a current connection at the time the request is made.

If ALL is specified, all known connections are closed and there is no current connection.

Disconnect CURRENT is the same as Disconnect.

If a connection name is specified with an identifier, the command disconnects the named connection. The name must be the name of a connection in the current session provided with the ij.connection.connectionName property or with the Connect command.

If the ij.database property or the Connect command without the AS clause was used, you can supply the name the system generated for the connection. If the current connection is the named connection, when the command completes, there will be no current connection and you must issue a Set Connection or Connect command.

A Disconnect command issued against a Cloudscape connection does not shut down the database or Cloudscape (but the Exit command does).

Example

ij> connect 'jdbc:cloudscape:menuDB;create=true';
ij> -- we create a new table in menuDB:
CREATE TABLE menu(course CHAR(10), ITEM char(20), PRICE integer);
0 rows inserted/updated/deleted
ij> disconnect;

ij> protocol 'jdbc:cloudscape:';
ij> connect 'toursDB;autocommit=false' as tours1;
ij> connect 'newDB;create=true' as newDB;
TOURS1 - jdbc:cloudscape:toursDB;autocommit=false
NEWDB* - jdbc:cloudscape:newDB;create=true
* = current connection

```java
ij(NEWDB)> set connection tours1;
ij(TOURS1)> disconnect all;
ij>
```

```java
java -Dij.connection.toursDB=jdbc:cloudscape:toursDB
-Dij.connection.History=jdbc:cloudscape:History
COM.cloudscape.tools.ij
ij version 2.0 (c) 1997-1999 Cloudscape, Inc.
HISTORY* - jdbc:cloudscape:History
TOURSDB - jdbc:cloudscape:toursDB
* = current connection
ij(HISTORY)> disconnect toursDB;
ij> show connections;
HISTORY* - jdbc:cloudscape:History
* = current connection
ij>
```
Driver

Syntax

Driver String

Description

Takes the value of the string and issues a Class.forName request to load the named class. The class is expected to be a JDBC driver that registers itself with java.sql.DriverManager.

If the Driver command succeeds, a new ij prompt appears for the next command.

Example

ij> -- load the Cloudscape driver so that a connection
-- can be made:
driver 'COM.cloudscape.core.JDBCDriver';
ij> connect 'jdbc:cloudscape:menuDB;create=true';
ij>
Elapsed time

Syntax
ELAPSEDTIME { ON | OFF }

Description
When elapsed time is turned on, ij displays the total time elapsed during statement execution. The default value is OFF.

Example
ij> elapsedtime on;
ij> VALUES current_date;
SQLCol1
----------
1998-07-15
ELAPSED TIME = 2134 milliseconds
ij>
Execute

Syntax
EXECUTE { String | Identifier }
[ USING { String | Identifier } ]

Description
Has several uses:

- To execute an SQL command that has the same name as an ij command, using the Execute String style. The String is passed to the connection without further examination or processing by ij. Normally, you execute SQL commands directly, not with the Execute command.

- To execute a named command previously prepared with the ij Prepare command, using the Execute Identifier style.

- To execute either flavor of command when that command contains dynamic parameters, taking values from the Using portion of the command. In this style, the Using portion’s String or previously prepared Identifier is executed, and it must have a result set as its result. Each row of the result set is applied to the input parameters of the command to be executed, so the number of columns in the Using’s result set must match the number of input parameters in the Execute’s statement. The results of each execution of the Execute statement are displayed as they are made. If the Using’s result set contains no rows, the Execute’s statement is not executed.

When auto-commit mode is on, the Using’s result set is closed upon the first execution of the Execute statement. To ensure multiple-row execution of the Execute command, use the Autocommit command to turn auto-commit off.

Example
ij> autocommit off;
ij> prepare menuInsert as 'INSERT INTO menu VALUES (?, ?, ?)';
ij> execute menuInsert using 'VALUES (''entree'', ''lamb chop'', 14),
                   (''dessert'', ''creme brulee'', 6)';
1 row inserted/updated/deleted
1 row inserted/updated/deleted

Cloudscape Version 3.0
ij> commit;
Exit

Syntax
EXIT

Description
Causes the ij application to complete and processing to halt. Issuing this command from within a file started with the Run command or on the command line causes the outermost input loop to halt.

ij automatically shuts down a Cloudscape database running in an embedded environment (issues a Connect 'jdbc:cloudscape;shutdown=true’ request) on exit.

ij exits when the Exit command is entered or if given a command file on the Java invocation line, when the end of the command file is reached.

Example
ij> disconnect;
ij> exit;
C:\\>
First

Syntax
FIRST Identifier

Description
Moves the cursor to the first row in the ResultSet, then fetches the row. The cursor must have been created with the Get Scroll Insensitive Cursor command. It displays a banner and the values of the row.

NOTE: This command works only in a JDK 1.2 environment.
NEW: The First command is new in Version 3.0

Example
ij> get scroll insensitive cursor scrollCursor as
'SELECT * FROM menu FOR UPDATE OF price';
ij> first scrollcursor;

<table>
<thead>
<tr>
<th>COURSE</th>
<th>ITEM</th>
<th>PRICE</th>
</tr>
</thead>
<tbody>
<tr>
<td>entree</td>
<td>lamb chop</td>
<td>14</td>
</tr>
</tbody>
</table>
Get Cursor

Syntax

GET CURSOR Identifier AS String

Description

Creates a cursor with the name of the Identifier by issuing a java.sql.Statement.executeQuery request on the value of the String.

If the String is a statement that does not generate a result set, the behavior of the underlying database determines whether an empty result set or an error is issued. If there is an error in executing the statement, no cursor is created.

ij sets the cursor name using a java.sql.Statement.setCursorName request. Behavior with respect to duplicate cursor names is controlled by the underlying database. Cloudscape does not allow multiple open cursors with the same name.

Once a cursor has been created, the ij Next and Close commands can be used to step through its rows, and if the connection supports positioned update and delete commands, they can be issued to alter the rows.

Example

ij> -- autocommit needs to be off so that the positioned update
ij> -- can see the cursor it operates against.
ij> autocommit off;
ij> get cursor menuCursor as
'SELECT * FROM menu FOR UPDATE OF price';
ij> next menuCursor;
COURSE | ITEM  | PRICE
-----------------------------------------------
entree | lamb chop | 14
ij> next menuCursor;
COURSE | ITEM | PRICE
-----------------------------------------------
dessert | creme brulee | 6
ij> UPDATE SET price=price+1 WHERE CURRENT OF menuCursor;
1 row inserted/updated/deleted
ij> next menuCursor;
COURSE | ITEM | PRICE
-----------------------------------------------
appetizer | baby greens salad | 7
ij> close menuCursor;
ij> commit;
ij>
Get Scroll Insensitive Cursor

Syntax

GET SCROLL INSENSITIVE CURSOR Identifier AS String

Description

Creates a scroll insensitive cursor with the name of the Identifier. (It does this by issuing a a createStatement(ResultSet.TYPE_SCROLL_INSENSITIVE, ResultSet.CONCUR_READ_ONLY) call and then executing the statement with java.sql.Statement.executeQuery request on the value of the String.)

NEW: This command only works in a JDK 1.2 environment.

Scroll insensitive cursors are not updatable.

If the String is a statement that does not generate a result set, the behavior of the underlying database determines whether an empty result set or an error is issued. If there is an error in executing the statement, no cursor is created.

ij sets the cursor name using a java.sql.Statement.setCursorName request. Behavior with respect to duplicate cursor names is controlled by the underlying database. Cloudscape does not allow multiple open cursors with the same name.

Once a scrolling cursor has been created, you can use the follow commands to work with the result set:

• Absolute
• After Last
• Before First
• Close
• First
• Last
• Next
• Previous
• Relative

NEW: The Get Scroll Insensitive Cursor command is new in Version 3.0.

Example

\$> autocommit off;
ij Commands and Errors Reference

ij> get scroll insensitive cursor scrollCursor as
   'SELECT * FROM menu FOR UPDATE OF price';
ij> absolute 5 scrollCursor;

<table>
<thead>
<tr>
<th>COURSE</th>
<th>ITEM</th>
<th>PRICE</th>
</tr>
</thead>
<tbody>
<tr>
<td>entree</td>
<td>lamb chop</td>
<td>14</td>
</tr>
</tbody>
</table>

ij> after last scrollcursor;
No current row

ij> before first scrollcursor;
No current row

ij> first scrollcursor;

<table>
<thead>
<tr>
<th>COURSE</th>
<th>ITEM</th>
<th>PRICE</th>
</tr>
</thead>
<tbody>
<tr>
<td>entree</td>
<td>lamb chop</td>
<td>14</td>
</tr>
</tbody>
</table>

ij> last scrollcursor;

<table>
<thead>
<tr>
<th>COURSE</th>
<th>ITEM</th>
<th>PRICE</th>
</tr>
</thead>
<tbody>
<tr>
<td>dessert</td>
<td>creme brulee</td>
<td>6</td>
</tr>
</tbody>
</table>

ij> previous scrollcursor;

<table>
<thead>
<tr>
<th>COURSE</th>
<th>ITEM</th>
<th>PRICE</th>
</tr>
</thead>
<tbody>
<tr>
<td>entree</td>
<td>lamb chop</td>
<td>14</td>
</tr>
</tbody>
</table>

ij> relative 1 scrollcursor;

<table>
<thead>
<tr>
<th>COURSE</th>
<th>ITEM</th>
<th>PRICE</th>
</tr>
</thead>
<tbody>
<tr>
<td>dessert</td>
<td>creme brulee</td>
<td>6</td>
</tr>
</tbody>
</table>

ij> previous scrollcursor;

<table>
<thead>
<tr>
<th>COURSE</th>
<th>ITEM</th>
<th>PRICE</th>
</tr>
</thead>
<tbody>
<tr>
<td>dessert</td>
<td>creme brulee</td>
<td>6</td>
</tr>
</tbody>
</table>

ij> next scrollcursor;

<table>
<thead>
<tr>
<th>COURSE</th>
<th>ITEM</th>
<th>PRICE</th>
</tr>
</thead>
<tbody>
<tr>
<td>dessert</td>
<td>creme brulee</td>
<td>6</td>
</tr>
</tbody>
</table>
Help

Syntax
HELP

Description
Prints out a brief list of the ij commands.
Last

Syntax

LAST Identifier

Description

Moves the cursor to the last row in the ResultSet, then fetches the row. The cursor must have been created with the Get Scroll Insensitive Cursor command. It displays a banner and the values of the row.

**NOTE:** This command works only in a JDK 1.2 environment.

**NEW:** This command is new in Version 3.0.

Example

```java
ij> get scroll insensitive cursor scrollCursor as 'SELECT * FROM menu FOR UPDATE OF price';
ij> absolute 5 scrollCursor;
COURSE | ITEM                | PRICE
-----------------------------------------------
entree  | lamb chop           | 14
ij> last scrollCursor;
COURSE | ITEM                | PRICE
-----------------------------------------------
dessert | creme brulee        | 6
```
MaximumDisplayWidth

Syntax

MAXIMUMDISPLAYWIDTH integer_value

Description

Sets the display width for column to the specified value.

Examples

ij> maximumdisplaywidth 3;
ij> VALUES ‘NOW IS THE TIME!’;
SQL&
---
NO&
ij> maximumdisplaywidth 30;
ij> VALUES ‘NOW IS THE TIME!’;
SQLCol1
-------------
NOW IS THE TIME!
Next

Syntax
NEXT Identifier

Description
Fetches the next row from the named cursor created with the Get Cursor command or Get Scroll Insensitive Cursor. It displays a banner and the values of the row.

Example
ij> get cursor menuCursor as ‘SELECT * FROM menu’;
ij> next menuCursor;
COURSE |ITEM  |PRICE
-----------------------------------------------
entree |lamb chop  |14
ij>
Prepare

Syntax

PREPARE Identifier AS String

Description

Creates a java.sql.PreparedStatement using the value of the String, accessible in ij by the Identifier given to it. If a prepared statement with that name already exists in ij, an error will be returned and the previous prepared statement will remain. Use the Remove command to remove the previous statement first. If there are any errors in preparing the statement, no prepared statement is created.

Any SQL statements allowed in the underlying connection’s prepared statement can be prepared with this command.

Example

ij> prepare seeMenu as 'SELECT * FROM menu';
ij> execute seeMenu;

<table>
<thead>
<tr>
<th>COURSE</th>
<th>ITEM</th>
<th>PRICE</th>
</tr>
</thead>
<tbody>
<tr>
<td>entree</td>
<td>lamb chop</td>
<td>14</td>
</tr>
<tr>
<td>dessert</td>
<td>creme brulee</td>
<td>6</td>
</tr>
</tbody>
</table>

2 rows selected
ij>
**Syntax**

`PREVIOUS Identifier`

**Description**

Moves the cursor to the row previous to the current one, then fetches the row. The cursor must have been created with the Get Scroll Insensitive Cursor command. It displays a banner and the values of the row.

**NOTE:** This command works only in a JDK 1.2 environment.

**NEW:** This command is new in Version 3.0.

**Example**

```
ij> get scroll insensitive cursor scrollCursor as
   'SELECT * FROM menu FOR UPDATE OF price';
ij> last scrollcursor;
 COURSE | ITEM                | PRICE
---------|---------------------|--------
dessert  | creme brulee        | 6
ij> previous scrollcursor;
 COURSE | ITEM                | PRICE
---------|---------------------|--------
entree   | lamb chop           | 14
```
Protocol

Syntax

```
PROTOCOL String [ AS Identifier ]
```

Description

Specifies the protocol, as a String, for establishing connections and automatically loads the appropriate driver. *Protocol* is the part of the database connection URL syntax appropriate for your environment, including the JDBC protocol and the Cloudscape-specific protocol. For further information about the Cloudscape database connection URL, see the *Cloudscape Developer’s Guide* (embedded environment) or the *Cloudscape Server and Administration Guide* (client/server environment) for complete details. Only Cloudscape protocols are supported. Those protocols are listed in “Protocol” on page 1-5.

Providing a protocol allows you to use a shortened database connection URL for connections. You can provide only the database name instead of the full protocol. In addition, you do not need to use the Driver command or specify a driver at start-up, since the driver is loaded automatically.

If you name the protocol, you can refer to the protocol name in the Connect command.

Example

```
i=j> protocol ‘jdbc:cloudscape:’;
i=j> connect ‘toursDB;autocommit=false’;
i=j> -- another example, this one showing the use of a name
i=j> protocol ‘jdbc:cloudscape:weblogic:’ as wlp;
i=j> connect ‘toursDB’ protocol wlp;
```
ReadOnly

Syntax

`READONLY { ON | OFF }`

Description

Sets the current connection to a “read-only” connection, as if the current user were defined as a `readOnlyAccess` user. (For more information about database authorization, see the Cloudscape Developer’s Guide.)

**NEW:** This command is new in Version 3.0.

Example

```sql
ij> connect 'jdbc:cloudscape:toursDB';
ij> readonly on;
ij> SELECT * FROM menu;
```

```plaintext
<table>
<thead>
<tr>
<th>COURSE</th>
<th>ITEM</th>
<th>PRICE</th>
</tr>
</thead>
<tbody>
<tr>
<td>entree</td>
<td>lamb chop</td>
<td>14</td>
</tr>
<tr>
<td>dessert</td>
<td>creme brulee</td>
<td>6</td>
</tr>
<tr>
<td>appetizer</td>
<td>baby greens</td>
<td>7</td>
</tr>
<tr>
<td>entree</td>
<td>lamb chop</td>
<td>14</td>
</tr>
<tr>
<td>entree</td>
<td>lamb chop</td>
<td>14</td>
</tr>
<tr>
<td>dessert</td>
<td>creme brulee</td>
<td>6</td>
</tr>
</tbody>
</table>
```

6 rows selected

```sql
ij> UPDATE menu set price = 3;
ERROR 25502: An SQL data change is not permitted for a read-only connection, user or database.
```
Relative

Syntax

RELATIVE int Identifier

Description

Moves the cursor to the row that is \textit{int} number of rows relative to the current row, then fetches the row. The cursor must have been created with the Get Scroll Insensitive Cursor command. It displays a banner and the values of the row.

\textbf{NOTE:} This command works only in a JDK 1.2 environment.

\textbf{NEW:} This command is new in Version 3.0.

Example

\begin{verbatim}
ij> -- autocommit needs to be off so that the positioned update
ij> -- can see the cursor it operates against.
ij> autocommit off;
ij> get scroll insensitive cursor scrollCursor as
   'SELECT * FROM menu FOR UPDATE OF price';
ij> last scrollcursor;
   COURSE |ITEM                |PRICE
   -------------------------------
dessert |creme brulee        |6
ij> previous scrollcursor;
   COURSE |ITEM                |PRICE
   -------------------------------
entree  |lamb chop           |14
ij> relative 1 scrollcursor;
   COURSE |ITEM                |PRICE
   -------------------------------
dessert |creme brulee        |6
\end{verbatim}
Remove

Syntax

\texttt{REMOVE Identifier}

Description

Removes a previously prepared statement from \textit{ij}. The identifier is the name by which the statement was prepared. The statement is closed to release its database resources.

Example

\begin{verbatim}
ij> prepare seeMenu as 'SELECT * FROM menu';
ij> execute seeMenu;

\begin{tabular}{lll}
\hline
COURSE & ITEM & PRICE \\
\hline
entree & lamb chop & 14 \\
dessert & creme brulee & 6 \\
\hline
\end{tabular}

2 rows selected
ij> remove seeMenu;
ij> execute seeMenu;

IJ ERROR: Unable to establish prepared statement SEEMENU
ij>
\end{verbatim}
Rollback

Syntax

ROLLBACK

Description

Issues a `java.sql.Connection.rollback` request. Use only if auto-commit is off. A `java.sql.Connection.rollback` request undoes the currently active transaction and initiates a new transaction.

Example

```sql
ij> INSERT INTO menu VALUES ('dessert', 'rhubarb pie', 4);
1 row inserted/updated/deleted
ij> SELECT * from menu;
COURSE    |ITEM                |PRICE
-----------------------------------------------
entree    |lamb chop           |14
dessert   |creme brulee        |7
appetizer |baby greens         |7
dessert   |rhubarb pie         |4
4 rows selected
ij> rollback;
ij> SELECT * FROM menu;
COURSE    |ITEM                |PRICE
-----------------------------------------------
entree    |lamb chop           |14
dessert   |creme brulee        |7
appetizer |baby greens         |7
3 rows selected
```
**Run**

**Syntax**

```
RUN String
```

**Description**

Assumes that the value of the string is a valid file name, and redirects ij processing to read from that file until it ends or an Exit command is executed. If the end of the file is reached without ij exiting, reading will continue from the previous input source once the end of the file is reached. Files can contain Run commands.

ij prints out the statements in the file as it executes them.

Any changes made to the ij environment by the file are visible in the environment when processing resumes.

**Example**

```
ij> run 'setupMenuConn.ij';
ij> -- this is setupMenuConn.ij
-- ij displays its contents as it processes file
ij> connect 'jdbc:cloudscape:menuDB';
ij> autocommit off;
ij> -- this is the end of setupMenuConn.ij
-- there is now a connection to menuDB and no autocommit.
-- input will now resume from the previous source.
;
ij>
```
Set Connection

Syntax

**SET CONNECTION Identifier**

Description

Allows you to specify which connection to make current when you have more than one connection open. Use the Show Connections command to display open connections.

If there is no such connection, an error results and the current connection is unchanged.

Example

```
ij> protocol 'jdbc:cloudscape:';
ij> connect 'toursDB;autocommit=false' as tours1;
ij> connect 'newDB;create=true' as newDB;
ij (NEWDB)> . show connections;
TOURS1 - jdbc:cloudscape:toursDB;autocommit=false
NEWDB* - jdbc:cloudscape:newDB;create=true
* = current connection
ij(NEWDB)> set connection tours1;
ij(TOURS1)> disconnect all;
ij>
```
Show Connections

**Syntax**

SHOW CONNECTIONS

**Description**

If there are no connections, returns “No connections available.”

Otherwise, displays a list of connection names and the URLs used to connect to them. The currently active connection, if there is one, is marked with an * after its name.

**Example**

```
ij> connect 'toursDB;autocommit=false' as tours1;
ij> connect 'newDB;create=true' as newDB;
ij(NEWDB)> show connections;
TOURS1  -      jdbc:cloudscape:toursDB;autocommit=false
NEWDB*  -      jdbc:cloudscape:newDB;create=true
* = current connection
ij(NEWDB)>
```
Wait For

Syntax

WAIT FOR Identifier

Description

Displays the results of a previously started asynchronous command.

The identifier for the asynchronous command must have been used in a previous Async command on this connection. Wait For waits for the SQL statement to complete execution, if it has not already, and then displays the results. If the statement returns a result set, the Wait For command steps through the rows, not the Async command. This may result in further execution time passing during the result display.

Example

See “Async” on page 5-6.
Comment

Syntax

-- Text

Description

You can use a comment anywhere within an ij command and as permitted by the underlying connection within SQL commands. The comment is ended at the first new line encountered in the text.

Comments are ignored on input and have no effect on the output displayed.

Example

ij> -- this is a comment;
-- the semicolons in the comment are not taken as the end
-- of the command; for that, we put it outside the --:
;
ij>
Some ij commands require identifiers. These ij identifiers are case-insensitive. They must begin with a letter in the range A–Z, and can consist of any number of letters in the range A–Z, digits in the range 0–9, and underscore (_) characters.

These identifiers exist within the scope of ij only and are distinct from any identifiers used in SQL commands, except in the case of the Get Cursor command. The Get Cursor command specifies a cursor name to use in creating a result set.

ij does not recognize or permit delimited identifiers in ij commands. They can be used in SQL commands.

**Example**

These are valid ij identifiers:

- fool
- exampleIdentifier12345
- another_one
String

Syntax

' Text '

Description

Some ij commands require strings. ij strings are represented by the same literal format as SQL strings and are delimited by single quotation marks. To include a single quotation mark in a string, you must use two single quotation marks, as shown in the examples below. ij places no limitation on the lengths of strings, and will treat embedded new lines in the string as characters in the string.

Some ij commands execute SQL commands specified as strings. Therefore, you must double any single quotation marks within such strings, as shown in the second example below.

The cases of letters within a string are preserved.

Example

This is a string in ij     And this is its value
'Mary’s umbrella'        Mary’s umbrella
'hello world'             hello world

--returns Joe’s
execute 'VALUES ''Joe''''s''';
ij Errors

ij issues the following messages to inform the user of errors during processing of statements:

- ERROR SQLState
- WARNING SQLState
- IJ ERROR
- IJ WARNING
- JAVA ERROR

**ERROR SQLState**

When the underlying JDBC driver returns an `SQLException`, ij displays the `SQLException` message with the prefix “ERROR SQLState”. If the `SQLException` has no SQLState associated with it, the prefix “ERROR (no SQLState)” is used.

**WARNING SQLState**

Upon completion of execution of any JDBC request, ij will issue a `getWarnings` request and display the SQLWarnings that are returned. Each `SQLWarning` message is displayed with the prefix “WARNING SQLState”. If an `SQLWarning` has no SQLState associated with it, the prefix “WARNING (no SQLState)” is used.

**IJ ERROR**

When ij runs into errors processing user commands, such as being unable to open the file named in a Run command or not having a connection to disconnect from, it prints out a message with the prefix “IJ ERROR”.

**IJ WARNING**

ij displays warning messages to let the user know if behavior may be unexpected. For example, using the Execute command’s USING clause while in auto-commit mode can cause unexpected results. ij warnings are prefixed with “IJ WARNING”.
JAVA ERROR

When an unexpected Java exception occurs, ij prints a message with the prefix “JAVA ERROR”.
Using the Bulk Import and Export Utilities

Often developers or system administrators want to import a large amount of data from files or export a large amount of data to files. Instead of having to use INSERT and SELECT statements, you can use utilities that Cloudscape provides to import data directly from files into tables and to export data from tables into files.

Bulk import is available in slow and fast modes. In slow mode, bulk import does not work much faster than a series of insert statements, but it provides the convenience of allowing you to work with files that are produced or read by other applications, such as spreadsheets or other desktop databases.

In fast mode, bulk import works much faster than a series of insert statements because Cloudscape minimizes logging and makes some under-the-covers performance enhancements. Fast mode is available in specific circumstances; see “Requirements for Fast Mode” on page 6-3.

Both slow- and fast-mode bulk import are transactional. If an error occurs during bulk import, all the changes made are rolled back.

**NEW:** In Version 3.0, you can import data into a table to replace existing data in fast mode.

- “Introduction” on page 6-1
- “How to Use the Utilities” on page 6-4

**Introduction**

- “Overview” on page 6-2
Overview

Cloudscape provides two classes in the \texttt{COM.cloudscape.tools} package that allow you to import data from flat ASCII files and export data to flat ASCII files. These 100\%-Java utilities are most easily run from Cloudview. You can also run them on the command line or invoke them from another Java application. When run in any of these ways, the utility by default tries to import data in fast mode.

The design of these utilities also allows you to “run” them within an SQL-J statement using \texttt{ij} or any Java application. Here’s how that is possible: The class \texttt{COM.cloudscape.tools.FileImport} fulfills the Virtual Table Interface (VTI) requirements. This class opens an ASCII text file and presents the content of the file to Cloudscape as a virtual external table that can be accessed within the FROM clause of a SELECT statement. \texttt{COM.cloudscape.tools.FileExport} exports the result set from a query to an ASCII text file. When run in this way, the utility runs by default in slow mode, unless you explicitly set a property.

Import/Export reads and writes only ASCII text files. If you export non-ASCII data, only the first byte of each character is written to the file. If you import non-ASCII data, each byte is read as a character. In addition, Import does not support read-once streams (live data feeds), because it reads the first line of the file to determine the number of columns, then reads it again to import the data.

\textbf{NOTE:} These server-side utilities exhibit different behavior in client/server mode. Typically, you use them to import data into and export data from a locally running Cloudscape. However, you can use the language-based version of the commands when Cloudscape is running in a server framework if you specify import and export files that are accessible to the server.

Preparation, Requirements, and Use

- \textit{Table must exist.}
  
  For you to import data into a table, the table must already exist in Cloudscape. The table does not have to be empty. If the table is not empty, bulk import runs in slow mode, unless you explicitly replace existing data with the new data.
• Create indexes and primary key, foreign key, and unique constraints first. Create the table’s indexes and primary keys first (before importing data) to avoid a separate create index step. However, if your memory and disk spaces resources are very limited, build the indexes and primary keys after importing data.

• Disable check constraints.
  You can disable check constraints with the SET CONSTRAINTS command. Re-enable constraints after the data are loaded. When replacing existing data, you may want to disable circular foreign key constraints.

• Format of data file.
  The data must be in a flat ASCII text file. There is a default file format (see “Default Input or Output File Format” on page 6-13). If the format of the file is different, you can use Cloudview to help you create a control file, which describes the alternate format of the file to the import and export utilities.

• Data types.
  You can import and export only data of the non-binary, built-in data types. For example, you can export objects of type INTEGER but not byte arrays or objects of type \texttt{JBMSTours.serializables.City}.
  \texttt{COM.cloudscape.tools.FileImport} sends all valid data types to Cloudscape as strings. Cloudscape implicitly converts the strings to the data type of the receiving column. If any of the implicit conversions fail, the whole import is aborted. For example, ‘3+7’ cannot be converted into an integer. An export that encounters a runtime error stops.

  \textbf{NOTE:} You cannot import or export the binary data types: BIT, BIT VARYING, LONG VARBINARY, LONG BIT VARYING, or LONG BINARY.

• Table locking.
  During import, the entire table is exclusively locked. During export, the entire table is locked with a read lock.

\section*{Requirements for Fast Mode}

Bulk import has a fast mode and a slow mode. To make a bulk-import operation run in fast mode, the import must fulfill these requirements:

• The table must not be in a target database.
• The table must not be a published table in a source database if you are replacing existing data.
Using the Bulk Import and Export Utilities

- From within the context of an SQL-J statement, you must explicitly specify fast import mode (insertMode=bulkInsert or insertMode=replace). See “Import Properties” on page 6-10.

- For non-published non-empty tables, you must explicitly specify that you are replacing existing data. In Cloudview, you select a checkbox; otherwise, you set the property insertMode to replace. See “Import Properties” on page 6-10. Otherwise, slow import mode is used.

NOTE: When using fast import mode and replacing existing data in a table that has a primary or unique constraint referenced by foreign keys in other tables, any foreign key constraints that have not been turned off are checked when Cloudscape completes the operation. If any of the foreign key constraints are not satisfied, an exception is thrown, and the statement is rolled back. Any constraints which have been turned off are not checked until the user explicitly turns them back on.

NEW: The ability to use fast mode for tables in a source database is new in Version 3.0.

The fast and slow import modes are both transactional; if the operations fail, all the inserts are rolled back. Fast import mode uses fewer resources.

How to Use the Utilities

Bulk Import and Export with Cloudview

To bulk-import data into a table with Cloudview, select the table, then choose File->Import.

To bulk-export data from a table with Cloudview, select the table, then choose File->Export.

To bulk-import and replace existing data with Cloudview, select the table, then choose File-Import. Select Replace Existing Data on the advanced tab.

Cloudview also allows you to create a control file to control the format of the imported and exported data.

NOTE: Cloudview’s on-line help has more details.
Bulk Import and Bulk Export with the Stand-Alone Utilities

The syntax for the standalone bulk-import utility is:

```
java [-DimportProperties ] COM.cloudscape.tools.FileImport
databaseConnectionURL tableName inputOrExportFileURL
[controlFileURL]
```

The syntax for the standalone bulk-export utility is:

```
java COM.cloudscape.tools.FileExport databaseConnectionURL
tableOrViewName inputOrExportFileURL [controlFileURL]
```

**NOTE:** You can set system properties such as `cloudscape.system.home` on the command line using parameters to the `java` command.

Arguments to Import and Export Utility Commands

- `databaseConnectionURL`
  The Cloudscape database connection URL (string) for the database containing the table.

- `tableName`
  The name of the table into which you want to bulk-import the data. Qualify the table name with the schema name if it is not in the default schema `APP`.

- `tableOrViewName`
  The name of the table or view from which you want to bulk-export the data. Qualify the table name with the schema name if it is not in the default schema `APP`.

- `inputOrExportFileURL`
  The path or the URL of the ASCII file from which to bulk-import or bulk-export the data. If a `controlFileURL` is not specified, the import or export file must be of the default file format (see “Default Input or Output File Format” on page 6-13). If a `controlFileURL` is specified, it must be of the input format described by the control file. The path separator in the path or the URL is `/` (forward slash), per the standard `file://` URL protocol. Absolute or relative paths are accepted. For local files, do not use the `file://` protocol; omit the protocol altogether. For export, only the `http://` protocol is not allowed. Import supports `http://`.

- `controlFileURL`
  The path or URL of the control file specifying the file format of the ASCII file for the bulk import or export.
Using the Bulk Import and Export Utilities

- `importProperties`
  The properties for the bulk load. See “Import Properties” on page 6-10. By default, the utility sets the property `insertMode` to `bulkInsert`. This property is ignored if the requirements for fast mode are not fulfilled. See “Requirements for Fast Mode” on page 6-3. (Where appropriate, you can also specify other properties such as `cloudscape.system.home`.)

**Bulk Import and Export Utility Examples**

**NOTE:** Type commands on a single line, with no carriage returns.

```java
java COM.cloudscape.tools.FileImport jdbc:cloudscape:toursDB
  APP.myints c:/mydata/mydata.dat
```
imports `c:\mydata\mydata.dat` into the table `myints` in the schema `APP` in the database `toursDB`.

```java
java -DinsertMode=replace COM.cloudscape.tools.FileImport
  jdbc:cloudscape:wombat APP.abc ../mydata/mydata2.dat
  ../mydata/mydata.ctl
```
Imports `mydata2.dat` into the table in `abc` in the schema `APP`, replacing existing data and running in fast mode in the database `wombat` using the control file `c:\data\myData.ctl`.

```java
java -Dcloudscape.system.home=c:\mydatabases
  COM.cloudscape.tools.FileImport
  jdbc:cloudscape:toursDB Flights
  http://airlines.com/data/flight.info
```
Imports `flight.info` via HTTP into the table `Flights` in the database `toursDB`, which is located in the specified system directory.

```java
java COM.cloudscape.tools.FileExport jdbc:cloudscape:wombat
  APP.abc c://mydata/mydata3.dat
```
Exports data from table or view `abc` in database `wombat` to file `c:\mydata\mydata3.dat`, using the default format (since no control file is specified).

```java
java COM.cloudscape.tools.FileExport jdbc:cloudscape:wombat
  APP.abc c://mydata/mydata4.dat c:/mydata/mydata.ctl
```
Exports from table or view `abc` in database `wombat` to file `mydata4.dat` using the file format described in `mydata.ctl`.

*Cloudscape Tools and Utilities Guide*
Bulk Import and Export Within an SQL-J Statement

You can use the bulk import and bulk export utilities within an SQL-J statement, which is executable from ij, Cloudview, or any Java program.

- “To Bulk-Import” on page 6-7
- “To Bulk-Export” on page 6-9
- “Import Properties” on page 6-10
- “Bulk Import and Export Within SQL-J Statement Examples” on page 6-12

To Bulk-Import

There are two ways to bulk-import within an SQL-J statement:

- Create a new instance of the `COM.cloudscape.tools.FileImport` class (aliased as `FileImport`) in the FROM clause of an insert. Such an instance constitutes an ExternalVirtualTable (described in the Cloudscape Reference Manual) that is allowed in a FROM clause instead of a tableExpression.

  ```sql
  INSERT INTO tableName [ PROPERTIES propertyList ]
  SELECT selectItems
  FROM NEW FileImport(constructorParameters)
  [ AS ] correlationName [ derivedColumnList ]
  ```

NEW: The alias `FileImport` is new in Version 3.0. In previous versions, you had to use the full class name.

- Call the static method Import, specifying the table name, file name, and control file name as parameters. (The method constructs and executes the SQL-J statement used above.)

  ```sql
  CALL FileImport.Import(staticMethodParameters)
  ```

Arguments to Import Command

- `tableName`
  The name of the table into which you want to bulk-import the data (qualified by the schema name if necessary).

- `propertyList`
  Properties for the bulk import. See “Import Properties” on page 6-10.

- `selectItems`
  Projects one or more columns to be inserted in the form of `derivedColumnName [, derivedColumnName]*`. 
• constructorParameters
  The parameters taken by the constructor for
  COM.cloudscape.tools.FileImport. See “Parameter Sets for
  COM.cloudscape.tools.FileImport Constructor” on page 6-8.

• staticMethodParameters
  The parameters taken by static Import method of
  COM.cloudscape.tools.FileImport. See “Parameter Sets for Static Method
  Import in COM.cloudscape.tools.FileImport” on page 6-8.

• correlationName
  A correlation name for the table, required by the FROM clause syntax.

• derived column list
  Optional names of the columns returned by the utility. By default, the
  import utility names the columns COLUMN1, COLUMN2, … ,
  COLUMNX. These are the names that you reference in selectItems.

Parameter Sets for COM.cloudscape.tools.FileImport Constructor

• InputFileURL
• InputFileURL, ControlFileURL

Parameter Sets for Static Method Import in COM.cloudscape.tools.FileImport

• java.sql.Connection, TableNameString, InputFileURL
• java.sql.Connection, TableNameString, InputFileURL, ImportPropertiesObject
• java.sql.Connection, TableNameString, InputFileURL, ControlFileURL
How to Use the Utilities

•  `java.sql.Connection`, `TableNameString`, `InputFileURL`, `ControlFileURL`, `importPropertiesObject`

**Table 6-1** Parameters to the Import Methods

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>InputFileURL</strong></td>
<td>The path name or URL of the ASCII input file accessible to the machine running Cloudscape. The input file must be of the default file format (see “Default Input or Output File Format” on page 6-13). If it is a literal, <code>InputFileURL</code> must be enclosed in single quotes. The path separator in the URL is <code>/</code> (forward slash), per the standard <code>file://</code> URL protocol. Absolute or relative paths are accepted.</td>
</tr>
<tr>
<td><strong>ControlFileURL</strong></td>
<td>If it is a literal, the URL of the control file must be enclosed in single quotes. For information about the control file, see “About the Control File” on page 6-14.</td>
</tr>
<tr>
<td><strong>TableNameString</strong></td>
<td>The name of the table into which you want to import data. If it is a literal, enclose it in single quotes.</td>
</tr>
<tr>
<td><strong>ImportPropertiesObject</strong></td>
<td>A Properties object representing import properties to use for the insert. See “Import Properties” on page 6-10.</td>
</tr>
<tr>
<td><strong>java.sql.Connection</strong></td>
<td>A connection to the pertinent database. Use <code>getCurrentConnection()</code> within an SQL-J statement.</td>
</tr>
</tbody>
</table>

For examples, see “Bulk Import and Export Within SQL-J Statement Examples” on page 6-12.

**To Bulk-Export**

Use an instance of the `COM.cloudscape.tools.FileExport` class (aliased as `FileExport`), specifying the table or view from which to export and the name of the output file. If you need to use a format different from the default format, also specify the name of the control file created with Cloudview.

**NEW:** The alias `FileExport` is new in Version 3.0. In previous versions, you had to use the full class name along with the CLASS keyword.

To export the entire table, specify a table name. To export data from any select statement, define a view as the SQL-J statement from which you want to export data. For example, to export data from a join, create a view to represent the join:

```sql
CREATE VIEW joinViewForExport AS
    SELECT country, city.getName()
```
FROM Countries JOIN Cities
USING (country_ISO_code)

Then export from the view.

CALL FileExport.Export(constructorParameters)

• parameters

COM.cloudscape.tools.FileExport.Export Parameter Sets

• (java.sql.Connection, TableOrViewNameString, OutputFileURL)
• (java.sql.Connection, TableOrViewNameString, OutputFileURL, ControlFileURL)

java.sql.Connection is the connection to the Cloudscape database.
TableOrViewName is the name of a table or a view in Cloudscape from which to export data. Qualify the table or view name with the schema name if it is not the default schema, APP.

OutputFileURL is the URL (of type file:// only) of the ASCII output file accessible to the machine running Cloudscape without going through any network transport protocol. The output file must be of the default file format (see “Default Input or Output File Format” on page 6-13).
OutputFileURL must be enclosed in single quotes.

ControlFileURL is the URL of the control file. It must be accessible to the machine running Cloudscape without going through any network transport protocol. ControlFileURL must be enclosed in single quotes.
See “About the Control File” on page 6-14.
TableOrViewName, OutputFileURL, and ControlFileURL must be enclosed in single quotes if they are constants.

For examples, see “Bulk Import and Export Within SQL-J Statement Examples” on page 6-12.

Import Properties

You can specify the insertMode property during bulk import in order to use Cloudscape’s fast import mode, which allows Cloudscape to use minimal logging and under-the-cover performance enhancements. Working in fast import mode results in a significant performance improvement.
If the table is empty, you only need to explicitly set this property when working within the context of an SQL-J statement. Otherwise, the utility sets this property by default. If the table is not empty, you must replace existing data to work in fast import mode. You always must explicitly set this property in that case.

The legal values for this property are bulkInsert and replace. If any other value is specified, an exception is thrown. Use bulkInsert to insert data into an empty table. Use replace to replace data in an existing table.

You can set this property in the following circumstances:

- When using the command-line utility, set the property on the command line as a -D parameter.
- When using an INSERT INTO SQL-J statement, set the property in a PROPERTIES clause after the table name.
- When using a CALL SQL-J statement, set the property in an ImportPropertiesObject.

**Syntax**

```
insertMode={ bulkInsert | replace }
```

When the insertMode property is set to either of the legal values, Cloudscape creates a new physical container for the table, which is more efficient than re-using the existing container.

**NOTE:** Cloudscape ignores the insertMode=bulkInsert property if any of the requirements for fast import mode are not met. See “Requirements for Fast Mode” on page 6-3. Cloudscape throws an exception if you specify insertMode=replace, and any of the requirements for fast import mode are not met.

If the insertMode property is set to bulkInsert, you can also specify an integer value for the bulkFetch property. This property sets the size of the bulk fetch for rebuilding any indexes associated with the table; the default value is 16.

**NOTE:** You can turn on bulkInsert even when you are not using the import utilities. For example, you may want to turn on bulkInsert when selecting data from another table or selecting data from an external virtual table. See “INSERT statement” in the Cloudscape Reference Manual.

**Syntax**

```
bulkFetch=intValue
```
Using the Bulk Import and Export Utilities

Setting this property to a higher value may speed up index rebuilding but requires more memory. Use the -mx parameter to the java command line to allocate more memory.

Bulk Import and Export Within SQL-J Statement Examples

-- This imports all columns and rows from
-- mydata3.dat into table1 in slow import mode
INSERT INTO myints SELECT *
FROM NEW COM.cloudscape.tools.FileImport
('/usr/share/data/mydata3.dat')
AS t

-- this imports the first two columns returned by
-- the file importer (whose columns are given the
-- correlation names a, b, and c) using the specified input
-- and control files. The example uses fast import mode.
INSERT INTO table2 PROPERTIES insertMode=bulkInsert
SELECT a, b FROM NEW COM.cloudscape.tools.FileImport(
  'file:////Jeeves/share/data/data2.asc',
  'file:////Jeeves/share/mapping/data2.ctl')
AS t (a, b, c)

-- Imports the first four columns from the specified file
-- (which are given correlation names)
-- into WorldCupStatistics using the default file
-- format in slow import mode
INSERT INTO WorldCupStatistics
SELECT a, b, c, d FROM NEW COM.cloudscape.tools.FileImport(
  '../cloudscape/demo/programs/tours/scripts/wc_stat.dat')
AS FI(a, b, c, d, e, f, g, h)

-- Imports the first four columns from the specified file
-- (which are given correlation names)
-- into WorldCupStatistics using the default file
-- format replacing the current data
INSERT INTO WorldCupStatistics PROPERTIES insertMode=replace
SELECT a, b, c, d FROM NEW COM.cloudscape.tools.FileImport(
  '../cloudscape/demo/programs/tours/scripts/wc_stat.dat')
AS FI(a, b, c, d, e, f, g, h)

-- use the static method to import data
-- into the WorldCupStatistics Table
CALL FileImport.Import(getCurrentConnection(),
  'WORLDCUPSTATISTICS', 'wc_stat.dat');
-- Exports data from the table table3 into the file
-- table3.dump using the default file format.
CALL FileExport.Export(getCurrentConnection(), 'APP.table3', '/temp/table3.dump');

Default Input or Output File Format

When no control file is specified, the input or output file takes on the default file format.

The default file format is a delimited ASCII file with the following characteristics:

- Rows are separated by a new line.
- Fields are separated by a comma (,).
- Field start is indicated by a double quote (").
- Field end is indicated by a double quote (").
- Field start and end are optional unless the field itself contains a field separator (comma) or row separator (CR-LF). For example, if a field contains the string “Hey, look over here!” the field start and end must be indicated by double quotation marks.

An example file of four rows and four columns using default file format:

```
1,abc,22,def
22,,,"a is a zero-length string, b is null"
13,hello,454,"world 
4,"b and c are both null",",
```

With no control file specified, this is what `COM.cloudscape.tools.FileExport` outputs:

```
"1", "abc", "22", "def"
"22", ",", "a is a zero-length string, b is null"
"13", "hello", "454", ",world 
"4", ",", "b and c are both null",",
```

To create a file in any other format, use Cloudview to create a control file that specifies the format. Cloudview allows you to specify the following:

- Fixed-width instead of delimited file. You will need to specify the width of each field.
- Field separator characters.
- Field start and end characters.
Using the Bulk Import and Export Utilities

- Row separators.
- If you are using a fixed-width file, the default output value for NULLs.

About the Control File

When you use Cloudview to specify a format other than the default, Cloudview allows you to name the control file and save it in the location of your choice.

**NOTE:** The control file is a text file generated and edited by Cloudview. Use Cloudview to maintain this file, because the format of this file is subject to change. Cloudview will handle upgrades to the file format if this format changes.

Treatment of NULLs

In a delimited file, a NULL value is exported as an empty field. The following example shows the export of a four-column row in which the third column is NULL:

```
7, 95, , Happy Birthday
```

Import works the same way; an empty field is imported as a NULL value.

When you customize the control file for import or export, you can choose fixed-width instead of delimited files. In fixed-width files, a NULL value is exported as the word **NULL**.

In fixed-width export, the example shown above would appear as follows:

```
7 95  NULL  Happy Birthday
```

During import, **NULL** is imported as a NULL value.

When using Cloudview, you can specify a different import or export value for NULL, which you will have to do if the width of the column that contains the NULL value is smaller than 4 (the number of characters in the word **NULL**).
Using the Database Class Utilities

$dbclasses$ is a server-side utility that allows you to store application logic in a database. Use it to add, update, and remove jar files in a database.

$DBClassLoad$ enables database class loading for applications using Cloudscape in embedded mode. Use it to boot an embedded application that loads classes from the Cloudscape classes loader instead of from the user’s class path. See the Cloudscape Developer’s Guide for more information on how it works.

You may wish to store jar files in a database so that Cloudscape and your application can load classes from the database instead of the user’s class path. Database class loading simplifies deploying applications. In a distributed Cloudscape system that uses synchronization, it also allows you to automate upgrading application logic.

For more information, see the Cloudscape Developer’s Guide.

NOTE: Cloudview makes it very easy to add, update, and remove jar files and to configure a database for database class loading.

Storing Jar Files in a Database Class Loading

$dbclasses$, or $COM.cloudscape.tools.dbclasses$, is a utility that allows you to store jar files in a database.

You can use the utility within an SQL-J statement or on the command line. When used within an SQL-J statement, the utility stores jar files in the database of the
current connection. When you use it on the command line, you specify the database connection URL of the database in which to store the jar files.

**Working with the Utility**

Your jar file has a *physical name* (the name you gave it when you created it) and a *Cloudscape name* (the Cloudscape identifier you give it when you load it into a particular schema). Its Cloudscape name is an *SQL92Identifier*; it can be delimited and must be unique within a schema. A single schema can store more than one jar file.

The syntax for the command-line utility to add a jar file is:

```
java COM.cloudscape.tools.dbclasses add databaseConnectionURL qualifiedJarNameInCloudscape jarFilePath
```

The SQL-J syntax to add a jar file is:

```
CALL dbclasses.addJar(' schema ', ' jarNameInCloudscape ', ' jarFilePath ')
```

The syntax for the command-line utility to remove a jar file is:

```
java COM.cloudscape.tools.dbclasses remove databaseConnectionURL qualifiedJarNameInCloudscape
```

The SQL-J syntax to remove a jar file is:

```
CALL dbclasses.removeJar ( ' schema ', ' jarNameInCloudscape ')
```

The syntax for the command-line utility to replace a jar file is:

```
java COM.cloudscape.tools.dbclasses replace databaseConnectionURL qualifiedJarNameInCloudscape jarFilePath
```

The SQL-J syntax to replace a jar file is:

```
CALL dbclasses.replaceJar(' schema ', ' jarNameInCloudscape ', ' jarFilePath ')
```

*databaseConnectionURL*

A database connection URL to connect to the database. Add any attributes necessary, such as user name and password. An example:

```
jdbc:cloudscape:toursDB;user=guest;password=cloudscape
```

*jarFilePath*

The path and physical name of the jar file to add or use as a replacement. An example:

```
d:/todays_build/tours.jar
```
Forcing Database Class Loading for an Application

• qualifiedJarNameInCloudscape
  The Cloudscape name of the jar file, qualified by the schema name if it is not APP. Two examples:
  
  APP.ToursLogic
  
  -- a delimited identifier. On the command-line you need to escape the double quotes
  APP."ToursLogic!"

• schema
  The schema name.

• jarNameInCloudscape
  The Cloudscape name of the jar file to add, remove, or update. Two examples:
  
  ToursLogic
  "ToursLogic!"

Complete Examples

• Complete SQL-J example for adding a jar:

  CALL dbclasses.addJar('APP', '"ToursLogic!"', 'd:/todays_build/tours.jar')

• Complete command-line example for adding a jar:

  java COM.cloudscape.tools.dbclasses add jdbc:cloudscape:toursDB
  APP."ToursLogic!"  d:/todays_build/tours.jar

For more information about storing classes in a database, see the Cloudscape Developer’s Guide.

Forcing Database Class Loading for an Application

Cloudscape provides an application bootstrap program that forces Cloudscape class loading for application-side logic. Use the bootstrap program instead of invoking your application on the command line if the application uses classes stored in the database.

Cloudscape Version 3.0
Using the Database Class Utilities

The program is called `COM.cloudscape.util.DBClassLoad`.

Typically, you would invoke your class like this:

```java
java COM.cloudscape.util.DBClassLoad databaseConnectionURL yourclassname classarguments
```

The `databaseConnectionURL` refers to the database in which the classes are stored.

In cases when user authentication is turned on and you are using database-level class loading, you will probably not want to put the user name and password on the command line in the database connection URL. In that case, you can invoke the class like this:

- Create a “wrapper” class. Don’t store this class in the database.
- In that wrapper class, ask the user to input his or her user name and password.
- At this point, you have two choices. Do one of the following:
  - Attach the user name and password to the database connection URL. Create a String array (`args`) that contains the database connection URL, the class name, and any arguments to your class. Invoke the application like this:

    ```java
    COM.cloudscape.util.DBClassLoad.main(args);
    ```

  - Get a connection to the database first (`theConnection`), passing in the user name and password. Create a String array (`args`) for any arguments to your class, then invoke the application like this:

    ```java
    COM.cloudscape.util.DBClassLoad.invokeMain(theConnection, "yourClassNameString", args);
    ```

For more information, see the javadoc for `COM.cloudscape.util.DBClassLoad`.
You can find out the version of Cloudscape with the following command:

```
java COM.cloudscape.tools.sysinfo
```

**NOTE:** This is the *product* version, not the database version. It uses information in the jar files, so verify that only `cloudscape.jar` or `cloudsync.jar` is in class path when you run this tool, not both.

```
D:>java COM.cloudscape.tools.sysinfo
--------------------------Java Info--------------------------
Java version:   1.2
Java vendor:    Sun Microsystems Inc.
Java home:      C:\Program Files\JavaSoft\JRE\1.2
Java classpath: d:\cloudscape\cloudscape.jar; d:\cloudscape\tools.jar
OS name:         Windows NT
OS architecture: x86
OS version:      4.0
Java user name:  janets
Java user home:  C:\WINNT\Profiles\janets
Java user dir:   D:\cloudscape
--------------------------Cloudscape Info--------------------------
[d:\cloudscape\cloudscape.jar] version 3.0.0 #9424
[d:\cloudscape\tools.jar] version 3.0.0 #9424
[License Type] Development. Valid.
```

When requesting help from Cloudscape technical support, send along the information provided by sysinfo.
Index

A
Absolute (ij command) 5-4
After Last (ij command) 5-5
Async (ij command) 5-6
Autocommit (ij command) 5-7

B
Before First (ij command) 5-8
Bulk import/export 6-1–6-14
and NULL 6-14
ascii only 6-2
description 1-2
not for client/server mode 6-2
performance gains of 6-1
replacing existing data 6-10
requirements 6-2
bulkFetch import property 6-11

C
Classes
  database vs. system storage 7-1
  storing in the database 7-1
Close (ij command) 5-9
Cloudconnector
  and Cloudview 2-3
  and ij 3-6
Cloudview 2-1
  and Cloudconnector 2-3
  and RmiJdbc 2-3
  and server frameworks 2-3
  and synchronization features 2-3
  JDK requirement 1-2
  properties 2-2
  starting 2-2
Cloudview on-line help
  specifying location of 2-3

Commit (ij command) 5-10
COM.cloudscape.tools package 6-2
COM.cloudscape.tools.dbclasses 7-2, 7-2
  using 7-1
COM.cloudscape.tools.FileExport 6-2
COM.cloudscape.tools.FileImport 6-2
COM.cloudscape.tools.sysinfo 8-1
COM.cloudscape.util.DBClassLoad 7-1
  using 7-3
Connect (ij command) 5-11

D
Database class utilities
  description 1-2
Database connection URL
  description 1-4
  specifying in ij 3-3
  supported protocols within 1-4
dbclasses
  using 7-1
dbclasses 7-1
DBClassLoad 7-1
  using 7-3
Disconnect (ij command) 5-13
Driver (ij command) 5-15
Drivers
  overview 1-3

E
Elapsed time (ij command) 5-16
Execute (ij command) 5-17
Exit (ij command) 5-19
Export 6-1–6-14

F
First (ij command) 5-20
Get Cursor (ij command) 5-21
Get Scroll Insensitive Cursor (ij command) 5-23

Help (ij command) 5-25

ij
and Cloudconnector 3-6
and RmiJdbc 3-6
commands 5-1—5-44
description 1-1
JDK requirement 1-2
properties 4-1—4-9
semicolon between statements required 3-7
starting 3-1
ij properties 3-2
ij (scripting tool)
using 3-1—3-8
ij.connection (ij property) 4-2
ij.database (ij property) 4-3
ij.driver (ij property) 4-4
ij.maximumDisplayWidth (ij property) 4-5
ij.outfile (ij property) 4-6
ij.protocol (ij property) 4-7
ij.protocol.protocolName (ij property) 4-8
ij.showErrorCode (ij property) 4-9
Import 6-1—6-14

JDBC
basics 1-3
JDBC driver
specifying in ij 3-3

Last (ij command) 5-26

MaximumDisplayWidth (ij command) 5-27

Next (ij command) 5-28

Prepare (ij command) 5-29
Previous (ij command) 5-29
Protocol 1-4
definition 1-5
Protocol (ij command) 5-31

Readonly (ij command) 5-32
Relative (ij command) 5-33
Remove (ij command) 5-34
RmiJdbc
and Cloudview 2-3
and ij 3-6
Rollback (ij command) 5-35
Run (ij command) 5-36

Scripts
running in ij 5-1
Set Connection (ij command) 5-37
Show Connections (ij command) 5-38
Shutdown
automatic with ij Exit command in embedded system 5-19
Storing classes in the database 7-1
sysinfo
description 1-2
using 8-1
SYSVISUAL schema
schema for Cloudview’s stored prepared statements 2-3
SysVisual.LOG 2-2
SysVisual.properties 2-2

Tools
how to use 1-1

Version
verifying 8-1

Wait For (ij command) 5-39