Oracle JDBC for Rdb User Guide

Release 7.4.1.1

March 2026

ORACLE

Oracle JDBC for Rdb User Guide

Release 7.4.1.1

Copyright 2005, 2026 Oracle and/or its affiliates. All rights reserved.
Oracle Corporation - Worldwide Headquarters, 2300 Oracle Way, Austin, TX 78741, United States.

Primary Author: Rdb Engineering and Documentation group.

This software and related documentation are provided under a license agreement containing restrictions on use and disclosure and are protected by intellectual property laws. Except as expressly permitted in your license agreement or allowed by law, you may not use, copy, reproduce, translate, broadcast, modify, license, transmit, distribute, exhibit, perform, publish, or display any part, in any form, or by any means. Reverse engineering, disassembly, or decompilation of this software, unless required by law for interoperability, is prohibited. The information contained herein is subject to change without notice and is not warranted to be error-free. If you find any errors, please report them to us in writing. If this is software or related documentation that is delivered to the U.S. Government or anyone licensing it on behalf of the U.S. Government, then the following notice is applicable:

U.S. GOVERNMENT END USERS: Oracle programs (including any operating system, integrated software, any programs embedded, installed or activated on delivered hardware, and modifications of such programs) and Oracle computer documentation or other Oracle data delivered to or accessed by U.S. Government end users are "commercial computer software" or "commercial computer software documentation" pursuant to the applicable Federal Acquisition Regulation and agency-specific supplemental regulations. As such, the use, reproduction, duplication, release, display, disclosure, modification, preparation of derivative works, and/or adaptation of i) Oracle programs (including any operating system, integrated software, any programs embedded, installed or activated on delivered hardware, and modifications of such programs), ii) Oracle computer documentation and/or iii) other Oracle data, is subject to the rights and limitations specified in the license contained in the applicable contract. The terms governing the U.S. Government s use of Oracle cloud services are defined by the applicable contract for such services. No other rights are granted to the U.S. Government.

This software or hardware is developed for general use in a variety of information management applications. It is not developed or intended for use in any inherently dangerous applications, including applications that may create a risk of personal injury. If you use this software or hardware in dangerous applications, then you shall be responsible to take all appropriate fail-safe, backup, redundancy, and other measures to ensure its safe use. Oracle Corporation and its affiliates disclaim any liability for any damages caused by use of this software or hardware in dangerous applications.

Oracle, Java, Oracle Rdb, Hot Standby, LogMiner for Rdb, Oracle SQL/Services, Oracle CODASYL DBMS, Oracle RMU, Oracle CDD/Repository, Oracle Trace, and Rdb7 are registered trademarks of Oracle and/or its affiliates. Other names may be trademarks of their respective owners.

Intel and Intel Inside are trademarks or registered trademarks of Intel Corporation. All SPARC trademarks are used under license and are trademarks or registered trademarks of SPARC International, Inc. AMD, Epyc, and the AMD logo are trademarks or registered trademarks of Advanced Micro Devices. UNIX is a registered trademark of The Open Group.

Contents

Preface ... vii

Chapter 1 Introduction ... 1
Chapter 2 Oracle JDBC for Rdb Drivers ... 3

2.1 Oracle JDBC for Rdb Native Driver ... 3
2.1.1 URL Specification Used with the Oracle JDBC for Rdb Native Driver ... 3
2.1.2 Class Used with the Oracle JDBC for Rdb Native Driver ... 4
2.1.3 Oracle JDBC for Rdb Native Driver JAR Files ... 4

2.2 Oracle JDBC for Rdb Thin Driver ... 5
2.2.1 URL Specification Used With the Oracle Rdb Thin Driver ... 5
2.2.2 Class Used with the Oracle JDBC for Rdb Thin Driver ... 6
2.2.3 Oracle JDBC for Rdb Thin Driver JAR Files ... 6

2.3 Connection Options ... 7
2.3.1 Oracle Rdb Second Password ... 14
2.3.2 Time Zones and Oracle Rdb Date/time Values ... 16
2.3.3 Timezone Connection Attribute ... 16
2.3.4 Sessiontimezone Connection Attribute ... 17
2.3.5 ConcurrentThread Connection Attribute ... 18

2.4 Oracle JDBC for Rdb System Properties ... 19
Chapter 3 Oracle JDBC for Rdb Servers ... 20

3.1 JDK Versioned Oracle JDBC for Rdb Server Jars ... 20
3.1.1 OpenJDK8 and Oracle JDBC for Rdb Servers ... 21

3.2 Oracle JDBC for Rdb Thin Server ... 22
3.2.1 Starting a Thin Server ... 22

3.3 Oracle JDBC for Rdb Multi-process Server ... 24
3.3.1 Starting a Multi-process Server ... 25
3.3.2 Shared Memory Usage ... 27
3.3.3 Pre-started Executors ... 27
3.3.4 Executor Naming ... 27
3.3.5 Executor Process Startup ... 28
3.3.6 Executor Maintenance ... 30
3.3.7 Lost Executor Sweep ... 31
3.3.8 Executor Balancing ... 31
3.3.9 Executor Reuse ... 32

3.4 Oracle JDBC for Rdb Pool Server ... 35
3.4.1 Starting a Pool Server ... 36
3.4.2 Pool Server Operation ... 37

3.5 Oracle JDBC for Rdb Manager Server ... 40
3.5.1 Utilizing a Manager Server ... 40
3.5.2 Starting a Manager Server ... 41
3.5.3 Adding a Remote Manager Server Configuration Entry ... 43
3.5.4 Using the Controller to Start Remote Servers ... 43
3.5.5 Manager Server Operations ... 44
Chapter 4 Server Configuration ... 46

4.1 Server Configuration Options ... 46

4.2 Multi-process Server Configuration Options ... *58*
4.3 Pool Server Configuration Options ... *60*
4.4 Manager Server Configuration Options ... *63*
4.5 Configuration Files ... *65*
4.5.1 Standard Properties File ... 65
4.5.2 XML-Formatted Configuration File ... 66
4.5.3 Using filenames in the configuration file ... 80
Chapter 5 Using SSL ... **81**
5.1 SSL Configuration ... *81*
5.1.1 Client SSL Configuration ... 81
5.1.2 Server SSL Configuration ... 82
5.2 SSL and the Controller ... *84*
5.3 SSL Configuration Options ... *84*
5.4 Using Self-Signed Certificates for Testing ... *85*
Chapter 6 Oracle JDBC for Rdb Controller ... **87**
6.1 JDK Versioned Oracle JDBC for Rdb Controller Jars ... *89*
6.2 Running the Controller ... *90*
6.2.1 Controller Command Line ... 93
6.3 Connecting to Servers ... *98*
6.3.1 Connect Command ... 99
6.3.2 Implicit Connection ... 100
6.4 Control Password ... *100*
6.5 Multicast Polling ... *101*
6.6 Server Matching ... *102*
6.6.1 Type Match ... 103
6.6.2 Name Match ... 104
6.6.3 Port Match ... 104
6.6.4 Stat Match ... 104
6.6.5 Node Match ... 105
6.6.6 Vers Match ... 106
6.6.7 Handling Servers From Prior Releases ... 106
6.7 Server Operations ... *107*
6.7.1 Closing Servers ... 107
6.7.2 Opening Servers ... 109
6.7.3 Showing Servers ... 110
6.7.4 Starting Servers ... 114
6.7.5 Stopping Servers ... 116
6.7.6 Watching Servers ... 117
6.7.7 Watching Events ... 118
6.7.8 Polling Servers ... 119
6.7.9 Poll Sub-commands ... 120
6.7.10 Showing Executors ... 122
6.7.11 Showing Server Pool ... 124
6.8 Client Operations ... *126*
6.8.1 Showing Clients ... 126
6.8.2 Stopping Clients ... 131
6.9 Other Commands ... *134*
6.9.1 Digest ... 134

6.9.2 Obfuscate ... 134
Chapter 7 Oracle SQL/Services and Oracle JDBC for Rdb Servers ... 136
7.1 JDBC Dispatcher ... 136
7.1.1 Creating an Oracle SQL/Services JDBC Dispatcher ... 136
7.1.2 Associating an Oracle SQL/Services JDBC Dispatcher to a Server ... 137
7.1.3 Legacy Behaviour Using the PORT_ID ... 138
7.1.4 Starting a JDBC Dispatcher ... 143
7.1.5 Stopping a JDBC Dispatcher ... 145
7.2 Command Procedures used by Oracle SQL/Services ... 145
7.2.1 JDBC Dispatcher Setup Procedure ... 146
7.3 Using Pool Servers ... 147
Chapter 8 Performance ... 149
8.1 Performance Features ... 149
8.1.1 FetchSize ... 150
8.1.2 Lockwait and Maxtries ... 150
8.1.3 Inactivity Timeouts ... 152
8.1.4 SQL Statement Cache ... 154
8.1.5 Results Cache ... 156
Chapter 9 Event Logging and Notification ... 158
9.1 Event Logging ... 158
9.2 Defining and Enabling Events ... 158
9.2.1 Defining Events ... 160
9.3 Event Types ... 161
9.3.1 Denial Events ... 162
9.3.2 Exception Events ... 162
9.3.3 Threshold Events ... 163
9.3.4 General Events ... 168
9.4 Watch Events Using the Controller ... 169
9.5 WatchEvent Sample Application ... 170
9.5.1 WatchEvent Sample Application Setup ... 170
9.5.2 Invoking the WatchEvent Sample Application ... 173
Chapter 10 Other Features ... 174
10.1 Anonymous Usernames ... 174
10.2 BYPASS Privilege ... 174
10.2.1 BYPASS and Multi-process Servers ... 175
10.2.2 Rights Identifiers ... 175
10.3 Persona ... 176
10.3.1 Persona and Server Operations ... 176
10.4 Default Transaction ... 177
10.4.1 Autocommit Transaction Duration ... 177
10.5 Executor Sub-process Used with the Rdb Native Driver ... 178
10.5.1 Starting a Detached Executor Process ... 179
10.5.2 Setting an Alternative ExecStartup Command Procedure ... 179
10.5.3 Setting Maximum Handshake Tries and Wait Duration ... 179
10.6 JDBC Hint Methods ... 180
10.7 Logging and Tracing ... 180
10.7.1 *Logfile Pattern* ... 181
10.8 Ignoring Statement.cancel() Method Calls ... 182

10.9 Server Name ... *182*
10.10 Named Databases ... *184*
10.11 On Start Commands ... *185*
10.11.1 srv.onStartCmd ... 185
10.11.2 srv.onExecStartCmd ... 186
10.11.3 srv.onCliStartCmd ... 187
10.12 Password Obfuscation in Server Configuration Files ... *188*
10.12.1 Control Passwords ... 188
10.12.2 User Passwords ... 189
10.13 Restricting Server, Database and Operational Access ... *190*
10.13.1 Restricting Database Access ... 191
10.13.2 Restricting User Access ... 192
10.13.3 Restricting IP Access ... 193
10.13.4 Privileged Users Access ... 196
10.13.5 Access to the Command Line ... 197
10.13.6 Access to the Server Root ... 197
10.13.7 Create and Drop Database Entitlement ... 198
10.13.8 Further Server Access Protection ... 200
10.13.9 Restricting SQL Statements ... 201
10.14 Scope of CONNECTION.setReadOnly() ... *202*
10.15 Server Command Procedures ... *203*
10.15.1 Server Startup Command Procedure ... 203
10.15.2 Executor Startup Command Procedure ... 204
10.15.3 CLI Startup Command Procedure ... 205
10.16 Server/Client Protocol Checking ... *206*
10.17 Using OpenVMS failSAFE IP ... *206*
10.18 Attaching to Multiple Databases in the Same Connection ... *207*
10.19 Shutdown Thread ... *208*
10.20 Getting a List of Known Databases from the Server ... *209*
10.20.1 Show Databases SQL Statement ... 209
10.20.2 getDatabases() ... 210
10.21 Create and Drop Database ... *210*
10.21.1 Server Root ... 211
10.21.2 Server Configuration Requirements ... 211
10.21.3 Create Database ... 211
10.21.4 Drop Database ... 213
10.22 Trace ... *213*
10.22.1 Setting Tracelevel ... 214
10.22.2 Abbreviated form of Tracelevel ... 216
10.22.3 Trace Values ... 216
10.23 File and Directory Access Requirements ... *216*
Chapter 11 JDBC Extensions for Oracle Rdb ... 218
11.1 Blob Class ... *218*
11.1.1 setSegSeparator() Public Method ... 218
11.2 Driver Class ... *219*
11.2.1 attach() Public Method ... 220
11.2.2 getDatabases() Public Static Method ... 222
11.3 ResultSet Class ... *223*

11.3.1 getBytes() Public Method ... 223
11.4 Extended SQL Syntax - SET ... 223
11.5 Extended SQL Syntax SHOW DATABASES ... 224
11.6 Extended SQL Syntax {nodbk} escape ... 225
Chapter 12 Other Information ... 226
12.1 Disallowed Dynamic SQL Statements ... 226
12.2 Sample Setup, Starting and Using an Oracle JDBC for Rdb Thin Server. ... 226
12.3 Sample Setup, Starting an Oracle JDBC for Rdb Thin Server from Oracle SQL/Services ... 234
12.4 Sample configuration file MY_SERVERS.XML ... 239
12.5 Datatype Mapping from Oracle Rdb to java.sql.Types ... 242
12.6 Datatype Mapping from java.sql.Types to Oracle Rdb ... 242
12.7 JDBC Specification SQL to Java Datatype Mappings ... 243
12.8 JDBC Specification Java to SQL Datatype Mappings ... 244

Tables

Table 1-1 JDBC driver JARs ... 1
Table 2-1 rdbNative Format Elements ... 3
Table 2-2 rdbNative JARs ... 4
Table 2-3 rdbThin Format Elements ... 5
Table 2-4 rdbThin JARs ... 7
Table 2-5 Connection Options ... 8
Table 3-1 Versioned server JARs ... 21
Table 3-2 Substatus Descriptions ... 30
Table 3-3 Executor Balancing ... 32
Table 3-4 Reuse Executors ... 32
Table 3-5 Pool Balancing ... 38
Table 4-1 Server Configuration Options ... 46
Table 4-2 Server Configuration Options2 ... 55
Table 4-3 Multi-process Server Configuration Options ... 59
Table 4-4 Pool Server Configuration Options ... 61
Table 4-5 Pool Server Configuration Options2 ... 63
Table 4-6 Manager Server Configuration Options ... 63
Table 4-7 Session Section Properties ... 70
Table 4-8 Database Section Properties ... 72
Table 4-9 EventSection Properties ... 73
Table 5-1 SSL Configuration Options ... 84
Table 6-1 Versioned server JARs ... 90
Table 6-2 Controller Options ... 90
Table 6-3 Controller Command Line Commands With Connection ... 94
Table 6-4 Controller Command Line Commands Without Connection ... 96
Table 6-5 rdbThin Type Codes ... 103
Table 6-6 rdbThin Match Values ... 104
Table 6-7 Interactive Close Server ... 108
Table 6-8 Command Mode Close Server ... 108
Table 6-9 Interactive Open Server ... 109
Table 6-10 Command Mode Open Server ... 109
Table 6-11 Interactive Show Server ... 110

Table 6-12 Command Mode Show Server 111
Table 6-13 Show Server display Attributes 112
Table 6-14 Additional Multi-process Server Attributes 113
Table 6-15 Interactive Start Server 115
Table 6-16 Command Mode Start Server 115
Table 6-17 Interactive Stop Server 116
Table 6-18 Command Mode Stop Server 117
Table 6-19 Interactive Watch Server 118
Table 6-20 Command Mode Watch Server 118
Table 6-21 Interactive Watch Events 119
Table 6-22 Command Mode Watch Events 119
Table 6-23 Interactive Poll Server 120
Table 6-24 Command Mode Poll Server 120
Table 6-25 Interactive Poll reopenlogs 121
Table 6-26 Command Mode Poll reopenlogs 121
Table 6-27 Interactive Show Executors 122
Table 6-28 Command Mode Show Executors 122
Table 6-29 Show Executors Display Attributes 123
Table 6-30 Interactive Show ServerPool 124
Table 6-31 Command Mode Show Clients 125
Table 6-32 Show Executors Display Attributes 125
Table 6-33 Interactive Show Clients 126
Table 6-34 Command Mode Show Clients 127
Table 6-35 Show Clients display attributes 128
Table 6-36 Additional Multi-process Server Attributes 130
Table 6-37 Interactive Stop Clients 132
Table 6-38 Command Mode Stop Clients 133
Table 6-39 Interactive Mode Digest 134
Table 6-40 Command Mode Digest 134
Table 6-41 Interactive Mode Obfuscate 135
Table 6-42 Command Mode Obfuscate 135
Table 9-1 Event criteria 160
Table 9-2 Threshold Types for Watch 164
Table 9-3 Threshold Attributes 165
Table 9-4 TestFor Attributes 166
Table 9-5 WatchEvent Configuration Properties 171
Table 10-1 Logfile patterns sequences 181
Table 10-2 Trace Values 216
Table 11-1 SHOW DATABASES Columns 224
Table 12-1 Datatype Mappings from OracleRdb to java.sql.Types 242
Table 12-2 Datatype Mappings from java.sql.Types to OracleRdb 242
Table 12-3 Datatype Mappings from SQL to Java 243
Table 12-4 Datatype Mappings from Java to SQL 244

Preface

Purpose of This Manual

The Oracle JDBC for Rdb User Guide describes concepts, features and usage of the Oracle JDBC for Rdb drivers and servers. This user guide covers Oracle JDBC for Rdb for OpenVMS on both Alpha and Integrity Servers.

Audience

This document is intended for users responsible for:

- System management
- Database administration
- Application programming

Access to Oracle Support

Oracle customers that have purchased support have access to electronic support through My Oracle Support. For information, visit http://www.oracle.com/pls/topic/lookup?ctx=acc&id=info or visit http://www.oracle.com/pls/topic/lookup?ctx=acc&id=trs if you are hearing impaired.

Reader Comments

Oracle Corporation welcomes your comments and suggestions on the quality and usefulness of this publication. Your input is an important part of the information used for revision.

- Did you find any errors?
- Is the information clearly presented?
- Do you need more information? If so, where?
- Are the examples correct? Do you need more examples?
- What features did you like most?

If you find any errors or have any other suggestions for improvement, please indicate the document title, release date, chapter, section, and page number (if available).

Please direct all comments, and corrections to this email address: **infordb_mb@oracle.com.**

If you have problems with the software, please contact your local Oracle Support Services.

Document Structure

This document consists of the following chapters:

Chapter 1	Introduction.
Chapter 2	Describes the Oracle JDBC for Rdb drivers.
Chapter 3	Describes the Oracle JDBC for Rdb servers.
Chapter 4	Describes details on how to configure Oracle JDBC for Rdb servers.
Chapter 5	Describes details on how to use SSL with Oracle JDBC for Rdb.
Chapter 6	Describes how to use the Oracle JDBC for Rdb controller.
Chapter 7	Describes how to use Oracle JDBC for Rdb with Oracle SQL/Services.
Chapter 8	Describes performance features that are available.
Chapter 9	Describes Event Notification and Logging.
Chapter 10	Describes other features that are available.
Chapter 11	Describes the JDBC extensions available for use with Oracle Rdb.
Chapter 12	Shows general examples and data type compatibilities.

Conventions

- Oracle JDBC for Rdb is referred to as Rdb JDBC or sometimes just JDBC.
- Oracle Rdb refers to Oracle Rdb Server for OpenVMS Alpha and OpenVMS IA-64 software.
- Oracle Rdb Connectivity Manager is referred to as ORCM.
- Oracle Rdb Data Provider for .NET is referred to as ORDP.
- OpenVMS Industry Standard 64 for Integrity Servers is referred to as OpenVMS IA-64.
- OpenVMS refers to both the OpenVMS IA-64 and OpenVMS Alpha operating systems.
- Hewlett-Packard Enterprise Company is referred to as HPE.
- VMS Software Inc. is referred to as VSI.
- The Oracle JDBC for Rdb Multi-process server is referred to as the Multi-process server or MP server.
- The Oracle JDBC for Rdb Pool server is referred to as the Pool server.

The following conventions are used in this document:

word	A lowercase word in a format example indicates a syntax element that you supply.
[ ]	Brackets enclose optional clauses from which you can choose one or none.
{ }	Braces enclose clauses from which you must choose one alternative.
...	A horizontal ellipsis means you can repeat the previous item.
. . .	A vertical ellipsis in an example means that information not directly related to the example has been omitted.

Release Specific Information

Certain features require specific versions of JDBC. These features are highlighted by a banner like the following:

Since release 7.3.4

Note:

Only new features or changed behavior seen since (and including) release 7.3.1 are highlighted in this manner.

Conventions in Code Examples

Code examples illustrate SQL or other command-line statements. They are displayed in a monospace (fixed width) font and separated from normal text as shown in this example:

```
SELECT last_name FROM employees WHERE last_name = 'TOLIVER';
```

Contents

Chapter 1
Introduction

Oracle provides the following Oracle JDBC for Rdb drivers:

- Oracle JDBC for Rdb native driver for client-side use with an Oracle Rdb installation
- Oracle JDBC for Rdb thin driver, a 100 percent pure Java driver for client-side use without an Oracle Rdb installation. This is particularly useful with applets.

The Oracle JDBC for Rdb drivers provide the same basic functionality and both support the following standards and features:

- Minimum JDK 1.5 / JDBC 3.0
- Same syntax and APIs

Programmatic access to these drivers may be made by incorporating the following driver JAR files into your build environment:

- RDBNATIVE.JAR
- RDBTHIN.JAR

In Release 7.3.5 of Oracle JDBC for Rdb, the installation of JDBC on Integrity system will also install varianted versions of the native and thin drivers. The associated JAR files are built to run in conjunction with a minimum Java version and provide support for JDBC features as shown in the following table:

Table 1-1 JDBC driver JARs

Driver JAR file	Minimum version of Java required	JDBC supported
RDBNATIVEV5.JAR and RDBTHINV5.JAR	JDK 1.5	JDBC 3.0, as released in JDK 1.4
RDBNATIVEV6.JAR and RDBTHINV6.JAR	Java SE 6	JDBC 4.0 as released in Java SE 6
RDBNATIVEV8.JAR and RDBTHINV8.JAR	Java SE 8 or OpenJDK8	JDBC 4.2 as released in Java SE 8 or OpenJDK8

In keeping with prior versions, the JDBC installation procedures will continue to install non-versioned variants of the driver JAR files:

- RDBNATIVE.JAR
- RDBTHIN.JAR

These drivers are identical to the JDK 1.5 (*V5) varianted JAR files.

The Oracle JDBC for Rdb drivers implement standard Oracle java.sql interfaces. It is assumed that the reader of these notes is already familiar with Java and JDBC.

The JDBC drivers are discussed in the next section Oracle JDBC for Rdb Drivers.

General information on Java may be found at
http://www.oracle.com/technetwork/java/index.html

General information on JDBC may be found at http://www.oracle.com/technetwork/java/index-142838.html

Documentation and information about installing HPE Java for OpenVMS systems may be found at the following web site:

http://www.hpe.com/

An important note for HPE customers using Java may be found at the following information page:

OpenVMS - Java Support on HPE OpenVMS and VSI OpenVMS

Documentation and information about VSI Java for OpenVMS systems may be found at the following web site:

https://vmssoftware.com/products/openjdk/

In conjunction with the Oracle JDBC for Rdb thin driver, Oracle provides the following Oracle JDBC for Rdb servers:

- Oracle Rdb Thin server
- Oracle Rdb Multi-process server
- Oracle Rdb Pool server

The Oracle JDBC for Rdb servers carry out remote database access operations on behalf of the Oracle JDBC for Rdb thin driver.

These JDBC servers are discussed later in this document under the section Oracle JDBC for Rdb Servers.

Management of the Oracle JDBC for Rdb servers may be carried out using the Oracle JDBC for Rdb controller, Oracle Rdb Connectivity Manager or the Oracle SQL/Services manager.

Contents

Chapter 2
Oracle JDBC for Rdb Drivers

There are two types of Oracle JDBC for Rdb drivers:

- Oracle JDBC for Rdb native driver
- Oracle JDBC for Rdb thin driver

The following sections discuss these two driver types as well as how to connect to Oracle Rdb databases and use System Properties when using the drivers:

- Connection Options
- Oracle JDBC for Rdb System Properties

Contents

2.1 Oracle JDBC for Rdb Native Driver

The Oracle JDBC for Rdb native driver is a Type II driver intended for use with client-server Java applications.

The native driver, written in a combination of Java and C, converts JDBC invocations to calls to SQLMOD modules, using native methods to call C-entry points.

When you use the native driver, the driver connects directly to the Oracle Rdb database system using SQLMOD. If you are not using Rdb Remote Access, then there are no network connections involved. This means that the native driver is potentially the fastest JDBC access method available within the Oracle JDBC for Rdb drivers.

Because the driver uses SQLMOD libraries to carry out Oracle Rdb access, the driver can only be used on a client machine if Oracle Rdb has been setup on that machine and the appropriate Oracle Rdb and SQL shared images are available. In addition, it is necessary for the driver to dynamically load a shared image to use with its Java JNI interface. Thus, this driver is not suitable for use with applications that require Java applets.

2.1.1 URL Specification Used with the Oracle JDBC for Rdb Native Driver

When you use the JDBC DriverManager to connect to an Oracle Rdb database using the native driver the following connection URL format should be used:

Format

```
jdbc:rdbNative:<database_specification><connect_switches>
```

Elements

The format elements are described in the following table:

Table 2-1 rdbNative Format Elements

Element	Description
`<database_specification>`	Is the full file specification of the Rdb database that you wish to connect.

`<connect_switches>`	These optional switches may be used to specify certain settings that should be established when the database connection is made. See Connection Options for more details.

Remarks

The `<database_specification>` should be a valid OpenVMS file specification or logical name.

Example

To connect to MYDIR:PERSONNEL:

```
Connection conn = DriverManager.getConnection(
"jdbc:rdbNative:myDir:personnel",user, pass);
```

2.1.2 Class Used with the Oracle JDBC for Rdb Native Driver

The Rdb native driver can be found in the following class:

```
oracle.rdb.jdbc.rdbNative.Driver
```

2.1.3 Oracle JDBC for Rdb Native Driver JAR Files

The Oracle JDBC for Rdb driver classes may be found in the Rdb native driver JAR file.

Prior to Oracle JDBC for Rdb release 7.3.5 only one Rdb native driver JAR file was available to use:

`rdbNative.jar`

Starting with release 7.3.5 you have the choice of three variants of the Rdb native driver JAR:

- rdbNativeV5.jar
- rdbNativeV6.jar
- rdbNativeV8.jar

These JARs provide JDBC and Java version specific variants of the driver classes:

Table 2-2 rdbNative JARs

Driver JAR file	Minimum version of Java required	JDBC supported
RDBNATIVEV5.JAR	JDK 1.5	JDBC 3.0, as released in JDK 1.4
RDBNATIVEV6.JAR	Java SE 6	JDBC 4.0 as released in Java SE 6

RDBNATIVEV8.JAR	Java SE 8 or OpenJDK8	JDBC 4.2 as released in Java SE 8 or OpenJDK8

To allow compatibility with prior versions, the file `rdbNative.jar` is also installed with the JDBC installation. This JAR is identical to the `rdbNativeV5.jar` file.

These JAR files are available in the `rdb$jdbc_home` directory created during JDBC installation. They are also available in the Windows MSI installation file provided during Oracle JDBC for Rdb installation.

Contents

2.2 Oracle JDBC for Rdb Thin Driver

The Oracle JDBC for Rdb thin driver is a 100 percent pure Java, Type IV driver. Because it is written entirely in Java, this driver is platform independent. It does not require any additional Oracle software on the client side.

For use with applets, the thin driver can be downloaded into a browser along with the Java applet being run. The HTTP protocol is stateless, but the thin driver is not. The initial HTTP request to download the applet and the thin driver is stateless. Once the thin driver establishes the database connection, the communication between the browser and the database is stateful and in a two-tier configuration.

The thin driver allows a direct connection to any Oracle Rdb database via an Oracle JDBC for Rdb server using TCP/IP on Java sockets.

Note:
When the thin driver is used with an applet, the client browser must have the capability to support Java sockets.

2.2.1 URL Specification Used With the Oracle Rdb Thin Driver

When you use the JDBC DriverManager to connect to an Oracle Rdb database using the thin driver the following connection URL format should be used:

Format

```
jdbc:rdbThin://<node>:<port>/<database_specification><connect_options>
```

Elements

The format elements are described in the following table:

Table 2-3 rdbThin Format Elements

Element	Description
`<node>`	Is the node name or IP address of the node where the Oracle JDBC for Rdb server you wish to connect to is running.

`<port>`	Is the port where the Oracle JDBC for Rdb server you wish to connect to is listening.
`<database_specification>`	Is the full file specification of the Rdb database to which you wish to connect.
`<connect_options>`	These optional switches may be used to specify certain settings that should be established when the database connection is made. See Connection Options for more details.

Example

To connect using the thin driver via an Oracle Rdb Thin server to MYDIR:PERSONNEL on node BRAVO using port 1701:

```
Connection conn = DriverManager.getConnection(
"jdbc:rdbThin://bravo:1701/myDir:personnel",
user, pass);
```

Note:

The <database_specification> should be a valid OpenVMS-style file specification or logical name, for example:

```
my_disk:[my_directory]my_database
```

When you use an Oracle Rdb thin driver connection, any logical names and relative directory specifications used in the database specification must be valid for the account and directory from which the Oracle Rdb Thin server was started.

2.2.2 Class Used with the Oracle JDBC for Rdb Thin Driver

The Rdb thin driver can be found in the following class:

```
oracle.rdb.jdbc.rdbThin.Driver
```

2.2.3 Oracle JDBC for Rdb Thin Driver JAR Files

The Oracle JDBC for Rdb thin driver classes may be found in the Rdb thin driver JAR file.

Prior to Oracle JDBC for Rdb release 7.3.5 only one Rdb thin driver JAR file was available to use:

`rdbThin.jar`

Starting with release 7.3.5, on Integrity systems you have the choice of three variants of the Rdb thin driver JAR:

- `rdbThinV5.jar`
- `rdbThinV6.jar`
- `rdbThinV8.jar`

These JARs provide JDBC and Java version specific variants of the driver classes:

Table 2-4 rdbThin JARs

Driver JAR file	Minimum version of Java required	JDBC supported
RDBTHINV5.JAR	JDK 1.5	JDBC 3.0, as released in JDK 1.4
RDBTHINV6.JAR	Java SE 6	JDBC 4.0 as released in Java SE 6
RDBTHINV8.JAR	Java SE 8 or OpenJDK8	JDBC 4.2 as released in Java SE 8 or OpenJDK8

To allow compatibility with prior versions, the non-varianted file `rdbThin.jar` is also installed with the JDBC installation. This JAR is identical to the `rdbThinV5.jar` file.

These JAR files are available in the `rdb$jdbc_home` directory. They are also available in the Windows MSI installation file provided during Oracle JDBC for Rdb installation.

Contents

2.3 Connection Options

The Oracle JDBC for Rdb drivers recognize several options that may be added to the connection string to specify certain default behavior and settings to be established when the connection is made.

Connection options may be either added directly to a connection URL using the `@` character as a separator, or as property values in the properties block that may be passed to the `DriverManager.getConnection()` method.

Format (In connection URL only)

```
@<option_name>=<value>
```

Options

The connections options that may be used are described in the following table:

Table 2-5 Connection Options

<option_name>	<value>	Default	Description
alias	string	**null**	Sets the alias for this database attach. This option is used only when attaching to a second database within the same Connection. See Attaching to Multiple Databases in the Same Connection for more details.
app	string	**null**	***Since release 7.3.2*** Sets the application name using this connection. This optional switch may be used to help identify application use of JDBC connections. The application name, if provided, will be displayed in the JDBC server logs and in the display of client information when using the controller `show clients` command.
cli.idleTimeout	decimal or hex integer	**0**	Sets the maximum time (in milliseconds) this client connection may be idle. If no operation is conducted using this connection within the time specified, the connection will be forcibly disconnected. The value **0** means unlimited idle time allowed. See Client connection timeout for more details.
concurrentThread	**true** or **false**	**false**	***Since release 7.3.5.1*** If **true** , this option tells the JDBC drivers that there is a potential of concurrent thread access to a single connection object. If this option is **true** , the JDBC drivers will disable some optimizations used to improve performance, to ensure that access to the JDBC drivers is thread safe. See ConcurrentThread Connection Attribute for more details.
handshakeTries	decimal or hex integer	**500**	Sets the maximum number of times the main process will attempt to establish handshake with its associated executor sub-process. This option is only valid on connections using the rdbNative driver and when ***multiprocess*** is enabled on the native connection.

<option_name>	<value>	Default	Description
			See Executor Sub-process used with the Rdb Native driver for more details.
handshakeWait	decimal or hex integer	**10**	Sets the time (in milliseconds) between handshake tries attempted between the main process and its associated executor sub-process. This option is only valid on connections using the rdbNative driver and when ***multiprocess*** is enabled on the native connection. See Executor Sub-process used with the Rdb Native driver for more details.
lockwait	decimal or hex integer	**-1**	Sets the lockwait (in seconds) for transactions. The value **1** means that the server will wait indefinitely for the lock. See Lockwait and Maxtries for more details.
multiProcess	**true** or **false** or **detached** or **spawn**	**false**	If **true** , a new executor process will be created for this connection. This option is only valid when used with an Rdb Native driver connection and will be ignored by the Rdb Thin driver. ***Since release 7.3.4.2.3*** If the value **detached** is specified a separate detached process will be used for the executor. If the value **spawn** is specified, a sub-process is used. This is the same as specifying the value **true** . See Executor Sub-process used with the Rdb Native driver for more details.
networkKeepalive	**true** or **false**	**false**	If **true** the socket used to connect the client to the server will have **SoKeepAlive** enabled See your operating system s socket documentation for more information on **SoKeepAlive**.
networkTimeout	decimal or hex integer	**0**	Sets the maximum time (in milliseconds) this client connection will wait on the completion of a read or write on the network. If the read or write does not

<option_name>	<value>	Default	Description
			complete within the designated time an exception will be raised. The value **0** means unlimited time allowed. This timeout is only applicable to the thin driver and is only placed on the client-side socket operations.
nlslang	Character set name	**DEC_MCS**	***Since release 7.3.3*** Sets the character set to use when converting DEC_MCS columns or literals for access by JDBC. If the column or literal is designated as having the character set DEC_MCS but contains non-MCS characters, when converting to Unicode during string operations JDBC will convert these characters as if they are DEC_MCS resulting in unexpected encoding of these characters. Setting **nlslang** tells JDBC to encode these strings using the specified character set instead of DEC_MCS. The value provided must be a valid character set name as found in the ***Supported Character Sets*** table of the ***Rdb SQL Reference Manual***. If not provided the default character set used will be DEC_MCS. Note: This option does not alter the SQL National Language.
password	The database password in plain text	**none**	Specifies the database password to use when attaching to the database.
password2	The password in plain text or obfuscated form	**none**	***Since release 7.3.4.2*** Specifies a second password to use during database authorization. Oracle recommends using an obfuscated password rather than plain text. The password should be obfuscated using the thin controller **Obfuscate** command. If the plain text password contains the character **@** then you **must** use an obfuscated password as the **@** may cause problems with internal JDBC connection string parsing.

<option_name>	<value>	Default	Description
			See Oracle Rdb Second Password for details.
reversetimestampdiff	true or false	false	***Since release 7.3.5.3.4*** If **true** , specifies that the extended function TIMESTAMPDIFF should subtract the first timestamp from the second timestamp rather than the default behaviour of subtracting the second from the first. The default is **false**.
rollbackOnDeadlock	true or false	**false**	***Since release 7.3.5.0.8*** If **true** , specifies that if a deadlock occurs, JDBC should force a rollback of the transaction even if **autoCommit** is disabled. The default is **false**.
sessiontimezone	The time zone to use for data conversion between client and database	**none**	***Since release 7.3.4.2.2*** Sets the JDBC Rdb database time zone. See Sessiontimezone Connection Attribute for more details.
silentTruncate	true or false	**false**	***Since release 7.3.5.1*** If **true** , specifies that JDBC should truncate strings that have been passed as input parameters to PreparedStatements, if the length is greater than expected by Rdb. For example, an insert of a string to a text column will truncate the value to the maximum length of that column. This will prevent SQL from raising string truncation conditions. The default is **false** , i.e., strings will not be truncated automatically. This option should be set to **true** if the string truncation semantics as seen in prior versions of JDBC is required.
simplepn	true or false	**false**	***Since release 7.3.4.2*** If **true** , specifies that the database product name retrieved when using **DatabaseMetaData** calls should be Rdb rather than Oracle Rdb .

<option_name>	<value>	Default	Description
			The default is **false** , i.e., Oracle Rdb will be returned as the database product name. This option may be required to be set to **true** for use with applications that may incorrectly assume that the database is an Oracle RDBMS database due to the word Oracle being present in the product name.
sqlcache	decimal or hex integer	**0**	Specifies the number of statements that may be maintained in the SQL cache. If less than or equal to **0**, SQL statement cache is disabled. Positive values specify the size of the SQL statement cache.
sqldialect	string value	**none**	***Since release 7.3.5.1*** Specifies SQL Dialect that should be used for this connection. By default, JDBC enables the SQL92 dialect for new connections. This may be changed for the connection by specifying a valid SQL dialect option. See the ***SET DIALECT*** section of the ***Oracle Rdb SQL Reference Guide*** for more details.
srv.execStartup	string value	**none**	Used in conjunction with the ***multiprocess*** option, tells the native driver where to find the command procedure to use to start an executor process. This option is only valid on connections using the rdbNative driver and when ***multiprocess*** is enabled on the native connection. See Setting an Alternative ExecStartup command procedure for more details.
srv.password	string value	**none**	Specifies the server password to be used for the connection. See Further server access protection for more details.
ssl*	various	**none**	Sets one or more SSL characteristics. See Using SSL for more details on these characteristics.
suppressAddingDbkey	**true** or **false**	**false**	***Since release 7.3.5***

<option_name>	<value>	Default	Description
			If **true** , this option tells the JDBC drivers to not add additional dbkey retrieval code to SQL statement text prior to execution. See Extended SQL Syntax {nodbk} escape for more information.
ticks	true or false	**true**	***Since release 7.3.3*** If **true** (the default), JDBC timestamp values will use seven (7) decimal places for the precision of fractions of seconds, otherwise three (3) decimal places will be used. By default, OpenVMS timestamp precision is in ticks or 100 nanosecond units which use seven (7) decimal places, but Rdb SQL may limit timestamp precision to 2 decimal places for timestamp values generated by SQL, for example, CURRENT_TIMESTAMP.
timezone	The default time zone to use	**none**	***Since release 7.3.4.2.2*** Sets the Java session time zone. See Timezone for more details.
transaction	**readonly** or **readonlydefer** or **readwrite** or **readwritedefer** or **automatic** or **oracle** or **manual** or **autofetch or autoreadfetch**	**automatic**	Specifies the default transaction for this connection. See Default Transaction and Scope of CONNECTION.setReadOnly() for more details. ***Since release 7.3.4.0.4*** The **autofetch** and **autoreadfetch** transaction options are only available when using the JDBC thin driver.
useHints	true or false	**true**	If **true** , the optional JDBC hint methods will be observed. If **false** , the optional JDBC hint methods will be silently ignored. See JDBC Hint Methods for more details.
useQueryHeader	true or false or character set name	**false**	***Since release 7.3.3*** If **true** or a valid character set name, JDBC will use the RDB$QUERY_HEADER column of the RDB$RELATION_FIELDS system table information to derive the Remarks property for the table columns returned in the DatabaseMetadata.getColumns() method.

<option_name>	<value>	Default	Description
			If **false** (the default) the Remarks field will be derived from the RDB$DESCRIPTION column of RDB$RELATION_FIELDS. If a character set name is used and is valid that character set will be used when converting the contents of the query header segmented string. The character set name provided must be a valid character set name as found in the ***Supported Character Sets*** table of the ***Rdb SQL Reference Manual***. If a character name is provided but is not valid the query header value will be converted assuming DEC_MCS encoding.
`user or u`	Database user name	**none**	The username to use when attaching to the database.
`username`	Database user name	**none**	***Since 7.3.3.2*** alternate keyword for user

Example

To connect using the thin driver via an Oracle JDBC for Rdb server to MYDIR:PERS on node BRAVO using port 1755 and enabling full trace logging for this connection:

```
Connection conn = DriverManager.getConnection(
    "jdbc:rdbThin://bravo:1755/myDir:pers@tracelevel=-1",
    user, pass);
```

Alternatively, these options may be placed in a properties block:

```
Properties info = new Properties();
info.put("user", user);
info.put("password", password);
info.put("tracelevel", traceLevel);

Connection conn = DriverManager.getConnection(
    "jdbc:rdbThin://bravo:1755/myDir:pers", info);
```

2.3.1 Oracle Rdb Second Password

Since release 7.3.4.2

Starting with Oracle Rdb release 7.3.2, Oracle Rdb allows you to specify a second password on the database attach and connect statements. This password must be provided at the same time the database password is provided. Oracle Rdb uses both passwords to authorization the database attach request.

If a second password is required, you must provide it during the JDBC connection request.

The connection attribute password2 may be used to provide this information. This password may be in plain text or obfuscated text created using the Controller Obfuscate command.

Note:

If the plain text password contains the **@** character, the obfuscated form of the password **must** be used, otherwise JDBC may fail to parse the connection string correctly.

Example

```
Connection conn = DriverManager.getConnection(
    "jdbc:rdbThin://bravo:1755/myDir:pers@password2=mypass",
    user, pass);
```

Preferably, the password should be obfuscated:

```
rdbthincontrol> obfuscate mypass
obfuscation : ##01249C19B8226825E92313CBC2AB
```

```
Connection conn = DriverManager.getConnection(
"jdbc:rdbThin://bravo:1755/myDir:pers@password2="+
"##01249C19B8226825E92313CBC2AB", user, pass);
```

Alternatively, these options may be placed in a properties block:

```
Properties info = new Properties();
info.put("user", user);
info.put("password", password);
info.put("password2", password2);
info.put("tracelevel", traceLevel);

Connection conn = DriverManager.getConnection(
    "jdbc:rdbThin://bravo:1755/myDir:pers", info);
```

Note:

If you specify a second password in the JDBC connection but the version of the Oracle Rdb database is earlier than release 7.3.2., the following exception will be returned on the connection request:

```
%SQL-F-DBENVSYNERR, Syntax error in database environment specification
```

2.3.2 Time Zones and Oracle Rdb Date/time Values

When working with date/time data in client applications, especially when the data must be displayed or entered in textual format, it is important to know in which time zone the data should be interpreted. This is especially important where applications need to work with data originating from more than one time zone.

Although several Java classes and their methods may accept time zone information as a direct parameter, many use the default Java VM time zone information when interpreting date time information.

This default time zone may be set by the client using one or more of the following techniques:

- `-Duser.timezone` on the Java command line or
- The Java `TimeZone.setDefault()` method or
- The appropriate Operating System time zone setter such as `SYS$MANAGER:UTC$TIME_SETUP.COM` on OpenVMS or
- `timezone` JDBC connection attribute (since release 7.3.4.2.2 JDBC)

(See the Java documentation for information on setting time zones and the `TimeZone` class.)

Although time zone information is not stored in the Rdb database, the time zone information obtained from the host system on which the database resides is used by Rdb to carry out the conversion of date/time values to the internal numeric format that this data is stored as within the database.

In general, the time zone used by Rdb is the time zone of the host system where the database resides i.e., the time zone that has been set by using OpenVMS tools such as `SYS$MANAGER:UTC$TIME_SETUP.COM`.

During the processing of date/time data to and from the Rdb database JDBC needs to correctly convert between the Java internal datetime format and the Oracle Rdb datetime forms. To do this JDBC must consider the time zone under which the Rdb data is to be stored.

JDBC uses the client application default time zone (as discovered using `TimeZone.getDefault`) to carry out these conversions. JDBC assumes that the client s time zone setting matches the database data time zone.

If the time zones do not match, the date/time values may not be converted correctly during insertion or retrieval and may not be correctly interpreted by other applications such as interactive SQL. Thus, it is important when accessing Rdb data, that the time zone setting at the time of the execution of update or query statements correctly match the time zone of the underlying Rdb data.

To ensure that time data is handled correctly you may have to set your client application environment default time zone using one of the methods cited above.

Client applications may also require interpreting date/times using time zones other than the one associated with the Rdb data, i.e., the default time zone required for the application might not match the Rdb database time zone.

Beginning with release 7.3.4.2.2 client applications may tell JDBC within which time zone the Rdb data should be interpreted irrespective of the default time zone of the application. See the section Sessiontimezone Connection Attribute for more details.

2.3.3 Timezone Connection Attribute

Since release 7.3.4.2.2

Oracle JDBC for Rdb allows you to specify the time zone to use when formatting and interpreting date/time strings. By default, JDBC will use the current Java session time zone when carrying out date/time formatting operations. This can be overridden for each connection the JDBC client makes.

The connection attribute ***timezone*** may be used to provide this information.

The time zone names used are the standard Java time zone IDs. See java.util.TimeZone#getAvailableIDs().

Invalid time zone values will be silently ignored, and the time zone will be reset to GMT.

Note:

This change in time zone is local to the JDBC client; the JDBC server time zone setting is not changed using this configuration option.

Example 1

```
Connection conn = DriverManager.getConnection(
    "jdbc:rdbThin://bravo:1755/myDir:pers"+
    "@timezone=Europe/Paris",
    user, pass);
```

Use the `@timezone` option if you wish the Java default time zone to be changed.

By changing the Java default time zone all datetime formatting operations carried out within the current Java session may be affected. This time zone change will also apply to all JDBC connections made within this session that have not had their time zone explicitly set by using a time zone configuration option.

See Connection Options for more information.

2.3.3.1 Setting Time Zone Application-wide

The default time zone may be set application-wide by using one or more of the following techniques:

- `-Duser.timezone` system property on the Java command line or
- The Java `TimeZone.setDefault()` method or
- The appropriate Operating System time zone setter such as SYS$MANAGER:UTC$TIME_SETUP.COM on OpenVMS or
- `-Doracle.rdb.jdbc.timezone` JDBC system property (associated with the `@timezone` connection attribute described above) on the Java command line.

For example:

```
$ java Doracle.rdb.jdbc.timezone=GMT my_application
```

(See the Java documentation for information on setting time zones and the `TimeZone` class.)

2.3.4 Sessiontimezone Connection Attribute

Since release 7.3.4.2.2

Client applications may inform JDBC of the time zone under which date/time data within the Rdb database should be interpreted. This time zone information is used to correctly carryout data conversions between Rdb and the Java subsystem. (See Timezones and Oracle Rdb Date/time Values for information about this data conversion).

By default, on the client, JDBC uses the Java `TimeZone.getDefault()` method to obtain the application s default time zone and uses this time zone during data conversions.

If this time zone is not the same as the one used for the Rdb data, you may use the `@sessiontimezone` connection parameter to correctly set the appropriate time zone for conversions, for example:

```
Connection conn = DriverManager.getConnection(
    "jdbc:rdbThin://bravo:1755/myDir:pers"+
    "@sessiontimezone=Europe/Paris",
    user, pass);
```

Use the `@sessiontimezone` option if you wish to change the time zone used to convert datetime values between Rdb and Java for this connection only. This will change how JDBC will convert datetime values for that connection but will not change the Java session default time zone.

This time zone is only used for data conversion and does not affect the Java VM default time zone on either the client or the server.

See Connection Options for more information.

2.3.4.1 Setting sessiontimezone Application Wide

If you wish to set the *sessiontimezone* for all Oracle JDBC for Rdb connections made by your application, you can use the following Java command line system properties.

Doracle.rdb.jdbc.sessiontimezone

```
$ java Doracle.rdb.jdbc.sessiontimezone=GMT my_application
```

or

Drdb.timezone

```
$ java Drdb.timezone=GMT my_application
```

2.3.5 ConcurrentThread Connection Attribute

Since release 7.3.5.1

Although JDBC is thread-safe when concurrent threads use separate connections, several optimization features introduced into JDBC drivers over the last few releases may cause threading issues when concurrent threads are using the same JDBC connection.

Concurrent thread access of a single JDBC connection may fail with several symptoms including Access Violations and prematurely closed connections.

Third-party or client applications using JDBC drivers may carry out concurrent threading on connections to optimize data retrieval. When these applications are used with Oracle JDBC for Rdb drivers, it is possible that operations may fail due to threading issues.

If your application intends to carry out concurrent thread operations on the same JDBC connection, the *concurrentThread* connection attribute may be used to inform JDBC of this fact.

When set to **true** , this attribute will cause JDBC drivers to disable some of the enhancements to statement preparation and record retrieval that are normally carried out but that are not thread-safe if interrupted by another thread using the same connection object.

See Connection Options for more information.

Contents

2.4 Oracle JDBC for Rdb System Properties

The Oracle JDBC for Rdb drivers and servers can recognize configuration or connection properties passed in as Java System Properties from the operating system command line during application invocation.

When used in conjunction with an application invoking the Rdb native or Rdb thin driver, the drivers will recognize system properties with an `<option_name>` similar to a valid Connection option, see Connection Options for more details of these options.

If the same configuration option is specified as both an Rdb system property and within the connection URL, then the value within the connection URL will take precedence.

When used in conjunction with an Rdb server invocation the server will recognize system properties with any `<option_name>` that may be used as a server configuration option, see Server Configuration Options and Pool Server Configuration Options for more details of these options.

Any Rdb system property specified during the invocation of a server will take precedence over the same property specified on the command line as a standard configuration option or in a configuration file.

Format

```
-Doracle.rdb.jdbc.<option_name>=<value>
```

Example

To set trace level to trace everything for your application that utilizes either the Rdb native or Rdb thin driver:

```
$ java Doracle.rdb.jdbc.tracelevel=-1 my_application
```

Contents

Chapter 3
Oracle JDBC for Rdb Servers

Oracle JDBC for Rdb servers are the server-side components that service JDBC requests issued by applications using the Oracle Rdb thin driver.

There are four types of Oracle JDBC for Rdb servers:

- Oracle JDBC for Rdb Thin server
- Oracle JDBC for Rdb Multi-process server
- Oracle JDBC for Rdb Pool server
- Oracle JDBC for Rdb Manager Server

Each server is multi-threaded, able to handle multiple client requests at the same time.

Oracle JDBC for Rdb servers should be installed and invoked on each node from which you wish to serve Oracle Rdb databases.

The Oracle JDBC for Rdb thin driver communicates with the Oracle JDBC for Rdb servers using Java sockets over TCP/IP.

Any of the servers, except for the Oracle JDBC for Rdb Manager server, may be used by thin clients to connect to Rdb databases. The Oracle JDBC for Rdb Manager server cannot be used for thin client access, it is a special server that may be used to aid in the management of your JDBC server environment.

The following sections provide information about each of the server types and the various ways you may start-up a server on your system.

Note:

To start Oracle JDBC for Rdb servers you will require certain access to the Oracle JDBC for Rdb directories and files. See File and Directory access Requirements for more details.

3.1 JDK Versioned Oracle JDBC for Rdb Server Jars

Since release 7.4.2

Prior to Oracle JDBC for Rdb release 7.4.2 only one version of each specific server-type JAR file was available to use when starting Oracle JDBC of Rdb servers:

- `rdbThinSrv.jar`
- `rdbThinPoolSrv.jar`
- `rdbManSrv.jar`

Beginning with release 7.4.2 you have the choice of three variants of each of the server JARs:

- rdbThinSrvV5.jar
- rdbThinSrvV6.jar
- rdbThinSrvV8.jar

- rdbThinPoolSrvV5.jar
- rdbThinPoolSrvV6.jar
- rdbThinPoolSrvV8.jar
- rdbManSrvV5.jar
- rdbManSrvV6.jar
- rdbManSrvV8.jar

These JARs provide JDBC and Java version specific variants of the driver classes:

Table 3-1 Versioned server JARs

Driver JAR file	Minimum version of Java required	JDBC supported
RDBTHINSRVV5.JAR RDBTHINPOOLSRVV5.JAR RDBMANSRVV5.JAR	JDK 1.5	JDBC 3.0, as released in JDK 1.4
RDBTHINSRVV6.JAR RDBTHINPOOLSRVV6.JAR RDBMANSRVV6.JAR	Java SE 6	JDBC 4.0 as released in Java SE 6
RDBTHINSRVV8.JAR RDBTHINPOOLSRVV8.JAR RDBMANSRVV8.JAR	Java SE 8 or OpenJDK8	JDBC 4.2 as released in Java SE 8 or OpenJDK8

To allow compatibility with prior Oracle JDBC for Rdb releases, non-versioned variants of each server JAR are also installed by the JDBC installation. These JARs are identical to the V5 versions with the same name:

- `rdbThinSrv.jar`
- `rdbThinPoolSrv.jar`
- `rdbManSrv.jar`

The server JAR files are available in the `rdb$jdbc_home` directory created during JDBC installation.

3.1.1 OpenJDK8 and Oracle JDBC for Rdb Servers

Since release 7.4.2

In prior releases of Oracle JDBC for Rdb only one variant of each server-type JAR file was available. The JAR files were built using the JDK1.5 development environment and were compatible with OpenVMS Java environment up to and including Java SE 8.

Unfortunately, these JAR files are not compatible with the OpenJDK8 runtime environment on OpenVMS and fail to load properly when invoked.

If you are running Java under OpenJDK8 on OpenVMS, to run JDBC servers or the controller you must use the new V8 versions of the various JAR files installed by Oracle JDBC for Rdb release 7.4.2 and above.

This in turn may require you to adapt your JDBC scripts and procedures accordingly by either:

- changing your scripts and procedures to add the V8 version of the JAR file to the classpath used for invocation of the server or

- if you will be using OpenJDK8 primarily or exclusively on your system you could copy and rename the V8 JAR files to remove the V8 from their names.

Note:

The JAR file containing the JDBC controller code is also available as a V8 variant. If you use OpenJDK8 on your OpenVMS system you must use the V8 variant of the controller jar , `rdbThinControlV8.jar`.

See Oracle JDBC for Rdb Controller for more details.

Contents

3.2 Oracle JDBC for Rdb Thin Server

The Oracle JDBC for Rdb Thin server is a server-side component that services JDBC requests issued by applications using the Oracle Rdb thin driver.

The standard thin server is multi-threaded, able to handle multiple client requests at the same time. Because the server is maintained as a single OpenVMS process, database access for each of the threads is synchronized.

A thin server is installed and invoked on each node from which you wish to serve Oracle Rdb databases. Oracle Rdb must be already installed and running on these nodes.

The server communicates with the Oracle Rdb thin driver using Java sockets over TCP/IP with the default Port ID 1701.

3.2.1 Starting a Thin Server

A thin server may be started by using the appropriate start statement within the controller, as an Oracle SQL/Services JDBC dispatcher or directly from the operating system command line.

3.2.1.1 Starting a Thin Server from the Oracle JDBC for Rdb Controller

A thin server may be started from the controller by referencing a thin server definition in an XML-formatted configuration file. See Starting Servers within Oracle JDBC for Rdb Controller for more details.

Format

```
START SERVER <server_name>
```

Example

Given the following server section in the XML-formatted configuration file mycfg.xml:

```
<server
    name="serv1"
    type="RdbThinSrv"
    url="//localhost:1707/"
    logfile="myLogs:serv1.log"
/>
```

The following command may be used to start this server from within the controller:

```
rdbthincontrol> start server serv1
```

Alternatively, the controller may be used in command mode to start a server:

```
$ java jar rdbthincontrol[<variant>].jar -cfg mycfg.xml -name serv1 -startserver
```

The optional <variant> if present must be the value V5 , V6 or V8 , See JDK Versioned Oracle JDBC for Rdb Controller Jars

3.2.1.2 Starting a Thin Server from Oracle SQL/Services

A thin server may be started from the Oracle SQL/Services manager.

Using the Oracle SQL/Services manager, you must first establish a connection to the Oracle SQL/Services server. Once connected you may then start a JDBC dispatcher.

Before you can start a JDBC dispatcher, you must first create its definition in the Oracle SQL/Services environment.

See Oracle SQL/Services and Oracle JDBC for Rdb Servers for more details on creating a JDBC dispatcher.

Format

```
START DISPATCHER <jdbc_dispatcher_name>
```

Example

```
$ run sys$system:SQLSRV_MANAGE74
SQLSRV> connect server;
Connecting to server
Connected
SQLSRV> start disp JDBC_MPDISP;
SQLSRV>
```

3.2.1.3 Starting a Thin Server from the Command Line

You may invoke a thin server from the OpenVMS command line.

Instead of placing several options on the command line, you may wish to create a server definition within an XML-formatted configuration file and then start the server using its server name. The server type for this server definition must be set to RdbThinSrv for a standard thin server.

Format

```
$ java jar rdbthinsrv[<variant>].jar [<-option>]
```

Elements

The optional <variant> if present must be the value V5 , V6 or V8 , See JDK Versioned Oracle JDBC for Rdb Server Jars.

See Server Configuration Options for a list of valid `<-option>` values. Remember that, on the DCL command line, each configuration option must have a hyphen `(-)` prepended to it.

Remarks

By default, the server is assumed to be of type RdbThinSrv, a standard thin server.

See XML formatted Configuration File for more details on server definitions within configuration files.

Example 1

```
$ java jar rdbthinsrv.jar port 1707
```

Example 2

Given the following server section in the XML-formatted configuration file `mycfg.xml`:

```
<server
    name="serv1"
    type="RdbThinSrv"
    url="//localhost:1707/"
    logfile="myLogs:serv1.log"
/>
```

the following method may be used to start this thin server:

```
$ java jar rdbthinsrv.jar cfg mycfg.xml name serv1
```

Contents

3.3 Oracle JDBC for Rdb Multi-process Server

The Oracle JDBC for Rdb Multi-process server is a server-side component that processes requests from the Oracle JDBC for Rdb thin driver using small memory footprint sub-processes to carry out the requested operations on the Oracle Rdb database.

A Multi-process server is multi-threaded and may handle multiple concurrent clients allocating each client its own subprocess for database access, thus allowing better concurrency and availability.

Most of the Multi-process server operations are carried out in a client thread context within the main server process, handing off control to the client s allocated subprocess only when direct Oracle Rdb database operations are required. Each client has its own OpenVMS subprocess, thus concurrency is improved, as the server does not need to synchronize database operations.

By default, the allocated subprocess is terminated on client disconnect. Executors may also be retained for re-use after client disconnect by specifying a positive *maxFreeExecutors* configuration option.

An idle timeout value may be placed on executors to request the server to close a free executor after a predefined amount of time of non-use, even if the number of free executors is less than the `maxFreeExecutors` limit. If the specified idle time elapses, the executor will be a candidate for termination if the number of free executors is greater than the *minFreeExecutors*.

A Multi-process server is installed and invoked on each node from which you wish to serve Oracle Rdb databases. Oracle Rdb must be already installed and running on these nodes.

The server communicates with the thin driver using Java sockets over TCP/IP with the default Port ID 1701.

3.3.1 Starting a Multi-process Server

A Multi-process server may be invoked by using the appropriate start statement within the controller, as an Oracle SQL/Services JDBC dispatcher, or directly from the operating system command line.

3.3.1.1 Starting a Multi-process Server from the Controller

A Multi-process server may be started from the controller by referencing a Multi-process server definition in an XML-formatted configuration file.

See Starting Servers within Oracle JDBC for Rdb Controller for more details.

Format

```
START SERVER <server_name>
```

Example 1

Given the following server section in the XML-formatted configuration file mycfg.xml:

```
<server
    name="Mpserv1"
    type="RdbThinSrvMP"
    url="//localhost:1799/"
    logfile="myLogs:serv1.log"
/>
```

the following command may be used to start this server from within the controller:

```
rdbthincontrol> start server Mpserv1
```

Example 2

Given the same configuration file as shown above, the controller may be used in command mode to start a server:

```
$ java jar rdbthincontrol.jar -cfg mycfg.xml -name Mpserv1 -startserver
```

3.3.1.2 Starting a Multi-process Server from Oracle SQL/Services

A Multi-process server may be started from Oracle SQL/Services manager.

Using the Oracle SQL/Services manager, you must first establish a connection to the SQL/Services server. Once connected you may then start a JDBC dispatcher.

Before you can start a JDBC dispatcher, you must first create its definition in the Oracle SQL/Services environment.
See Oracle SQL/Services and Oracle JDBC for Rdb Servers for more details.

Format

```
START DISPATCHER <jdbc_dispatcher_name>
```

Example

```
$ run sys$system:SQLSRV_MANAGE74
SQLSRV> connect server;
Connecting to server
Connected
SQLSRV> start disp JDBC_MPDISP;
SQLSRV>
```

See Oracle SQL/Services and Oracle JDBC for Rdb Servers for more details.

3.3.1.3 Starting a Multi-process Server from the Command Line

You may invoke a Multi-process server from the OpenVMS command line.

Format

```
$ java jar rdbthinsrv[<variant>].jar [<-option>]
```

Elements

The optional <variant> if present must be the value V5 , V6 or V8 , See JDK Versioned Oracle JDBC for Rdb Server Jars.

See Server Configuration Options for a list of valid <-option>s. Remember that, on the DCL command line, each configuration option must have a hyphen (-) pre-pended to it.

Remarks

Both the thin server and Multi-process server are started using the same rdbthinsrv.jar file. It is the server type that determines the style of server that will be started.

By default, the server is assumed to be of type *RdbThinSrv*, a standard thin server. To start a Multi-process server, the server type must be set to *RdbThinSrvMP*.

Alternatively, the developer may wish to create a server definition within an XML-formatted configuration file and then start the server using its server name. Again, the server type must be set to *RdbThinSrvMP*.

On the DCL command line you must use double quotes to preserve the case-sensitive type name.

Example 1

```
$ java jar rdbthinsrv.jar port 1755 type "RdbThinSrvMP"
```

Example 2

Given the following server section in the XML-formatted configuration file `mycfg.xml`:

```
<server
    name="Mpserv1"
    type="RdbThinSrvMP"
    url="//localhost:1799/"
    sharedmem="102400"
    logfile="myLogs:serv1.log"
```

```
/>
```

the following method may be used to start this Multi-process server:

```
$ java jar rdbthinsrv.jar cfg mycfg.xml name Mpserv1
```

3.3.2 Shared Memory Usage

The Multi-process server needs to allocate shared global memory for communication with its executors, the size of which you may specify using the *sharedmem* server configuration option.

The default allocation for shared memory is 1024 KB and is only adequate for one or two executors.

A rule of thumb that can be used is to allow 1024 KB for each concurrent executor you expect to be running in conjunction with the server, but this will depend on the complexity of the queries, the number of columns involved and the size of the data area that will have to be created to hold the data returned to the executor by Rdb.

3.3.3 Pre-started Executors

With a Multi-process server, you may also specify the number of executor processes that may be pre-started when the server starts running, by specifying a positive *prestartedExecutors* configuration option.

This is particularly useful if your system takes a while to start OpenVMS processes and sub-processes due to system load.

In addition, you can also specify the maximum number of free executor processes that may be kept around while the server is running by specifying a positive *maxFreeExecutors* configuration option.

By pre-starting executor processes you may reduce the overall elapsed time it takes for a client to make its initial database connection, but keep in mind that free executors, although they may not have database connections active, may still have OpenVMS system resources allocated.

3.3.4 Executor Naming

Each executor started up on a single system requires a unique process name on that system. By default, a name will be created for the executor based on the name of the server that started it and a hexadecimal value that represents the instance of the executor process with relation to the server.

By default, the name of the executor subprocess has the following format:

Format

```
First seven (7) characters of server name +
eight (8) character hexadecimal id.
```

Remarks

Names of executors are not case-sensitive.

The first seven (7) characters of the names of Multi-process servers started up within the same system should be unique irrespective of character casing; otherwise, executor process names may clash.

Example 1

```
RDBTHNS00000220
```

The format of the executor names may be changed by using the `srv.execPrefix` configuration option:

Format

`srv.execPrefix` + `up to eight (8) character hexadecimal id.`

Remarks

If the `srv.execPrefix` configuration option is specified for a Multi-process server, all executors for that server will have this name prefix. The server will try to provide a unique name for each executor instance by appending to the given prefix as many characters of the hexadecimal numeric id of the executor that will keep the executor s name within the Process name sized expected by OpenVMS.

See XML Formatted Configuration File for more details on server definitions.

Example 2

Given the `srv.execPrefix` of "`MY_EXECUTOR_`" the fourth executor will be named:

```
MY_EXECUTOR_004
```

Note:

The longer the prefix, the smaller the number of characters that may be used to provide uniqueness, so consideration should be given to the number of concurrent executors that you expect a server to maintain when specify a customized executor name prefix.

3.3.5 Executor Process Startup

By default, the Multi-process server will create a subprocess for each executor it allocates and starts. OpenVMS command procedures are used during this subprocess creation. Information about these command procedures may found in the Server Command Procedures and On Start Commands sections of this document.

If a `persona` is specified for the server (see Persona for more information) or if the `srv.usecreprc` configuration attribute is set to **true** , the server will use the OpenVMS `SYS$CREPRC` system service to start the process as detached.

If a `persona` is not specified and the `srv.usecreprc` is set to **false** then the Java `System.exec()` method will be used instead which will start a subprocess.

Note:

Some OpenVMS quotas are shared across the entire process subtree (i.e., shared between the server process and its subprocesses), thus it is possible that the increasing the number of concurrent executors available for a server may exceed process or system quotas and some executors may fail to start correctly. (See the OpenVMS documentation on process and system level quota limits.)

If shared process level quotas are found to be an issue, then consider using the `srv.usecreprc` configuration attribute to tell the MP server to use detached processes for the executors. As the executor will run within its own detached process the operating system should allocate a separate set of process quotas to it.

If the server environment or the JDBC directories are not appropriately setup, errors may occur during the startup of the executor process.

See File and Directory access Requirements for more details on JDBC directories access requirements.

The steps taken during the startup of an executor process depend on whether `persona` or `srv.usecreprc` are used with the server.

3.3.5.1 Executor Start-up Steps

Since release 7.3.4.1.2

Without Persona and when `srv.usecreprc` is false

If persona is not used and `srv.usecreprc` is set to **false ,** the following steps are carried out by the server to start an executor:

1. An executor name based on the server s name is generated for the new process; see Executor Naming for more details.
2. An attached process is created using the `System.exec()` method.
3. The command procedure designated by the `srv.execStartup` option for the Multi-process server is executed. If this option has not been specified for the server nor for the `DEFAULT` server in the configuration file, then `RDB$JDBC_HOME:RDBJDBC_STARTEXEC.COM` is used. See Server Startup Command Procedure.
4. If the `srv.onExecStartCmd` option is present for this server or for the `DEFAULT` server then this command is executed. This is generally used to setup server and site-specific environments. See srv.onExecStartCmd.
5. The executor image pointed to by the logical name `RDBJDBCEXEC` is executed.
6. The executor and server establish communication channels using global shared memory.

With Persona or when `srv.usecreprc` is true

If persona is used or `srv.usecreprc` is set to **true** :

1. An executor name based on the server s name is generated for the new process; see Executor Naming for more details.
2. Process quotas are determined for the executor process based on the current quotas of the executing server.
3. A termination mailbox is setup for the executor process and a read is issued.

4. CREPRC is used to create a process and SYS$SYSTEM:LOGINOUT.EXE is executed.

5. Steps 3 thru 6 as described in the previous list above.

3.3.5.2 Executor Process Start-up Problems

If a problem occurs during executor subprocess creation, the status codes relating to the problem will be written to the server log, for example:

```
java.sql.SQLException: Unable to start process,
status: 0x56EC03C : substatus 65535
```

The status code shown is a VMS status code or an Rdb specific status code; see the OpenVMS and Oracle Rdb documentation for more information on this status code.

The substatus indicates more information about the problem found. The following table lists the substatus values and their meanings.

Table 3-2 Substatus Descriptions

Substatus	Description
12	No more memory, check quotas.
13	Unable to create command procedure in rdb$jdbc_com: directory, either insufficient privilege or access denied or there already exists an earlier version of the file with the same name but created by another user.
19	Problem in pathname pointed to by rdb$jdbc_com logical name, invalid device specified.
20	Problem in pathname pointed to by rdb$jdbc_com logical name, invalid directory specified.
24	Too many files open by server already, check quotas.
28	Disk full, check the disk pointed to by rdb$jdbc_com.
30	Disk or directory is write-protected, check the disk/directory pointed to by rdb$jdbc_com.
65530	Process terminated prematurely.
65531	Problem reading termination mailbox.
65532	Problem during call to CREPRC.
65533	Problem getting information about termination mailbox.
65534	Problem creating termination mailbox.

Note:

It is important that the server process has appropriate access rights to the directories specified by JDBCRDB_HOME, RDBJDBC_COM and RDB$JDBC_LOGS logical names, see File and Directory access Requirements for more details.

3.3.6 Executor Maintenance

The Multi-process server maintains a list of executor processes that it has started to allow it to do basic housekeeping on these processes.

At process start up, the server will attempt to create the number of executors specified in the server configuration parameter `prestartedExecutors`. Once started, these executors will be placed in the server s free executor list.

Executors may be either *free*, meaning that currently no client is using it, or *occupied*, meaning that the executor has been allocated to a client.

When an executor is freed, for example when the occupying connection disconnects, the server must decide if it should be kept in its free executor list, or alternatively, be closed. The server configuration parameter `maxFreeExecutors` specifies the maximum number of free executors to keep around.

If the server has a positive, non-zero value for its configuration parameter `minFreeExecutors`, depending on the executor reuse scheme specified for the server, it may attempt to prestart enough executors to meet the minimum free executor limit specified. The server may check the minimum executor level whenever an executor is allocated from the free list, returned to the free list, or periodically if lost executor sweeping is enabled.

The server may also periodically check to see if any executors in its free list have been free for longer than the specified `srv.execTimeout`. Executors that have exceeded this limit will be candidates for termination. However, if the number of free executors is less than or equal to the server configuration parameter `minFreeExecutors`, the idle executor will not be terminated.

3.3.7 Lost Executor Sweep

The server may periodically check occupied executors to ensure that they are still viable.

If an executor process has been terminated unexpectedly or appears to be unresponsive, the server will try to ensure that the executor s OpenVMS process is terminated correctly, and the client connection will be flagged as lost. The next time the client attempts to carry out an action, a lost connection exception will be returned to them.

The server configuration parameter `srv.lostExecSweep` controls how often the server will do its executor viability checks.

The server may also use the lost executor sweep to ensure that the required minimum number of free executors are available. See Executor Reuse for more details.

3.3.8 Executor Balancing

Since release 7.3.2

By default, the Multi-process server uses a first-in/first-out (FIFO) scan of its free executor list to select the next executor process to allocate to the new connection request.

The way the free executor list is scanned to select the next executor process may be changed to suit your requirements; this is called *executor balancing.*

Executor balancing specifies the order in which the Multi-process server should select the next candidate from the free executor list and may be changed by specifying a value for the `srv.execBalance` configuration attribute for that server.

The `srv.execBalance` configuration attribute will accept either a numeric value or the appropriate keyword as shown in the table below. Keywords are not case sensitive.

The following table shows the possible balance values recognized by the Multi-process server:

Table 3-3 Executor Balancing

Value	Keyword	Description
0	Default or FIFO	Use a FIFO method to select the next free executor. This is the default balancing style.
1	MostMemory	Select the executor that has the most memory already allocated to it. This is determined by checking the PAGEFILE count for the executor process.
2	LeastMemory	Select the executor that has the least memory already allocated to it. This is determined by checking the PAGEFILE count for the executor process.
3	Oldest	Select the executor that has spent the most amount of time in the current free list.
4	Youngest	Select the executor that has spent the least amount of time in the current free list.

3.3.9 Executor Reuse

By default, the Multi-process server will reuse executors when they are free, however, the server s reuse of executors may be changed to suit your application and environmental requirements.
The `srv.execReuse` configuration attribute is used to set the type of executor reuse the server should carry out.

The following table shows the possible executor reuse values recognized by the Multi-process server:

Table 3-4 Reuse Executors

Value	Keyword	Description
0	Serial	Reuse executors. The scope of executor use is a single server connection request. This is the default reuse type.
1	Preemptive	Reuse executors. The scope of executor use is a single server connection request. Whenever an executor is allocated to a connection, and the number of free executors is below the minimum number set by `minFreeExecutors`, another executor will be started concurrently to replace the one used. If the lost executor sweeper is active, the number of free executors will be checked on each sweep cycle. If the number of free executors is below the minimum number set by

Value	Keyword	Description
		minFreeExecutors, executors will be started until the required minimum number of executors are present.
-1	None	Do not reuse executors. On disconnect, the executor will be destroyed, and if the number of free executors is below the minimum number set by minFreeExecutors, another executor will be started to replace the one destroyed.
-2	Fresh	Do not reuse executors. On disconnect, the executor will be destroyed, and if the number of free executors is below the minimum number set by minFreeExecutors, another executor will be started to replace the one destroyed. If the lost executor sweeper is active, the number of free executors will be checked on each sweep cycle. If the number of free executors is below the minimum number set by minFreeExecutors, executors will be started until the required minimum number of executors are present.
-3	Fresh_Preemptive	Do not reuse executors. Whenever an executor is allocated to a connection, and the number of free executors is below the minimum number set by minFreeExecutors, another executor will be started concurrently to replace the one used. On disconnect, the executor will be destroyed, and if the number of free executors is below the minimum number set by minFreeExecutors, another executor will be started to replace the one destroyed. If the lost executor sweeper is active, the number of free executors will be checked on each sweep cycle. If the number of free executors is below the minimum number set by minFreeExecutors, executors will be started until the required minimum number of executors are present.

The srv.execReuse configuration attribute will accept either a numeric value or the appropriate keyword as shown in the table above. Keywords are not case sensitive.

The different types of executor reuse are described in the following sections.

3.3.9.1 Executor Reuse Serial

When a client disconnects, the executor used for the connection will be returned to the server s free list to be reused later. This is termed serial reuse and is the default reuse behaviour.

Before returning the executor to the free list, the Multi-process server also checks the number of free executors currently in the free list. If there are already more free executors than the `maxFreeExecutors` server configuration attribute, the executor process will be terminated instead of being returned to the free list.

If the executor reuse is set to serial, the server will disregard any `minFreeExecutors` configuration attribute set.

3.3.9.2 Executor Reuse Preemptive

If the `srv.execReuse` configuration attribute is set to `preemptive` , the MP server will attempt to retain a minimum number of free executors as set by `minFreeExecutors.`

As executors are allocated to connections, if the number of free executors is below the `minFreeExecutors` value, the server will concurrently start a new executor to replace the allocated one.

If executor processes are prematurely terminated by any means and the lost executor sweeper is active, the server will attempt to restart enough executors to maintain the required minimum number.

Before returning the executor to the free list, the Multi-process server also checks the number of free executors currently in the free list. If there are already more free executors than the `maxFreeExecutors` server configuration attribute, the executor process will be terminated instead of being returned to the free list.

3.3.9.3 Executor Reuse None

There may be situations where you require pre-started and available executor processes, for example, to reduce the initial connection time, but require that a fresh OpenVMS process be used for each new connection.

In these situations, it is required that the executor process be pre-started prior to the connection request but be terminated after the connection has been released.

The `srv.execReuse` configuration attribute used in conjunction with the `minFreeExecutors` configuration attribute allows you to setup this type of configuration.

Setting the `srv.execReuse` configuration attribute to `none` , tells the MP server to terminate each executor after the client has disconnected from it. At this time, if the current number of free executors is less than the server s `minFreeExecutors` configuration attribute, and less than `maxFreeExecutors`, the server will prestart new executor processes until the `minFreeExecutors` number of free executor processes has been reached.

Note:

If the executor reuse is none , idle executors are not placed back on the free list and the associated OpenVMS processes will be terminated.

3.3.9.4 Executor Reuse Fresh

If the srv.execReuse configuration attribute is set to fresh , the MP server will terminate each executor after the client has disconnected from it. At this time, if the current number of free executors is less than the server s minFreeExecutors configuration attribute, and less than maxFreeExecutors, the server will prestart new executor processes until the minFreeExecutors number of free executor processes has been reached.

In addition, if the lost executor sweeper is active, the number of free executors will be checked on each sweep cycle. If the number of free executors is below the minimum number set by minFreeExecutors, executor processes will be started until the required minimum number of executor processes are present.

3.3.9.5 Executor Reuse Fresh_Preemptive

If the srv.execReuse configuration attribute is set to fresh_premptive , the MP server will terminate each executor after the client has disconnected from it. At this time, if the current number of free executors is less than the server s minFreeExecutors configuration attribute, and less than maxFreeExecutors, the server will prestart new executor processes until the minFreeExecutors number of free executor processes has been reached.

As executors are allocated to connections, if the number of free executors is below the minFreeExecutors value, the server will concurrently start a new executor to replace the allocated one.

If executor processes are prematurely terminated by any means and the lost executor sweeper is active, the server will attempt to restart enough executors to maintain the required minimum number.

Contents

3.4 Oracle JDBC for Rdb Pool Server

The Oracle JDBC for Rdb Pool server is a server-side component that accepts connection requests from the thin driver and redirects the requests to the next available Oracle JDBC for Rdb server for processing.

Using the Pool server, you can designate a single Port ID that can be used by client-side applications to connect to the next available server. The Pool server selects the next available server from a table of candidate servers in a round-robin fashion.

Once the connection request has been redirected, the thin driver and the designated server communicate directly with each other; the Pool server is no longer involved with that connection.

A Pool server is installed and invoked on each node from which you wish to direct the access to Oracle JDBC for Rdb servers. Oracle Rdb need not be present on these nodes, as the Pool server does not communicate directly with Oracle Rdb.

The Pool server and its pooled servers do not need to be on the same node, but they must be able to communicate with each other over TCP/IP. Firewalls and other network security devices may interfere with this communication.

The Pool server communicates with the thin driver using Java sockets over TCP/IP with the default Port ID 1702.

Note:

The Pool server conducts server pooling NOT connection pooling. Connections are created in each connection request and are not reusable.

3.4.1 Starting a Pool Server

A Pool server must be invoked on each node on which you wish to provide server redirection. The Pool server does not need to be on the same node as its pooled servers.

A Pool server may be invoked by using the appropriate start statement within the controller, as an Oracle SQL/Services JDBC dispatcher or directly from the operating system command line.

3.4.1.1 Starting a Pool Server from the Controller

A Pool server may be started from the controller by referencing a Pool server definition in an XML-formatted configuration file. See Starting Servers within Oracle JDBC for Rdb Controller for more details.

Format

```
START SERVER <server_name>
```

Example

Given the following server section in the XML-formatted configuration file `mycfg.xml`:

```
<server
    name="mypoolserver"
    type="RdbThinSrvPool"
    url="//localhost:1702/" >
    <pooledServer name="srv1forRdb"/>
    <pooledServer name="srv2forRdb"/>
    <pooledServer name="srvMPforRdb"/>
</server>
```

The following command may be used to start this server from within the controller:

```
rdbthincontrol> start server mypoolserver
```

Alternatively, the controller may be used in command mode to start a server:

```
$ java jar rdbthincontrol.jar -cfg mycfg.xml -name mypoolserver -startserver
```

3.4.1.2 Starting a Pool Server from Oracle SQL/Services

A Pool server may be started from the Oracle SQL/Services manager.

Using the Oracle SQL/Services manager, you must first establish a connection to the SQL/Services server. Once connected you may then start a JDBC dispatcher.

Before you can start a JDBC dispatcher, you must first create its definition in the Oracle SQL/Services environment.

See Oracle SQL/Services and Oracle JDBC for Rdb Servers for more details.

Format

```
START DISPATCHER <jdbc_dispatcher_name>
```

Example

```
$run sys$system:SQLSRV_MANAGE74
SQLSRV> connect server;
Connecting to server
Connected
SQLSRV> start disp JDBC_DISP;
SQLSRV>
```

See Oracle SQL/Services and Oracle JDBC for Rdb Servers for more details.

3.4.1.3 Starting a Pool Server from the Command Line

You may invoke a Pool server from the OpenVMS command line.

Format

```
$ java jar rdbthinsrvpool[<variant>].jar [-option]
```

Elements

The optional <variant> if present must be the value V5 , V6 or V8 , See JDK Versioned Oracle JDBC for Rdb Server Jars.

See Pool Server Configuration Options for a list of valid options.
Each option must have a hyphen ( ) prepended to it.

Example

```
$ java jar rdbthinsrvpool.jar cfg mycfg.xml
name mypoolserver
```

3.4.2 Pool Server Operation

Once started, by default, the Pool server will scan the list of pooled servers in a round-robin fashion to select the next available server. See the section Pool Balancing on how this scan order may be changed to suit your needs.

You can start and stop the servers in the pool at any time. If a server is not available, then the Pool server will bypass it. The Pool server also can automatically start up one or more pooled servers when the Pool server itself starts up.

During Pool server startup, a check is made on each server within its pool to see if the pooled server has the `autoStart` option enabled. If `autoStart` is enabled, then the command procedure pointed to by the `srv.startup` option of that pooled server will be executed. See Server Command Procedures for more details.

While the Pool server is running, it will periodically check to see that each pooled server within its pool of servers with `autoRestart` option enabled is still running. If `autoRestart` is enabled for a non-running pooled server, the command procedure pointed to by the `srv.startup` option of that pooled server will be executed to restart the server.

You can use the `srv.keepAliveTimer` option on Pool server start-up to specify the time between checks for non-running `autoRestart` servers. See Pool Server Configuration Options for more details.

If the Pool server is shutdown using the controller or the Oracle SQL/Services manager, then during server shutdown all pooled servers within the pool that were started by the pool server will also be shut down.

However, if the Pool server is terminated forcibly by stopping or deleting the OpenVMS process, it will not be able to carry out an orderly shutdown, in which case its pooled servers will not be shutdown automatically.

3.4.2.1 Pool Server Redirection and *failSAFE IP*

During connection redirection by the Pool server, the IP of the chosen pooled server will be returned to the thin driver so that it may redirect the client's connection request to that chosen server. As the DNS node name conversions may differ on the client and server node, the Pool server will implicitly convert any named nodes to IP addresses before returning the resultant IP to the thin driver.

The conversion to IP addresses may limit the failover to a standby address carried out by *failSAFE IP*. *failSAFE IP* is an optional service provided by TCP/IP Services on OpenVMS to allow IP addresses to fail over when nodes fail.

You may specify that the Pool server should not carry out the translation of named nodes to IP addresses during the connection redirection. This will then maintain the "logical" nature of the named IP and thus allowing *failSAFE IP* to correctly redirect to a standby node. See `srv.useLogicalIPs` in Pool Server Configuration Options for more details.

3.4.2.2 Server Pool Balancing

By default, the Pool server uses a round-robin scan of the pool to select the next candidate pooled server for allocation. The order in which the pool server scans the pool may be changed to suit your requirements; this is called *server pool balancing.*

Server pool balancing specifies the order in which the pool server should select the next candidate from the pool and may be changed by specifying a value for the `srv.balance` configuration attribute for that pool server.

The `srv.balance` configuration attribute will accept either a numeric value or the appropriate keyword as shown in the table below. Keywords are not case sensitive.

The following table shows the possible balance values recognized by the Pool server:

Table 3-5 Pool Balancing

Value	Keyword	Description
0	Default	Use a round-robin to select the next server from the pool. This is the default balancing style.
1	Memory	Select the server that has the least percentage of its total global shared memory allocation currently in use.

Value	Keyword	Description
2	Users	Select the server that has the least number of users currently connected. The servers are searched using round-robin. If the server has no clients currently connected it will be selected, otherwise the first server encountered with the least number of current clients will be selected.
4	Executors	Select the server that has the greatest number of free executors. This is only applicable to Multi-process pooled servers. Pooled servers that are not Multi-process are deemed to have zero (0) free executors.
8	FreeExecutors	Using round-robin, select the server that has at least one (1) free executor. If no server is found to have a free executor, then the next server is selected using round-robin.
16	Usage	Select the server that has the greatest number of free client slots available. The number of free slots is determined by subtracting the current number of clients connected from the maximum number of clients allowed (`maxClients`). The first server found by a round-robin search to have no clients currently connected will be selected irrespective of its `maxClients` setting. Servers that have no `maxClients` limit will be selected as primary candidates, in which case the server with the least number of current clients will be selected.
32	PoolOrder	***Since release 7.3.4.0.5*** Using the implicit order of the pooled server pool (i.e. the order that the pooled server names are specified in the Pooled Server Subsection of the pool server s definition), select the first server that has not reached its maximum number of clients allowed (`maxClients`).

3.5 Oracle JDBC for Rdb Manager Server

Since release 7.3.2

The Oracle JDBC for Rdb Manager server is a server-side component that services JDBC management requests. The Manager server is multi-threaded, able to handle multiple management requests at the same time.

As the JDBC thin controller can only start JDBC servers on the node that the controller is executing on, the Manager server has been introduced to allow applications like the controller to start JDBC servers on other nodes.

A Manager server cannot be used to access databases. Its purpose is to provide a JDBC management capability on nodes that may be remote to where the controller application is running, and its main function is to handle the starting and maintenance of JDBC servers on those nodes.

A Manager server is installed and invoked on each node on which you wish to start JDBC servers. If these JDBC servers will be accessing Rdb databases, then Oracle Rdb must also be installed and running on those same nodes.

The Manager server communicates with management applications using Java sockets over TCP/IP with the default Port ID 2060.

3.5.1 Utilizing a Manager Server

The main purpose of the Manager server is to provide a mechanism that will allow JDBC servers to be started up on nodes remote to the one the controlling application is running.

The Manager server can also be used to automatically start servers on its local host and to periodically check to ensure each server is operational. If the Manager server finds a failed server, it can attempt to restart that server. See Manager Server Operations for more details.

Although any JDBC server running on your network may be stopped by the controller (assuming the controller has the appropriate server control password for that server and has network access to that node), the invocation or startup of a JDBC server using the controller is limited to the local node on which the controller application is running. This is because an OpenVMS process must be started for the server to run within, which is more difficult when the node on which the process is to be started is remote to the invoking application.

Oracle JDBC for Rdb does not use features such as Java RMI to invoke remote events as this may introduce greater security risks on your network. Thus, without any additional components, the controller can only start server processes locally.

To provide a controlled mechanism to start remote server processes, Oracle JDBC for Rdb introduced the Manager server.

Although the Manager server itself needs to be started on the remote node, which is a necessary bootstrapping step, once it is started it can service requests from the remote controller application to startup new JDBC server instances.

The Manager server will only respond to management requests made from applications that have connected as control users. That is, the application must know the Manager server s control password to connect successfully to it.

To utilize a Manager server for remote server startup, the following steps are required:

1. Startup an instance of a Manager server on the remote node you wish to later start JDBC servers on. See Starting a Manager Server for more details.
2. Add a Manager server entry to the local configuration file that you will be using with the controller.
3. Request the controller to start the remote server.

3.5.2 Starting a Manager Server

A Manager server may be started by using the appropriate start statement within the controller, as an Oracle SQL/Services JDBC dispatcher or directly from the operating system command line.

The configuration file used to start the Manager server should also contain server definitions for all JDBC servers that you wish to be able to start remotely using this Manager server, otherwise the DEFAULT server definition found in this configuration file will be used.

3.5.2.1 Starting a Manager Server from the Oracle JDBC for Rdb Controller

A Manager server may be started from the controller by referencing a Manager server definition in an XML-formatted configuration file. See Starting Servers within Oracle JDBC for Rdb Controller for more details.

Example

Given the following server section in the XML-formatted configuration file mycfg.xml:

```
<server
    name="manserv1"
    type="RdbManSrv"
    url="//localhost:2060/"
    logfile="myLogs:manserv1.log"
/>
```

The following command may be used to start this server from within the controller:

```
rdbthincontrol> start server manserv1
```

Alternatively, the controller may be used in command mode to start a server:

```
$ java jar rdbthincontrol.jar -cfg mycfg.xml -name manserv1 -startserver
```

3.5.2.2 Starting a Manager Server from Oracle SQL/Services

A Manager server may be started from the Oracle SQL/Services manager.

Using the Oracle SQL/Services manager, you must first establish a connection to the SQL/Services server. Once connected you may then start a JDBC dispatcher.

Before you can start a JDBC dispatcher, you must first create its definition in the Oracle SQL/Services environment.

The associated JDBC server type for this dispatcher definition must be set to RdbManSrv for a Manager server.

See Oracle SQL/Services and Oracle JDBC for Rdb Servers for more details.

Format

```
START DISPATCHER <jdbc_dispatcher_name>
```

Example

```
$run sys$system:SQLSRV_MANAGE74
SQLSRV> connect server;
Connecting to server
Connected
SQLSRV> start disp JDBC_MANDISP;
SQLSRV>
```

3.5.2.3 Starting a Manager Server from the Command Line

You may invoke a Manager server from the OpenVMS command line.

Instead of placing several options on the command line, you may wish to create a server definition within an XML-formatted configuration file and then start the server using its server name.

The server type for this server definition must be set to RdbManSrv for a Manager server.

Manager servers are invoked using the `rdbmansrv` JAR.

Format

```
$ java jar rdbmansrv[<variant>].jar [<-option>]
```

Elements

The optional <variant> if present must be the value V5 , V6 or V8 , See JDK Versioned Oracle JDBC for Rdb Server Jars.

See Server Configuration Options for a list of valid `<-option>`s. Remember that on the DCL command line, each configuration option must have a hyphen `(-)` prepended to it.

Remarks

By default, the server is assumed to be of type RdbManSrv.

See XML formatted Configuration File for more details on server definitions within configuration files.

Example 1

```
$ java jar rdbmansrv.jar port 2666
```

Example 2

Given the following server section in the XML-formatted configuration file `mymancfg.xml`:

```
<server
    name="manserv1"
    url="//localhost:2060/"
    logfile="myLogs:manserv1.log"
/>
```

The following method may be used to start this Manager server:

```
$ java jar rdbmansrv.jar cfg mymancfg.xml name manserv1
```

3.5.3 Adding a Remote Manager Server Configuration Entry

For the controller to know where Manager servers can be found, you must add one or more entries to your controller configuration file specifying the name, and URL of the Manager server or servers.

Once defined, this information will be used by the controller to connect to the appropriate Manager server and request that the designated remote server should be started.

The Manager server entry need only contain the name, and URL of the Manager server:

```
<server
    name="manserv1"
    url="//myremotenode.adir.com:2060/"
/>
```

Once defined you may then use the controller to start remote servers on the same node the Manager server is running.

3.5.4 Using the Controller to Start Remote Servers

If a remote node has a Manager server currently running, you can start a JDBC server on that node by utilizing the VIA qualifier of the controller start command.

The name of the server you wish to start will be sent to the Manager server, which in turn will lookup that name within the server definitions of its own configuration file to determine the server characteristics to use. It will then use these characteristics to startup that server on behalf of the controller.

Example 1

Assuming the local controller configuration file `mycfg.xml` contains the following server definition:

```
<server
    name="manserv1"
    url="//myremotenode.adir.com:2060/"
/>
```

And that the remote Manager server was started on node `myremotenode.adir.com`
using a configuration file that included the following server definitions:

```
<server
    name="manserv1"
    type="RdbManSrv"
    url="//localhost:2060/"
    controlpass="MYCONTROL"
/>
<server
    name="mysrv1"
    type="RdbThinSrv"
    url="//localhost:1701/"
/>
```

The following local command will start up a thin server called mysrv1 listening on port 1701 on node `myremotenode.adir.com`:

```
$ java -jar rdb$jdbc_home:rdbthincontrol.jar -cfg mycfg.xml
rdbthincontrol> start server mysrv1 via manserv1 jones MYCONTROL
rdbthincontrol>
```

Example 2

If you have a session control password in the configuration file that is the same as the control password of the Manager server then you do not need to give the control password on the command line.

Assuming the local controller configuration file `mycfg.xml` contains the following session and server definitions:

```
<session
    name="DEFAULT"
    controlUser="jones"
    controlPass="MYCONTROL" />

<server
    name="manserv1"
    url="//myremotenode.adir.com:2060/" />
```

And given the same remote configuration file used for the remote Manager server setup as shown in *Example 1*, the following local command will start up a thin server called mysrv1 listening on port 1701 on node `myremotenode.adir.com`:

```
$ java -jar rdb$jdbc_home:rdbthincontrol.jar -cfg mycfg.xml
rdbthincontrol> start server mysrv1 via manserv1
rdbthincontrol>
```

Oracle recommends not to store passwords in your configuration file, however if you choose to store them then an obfuscated form should be used. You may use the `obfuscate` function within the Controller application to generate an obfuscated password that is suitable to use with the `controlpass` property. See Password Obfuscation in Server Configuration Files for more details.

3.5.5 Manager Server Operations

Once started, by default, the Manager server will scan the list of servers found in its configuration file to determine if any require starting.

During the Manager server startup, a check is made on each server definition within its configuration file to locate any servers that have `autoStart` option enabled. For each server that has a URL indicating that it should run on the same host on which the Manager server is running, and that has `autoStart` enabled, the Manager server will attempt to start that server using the command procedure pointed to by the `srv.startup` option for that server. See Server Command Procedures for more details.

The Manager server will periodically check each server found in its configuration file. If a server has a URL configuring it for the same host as the one on which the manager server is running and has the `autoRestart` option enabled, it will verify that the server is still running and restart it if needed.

The Manager server will use the command procedure pointed to by the `srv.startup` option for that server to restart the server.

You can use the `srv.keepAliveTimer` option on Manager server start-up to specify the time between checks for non-running `autoRestart` servers. See Manager Server Configuration Options for more details.

You may also use the `srv.keepAliveWait` option on Manager server start-up to specify the time the Manager server will wait for a newly started server to respond that it has started successfully. See Manager Server Configuration Options for more details.

If the Manager server is shut down, any servers it has started will continue to run. You may use the controller to stop these servers if required.

Contents

Chapter 4
Server Configuration

There are several configuration options that apply to Oracle JDBC for Rdb servers that may be used as command line options or as server options inside a configuration file.

See Configuration Files for more details on how to use these options within a configuration file.

The following sections detail the configuration options and files:

- Server Configuration Options
- Multi-process Server Configuration Options
- Pool Server Configuration Options
- Manager Server Configuration Options
- Configuration Files

Contents

4.1 Server Configuration Options

The following server configuration options may be used on the command line or in configuration files in conjunction with standard thin servers:

Table 4-1 Server Configuration Options

Option	Default	Description
`anonymous`	**false**	If **true** , tells the server to allow anonymous connections, that is, connections where the user and password are not specified. Depending on how the Oracle Rdb database has been set up, Oracle Rdb may allow connection to the database without a username being explicitly specified, in which case the characteristics of the authorization account of the server invoker will be used by Oracle Rdb to determine database access. This switch may be used in conjunction with the *password* and *user* configuration options to provide default authorization on connections. By default, anonymous connections are disabled, and the client must specify a valid username and password combination to access the Rdb database.

Option	Default	Description
allowAccessToCL	**false**	If **true** , indicates that users may be allowed access to Command Line operations on the system on which the server is executing. This option should only be used within an XML formatted configuration file. See Access to the Command Line for more details.
allowAccessToRoot	**false**	***Since release 7.3.4*** If specified, indicates that users may be allowed access to the server root. Server root access is required to conduct operations such as CREATE and DROP databases. If the specified value is **true** , all users will have the ability to connect to the server root. If the specified value is **priv** , only privileged users will have the ability to connect to the server root. This option should only be used within an XML formatted configuration file. See Access to the Server Root for more details. By default, server root access is disabled.
allowCreateDatabase	**false**	***Since release 7.3.4*** If specified, indicates that the server will allow appropriate users the ability to create new Rdb databases. If the specified value is **true** , all users will have the ability to create new database if allowed by OpenVMS and Rdb. If the specified value is **priv** , only privileged users will have the ability to create new databases. See Create and Drop Database for more details. By default, create database is prohibited.
allowDropDatabase	**false**	***Since release 7.3.4*** If specified, indicates that the server will allow appropriate users the ability to drop existing Rdb databases.

Option	Default	Description
		If the specified value is **true** , all users will have the ability to drop databases if allowed by OpenVMS and Rdb. If the specified value is **priv** , only privileged users will have the ability to drop databases. See Create and Drop Database for more details. By default, drop database is prohibited.
`allowShowDatabases`	**false**	If specified **true** , indicates that the server will respond to user requests for a list of databases that are known to the server. The list of known named databases is specified in the Database section of the configuration file. See Named Databases for more details.
`autorestart`	**false**	If specified **true** , indicates to any Pool server that may include this server in its pool of servers to automatically restart this pooled server. See Oracle JDBC for Rdb Pool Server for more details. Manager servers may also restart servers that have this option enabled. See Oracle JDBC for Rdb Manager Server for more details. This option is only valid in an XML formatted Configuration File.
`autostart`	**false**	If specified **true** , indicates to any Pool server that may include this server in its pool of servers to automatically start up this pooled server. See Pool Server Operation for more details. Manager servers may also start servers that have this option enabled. See Oracle JDBC for Rdb Manager Server for more details. This option is only valid in an XML formatted Configuration File.

Option	Default	Description
`b or buffersize` ***send_buffer_size***	**see description**	Provides a hint to the server on sizing of the underlying network I/O buffers. Increasing buffer size can increase the performance of network I/O for high-volume connections, while decreasing it can help reduce the backlog of incoming data. The default buffer size is the current default network buffer size for TCP/IP set on the server system.
`bypass`	**false**	If specified **true** , then if the privilege is available, bypass will be an allowable privilege for the server process. Rdb checks for this privilege to determine the access rights to databases and database objects. If enabled, all validated users connected to databases via this server instance will be considered to have bypass privilege. The default is **false** where the bypass privilege is disabled for the server by default. Validated users who already possess the bypass privilege will still have bypass available. See BYPASS Privileges for more details.
`cfg or configfile` ***file_specification***	**none**	The file specification of a configuration file where server attributes may be found. Attributes set in this configuration file may be overridden by setting the same attribute at the command line level. If the file extension is XML, the configuration parameters are held in a XML format. See Configuration Files for more details. By default, no configuration file is used.
`cli.idleTimeout` ***timeout***	0	Sets the maximum time, in milliseconds, a client connection may be idle. If no operation is conducted using this connection within the time specified, the connection will be forcibly disconnected. A value of zero (0) means unlimited idle time allowed.

Option	Default	Description
		See Client connection timeout for more details.
`Controlpass` ***`control_password`***	**none**	Specifies the password that control users must use to be able to issue control commands on this server instance. This password may be either plain text or an obfuscated password value. See Control Password for more information on this password.
`fs or fetchsize` ***`default_fetch_size`***	100	Specifies the default fetch size to use. The *fetchsize* value provides a hint to the server indicating the number of records to retrieve and send back to the client at the one time. Increasing the *fetchsize* value may improve the network performance by reducing the average network overhead per record retrieved.
`Lockwait` ***`lock_wait`***	-1	Specifies the maximum number of seconds to wait on getting a record lock. This switch, used in conjunction with *maxtries* and *trywait*, specifies how often and how long to try to get a lock on a locked object before issuing a locked object exception. A value of minus one (-1) means wait indefinitely.
`log or logfile` ***`file_specification`***	**console**	Specifies the file specification of the log file for this server. If trace is enabled, the trace messages will be written to this file instead of the console. By default, trace messages will be written to the console.
`Maxclients` ***`max_num_of_clients`***	-1	Specifies the maximum number of concurrent clients this server instance may manage. A value of minus one (-1) means allow an unlimited number of clients.

Option	Default	Description
Maxtries ***max_num_of_lock_attempts***	10	Specifies the maximum number of times to try to get a record lock. This switch, used in conjunction with *lockwait* and *trywait*, specifies how often and how long to try to get a lock on a locked object before issuing a locked object exception.
Name ***server_name***	**see description**	Specifies a name for this server instance. This name need not be unique; however, the name may be used to lookup server information within the start-up configuration file. The value of this name is not case-sensitive. If not specified, a name will be created for the server based on the server type.
p or port ***port_num***	1701	Tells the server to listen on port ***port_num***.
pw or password ***default_user_password***	**none**	Used in conjunction with the *user* and *anonymous* switches, provides the password to use on an anonymous connection.
Persona ***username***	**none**	Specifies the Operating system username, which the process running the server will assume. If not specified, *persona* will not be used. See Persona for more details.
relay	**false**	If specified **true** , designates that this server should relay poll requests to all active servers in its network community.
restrictAccess	**false**	Used in conjunction with *allow*, *deny* and other options to restrict access to designated databases, users, and operations. This option should only be used within an XML formatted configuration file. See Restricting Server, Database and Operational Access for more details.
restrictSQL ***sql_verb_list***	**none**	Used to restrict allowable SQL statements to only those that start with the one of the verbs specified in the comma separated verb list.

Option	Default	Description
		This option should only be used within an XML formatted configuration file. See Restricting SQL Statements for more details.
`retainRdbSQLState`	**false**	***Since release 7.3.2*** If specified **true** , designates that this server should retain the SQLState values returned by Rdb and pass them unchanged to the client. The default behaviour is to alter the SQLState values returned under some circumstances, to a generic S1000 state.
`srv.bindTimeout` ***`timeout`***	0	Sets the timeout, in milliseconds, on waiting for a database connection to complete. If the database fails to connect within this time, an exception will be raised. A value of zero (0) means unlimited timeout.
`srv.cliStartup` ***`file_specification`***	**see description**	Specifies the start-up batch or command file that will be used to execute any CLI statements the server issues. If not specified, `rdb$jdbc_home:rdbjdbc_execcli.com` will be used. See Server Command Procedures for more details.
`srv.idleTimeout` ***`timeout`***	0	Sets the maximum time, in milliseconds, the server will wait for a new client connection request. If no new connection is made within the timeout period, the server will be closed due to inactivity. A value of zero (0) means unlimited idle time allowed. See Server Inactivity Timeout for more details.
`srv.mcBasePort` ***`base_port`***	5517	Specifies the base port number that will be used for multicast operations. A value of zero (0) will disable multicast operations.
`srv.mcGroupIP` ***`group_ip`***	**239.192.1.1**	Specifies the multicast IP group within which this server will participate.

Option	Default	Description
`srv.networkKeepAlive`	**false**	If specified **true** , the socket used to connect to the client will have **`SoKeepAlive`** enabled. See the operating system s socket documentation for more information on **`SoKeepAlive`** **(TCP** option **`SO_KEEPALIVE`**).
`srv.onStartCmd` ***command***	**none**	Specifies a DCL command statement that should be executed prior to starting up a server. The specified command will only be executed if the server is started using the controller or by a Pool server. See On Start Commands for more details.
`srv.password` ***server_password***	**none**	Specifies an additional password that clients need to provide before they may use the server for database connections. See Further server access protection for more details.
`srv.showPoll`	**false**	If specified **true** , information about POLL requests received should be traced, if server action tracing has been enabled. See Trace for further information.
`srv.startup` ***file_specification***	**see description**	Specifies the start-up batch or command file that will be used by the controller to start the process for this server. If not specified, `rdb$jdbc_home:rdbjdbc_startsrv.com` will be used. See Server Command Procedures for more details.
`tl or tracelevel` ***trace_level***	0	Sets the trace level for debugging purposes. See Trace for further information.
`Transaction` ***transaction_option***	**none**	***Since release 7.3.4.0.4*** Specifies the default transaction behaviour for all connections using this server. The value **autofetch** is the only transaction option recognized and specifies that transactions should be committed when the server sends FETCHSIZE groups of records back to the client.

Option	Default	Description
		See the section Default Transaction for more information on the **autofetch** transaction option.
`tracelocal`	**false**	If specified **true** , only local server base tracing should be enabled. If this option is set, any *tracelevel* values specified by a client connection will not affect the trace level of the server components.
`Trywait` ***wait_time***	10	Specifies the time in milliseconds to wait between lock tries. This switch, used in conjunction with *maxtries* and *lockwait*, specifies how often and how long to try to get a lock on a locked object before issuing a locked object Exception. A value of zero (0) or a negative value, indicates not to wait between lock tries.
`Type` ***server_type***	**RdbThinSrv**	Specifies the server type of this server. Valid values are: . `RdbThinSrv` - standard thin server. . `RdbThinSrvSSL` - thin server using SSL for communication. . `RdbThinSrvMP` Multi-process server. . `RdbThinSrvMPSSL` Multi-process server using SSL.
`u or user` ***default_user_name***	**none**	Used in conjunction with the *password* and *anonymous* switches, provides the username to use on an anonymous connection.
`url` ***connection_URL***	**none**	Specifies the node IP and port this server will run on. This switch overrides any *port* switch. The format of the <connection URL> is `//<node>:<port>/`

The following server configuration options may only be used in XML-formatted configurations:

Table 4-2 Server Configuration Options2

Option	Default	Description
<allow database="***database-name***">	**none**	***Since release 7.3.2***
		Specifies the name of a database to which this server will allow access. This is used in conjunction with the *restrictAccess* option. The named database should also be described in the same configuration file. A separate *allowDatabase* option should be used for each database to which this server will allow access. **Note**: This form of the *allow database* option is only available for release 7.3.2 of Oracle JDBC for Rdb and later. See Restricting Database Access for more details.
<allow IP="***ip-mask***">	**none**	***Since release 7.3.2***
		Specifies the IPs this server will allow access from. This is used in conjunction with the *restrictAccess* option. A separate *allowIP* option should be used for each IP or IP group from which this server will allow access. **Note**: The *allow IP* option is only available in release 7.3.2 of Oracle JDBC for Rdb and later. See Restricting IP Access for more details.
<allow privuser="***user-name***">	**none**	***Since release 7.3.2***
		Specifies the usernames to which this server will allow special access. This is used in conjunction with options such as the *allowAccessToCL* option. A separate *allowPrivUser* option should be used for each username to which this server will allow special access. **Note**: this form of the *allow privuser* option is only available in release 7.3.2 of Oracle JDBC for Rdb and later.

Option	Default	Description
		See Privileged Users for more details.
<allow user="***user-name***">	**none**	***Since release 7.3.2***
		Specifies the usernames to which this server will allow database access. This is used in conjunction with the *restrictAccess* option. A separate *allowUser* option should be used for each user to which this server will allow database access. This option should only be used within an XML formatted configuration file. **Note**: this form of the *allowUser* option is only available in release 7.3.2 of Oracle JDBC for Rdb and later. See Restricting User Access for more details.
<allowDatabase name="***db-name***">	**none**	Specifies the name of a database to which this server will allow access. This is used in conjunction with the *restrictAccess* option. The named database should also be described in the same configuration file. A separate *allowDatabase* option should be used for each database to which this server will allow access. See Restricting Database Access for more details.
<allowIP IP="***ip-mask***">	**none**	***Since release 7.3.2***
		Specifies the IPs this server will allow access from. This is used in conjunction with the *restrictAccess* option. A separate *allowIP* option should be used for each IP or IP group from which this server will allow access. **Note**: The *allow IP* option is only available option is only available in release 7.3.2 of Oracle JDBC for Rdb and later.

Option	Default	Description
		See Restricting IP Access for more details.
`<allowPrivUser name="`***`user-name`***`">`	**none**	Specifies the usernames to which this server will allow special access. This is used in conjunction with options such as the *allowAccessToCL* option. A separate *allowPrivUser* option should be used for each username to which this server will allow special access. See Privileged Users for more details.
`<allowUser name="`***`user-name`***`">`	**none**	Specifies the usernames to which this server will allow database access. This is used in conjunction with the *restrictAccess* option. A separate *allowUser* option should be used for each user to which this server will allow database access. This option should only be used within an XML formatted configuration file. See Restricting User Access for more details.
`<deny IP="`***`ip-mask`***`">`	**none**	***Since release 7.3.2*** Specifies the IPs from which this server will deny access. This is used in conjunction with the *restrictAccess* option. A separate *deny IP* option should be used for each IP or IP group from which this server will deny access. **Note**: The *deny IP* option is only available in release 7.3.2 of Oracle JDBC for Rdb and later. See Restricting IP Access for more details.
`<deny sql="`***`sql pattern`***`">`	**none**	Used to restrict allowable SQL statements to only those that do not match the provided regular expression pattern. This option should only be used within an XML formatted configuration file.

Option	Default	Description
		See Restricting SQL Statements for more details.
<deny user="***user-name***">	**none**	***Since release 7.3.3*** Specifies the usernames to which this server will deny database access. This is used in conjunction with the *restrictAccess* option. A separate *deny User* option should be used for each user to which this server will deny database access. This option should only be used within an XML formatted configuration file. **Note**: the *deny user* option is only available in release 7.3.3 of Oracle JDBC for Rdb and later. See Restricting User Access for more details.
<enableEvent name="***event-name***">	**none**	***Since release 7.3.1*** Specifies the events this server should log. This is used in conjunction with the *Events* configuration section. A separate *enableEvent* option should be used for each event this server will log. This option should only be used within an XML formatted configuration file. See Event Logging and Notification for more details.

See Multi-process Server Configuration Options for additional options that may be used with Multi-process servers.

See Pool Server Configuration Options for the options that may be used with Pool servers.

Contents

4.2 Multi-process Server Configuration Options

In addition to the options shown in Server Configuration Options , the following configuration options may be used on the command line or in configuration files in conjunction with Multi-process servers:

Table 4-3 Multi-process Server Configuration Options

Option	Default	Description
maxFreeExecutors ***max_num_of_free_executors***	0	Specifies the maximum number of free (unused) executor processes that may be maintained while the server is running.
minFreeExecutors ***min_num_of_free_executors***	0	***Since release 7.3.2*** Specifies the minimum number of free (unused) executor processes that should be maintained during executor cleanup due to excess idle time, or when the executor reuse is set to **none** . See Executor Maintenance and Executor Reuse for more details.
prestartedExecutors ***num_of_prestarted_executors***	0	Specifies the number of executor processes to start up when the Multi-process server starts.
sharedMem ***size_in_KB***	1024	Specifies the amount of global shared memory (in KB) that should be allocated by the server.
srv.execPrefix ***prefix***	**see description**	Specifies the prefix to use for executor names. If not specified, a standard prefix based on server name will be used. See Executor Naming for more details.
srv.execStartup ***file_specification***	**see description**	Specifies the start-up batch or command file that will be used to start the subprocess for each client connection. If not specified, `rdb$jdbc_home:rdbjdbc_startexec.com` will be used. See Server Command Procedures for more details.
srv.execBalance ***balance***	0	***Since release 7.3.2*** Sets the executor balancing to be used. A value of zero (0) means FIFO. See Executor Balancing for more details.
srv.execReuse ***reuse***	0	***Since release 7.3.2*** Sets the executor reuse to be used. A value of zero (0) means Serial reuse. See Executor Reuse for more details.

Option	Default	Description
`srv.execTimeout` ***timeout***	0	Sets the timeout, in milliseconds, that an unused executor process can remain idle in the free executor queue before being terminated. A value of zero (0) means unlimited timeout.
`srv.lostExecSweep` ***wait_time***	30000	***Since release 7.3.1*** Specifies the time, in milliseconds, to wait before checking for lost executor processes again. The default is 30 seconds. A value of zero (0) will disable lost executor sweeps.
`srv.mpMaxTries` ***maximum_tries***	500	Specifies the number of times the server should try to synchronize handshake with executor before giving up.
`srv.mpTryWait` ***wait_time***	10	Specifies the time in milliseconds to wait between server/executor handshake synchronization tries.
`srv.onExecStartCmd` ***command***	**none**	Specifies a DCL command statement that should be executed prior to starting up an executor. See On Start Commands for more details.
`srv.usecreprc`	**false**	***Since release 7.3.4.1.2*** If **true** , JDBC will use SYS$CREPRC to create processes as detached. If **false** (the default), JDBC will use internal Java methods to create subprocesses. See Executor Process Startup for more details.

See Executor Maintenance for more information on executor housekeeping carried out by the Multi-process server.

Contents

4.3 Pool Server Configuration Options

The valid configuration options that may be used with a Pool server can be found in the following table:

Table 4-4 Pool Server Configuration Options

Option	Default	Description
`cfg or configfile` ***configuration_filename***	**none**	The file specification of a configuration file where server attributes may be found. Attributes set in this configuration file may be overridden by setting the same attribute at the command line level. If the file extension is XML, the configuration parameters are held in an XML format. By default, no configuration file is used. See Configuration Files for more details.
`Controlpass` ***control_password***	**none**	Specifies the password that control users must use to be able to issue control commands on this server instance. See Control Password for more information on this password.
`log or logfile` ***file_specification***	**console**	Specifies the file specification of the log file for this server. If trace is enabled, the trace messages will be written to this file instead of the console. By default, trace messages will be written to the console.
`node<n>` ***node***	**none**	Specifies the node on which the thin server number **<n>** resides. This option is not valid for use in XML-formatted configuration files.
`poolserver`	**none**	Specifies that the server should function as a Pool server. This is a mandatory option if used on the command line or a non-XML formatted configuration file.
`poolsize` ***pool_size***	**none**	Specifies the number of thin servers that will be specified. This is a mandatory option if used on the command line or a non-XML formatted configuration file.
`port<n>` ***port_num***	**none**	Specifies the port for the thin server number **<n>** in server list.

Option	Default	Description
		This option is not valid for use in XML-formatted configuration files.
`p or port` ***port_num***	1701	Tells the Pool server to listen on port **port_num**.
`srv.balance` ***balance_type*** `or` `srv.balance` ***balance_keyword***	0 **Default**	Specifies the server pool balancing to use. The option value may be a numeric literal or a keyword. See Server Pool Balancing for more details.
`srv.keepAliveTimer` ***seconds***	60	Sets the time, in seconds, of the duration between Pool server checks for non-running pooled servers that have autoRestart enabled. See Oracle JDBC for Rdb Pool Server for more details.
`srv.mcBasePort` ***base_port***	5517	Specifies the base port number that will be used for multicast operations. A value of zero (0) will disable multicast operations.
`srv.mcGroupIP` ***group_ip***	**239.192.1.1**	Specifies the multicast IP group within which this server will participate.
`srv.password` ***server_password***	**none**	Specifies an additional password that clients need to provide before they may use the server for database connections. See Further server access protection for more details.
`srv.useLogicalIPs`	**false**	Only Valid for Pool servers. Specifies that the server should not translate named IP values to IP addresses prior to redirecting connection request. See Using OpenVMS failSAFE IP for more details.
`tl or tracelevel` ***trace_level***	0	Sets the trace level for debugging purposes. See Trace for further information.
`type` ***server_type***	**RdbThinSrvPool**	Specifies the server type of this server. Valid values are:

Option	Default	Description
		. **RdbThinSrvPool** - Pool server. . **RdbThinSrvPoolSSL** - Pool server using SSL.
`url` ***connection_URL***	**none**	Specifies the node IP and port this server will run on. This switch overrides any *port* switch.

The valid configuration options that may be used in conjunction with a Pool server, and used only within an XML-formatted configuration file, can be found in the following table:

Table 4-5 Pool Server Configuration Options2

Option	Default	Description
`<pooledserver name="`***server-name***`">`	**none**	Specifies the name of a server that will take part in the pool. The named server should also be described in the same configuration file.

As there may be several servers listed in the server pool, Oracle advises to use the configuration file to specify these options.

If you are using a standard configuration file, the number of servers in the pool is specified by the *poolsize* option. In the case of an XML-formatted configuration file, the number of servers in the pool will be the same as the number of *PooledServer* subsections within the server definition.

Each server participating in the pool must have both a node and a port id specified for it.

See Configuration Files for examples of configuring a Pool server.

Contents

4.4 Manager Server Configuration Options

The valid configuration options that may be used with a Manager server can be found in the following table:

Table 4-6 Manager Server Configuration Options

Option	Default	Description
`cfg or configfile` ***configuration_filename***	**none**	The file specification of a configuration file where server attributes may be found. Attributes set in this configuration file may be overridden by setting the same attribute at the command line level.

Option	Default	Description
		If the file extension is XML, the configuration parameters are held in an XML format. By default, no configuration file is used. See Configuration Files for more details.
`Controlpass` ***`control_password`***	**none**	Specifies the password that control users must use to be able to issue control commands on this server instance. See Control Password for more information on this password.
`log or logfile` ***`file_specification`***	**console**	Specifies the file specification of the log file for this server. If trace is enabled, the trace messages will be written to this file instead of the console. By default, trace messages will be written to the console.
`p or port` ***`port_num`***	1701	Tells the server to listen on port **port_num**.
`srv.keepAliveTimer` ***`seconds`***	60	Sets the time, in seconds, of the duration between Manager server checks for non-running servers that have autoRestart enabled. See Oracle JDBC for Rdb Manager Server for more details.
`srv.keepAliveWait` ***`seconds`***	60	Sets the time, in seconds, the Manager server will wait for a newly started server to confirm it has started correctly, before raising a failed server event. See Oracle JDBC for Rdb Manager Server for more details.
`srv.mcBasePort` ***`base_port`***	5517	Specifies the base port number that will be used for multicast operations. A value of zero (0) will disable multicast operations.
`srv.mcGroupIP` ***`group_ip`***	**239.192.1.1**	Specifies the multicast IP group within which this server will participate.
`tl or tracelevel` ***`trace_level`***	0	Sets the trace level for debugging purposes. See Trace for further information.

Option	Default	Description
`type` ***server_type***	**RdbManSrv**	Specifies the server type of this server. Valid values are: . **`RdbManSrv`** - Manager server. . **`RdbManSrvSSL`** Manager server using SSL for communication.
`url` ***connection_URL***	**none**	Specifies the node IP and port this server will run on. This switch overrides any *port* switch.

See Configuration Files for examples of configuring a Manager server.

Contents

4.5 Configuration Files

Instead of setting the switches on the command line, you can specify a configuration file that details the settings.

Two formats of configuration files are recognized:

- Standard Java Properties load file
- XML-formatted file

4.5.1 Standard Properties File

The following section describes the use of configuration file formatted as a standard Java Properties load file. See XML Formatted Configuration File for details on using an XML-formatted configuration file.

Most of the server configuration options as specified in configuration options tables can be used but with the following changes:

- Each keyword requires a value, even those that do not have values on the command line; these options are considered Booleans and thus should have the appropriate **true** value.
- Each keyword must be separated from its value by an equal sign `(=)`.

The `cfg` switch on the command line allows you to specify the file specification of this configuration file:

Format

```
$ java jar rdb$jdbc_home:rdbthinsrv.jar cfg thinsrv.cfg
```

Example

Java style comments and empty lines may be included in the file, for example:

```
   //
   // configuration file for our thin server
   //
   // the default port for the thin server is 1701 but we
   // want it to listen on another port

   port=1708

   // allow anonymous connections

   anonymous=true

   // enable password display
   showpass=true

   // limit the number of clients
   maxclients=10

   // set the locking keywords
   lockwait=2
   maxtries=20

   // end of config file
```

In addition, the configuration file for a Pool server should contain information about the list of servers to which it may delegate connection requests, for example:

```
   //
   // configuration file for Pool server
   //
   // the default port for the Pool server is 1702
   port=1702

   // show is a Pool server and the poolsize
   //(number of subservient servers)
   poolserver=true
   poolsize=4

   // now add the servers
   node1=MYNODE1
   port1=1704

   node2=MYNODE1
   port2=1705

   node3=MYNODE1
   port3=1706

   node4=MYNODE2
   port4=1704

// end of config file
```

4.5.2 XML-Formatted Configuration File

Instead of setting the switches on the command line, you can specify an XML-formatted configuration file that details the settings of these switches. The XML-formatted configuration file allows a greater number of configuration options to be specified than the standard CFG file and is the recommended configuration file format.

The XML-formatted configuration file differs from the standard CFG file in that it may contain information about multiple servers in the same configuration file.

Each server is specified within a separate server section and must be given a unique name. This name is used to get default configuration information about the server when the server starts-up, as well as how a server may be identified on your system and within the controller interface.

The `cfg` switch on the command line allows you to specify the file specification of this file.

The server configuration options as specified in configuration options tables can be used but with the following changes:

- Each keyword requires a value, even those that do not have values on the command line. These options are considered Boolean values and thus should have the appropriate **true** value.
- Each keyword must be separated from its value by an equal sign (=).
- All option values must be enclosed in double quotation marks.

The configuration document is a hierarchical XML object. Each keyword must be placed within its appropriate section or subsection.

Multiple servers may be specified within the same configuration file, but each server must have a unique name. The format of the contents of the configuration file is XML V1.0.

Format

```
$ java . . . cfg <config file>.xml
```

Example

```
<?xml version = '1.0'?>
<! Configuration file for Rdb Thin JDBC Drivers and Servers -->
<config>
<! SERVERS -->
<servers>
  <! DEFAULT server characteristics-->
  <server
    name="DEFAULT"
    type="RdbThinSrv"
    url="//localhost:1701/"
    maxClients="-1"
    srv.bindTimeout="1000"
    srv.idleTimeout="0"
    srv.mcBasePort="5517"
    srv.mcGroupIP="239.192.1.1"
    tracelevel = "0"
    autostart = "false"
    autorestart = "false"
    restrictAccess = "false"
    anonymous = "false"
    bypass = "false"
    tracelocal = "false"
    relay = "false"
    controlUser="control_user"
    controlPass="0x18E007C81F6B2E2EA02065F78A587BD3"
    cfg="rdb$jdbc_com:rdbjdbccfg.xml"
    srv.execStartup="rdb$jdbc_home:rdbjdbc_startexec.com"
    srv.startup="rdb$jdbc_home:rdbjdbc_startsrv.com"
    sharedmem = "0"
  />
  <! DEFAULT Secure socket server -->
```

```
<server
  name="DEFAULTSSL"
  type="RdbThinSrvSSL"
  ssl.default="false"
  ssl.context="TLS"
  ssl.keyManagerFactory="SunX509"
  ssl.keyStoreType="jks"
  ssl.keyStore="rdbjdbcsrv.kst"
  ssl.keyStorePassword="CHANGETHIS"
  ssl.trustStore="rdbjdbcsrv.kst"
  ssl.trustStorePassword="CHANGETHIS"
/>
<! now specific servers that will be started up by Pool server -->
<server
  name="srv1forRdb"
  type="RdbThinSrv"
  url="//localhost:1701/"
  autoStart="true"
  autoRestart="true"
  logfile="rdb$jdbc_logs:srv1forRdb.log"
  tracelevel="-1"
  maxClients="1"
/>
<server
  name="srv2forRdb"
  type="RdbThinSrv"
  url="//localhost:1708/"
  autoStart="true"
  logfile="rdb$jdbc_logs:srv2forRdb.log"
/>

<! MP server -->
<! sharedmem is in KB default = 1024 -->
<server
  name="srvMPforRdb"
  type="RdbThinSrvMP"
  url="//localhost:1705/"
  autoStart="true"
  maxClients="10"
  maxFreeExecutors="10"
  prestartedExecutors="10"
  sharedMem="10240"
/>
<! the Pool server -->
<server
  name="rdbpool"
  type="RdbThinSrvPool"
  url="//localhost:1702/" >
  <pooledServer name="srv1forRdb"/>
  <pooledServer name="srv2forRdb"/>
  <pooledServer name="srvMPforRdb"/>
</server>

<! Secure socket server -->
<server
  name="srvssl1forRdb"
  type="RdbThinSrvSSL"
  url="//localhost:1709/"
/>

<! a Manager server -->
<server
  name="rdbman"
  type="RdbManSrv"
  url="//localhost:2060/"
/>
```

```
</servers>
<! database -->
<databases>
  <database
    name="mf_pers"
    url="//localhost:1701/mydisk:[databases]mf_personnel"
    driver="oracle.rdb.jdbc.rdbThin.Driver"
    URLPrefix="jdbc:rdbThin:"
  />
  <database
    name="pers"
    url="//localhost:1702/mydisk:[databases]personnel"
    driver="oracle.rdb.jdbc.rdbThin.Driver"
    URLPrefix="jdbc:rdbThin:"
  />
</databases>

</config>
```

The XML-formatted configuration file is an XML document that is composed of several sections and sub-sections.

Description of the sections and sub-sections within an XML-formatted configuration file is now presented:

- Config Section
- Session Section
- Databases Section
- Database Section
- Events Section
- Event Section
- Servers Section
- Server Section
- Pooled Server Subsection
- Allow Database Subsection
- Allow IP Subsection
- Allow User Subsection
- Enable Events Subsection
- Deny IP Subsection
- Deny SQL Subsection

4.5.2.1 Config Section

This section covers the entire configuration settings and contains the specific configuration sections as described below.

Format

```
<config>
[ session section ]
[ databases section ]
[ servers section ]
[ events section ]
</config>
```

4.5.2.2 Session Section

This section describes session characteristics for an interactive session. Information within the session section is currently only used by the Oracle JDBC for Rdb controller. You can specify information such as passwords and usernames that may be used when you start up a controller session.

If it exists, the session named `DEFAULT` will be used to setup the default session characteristics.

These session properties provide an alternate way of specifying options you may have otherwise supplied on the command line during controller startup.

Format

```
<session
[ session property ]
/>
```

Options

Valid properties for the session section can be seen in the following table:

Table 4-7 Session Section Properties

Option	Default	Description
`controlPass`	**none**	Specifies the password that will be used by default when connecting to an active server for control purposes. **Note**: This password can be a plain-text or obfuscated password created using the obfuscate command. You should not use an obfuscated password created using the digest command, as the server will not recognize the password. See Password Obfuscation in Server Configuration Files for more details.
`controlUser`	**none**	Username to use on control connections.
`password`	**none**	Currently this has the same function as controlPass, however if both are present, controlPass will take precedence. **Note**: This password can be a plain-text or obfuscated password created using the obfuscate command. You should not use an obfuscated password created using the digest command as the server will not recognize the password. See Password Obfuscation in Server Configuration Files for more details.
`name`	**none**	Name for this session description; must be `DEFAULT` .

Option	Default	Description
`user`	**none**	Username to use on connection.
`tracelevel`	0	The sessions default trace level.
`srv.mcBasePort <base_port>`	5517	Specifies the base port number that will be used for multicast operations.
`srv.mcGroupIP <group_ip>`	**239.192.1.1**	Specifies the multicast IP group that will be used for multicast operations.
`ssl.*`	**none**	Specifies SSL configuration information for the session that may be used to connect to SSL-enabled thin servers. See Using SSL for more information.

Example

```
<session
   name="DEFAULT"
   controlPass="jdbc_control"
   user="jdcb_user"
   password="jdbc_control"
   tracelevel="0"
   srv.mcBasePort="5517"
   srv.mcGroupIP="239.192.1.1"
/>
```

Note:

1. The session properties `srv.mcBasePort` and `srv.mcGroupIP` specify the multicast attributes that should be used for polling servers. Only those servers participating in the specified multicast group will respond to any poll requests issued by the controller.

2. Although the user and control passwords may be stored in plain-text format in the configuration file as shown in the example above, this may be contrary to your organization's security policy. Oracle recommends to not store plain-text passwords in your configuration files, instead the appropriate command line switches should be used to provide the password.

3. User passwords and control passwords used within the session section of the configuration file may be stored as obfuscated values created using the Obfuscate command. Control passwords associated with servers may also be specified in the server section of the configuration file as obfuscated values created using the Digest command.

4.5.2.3 Databases Section

This section specifies one or more database sections.

Format

```
<databases>
   [ database section ]
</databases>
```

4.5.2.4 Database Section

This section specifies a named database with the given properties.

Format

```
<database
    [ database property ]
/>
```

Options

Valid properties for the database section can be seen in the following table:

Table 4-8 Database Section Properties

Option	Default	Description
name	**none**	The name by which the Oracle JDBC for Rdb drivers may recognize this database. This name is required and must be unique within the databases section of this configuration file.
url	**none**	The URL that may be used to access this database.
driver	**none**	The class path of the preferred JDBC driver that may be used to access this database.
URLPrefix	**none**	The prefix that needs to be added to the URL above to provide a complete JDBC Connection URL.

Example

```
<! database -->
<databases>
  <database
    name="mf_pers"
    url="//localhost:1701/mydisk:[databases]mf_personnel"
    driver="oracle.rdb.jdbc.rdbThin.Driver"
    URLPrefix="jdbc:rdbThin:"
  />
  <database
    name="pers"
    url="//localhost:1702/mydisk:[databases]personnel"
    driver="oracle.rdb.jdbc.rdbThin.Driver"
    URLPrefix="jdbc:rdbThin:"
  />
</databases>
```

4.5.2.5 Events Section

Since release 7.3.1

This section specifies one or more event sections.

Format

```
<events>
    [ event section ]
</events>
```

4.5.2.6 Event Section

This section specifies a named event with the given properties.

Format

```
<event
    [ event property ]
/>
```

Options

Valid properties for the event section can be seen in the following table:

Table 4-9 EventSection Properties

Option	Default	Description
`name`	**none**	The name by which the Oracle JDBC for Rdb servers may recognize this event. This name is required and must be unique within the events section of this configuration file.
`type`	**none**	The type or category of the event. See Event criteria for more details.
`watch`	**none**	The criteria that should be watched. The watch criteria depend on the event type. See Event criteria for more details.
`pattern`	**none**	The pattern that should be used to filter the event watch criteria. The pattern is a Java regular expression that will be used to match the watched criteria. See Event criteria for more details.
`threshold`	**Max**	The actual value that must be met, exceeded, or near to trigger a threshold event. See Event criteria for more details.
`deviation`	**none**	The allowed deviation that the watched criteria can have before the event will be triggered. See Event criteria for more details.
`testFor`	**none**	Criteria condition to evaluate for. See Event criteria for more details.

Option	Default	Description
message	**none**	This is the customized message that will be logged when this event is triggered.

Example

```
<! Rdb events -->
<events>
  <event
    name = "NO_TABLE"
    type = "EXCEPTION"
    watch = "SYSERROR"
    pattern = ".*RELNOTDEF.*"
    message = "Problem with table "
  />

  <event
    name = "Near Max"
    type = "THRESHOLD"
    watch = "NUMUSERS"
    deviation="-3"
    message = "The number of users nearing maximum allowed"
  />

</events>
```

Details on using the various event properties may be found in Defining and Enabling Events.

4.5.2.7 Servers Section

This section specifies one or more server property sections.

Format

```
<servers>
   [ server section ]
</servers>
```

4.5.2.8 Server Section

This section specifies one or more properties to assign to this server. See Server Configuration for details on the properties that may be set.

Format

```
<server
  <property="value"/>
/>
```

or

```
<server
```

```
  <property="value"/>
>
[ pooled server subsection ]
[ allow database subsection ]
[ allow IP subsection ]
[ allow user subsection ]
[ enabled event subsection ]
[ deny IP subsection ]
[ deny SQl subsection ]
</server>
```

Example 1

A standard thin server called serv1 listening on port 1799 could be described using the following Server Property section:

```
<server
    name="serv1"
    type="RdbThinSrv"
    url="//localhost:1799/"
    logfile="myLogs:serv1.log"
/>
```

Remarks

Default server characteristics for server configuration can be specified so that options need not be repeated within the specific server configuration sections. Default server options may be specified by declaring a server section with a name of `DEFAULT` or `DEFAULTSSL`:

```
<server
    name="DEFAULT"
    type="RdbThinSrv"
    url="//localhost:1701/"
    maxClients="-1"
    srv.bindTimeout="0"
    srv.idleTimeout="0"
    srv.mcBasePort="5517"
    srv.mcGroupIP="239.1.1.1"
    autoStart="false"
    controlUser="jdbc_user"
    controlPass="0x811B15F866179583EB3C96751585B843"
/>
```

The `DEFAULT` and `DEFAULTSSL` server definitions should only be used to define the default server characteristics and are not intended to represent actual server instances that can be started by the controller or Pool servers.

These default server properties will be assigned to each server found defined after them in the configuration file unless explicitly overridden in the specific server subsection.

The placement of the `DEFAULT` and `DEFAULTSSL` server sections within the configuration file is important. Only those servers defined in sections that occur after these default definitions will have these default characteristics. Any server section specified prior to the default server sections will not get these default characteristics. Oracle recommends that these two sections be the first two server sections within your configuration file.

If subsections such as `Pooled Server` or `Allowed Database` are required, then the second format for a `Server` section must be used.

Example 2

```
<server
    name="rdbpool"
    type="RdbThinSrvPool"
    url="//localhost:1702/" >
    <pooledServer name="srv1forRdb"/>
    <pooledServer name="srv2forRdb"/>
    <pooledServer name="srvMPforRdb"/>
</server>
```

4.5.2.9 Pooled Server Subsection

This subsection specifies a server that will take part in the Pool server's server pool and is valid only when used within an `RdbThinSrvPool` server declaration.

The declared server name must reference a server already named in this configuration file.

Multiple `PooledServer` subsections may be present in a single server declaration.

The set of `pooledServer`s provided will make up the pool of servers that the Pool server may try to access.

Format

```
<pooledServer name="declared server"/>
```

Example

```
<server
    name="rdbpool"
    type="RdbThinSrvPool"
    url="//localhost:1702/" >
    <pooledServer name="srv1forRdb"/>
    <pooledServer name="srv2forRdb"/>
    <pooledServer name="srvMPforRdb"/>
</server>
```

4.5.2.10 Allow Database Subsection

This subsection specifies the database that clients using the server may access and is only valid when used within a server declaration.

The declared database name must either reference a database already named in the database section of this configuration file or must be a valid database file specification or logical name.

Multiple `AllowDatabase` subsections may be present in a single server declaration.

For database access to be restricted the server attribute `restrictAccess` must be set to **true**.

See the section Restricting Database Access for more details.

Format

```
<allowDatabase name="db-specification" />
```

Since release 7.3.2

Alternatively, from release 7.3.2 onwards:

```
<allow database="db-specification" />
```

Example

```
<server
    name="srv2restrict"
    type="RdbThinSrv"
    url="//localhost:1701/"
    restrictAccess="true"
    >
    <allowDatabase name="mf_pers"/>
    <allowDatabase name="disk1:[databases]customers"/>
</server>
```

4.5.2.11 Allow IP Subsection

Since release 7.3.2

This subsection specifies the IPs that may be allowed to access this server and is only valid when used within a server declaration.

The declared IP must be a valid IPv4 *dot-decimal* formatted IP string, or a Java regular expression based on the same standard IPv4 *dot-decimal* notation.

Multiple `AllowIP` subsections may be present in a single server declaration.

For IP access to be restricted the server attribute `restrictAccess` must be set to **true**.

This feature is only available when using Oracle JDBC for Rdb release 7.3.2 and later.

See the section Restricting IP Access for more details.

Format

```
<allowIP IP="IP allowed" />
```

or

```
<allow IP="IP allowed" />
```

Example

```
<server
    name="srv2restrict"
    type="RdbThinSrv"
    url="//localhost:1701/"
    restrictAccess="true"
    >
    <allow IP ="170.0.0.1"/>
```

```
    <allow IP ="170\.0\.0\.([5-7]|[8-9])"/>
</server>
```

4.5.2.12 Allow User Subsection

This subsection specifies the usernames the server will allow access to and is only valid when used within a server declaration.

The declared username must be a valid username recognized by Rdb. The matching of usernames by the server for this level of restriction is not case-sensitive.

Multiple `AllowUser` subsections may be present in a single server declaration.

For user access to be restricted the server attribute `restrictAccess` must be set to **true**.

See the section Restricting User Access for more details.

Format

```
<allowUser name="username" />
```

Since release 7.3.2

Alternatively, from release 7.3.2 onwards:

Format

```
<allow user="username" />
```

Example

```
<server
    name="srv2restrict"
    type="RdbThinSrv"
    url="//localhost:1701/"
    restrictAccess="true"
    >
    <allowUser name="smith"/>
    <allowUser name="jones"/>
</server>
```

4.5.2.13 Enable Events Subsection

Since release 7.3.1

This subsection specifies the events the server should log and is only valid when used within a server declaration.

Multiple `enableEvent` subsections may be present in a single server declaration.

See Event Logging and Notification for more details.

Format

```
<enableEvent name="event name" />
```

Example

```
<server
    name="srv2"
    type="RdbThinSrv"
    url="//localhost:1701/"
    >
    <enableEvent name="BAD_TABLE"/>
    <enableEvent name="DENIED_SQL"/>
</server>
```

The name provided in the enableEvent must match a valid Event specified in the Events Section of the same configuration file.

4.5.2.14 Deny IP Subsection

Since release 7.3.2

This subsection specifies the IPs that may be denied access to this server and is only valid when used within a server declaration.

The declared IP must be a valid IPv4 *dot-decimal* formatted IP string, or a Java regular expression based on the same standard IPv4 *dot-decimal* notation.

Multiple `denyIP` subsections may be present in a single server declaration.

For IP access to be restricted the server attribute `restrictAccess` must be set to **true**.

This feature is only available when using Oracle JDBC for Rdb release 7.3.2 and later.

See the section Restricting IP Access for more details.

Format

```
<denyIP IP="IP denied" />
```

or

```
<deny IP="IP denied" />
```

Example

```
<server
    name="srv2restrict"
    type="RdbThinSrv"
    url="//localhost:1701/"
    restrictAccess="true"
    >
    <deny IP ="170.0.0.1"/>
    <deny IP ="170.0.0.2"/>
</server>
```

4.5.2.15 Deny SQL Subsection

Since release 7.3.1

This subsection specifies the SQL statement that should be denied by the server and is only valid when used within a server declaration.

Multiple `Deny` subsections may be present in a single server declaration.

See Restricting SQL Statements for more details.

Format

```
<deny SQL="sql-pattern" />
```

Example

```
<server
    name="srv2"
    type="RdbThinSrv"
    url="//localhost:1701/"
    >
    <deny SQL =" (?i).*select.*jobs.* "/>
    <deny SQL =" (?i).*select.*from.*employees.* "/>
</server>
```

4.5.3 Using filenames in the configuration file

Several attributes within the configuration file sections require the specification of a filename, for example:

- `cfg="<filename>"`
- `log="<filename>"`
- `srv.execStartup="<filename>"`
- `srv.startup="<filename>"`

The filename must be a valid OpenVMS file specification that may contain a full or partial file path and may include logical names.

You must ensure that, if logical names are used, they are available to the context within which the server will be started, and that the file is accessible by the VMS user that starts up the server.

If a server defined in the configuration will be started up using the controller, as a pooled server by a Pool server, or by Oracle SQL/Services, a detached process will be created for the server and the `LOGINOUT.EXE` will be run to ensure a valid process environment under which Java and Oracle Rdb can be accessed.

Because the `LOGINOUT.EXE` program is run, any file specification using relative file paths must be relative to the login directory of the invoker, otherwise a full file specification must be used.

Contents

Chapter 5
Using SSL

Secure Sockets Layer (SSL) was developed to provide security for Web traffic, including confidentiality, message integrity, and authentication. SSL achieves this using cryptography, digital signatures, and certificates. Oracle JDBC for Rdb servers and thin clients may use SSL for communication over TCP/IP. SSL allows an SSL-enabled server to authenticate itself to an SSL-enabled client, allows the client to authenticate itself to the server, and allows both machines to establish an encrypted connection.

Before trying to use SSL with the thin driver, you should familiarize yourself with general Java security and SSL concepts. Please refer to the Java documentation for general information on SSL and Java Security.

The following sections provide SSL information specific to using SSL with the thin driver and assume a basic understanding of Java Security and SSL:

- SSL Configuration
- SSL and the Controller
- SSL Configuration Options

Contents

5.1 SSL Configuration

Information about SSL connection characteristics must be provided to both the client and server and, for a communication channel to be established, both the server and client must agree on the SSL security characteristics.

In addition, it is important that both the client and the server have the same security certificate for authorization. The following sections detail how to provide SSL characteristics in a client connection request and to an SSL-enabled Oracle JDBC for Rdb server:

- Client SSL Configuration
- Server SSL Configuration

5.1.1 Client SSL Configuration

The client application must specify its SSL characteristics during its connection request to the thin driver. The simplest way of doing this is by providing extra SSL information in the properties block that is passed to the `DriverManager.getConnection()` method.

The SSL information provides information such as where to find the appropriate certificate for SSL connections and what context and protocols should be used to carry out the SSL handshake during connection set-up.

Example 1

```
Properties info = new Properties();
info.put("user", user);
info.put("password", password);
```

```
info.put("tracelevel", traceLevel);
info.put("ssl", "true");
info.put("ssl.default", "false");
info.put("ssl.context", "TLS");
info.put("ssl.keyManagerFactory", "SunX509");
info.put("ssl.keyStoreType", "jks");
info.put("ssl.keyStore", "rdbjdbccli.kst");
info.put("ssl.keyStorePassword", "CHANGETHIS");
info.put("ssl.trustStore", "rdbjdbccli.kst");
info.put("ssl.trustStorePassword", "CHANGETHIS");
Connection conn = DriverManager.getConnection(
  "jdbc:rdbThin://bravo:1755/myDir:pers", info);
```

Remarks

The properties block must have the property **ssl** set to **true** for SSL connections to be attempted.

In addition, the SSL characteristics can be specified explicitly as properties, or you may use `ssl.default` set to **true** to request that the default SSL characteristics for your system should be used.

Example 2

```
Properties info = new Properties();
info.put("user", user);
info.put("password", password);
info.put("tracelevel", traceLevel);
info.put("ssl", "true");
info.put("ssl.default", "true");
Connection conn = DriverManager.getConnection(
  "jdbc:rdbThin://bravo:1755/myDir:pers", info);
```

Remarks

See SSL configuration options for details of the ssl.* options.

For an SSL connection to be made, the appropriate certificate for the server to which you are trying to attach to should be in the keystore you have designated in the SSL properties for the connection.

If no certificate is found the following exception will be raised:

```
javax.net.ssl.SSLException: No available certificate corresponds to the SSL cipher
suites, which are enabled.
```

See the Java Security documentation for more information on certificates.

5.1.2 Server SSL Configuration

An SSL-enabled server must also be provided with SSL configuration information. This is usually provided within the server section for the named server in an XML-based configuration file.

To indicate that the server should be SSL-enabled, the server must be defined as one of the following SSL server types:

- `RdbThinSrvSSL`
- `RdbThinSrvMPSSL`
- `RdbThinSrvPoolSSL`

- RdbManSrvSSL

Example 1

```
<server
    name="MYSSL"
    type="RdbThinSrvSSL"
    ssl.default="false"
    ssl.context="TLS"
    ssl.keyManagerFactory="SunX509"
    ssl.keyStoreType="jks"
    ssl.keyStore="rdbjdbcsrv.kst"
    ssl.keyStorePassword="CHANGETHIS"
    ssl.trustStore="rdbjdbcsrv.kst"
    ssl.trustStorePassword="CHANGETHIS"
/>
```

Remarks

If you wish to define several SSL-enabled servers with the same SSL characteristics, then you can use the special DEFAULTSSL server definition to define the default characteristics. Each subsequent server definition that has one of the SSL server types will use these characteristics, unless explicitly overridden in the server definition.

Example 2

```
<server
    name="DEFAULTSSL"
    type="RdbThinSrvSSL"
    ssl.default="false"
    ssl.context="TLS"
    ssl.keyManagerFactory="SunX509"
    ssl.keyStoreType="jks"
    ssl.keyStore="rdbjdbcsrv.kst"
    ssl.keyStorePassword="CHANGETHIS"
    ssl.trustStore="rdbjdbcsrv.kst"
    ssl.trustStorePassword="CHANGETHIS"
/>
<server
    name="SSLsrv1"
    type="RdbThinSrvSSL"
    url="//localhost:1707/"
/>
<server
    name="SSLsrv2"
    type="RdbThinSrvMPSSL"
    url="//localhost:1708/"
    sharedMem="10000"
/>
<server
    name="SSLmansrv"
    type="RdbManSrvSSL"
    url="//localhost:2061/"
/>
```

Remarks

If a Pool server is SSL-enabled, for security reasons it will only communicate with pooled servers within its pool that are also SSL-enabled. Non-SSL-enabled pooled servers within the pool will be ignored and will not be considered candidates for redirection of connection requests.

See SSL Configuration Options for details of these options.

Contents

5.2 SSL and the Controller

All connections made to SSL-enabled servers must be made using SSL connections. This also includes the controller.

If the controller will be used to manage SSL-enabled servers, then the controller session must also have the correct SSL information to make the secure connection to the server.

You can specify the SSL information that the controller uses for connecting to SSL-enabled thin servers by starting the controller using an XML-formatted configuration file that has the appropriate SSL information in its SESSION section.

Example

```
<session
    name="DEFAULT"
    controlPass="jdbc_user"
    user="cts1"
    password="jdbc_user"
    tracelevel="0"
    srv.mcBasePort="5518"
    srv.mcGroupIP="239.192.1.2"
    ssl.default="false"
    ssl.context="TLS"
    ssl.keyManagerFactory="SunX509"
    ssl.keyStoreType="jks"
    ssl.keyStore="rdbjdbccli.kst"
    ssl.keyStorePassword="CHANGETHIS"
    ssl.trustStore="rdbjdbccli.kst"
    ssl.trustStorePassword="CHANGETHIS"
/>
```

Remarks

This is the same SSL information that you would have provided for a client SSL configuration as described in Client SSL configuration.

If this information is provided, the controller will use the SSL configuration to connect to any server that responds to a poll request as an SSL-enabled server.

Contents

5.3 SSL Configuration Options

The various SSL configuration options that may be set can be found in the following table:

Table 5-1 SSL Configuration Options

Option	Default	Description
`ssl.default`	**false**	If specified, indicates that the default SSL socket factory should be used to create an SSL socket. The default SSL socket factory can be changed by setting the value of the

Option	Default	Description
		"ssl.ServerSocketFactory.provider" security property (in the Java security properties file) to the desired class. All other ssl.* configuration options will be ignored, if ssl.default is specified and set to **true** . If ssl.default is not specified, or specified as **false** , then the values of the following ssl.* properties should be used to create an SSL socket factory.
ssl.context <ssl context>	**none**	Indicates the SSL context to use, for example "TLS".
ssl.keyManagerFactory <keymanager factory>	**none**	Indicates the keymanager factory to use, for example "SunX509".
ssl.keyStoreType <store type>	**none**	Indicates the type of the key store, for example "jks".
ssl.keyStore <store filename>	**none**	Indicates the filename of the keystore.
ssl.keyStorePassword <password>	**none**	Indicates the password for the keystore.
ssl.trustStore <trust store filename>	**none**	Indicates the filename of the trust store.
ssl.trustStorePassword <password>	**none**	Indicates the password of the trust store.

Contents

5.4 Using Self-Signed Certificates for Testing

The following code is how you may build and copy certificates that may be used for SSL communications where the client and server are on OpenVMS nodes that have Java environments already set up.

Note:
A copy of this code is in the JDBC installation directory within BUILD_CERTS.COM.

Information such as the names, company, country, keystore and password should be changed appropriately for your own situation.

```
$! BUILD_CERTS.COM
$! Example command procedure for creating self-signed certificates
$! for use with Oracle JDBC For Rdb
$!
$! ----
$! The following should be done on the Server node
$! ----
$!
$ write sys$output -
"Generating the Server KeyStore in file rdbjdbcsrv.kst"
$ keytool -genkey -alias rdbjdbc-sv -
-dname "CN=MY NAME, OU=MY UNIT, O=MY COMPANY, c=MY REGION" -
```

```
-keypass "CHANGETHIS" -storepass "CHANGETHIS" -
-KeyStore rdbjdbcsrv.kst
$!
$ write sys$output -
"Exporting the certificate from keystore to file server.cer"
$ keytool -export -alias rdbjdbc-sv -
-storepass "CHANGETHIS" -file server.cer -keystore rdbjdbcsrv.kst
$!
$! ----
$! The following should be done on the Client node
$! ----
$!
$ write sys$output -
"Generating the Client KeyStore in file rdbjdbccli.kst"
$ keytool -genkey -alias rdbjdbc-cl -
-dname "CN=Rdbjdbc Client, OU=X, O=Y, L=Z, S=XY, C=YZ" -keyalg RSA -
-keypass "CHANGETHIS" -storepass "CHANGETHIS" -keystore rdbjdbccli.kst
$!
$ write sys$output -
"Exporting the certificate from keystore to file client.cer"
$ keytool -export -alias rdbjdbc-cl -storepass "CHANGETHIS" -
-file client.cer -keystore rdbjdbccli.kst
$!
$! ----
$! Exchange the certificates by copying the client certificate file
$! (client.cer) to the server node, and the server certificate file
$! (server.cer) to the client node
$! ----
$!
$! ----
$! Now back on the Server node
$! ----
$!
$ write sys$output -
"Importing Client's certificate into Server's keystore"
$ keytool -import -v -trustcacerts -alias rdbjdbc -file client.cer -
-keystore rdbjdbcsrv.kst -keypass "CHANGETHIS" -storepass "CHANGETHIS"
yes
$!
$! ----
$! And back on the Client node
$! ----
$!
$ write sys$output -
"Importing Server's certificate into Client's keystore"
$ keytool -import -v -trustcacerts -alias rdbjdbc -file server.cer -
-keystore rdbjdbccli.kst -keypass "CHANGETHIS" -storepass "CHANGETHIS"
Yes
```

The **keytool** command should work as shown above on most operating systems that have Java installed.

Once the keystores have been set up, if you have setup the SSL properties correctly for the client and the server as shown in previous sections, you can use SSL for client/server communication within the thin driver.

Note:

It is important to use double quotes to maintain values such as passwords exactly as you specify them in the server or client SSL connection configuration properties.

Contents

Chapter 6
Oracle JDBC for Rdb Controller

The Oracle JDBC for Rdb controller (here-on referred to as the controller) allows basic management of Oracle JDBC for Rdb servers.

Contained in the `rdbthincontrol.jar` file, this application allows local and remote password-protected server management operations to be carried out on a thin server or Pool server. These operations can include showing the clients that are currently connected, stopping client threads, and starting and stopping thin servers.

The controller can be run either in interactive mode or single command mode. In interactive mode you typically connect to the server you wish to manage and then issue the management requests. When you are finished using the controller, you can issue the exit command to terminate the image.

In single command mode, you provide command line switches to tell the controller what action must be performed. When the action is complete, the controller image will terminate.

The controller is typically used in conjunction with an XML-formatted configuration file that provides information about the Oracle JDBC for Rdb servers running on your system.

In addition, the configuration file may provide session information such as the broadcast port information to use when doing poll operations. See Configuration Files for more information about configuration files.

The controller may be used to start and stop servers, as well as other operations pertaining to servers and connected clients. In addition, the controller may be used to show the status of Oracle JDBC for Rdb servers running throughout your network.

Below is a sample session using the controller in interactive mode:

Example

```
rdbthincontrol> show stored servers
Stored server info

RDB$NODE : localhost
RDB$PORT : 1702
RDB$STATUS : not available
RDB$SERVER_NAME : SRV2
RDB$SERVER_TYPE : RdbThinSrv
RDB$SERVER_VERSION : not available
RDB$SERVER_SHR_VERSION : not available
RDB$SERVER_PID : not available
RDB$ALLOWS_ANON : false
RDB$ALLOWS_BYPASS : false
RDB$NUMBER_OF_CLIENTS : 0
RDB$MAX_CLIENTS : -1


RDB$NODE : localhost
RDB$PORT : 1701
RDB$STATUS : not available
RDB$SERVER_NAME : SRV1
RDB$SERVER_TYPE : RdbThinSrv
RDB$SERVER_VERSION : not available
RDB$SERVER_SHR_VERSION : not available
```

```
RDB$SERVER_PID : not available
RDB$ALLOWS_ANON : false
RDB$ALLOWS_BYPASS : false
RDB$NUMBER_OF_CLIENTS : 0
RDB$MAX_CLIENTS : -1


RDB$NODE : localhost
RDB$PORT : 1701
RDB$STATUS : not available
RDB$SERVER_NAME : DEFAULT
RDB$SERVER_TYPE : RdbThinSrv
RDB$SERVER_VERSION : not available
RDB$SERVER_SHR_VERSION : not available
RDB$SERVER_PID : not available
RDB$ALLOWS_ANON : false
RDB$ALLOWS_BYPASS : false
RDB$NUMBER_OF_CLIENTS : 0
RDB$MAX_CLIENTS : -1
rdbthincontrol> start server srv1
Starting server ...


RDB$NODE : nnn.nnn.nnn.nnn
RDB$PORT : 1701
RDB$STATUS : Idle
RDB$SERVER_NAME : srv1
RDB$SERVER_TYPE : RdbThinSrv
RDB$SERVER_VERSION : V7.4-100 20250113 BP1D
RDB$SERVER_SHR_VERSION : V7.4-100 20250113 BP1D
RDB$SERVER_PID : 0x20238378(539198328)
RDB$ALLOWS_ANON : false
RDB$ALLOWS_BYPASS : false
RDB$NUMBER_OF_CLIENTS : 0
RDB$MAX_CLIENTS : -1
RDB$TRACE_LEVEL : 0
RDB$LOG_FILE : rdbjdbclog
RDB$RESTRICT_ACCESS : false
rdbthincontrol> poll
Polling servers ...
srv1(0) //nnn.nnn.nnn.nnn:1701/ (0x20238378<539198328>)
rdbthincontrol> start server srv2
Starting server ...


RDB$NODE : nnn.nnn.nnn.nnn
RDB$PORT : 1702
RDB$STATUS : Idle
RDB$SERVER_NAME : srv2
RDB$SERVER_TYPE : RdbThinSrv
RDB$SERVER_VERSION : V7.4-100 20250113 BP1D
RDB$SERVER_SHR_VERSION : V7.4-100 20250113 BP1D
RDB$SERVER_PID : 0x2033137C(540218236)
RDB$ALLOWS_ANON : false
RDB$ALLOWS_BYPASS : false
RDB$NUMBER_OF_CLIENTS : 0
RDB$MAX_CLIENTS : -1
RDB$TRACE_LEVEL : 0
RDB$LOG_FILE : rdbjdbclog
RDB$RESTRICT_ACCESS : false
rdbthincontrol> poll
Polling servers ...
srv2(0) //nnn.nnn.nnn.nnn:1702/ (0x2033137C<540218236>)
srv1(0) //nnn.nnn.nnn.nnn:1701/ (0x20238378<539198328>)
rdbthincontrol> show active servers
Active server info
```

```
RDB$NODE : nnn.nnn.nnn.nnn
RDB$PORT : 1702
RDB$STATUS : Idle
RDB$SERVER_NAME : srv2
RDB$SERVER_TYPE : RdbThinSrv
RDB$SERVER_VERSION : V7.4-100 20250113 BP1D
RDB$SERVER_SHR_VERSION : V7.4-100 20250113 BP1D
RDB$SERVER_PID : 0x2033137C(540218236)
RDB$ALLOWS_ANON : false
RDB$ALLOWS_BYPASS : false
RDB$NUMBER_OF_CLIENTS : 0
RDB$MAX_CLIENTS : -1
RDB$TRACE_LEVEL : 0
RDB$LOG_FILE : rdbjdbclog
RDB$RESTRICT_ACCESS : false

RDB$NODE : nnn.nnn.nnn.nnn
RDB$PORT : 1701
RDB$STATUS : Idle
RDB$SERVER_NAME : srv1
RDB$SERVER_TYPE : RdbThinSrv
RDB$SERVER_VERSION : V7.4-100 20250113 BP1D
RDB$SERVER_SHR_VERSION : V7.4-100 20250113 BP1D
RDB$SERVER_PID : 0x20238378(539198328)
RDB$ALLOWS_ANON : false
RDB$ALLOWS_BYPASS : false
RDB$NUMBER_OF_CLIENTS : 0
RDB$MAX_CLIENTS : -1
RDB$TRACE_LEVEL : 0
RDB$LOG_FILE : rdbjdbclog
RDB$RESTRICT_ACCESS : false
rdbthincontrol> stop all servers
Successfully stopped Thin Server : srv1 (//nnn.nnn.nnn.nnn:1701/)
Successfully stopped Thin Server : srv2 (//nnn.nnn.nnn.nnn:1702/)
rdbthincontrol> poll
Polling servers ...
rdbthincontrol> exit
```

The following sections detail how to run the controller and carry out various server and client control functions on active servers within your network:

- Running the Controller
- Connecting to Servers
- Control Password
- Multicast Polling
- Server Matching
- Server Operations
- Client Operations

Contents

6.1 JDK Versioned Oracle JDBC for Rdb Controller Jars

Since release 7.4.2

Prior to Oracle JDBC for Rdb release 7.4.2 only one version of the controller JAR file was available to use:

```
rdbThinControl.jar
```

Starting with release 7.4.2 you have the choice of three variants of this JAR:

- rdbThinControlV5.jar
- rdbThinControlV6.jar
- rdbThinControlV8.jar

These JARs provide JDBC and Java version specific variants of the server classes:

Table 6-1 Versioned server JARs

Driver JAR file	Minimum version of Java required	JDBC supported
RDBTHINCONTROLV5.JAR	JDK 1.5	JDBC 3.0, as released in JDK 1.4
RDBTHINCONTROLV6.JAR	Java SE 6	JDBC 4.0 as released in Java SE 6
RDBTHINCONTROLV8.JAR	Java SE 8 or OpenJDK8	JDBC 4.2 as released in Java SE 8 or OpenJDK8

To allow compatibility with prior Oracle JDBC for Rdb releases, a non-versioned variant of the controller JAR is also installed by the JDBC installation. This JAR is identical to the V5 version with the same name, `rdbThinControlV5.jar` file.

The controller JAR files are available in the `rdb$jdbc_home` directory created during JDBC installation.

6.2 Running the Controller

The controller allows basic management of Oracle JDBC for Rdb servers.

The controller can be run from the OpenVMS DCL command line either in single command mode or as a command line interface:

Format

```
$java jar rdb$jdbc_home:rdbthincontrol[<variant>].jar [<option> | <command_keyword>]...
```

Remarks

The optional <variant> if present must be the value V5 , V6 or V8 , See JDK Versioned Oracle JDBC for Rdb Controller Jars
Valid values for `<option>` can be found in Table 6.1 1.
Valid settings for `<command_keyword>` can be found in Table 6.1 2.

For the controller to be able to manage an Oracle JDBC for Rdb server the server must have a control password.

See Server Configuration Options for more details on specifying the control password.

Table 6-2 Controller Options

Option	Default	Description
`-active`	**false**	Used in conjunction with a *command_keyword* to specify that the

Option	Default	Description
		action applies to only active designated entities.
`-all`	**n/a**	Used in conjunction with a *command_keyword* to specify that the action applies to all designated entities.
`-configfile or -cfg <configuration_filename>`	**none**	The file specification of a configuration file where session and server attributes may be found. Attributes set in this configuration file may be overridden by setting the same attribute at the command line level. See Configuration Files for more details. By default, no configuration file is used.
`-controlpass <control password>`	**none**	Specifies the control password to use when connecting to servers. This password takes precedence over any *password* option provided on the same command line.
`-n or -node <node>`	**none**	Specifies the node where the server to be connected to is running.
`-name <server name>`	**none**	Specifies a name for the server. The name is used to lookup server information within the start-up configuration file. The value of this name is not case-sensitive.
`-oem`	**n/a**	Used by OEM to indicate that the return status and messages should be formatted for OEM usage.
`-password or -pw <password>`	**none**	Specifies the password to send to the thin server when requesting a control connection. If a *controlpass* option is also found on the same command line the *controlpass* option will take precedence.
`-pollTimeout <timeout>`	2000	Specifies the time, in milliseconds, that the controller should wait for `POLL` replies from servers. See Polling Servers for more information.

Option	Default	Description
`port or -p <port_num>`	**none**	Specifies the port on which the server to be connected to is listening.
`-srvargs <server_arguments>`	**none**	Additional arguments to be passed on the connection URL when connecting to the server. For example: `@tracelevel=-1`
`-srv.mcBasePort <base_port>`	5517	Specifies the base port number that will be used for multicast operations.
`-srv.mcGroupIP <group_ip>`	**239.192.1.1**	Specifies the multicast IP group within which this server will participate.
`-stored`	**n/a**	Used in conjunction with a *command_keyword* to specify that the action applies to the stored designated entities as found in the XML configuration file.
`-tracelevel or -tl <trace_level>`	0	Specifies the default tracelevel for the session. The value zero (0) means no tracing. See Trace Values for valid trace level values.
`-user or -u <user_name>`	**none**	Specifies the username to use for connection to the server.
`-url <connection URL>`	**none**	Specifies the node IP and port of the server to connect. This switch overrides any `port` and `node` switch specified. The format of the <connection URL> is //<node>:<port>/

Note:
A number of these options may also be specified in a session section of the XML-formatted configuration file used to start an interactive controller session. See Session Section within XML Formatted Configuration File for more details.

Table 6-2 Controller Command Keywords

Option	Description
`-poll`	Sends a poll request out to locate active servers. See Polling Servers for more information.

`-startserver`	Starts the server as specified by the other options given on the command line. See Starting Servers for more information.
`-openserver`	Opens the server as specified by the other options given on the command line. See Opening Servers for more information.
`-closeserver`	Closes the server as specified by the other options given on the command line. See Closing Servers for more information.
`-showserver`	Issues the `Show Server` command that gets server information from the connected server. See Showing Servers for more information.
`-showclients`	Issues the `Show Clients` command, which gets client information from the connected server. See Showing Clients for more information.
`-stopserver`	Stops the server as specified by the other options given on the command line. See Stopping Servers for more information.
`-stopclient <client_id>`	Issues the `Stop Client` command which requests the connected server to terminate the specified client thread. The `<client_id>` is an id of a client as displayed by the Show Clients command See Stopping Clients for more information. There is no default value for `<client_id>`.

If the controller is invoked with the appropriate connect information and one of command keywords, the controller will issue the desired request to the server, optionally display the results, and terminate immediately.

If more than one command keyword is present, only one will be issued using the precedence as shown in the preceding table.

Example

An example of issuing command keyword to the controller:

```
$ java -jar rdb$jdbc_home:rdbthincontrol.jar -u jan
-controlpass mpass -node nd1 -port 1701 -stopserver
```

6.2.1 Controller Command Line

If no command keyword is used on the controller invocation, the application will go into command line prompt mode allowing multiple commands to be issued.

If valid connection information is provided at the controller invocation (i.e., node, port, user and password), the controller will automatically attempt to connect to the specified server.

If a connection has not been established or a different server connection is required, then the Connect command can be issued at the control command line. See Connecting to Servers for more information.

If username and password are not provided on the connect command line, then the values of the configuration options when the controller was invoked will be used.

If a configuration file is specified, the configuration file session characteristics will be used. See Session Section within XML formatted Configuration File for more information on session characteristics.

Commands may be issued at the control command line either within the context of a server connection or outside the context of a specific server connection.

The commands that may be issued once a connection has been established to a server are discussed in Commands requiring server connection.

The commands that do not require a server connection are discussed in Commands Not requiring a server connection.

Format

```
$ java -jar rdb$jdbc_home:rdbthincontrol[<variant>].jar -cfg my_servers.xml
```

The optional <variant> if present must be the value V5 , V6 or V8 , See JDK Versioned Oracle JDBC for Rdb Controller Jars

6.2.1.1 Commands Requiring a Server Connection

Once a valid server connection has been established the commands shown in the following table may be issued:

Table 6-3 Controller Command Line Commands With Connection

Command	Description
`close server`	Closes the currently connected server. See Closing Servers for more details.
`disconnect`	Disconnects from the currently connected server.
`open server`	Opens the currently connected server. See Opening Servers for more details.
`set logfile [<filename>]`	Sets the logfile for the currently connected active server. This may be used to redirect trace log message to a different log file, which will close the current log file. If `<filename>` is missing or is equal to the value `OFF` the current logfile is still closed and log messages will no longer be sent to the log file.
`set default tracelevel <int>`	Sets the default tracelevel on the currently connected active server.

	This does not affect currently connected clients. Only clients connecting after the set default tracelevel is issued will be affected. The value zero (0) means no tracing. See Trace Values for valid trace level values.
`set tracelevel <int>`	Sets the tracelevel on the currently connected active server. This will set the trace level for all clients that are currently connected to the server. Clients connecting after the set is issued will not be affected. The value zero (0) means no tracing. See Trace Values for valid trace level values.
`show clients`	Show all clients on the currently connected server. See Showing Clients for more details.
`show server`	Show the details of the currently connected server. See Showing Servers for more information.
`stop client <client_id>`	Stops the client matching the specified `<client_id>` on the currently connected server. See Stopping Clients for more details.
`stop clients`	Stops all clients on the currently connected server. See Stopping Clients for more details.
`stop server`	Stops the currently connected server.
`watch [server]`	Send trace logging from connected server to the current console. See Watching Servers for more details.

Example

```
$ java -jar rdb$jdbc_home:rdbthincontrol.jar
rdbthincontrol> connect //localhost:1701/ jones mypassword
rdbthincontrol> show server

RDB$NODE : localhost
RDB$PORT : 1701
RDB$STATUS : Idle
RDB$SERVER_NAME : rdbthnsrv1
RDB$SERVER_TYPE : RdbThinSrv
RDB$SERVER_VERSION : V7.4-100 20250113 BP1D
RDB$SERVER_SHR_VERSION : V7.4-100 20250113 BP1D
RDB$SERVER_PID : 0x0B24(2852)
RDB$ALLOWS_ANON : false
RDB$ALLOWS_BYPASS : false
RDB$NUMBER_OF_CLIENTS : 0
RDB$MAX_CLIENTS : -1
rdbthincontrol>
rdbthincontrol> stop server
```

```
Successfully stopped Rdb Thin Server : //localhost:1701/
rdbthincontrol> exit
$
```

6.2.1.2 Commands not Requiring a Server Connection

Several commands may be issued that do not require you to have a connection established. However, for all commands other than *poll* and *quit*, you will have to provide a username and control password which will be used to connect to the servers to obtain the required information.

The commands that do not require a server connection are listed in the following table:

Table 6-4 Controller Command Line Commands Without Connection

Command	Description
`poll`	Multicast Poll for responding servers. See Polling Servers for more details.
`set session controlpass <pwd>`	Sets the sessions control password. See Control Password for more information.
`set default tracelevel <int> <server_ident>`	Sets the default tracelevel on the identified active server. This does not affect currently connected clients. Only clients connecting after the set default tracelevel is issued will be affected.
`set logfile <filename> <server_ident>`	Sets the log file specification for the identified active server. This may be used to redirect trace log messages to a different log file, which will close the current log file. If `<filename>` is the value **OFF** then the current logfile will be closed and log messages will no longer be sent to the log file.
`set polltimeout <int>`	Sets the time, in milliseconds that the controller should wait for `POLL` replies from servers. See Polling Servers for more information.
`set tracelevel <int> <server_ident>`	Sets the tracelevel on the identified active server. This will set the trace level for all clients that are currently connected to the server. Clients connecting after the set is issued will not be affected. The value zero (0) means no tracing. See Trace Values for valid trace level values.

`show active servers`
`show all servers`
`show server <server_ident>`
	Shows information about servers.
	See Showing Servers for more details.

`show active clients` `show all clients`	Shows information about clients on all responding servers. See Showing Clients for more details.
`show active clients <name>` `show all clients <name>`	Shows information about clients with username `<name>` on all responding servers. See Showing Clients for more details.
`stop active clients` `stop all clients`	Stops all clients on all responding servers. See Stopping Clients for more details.
`stop active clients <name>` `stop all clients <name>`	Stops all clients with username `<name>` on all responding servers. See Stopping Clients for more details.
`stop active clients in <database spec>` `stop all clients in <database spec>`	Stops all clients on all responding servers if the client is currently connected to the specified database. See Stopping Clients for more details.

`stop active servers`
`stop all servers`
`stop server <server_ident>`
	Stops active servers.
	See Stopping Servers for more details.

`open active servers`
`open all servers`
`open server <server_ident>`
	Opens active servers.
	See Opening Servers for more details.

`close active servers`
`close all servers`
`close server <server_ident>`
	Closes active servers.
	See Closing Servers for more details.

`watch [server] <server_ident>`	Watches active servers. See Watching Servers for more details.
`quit or exit`	Exits the controller application.

Example

```
$ java -jar rdb$jdbc_home:rdbthincontrol.jar -user jones -controlpass jdbc_user
rdbthincontrol> show active servers

RDB$NODE : localhost
RDB$PORT : 1701
RDB$STATUS : Idle
RDB$SERVER_NAME : rdbthnsrv1
RDB$SERVER_TYPE : RdbThinSrv
RDB$SERVER_VERSION : V7.4-100 20250113 BP1D
RDB$SERVER_SHR_VERSION : V7.4-100 20250113 BP1D
RDB$SERVER_PID : 0x0B30(2864)
RDB$ALLOWS_ANON : false
RDB$ALLOWS_BYPASS : false
RDB$NUMBER_OF_CLIENTS : 0
RDB$MAX_CLIENTS : -1

RDB$NODE : localhost
RDB$PORT : 1711
RDB$STATUS : Idle
RDB$SERVER_NAME : myserver
RDB$SERVER_TYPE : RdbThinSrv
```

```
RDB$SERVER_VERSION : V7.4-100 20250113 BP1D
RDB$SERVER_SHR_VERSION : V7.4-100 20250113 BP1D
RDB$SERVER_PID : 0x0B88(2952)
RDB$ALLOWS_ANON : false
RDB$ALLOWS_BYPASS : false
RDB$NUMBER_OF_CLIENTS : 0
RDB$MAX_CLIENTS : -1
rdbthincontrol>
```

If a server does not recognize the provided control password, it will respond with a failure message:

```
rdbthincontrol> show active servers
Failed to connect <CONTROL>
No Rdb Thin Server connection has been established
Unable to connect to server //localhost:1701/
Failed to connect <CONTROL>
No Rdb Thin Server connection has been established
Unable to connect to server //localhost:1711/
rdbthincontrol>
```

Contents

6.3 Connecting to Servers

Many commands that can be issued from the controller require a valid control connection to be established with a server. If valid connection information is provided at the controller invocation (node, port, user, and password), the controller will automatically attempt to connect to the specified server when the controller starts up.

If user and password are provided at the controller invocation, this information will be maintained for the entire controller session and will be used in subsequent connection requests unless explicitly overridden on the command statement.

Commands will only be carried out on a server if a control connection has been established, which requires the correct control password to be provided during the connect request. See Control Password for more information of this password.

This control connection may be an explicit connection established for the session by using the Connect command or may be implicitly established if a command is issued to a server that requires control access to execute successfully. Many controller commands allow server connection information to be specified, indicating which server to apply the command. In addition, the connection information may provide a username and password to use for that server.

Format

```
<command> <server_connection>
```

The <server_connection> information is comprised of a server identification string and optional connection username and control password:

```
<server_ident> [<server_uid>]
```

The `<server_ident>` string can be one of the following:

- Port ID - this is the same as issuing: `//localhost:<port>/`
- full URL with the format: `//<node>:<port>/`
- name of server as found in the configuration used to start the controller

The <server_uid> is:

```
<username> [<password>]
```

The `<password>` must match the control password of the server for the control connection to be carried out successfully.

If a *username* or *password* is not provided on the command line then the current session information is used.

This connection, once established, will be maintained until either an explicit `Disconnect` is issued, or a new connection is established to another server or the controller exits.

Example

If an attempt is made to issue a controller command without a connection being established, then an error condition will be raised:

```
rdbthincontrol> watch
No Rdb Thin Server connection has been established
```

If *username* and *password* are not provided on the connect command line, then the values of the appropriate configuration options set when the controller was invoked will be used.

If a configuration file is specified, the configuration file session characteristics will be used.

See Session Section within XML formatted Configuration File for more information on session characteristics.

6.3.1 Connect Command

If a connection has not been established, or the current connection has been disconnected, or a different server connection is required, then the `Connect` command can be issued at the control command line.

Format

```
connect [server] <server_connection>
```

This command connects to the server specified by the `<server_connection>` information.

Example

The following examples use the Connect command:

```
rdbthincontrol> connect //localhost:1701/ jones mypassword
rdbthincontrol> connect server 1701
rdbthincontrol> connect myServer jim xxxxx
```

Remarks

If *username* and *password* are not provided on the `Connect` command line, then the values entered in the configuration options when the controller was invoked will be used or, if a configuration file is specified, the configuration file session characteristics will be used. See Session Section within XML formatted Configuration File for more information on session characteristics.

6.3.2 Implicit Connection

Several of the control commands require a control connection to be established with the target server. If the target server is not currently connected, both explicitly provided connection information and session connection information may be used to attempt to establish a control connection.

Example

Connection information may be provided on the command line along with the command:

```
rdbthincontrol> stop server //localhost:1701/ jones mypassword
```

Once an implicit connection is made, this connection will be established as the current session connection until overridden by another implicit or explicit connection.

Contents

6.4 Control Password

To carry out any operations on active servers or clients, you are required to provide a control password. This password must match the control password for that active server, otherwise, an exception will be raised, and the operation will fail.

When you start up the controller you may provide a `password` as a command line option or in the session section of an XML-formatted Configuration file. If you provide both a `password` and a `controlPass` the `controlPass` will take precedence.

Example

```
rdbthincontrol> stop server myMPServer
Failed to connect <CONTROL>
No Rdb Thin Server connection has been established
Unable to connect to server //localhost:1788/
```

In addition, the control password may be set for a session by using the `Set Session Controlpass` statement at the controller command line prompt.

```
rdbthincontrol> set session controlpass badpassword
rdbthincontrol> show server 1701
Failed to connect <CONTROL>
No Rdb Thin Server connection has been established
rdbthincontrol> set session controlpass mypassword
rdbthincontrol> show server 1701

RDB$NODE : nnn.nnn.nnn.nnn
RDB$PORT : 1701
RDB$STATUS : Idle
RDB$SERVER_NAME : jimserv
RDB$SERVER_TYPE : RdbThinSrv
RDB$SERVER_VERSION : V7.4-100 20250113 BP1D
RDB$SERVER_SHR_VERSION V7.4-100 20250113 BP1D
RDB$SERVER_PID : 0x1728(5928)
RDB$ALLOWS_ANON : false
RDB$ALLOWS_BYPASS : false
```

```
RDB$NUMBER_OF_CLIENTS : 0
RDB$MAX_CLIENTS : -1
```

Note:

A session *password* or *controlPass* specified on the controller command line should not start with the following strings:

- `0x` or
- `##`

A session *password* or *controlPass* specified in a configuration file starting with the prefixes as designated above, will be considered to be a digested or obfuscated password.

Contents

6.5 Multicast Polling

The controller uses multicast polling to discover Oracle JDBC for Rdb servers that may be available on the network. Multicasting is a style of efficiently broadcasting data over a network connection to many connected servers. Any server listening in to the multicast IP address will receive the data packets that are broadcast, such as poll requests.

Oracle JDBC for Rdb servers use the Administrative Scoping range of addresses that allow easy limiting of multicast transmission to well defined boundaries within your network.

Administrative Scoping is the restriction of multicast transport based on the address range of the multicast group. It is defined by RFC 2365 "Administratively Scoped IP Multicast." and is restricted to the address range:

239.0.0.0 to 239.255.255.255

The IP address for server multicast polling should be chosen from within the following range:

239.192.0.0 to 239.192.255.255

This range is known as the IPv4 Organization Local Scope and has a subnet mask of 255.252.0.0. It is intended for use by an entire organization setting multicast scopes privately for its own internal or organizational use and allows up to 262,144 group addresses.

By default, Rdb servers use the multicast IP 239.192.1.1 with a base port of 5517.

Multicast Group IP addresses can be assigned to a server using the `srv.mcGroupIP` option within a server configuration file or the server start-up command line.

The `srv.mcBasePort` option allows you to change the Multicast Base port.

Note:

When a server participates in a multicast group, as part of the standard multicast protocol, its presence in the group will be broadcast at regular intervals. This may conflict with the network policy and procedures of your network administration.

Please consult your network manager to ensure that multicast polling is allowed on your system. Your network manager may also allocate a specific IP address and Port range that may be used by the Rdb Native Drivers, and you should change your server and session configuration files to reflect these allocated addresses.

Setting the Multicast Base port to zero (0) will effectively disable multicast broadcast and receipt for that server. This also means that the server will not respond to any POLL requests issued by the Controller.

See Polling Servers for more details on how the controller may be used to `POLL` servers.

Contents

6.6 Server Matching

To allow the selectivity of servers when issuing controller commands, certain commands may take a server matching pattern.

The server matching pattern takes the form of a standard Java Regular Expression (REGEX) and may be used to select those servers on the system that should respond and/or carry out the required operation.

Note:

In addition to the server matching criteria, only servers that are using the same Group IP will respond to controller commands such as POLL. See Multicast Polling for more details.

Currently the only controller command allowing server matching is the POLL command. See Polling Servers for more details.

Example 1

```
rdbthincontrol> poll #type:RdbThinSrvPool#node:ALPHA.*
```

In this example, the server matching pattern refines the server selection for the `POLL` request to target only Pool servers currently running on nodes starting with names "ALPHA".

Format

The format of the server match pattern is:

```
[#<match type>:<match value>]
```

where `<match type>` is one of:

- `type` the type of server
- `name` the server name

- `port` the port the server is listening on
- `stat` the state of the server
- `node` the node the server is running on
- `vers` the version of JDBC the server is running

and `<match value>` depends on the `<match type>`. Some values for `<match type>` will allow a regular expression pattern to match on. The following sections describe each `<match type>` and the allowable values for `<match value>`s.

See the Java documentation for more information on Regular Expressions.

6.6.1 Type Match

The `type <match type>` specifies the type of server that should respond. The following table shows the allowed values of `<match value>` for a `type` match:

Table 6-5 rdbThin Type Codes

Type name	Type code	Description
`RdbThinSrv`	0	Standard thin server.
`RdbThinSrvMP`	1	Multi-process server.
`RdbThinSrvSSL`	2	Thin server using SSL for communication.
`RdbThinSrvMPSSL`	3	Multi-process server using SSL.
`RdbThinSrvPool`	4	Pool server.
`RdbThinSrvPoolSSL`	5	Pool server using SSL
`RdbManSrv`	102	JDBC Manager server
`RdbManSrvSSL`	103	JDBC Manager server using SSL

The `<match value>` for server type may be either an `int` value, representing the server type code as shown above, or a string value to match the server type name.

The `<match value>` may be a regular expression. Character case will be ignored.

Example 1

All SSL server types (using a regular expression designating that the server s type name should end with SSL).

```
rdbthincontrol> poll #type:.*SSL
```

Example 2

All Multi-process server types (using a regular expression designating that the server s type code should match either the value 1 or the value 3).

```
rdbthincontrol> poll #type:[13]
```

Example 3

Any Multi-process server using SSL.

```
rdbthincontrol> poll #type:rdbthinsrvmpssl
```

6.6.2 Name Match

The name `<match type>` specifies the name of servers that should respond.
The `<match value>` may be a regular expression. Character casing will be ignored.

Example 1

All servers named "MY_MP_SRV"

```
rdbthincontrol> poll #name:MY_MP_SRV
```

Example 2

All servers with names starting with "MP_", ending with "_SRV" and any 2 characters in between.

```
rdbthincontrol> poll #name:MP_.._SRV
```

Example 3

All servers with names starting with "P_", ending with "_SRV" and 1 or more digits in between.

```
rdbthincontrol> poll #name:P_(\d+)_SRV
```

6.6.3 Port Match

The `port <match type>` specifies the port numbers for servers that should respond.
The `<match value>` may be a regular expression.

Example 1

All servers listening on port 1701

```
rdbthincontrol> poll #port:1701
```

Example 2

All servers listening on port 1700 though 1709

```
rdbthincontrol> poll #port:170\d
```

6.6.4 Stat Match

The `stat <match type>` specifies only servers currently in the specified state should respond. The following table shows the values for `<match value>` allowed for a `stat` match:

Table 6-6 rdbThin Match Values

Match Value	Description
AVAILABLE	The server is currently available and has at least one client connected.

BUSY	The server has the maximum number of clients connected. See maxClients server property.
CLOSED	The server is currently closed with no clients connected. See Closing Servers.
CLOSING	The server is currently closing but still has at least one client connected. See Closing Servers.
IDLE	The server is currently available, and no clients connected.

The <match value> may be a regular expression. Character casing will be ignored.

Example 1

All idle servers.

```
rdbthincontrol> poll #stat:idle
```

Example 2

All closed servers or servers in the process of closing (using a regular expression designating that the state should start with clos).

```
rdbthincontrol> poll #stat:clos.*
```

Example 3

All available servers irrespective of the number of clients connected.

```
rdbthincontrol> poll #stat:IDLE|BUSY|AVAIL.*
```

6.6.5 Node Match

The node <match type> specifies that only servers running on the specified node(s) should respond. The <match value> may be a regular expression. Character casing will be ignored.

The <match value> pattern can represent either the name of the node or the IP address of the node.

Example 1

All servers running on node ALPHA1.

```
rdbthincontrol> poll #node:ALPHA1
```

Example 2

All servers running on node ALPHA1, ALPHA2 and ALPHA3.

```
rdbthincontrol> poll #node:ALPHA[123]
```

Example 3

All servers running in the sub-network 192.169.1.*.

```
rdbthincontrol> poll #node:192.169.1.(\d+)
```

6.6.6 Vers Match

The `vers <match type>` specifies that only servers running under the specified Oracle JDBC for Rdb version should respond.

The `<match value>` may be a regular expression. Character casing will be ignored.

The `<match value>` pattern may optionally include the "V" prefix for the version.

The `<match value>` pattern strings **`*`**, **`.*`** and **`ALL`** are handled as special patterns and indicate that ALL versions should be matched. This wildcard version specification may be used to allow prior-version servers to respond to controller commands containing match selectivity. See Handling prior version Servers for more details.

Example 1

All servers with release 7.3 Oracle JDBC for Rdb.

```
rdbthincontrol>poll #vers:73
```

Example 2

All servers running on node ALPHA1 including prior-version servers.

```
rdbthincontrol>poll #vers:ALL#node:ALPHA1
```

6.6.7 Handling Servers From Prior Releases

Server matching relies on server functionality that is only available from release 7.3 Oracle JDBC for Rdb and later. Servers from releases prior to 7.3 will silently ignore any control requests containing server matching criteria.

If your network has prior-version JDBC servers running, while these servers may respond to standard control requests such as `POLL,` they may not respond to control statements containing server matching patterns.

For example, assuming the network has several 7.4 servers and one 7.3 server, named "`V73SRV,` the standard `POLL` request will discover all the servers:

```
rdbthincontrol> poll
Polling servers ...
mpx(1) //192.168.1.2:1899/ (0xC6C<3180>) node = froggy2
rdbthnsrv1(0) //192.168.1.2:1701/ (0x9A8<2472>) node = froggy2
rdbthnsrv2(0) //192.168.1.2:1702/ (0xB30<2864>) node = froggy2
v73srv(0) //192.168.1.2:1800/ (0xC10<3088>) node = froggy2
```

But a POLL request with matching server criteria will not:

```
rdbthincontrol> poll #node:192.168.1.\d+
Polling servers (using qualifiers : #node:192.168.1.\d+ ) ...
mpx(1) //192.168.1.2:1899/ (0xC6C<3180>) node = froggy2
rdbthnsrv1(0) //192.168.1.2:1701/ (0x9A8<2472>) node = froggy2
rdbthnsrv2(0) //192.168.1.2:1702/ (0xB30<2864>) node = froggy2
```

If match criteria are required and prior version servers are present, then the `#VERS` match condition set to `ALL` may be used to indicate to the controller that matching should be done by the controller, rather than the servers. This will allow prior version servers to respond correctly:

```
rdbthincontrol> poll #node:192.168.1.\d+#vers:all
Polling servers (using qualifiers : #node:192.168.1.\d+#vers:all ) ...
mpx(1) //192.168.1.2:1899/ (0xC6C<3180>) node = froggy2
rdbthnsrv1(0) //192.168.1.2:1701/ (0x9A8<2472>) node = froggy2
rdbthnsrv2(0) //192.168.1.2:1702/ (0xB30<2864>) node = froggy2
v73srv(0) //192.168.1.2:1800/ (0xC10<3088>) node = froggy2
```

Contents

6.7 Server Operations

This section details the operations you may carry out on servers using the controller both interactively and in command mode.

The following sub-sections describe:

- Closing Servers
- Opening Servers
- Showing Servers
- Starting Servers
- Stopping Servers
- Watching Servers
- Watching Events
- Polling Servers
- Showing Executors

Note:

The examples in this section assume that Java has been set up and the following DCL symbol has been set in the environment.

```
$ thincontrol :== 'java' -jar rdb$jdbc_home:rdbthincontrol.jar
-cfg my_servers.xml -controlpass "MySecretPassword"
```

The configuration file contents used for these examples may be seen in Sample configuration file MY_SERVERS.XML

Server Matching may be used in conjunction with the following server commands to target specific servers or groups of servers on your network.

6.7.1 Closing Servers

Active servers may be closed using the controller. You must provide a valid control password for the server.

Closing a server sets its *maxClients* attribute to zero (0) thus preventing any further connections to be made. Already established connections are not affected.

You may issue an open command later to re-open a closed server, which will reestablish the *maxClients* value for the server back to the value it was prior to closing. See Opening Servers for more details.

Only those servers where the control password matches the control session control password will be closed.

6.7.1.1 Interactive Mode

The interactive control commands available to close servers can be seen in the following table:

Table 6-7 Interactive Close Server

Command	Description
`close active servers` `close all servers`	Closes all responding servers. Keywords *all* and *active* in this context are considered synonyms.
`close server`	Closes the currently connected server.
`close server <server_connection>`	Closes the active server specified by the server connection information. See Connecting to Servers for more information.

Example

```
rdbthincontrol> close server myserver
rdbthincontrol> close server //prod_node:1766/
rdbthincontrol> close server 1701
rdbthincontrol> close active server
rdbthincontrol> close server myserver george MySecretPassword
```

6.7.1.2 Command Mode

The command mode commands available to close servers can be seen in the following table:

Table 6-8 Command Mode Close Server

Command	Required options	Additional options	Ignored options	Description

`-closeServer`		`-name`
		`-node`
		`-port`
		`-URL`
			`-active`
			`-all`
			`-using`
			`-in`
				Close the server as specified by other command line options.

`-closeServer`	`-active` or `-all`		`-name`
			`-node`
			`-port`
			`-URL`
			`-using`
			`-in`
				Close all servers that are responding to the multicast poll request.

Qualifiers specified in the **Ignored Options** column are silently ignored if present on the Command line.

Example

```
$ thincontrol -closeServer -url //prod_node:1766/
$ thincontrol -closeServer -port 1701 -node localhost
$ thincontrol -closeServer -active
$ thincontrol -closeServer -name myserver
```

6.7.2 Opening Servers

Active servers may be opened using the controller. You must provide a valid control password for the server.

Opening a server allows new client connections to be made using that server.

You may issue an open command to re-open a closed server, which will reestablish the *maxClients* value for the server back to the value it was prior to closing.

Only those servers where the control password matches the control session control password will be opened.

6.7.2.1 Interactive Mode

The control commands available to open servers can be seen in the following table:

Table 6-9 Interactive Open Server

Command	Description
`open active servers` `open all servers`	Opens all responding servers.
`open server`	Opens the currently connected server.
`open server <server_connection>`	Opens the active server specified by the server connection information. See Connecting to Servers for more information.

Example

```
rdbthincontrol> open server
rdbthincontrol> open server myserv
rdbthincontrol> open server //prod_node:1766/
rdbthincontrol> open server 1701
rdbthincontrol> open all servers
rdbthincontrol> open server //prod_node:1766/ fred mypass
```

6.7.2.2 Command Mode

The command mode commands available to open servers can be seen in the following table:

Table 6-10 Command Mode Open Server

Command	Required options	Additional options	Ignored options	Description

`-openServer`		`-name`
		`-node`
		`-port`
		`-URL`
			`-active`
			`-all`
			`-using`
			`-in`
				Opens the server as specified by other command line options.

`-openServer`	`-active` or `-all`		`-name`
			`-node`
			`-port`
			`-URL`
			`-using`
			`-in`
				Open all servers that are responding to the multicast poll request.

Qualifiers specified in the Ignored Options column are silently ignored if present on the Command line.

Example

```
$ thincontrol -openServer -url //prod_node:1766/
$ thincontrol -openServer -port 1701 -node localhost
$ thincontrol -openServer -active
$ thincontrol -openServer -name myserver
```

6.7.3 Showing Servers

Information about active and known servers may be displayed using the controller. You must provide a valid control password for the server before information will be displayed.

If showing all or active servers only those servers where the control password matches the control session control password will have information displayed.

All server definitions as stored in the configuration file will be displayed when showing stored or all servers irrespective of the control password.

6.7.3.1 Interactive Mode

The control commands available to show servers can be seen in the following table:

Table 6-11 Interactive Show Server

Command	Description
`show active servers`	Show all servers that are responding to the multicast poll request.
`show all servers`	Shows active servers as well as the server definitions as found in the configuration file used to start the controller.
`show stored servers`	Shows the server definitions as found in the configuration file used to start the controller.
`show server`	Shows information about the currently connected server.
`show server <server_connection>`	Shows information about the active server specified by the server connection information. See Connecting to servers for more information.

Example

```
rdbthincontrol> show server
rdbthincontrol> show server myserv
rdbthincontrol> show server //prod_node:1766/
rdbthincontrol> show server 1701
rdbthincontrol> show active servers
rdbthincontrol> show server //prod_node:1766/ fred mypass
```

6.7.3.2 Command Mode

The command mode commands available to show servers can be seen in the following table:

Table 6-12 Command Mode Show Server

Command	Required options	Additional options	Ignored options	Description
-showServer		-name -node -port -URL	-active -all -using -in	Shows the server as specified by other command line options.
-showServer	-active		-name -node -port -URL -using -in	Show all servers that are responding to the multicast poll request.
-showServer	-all		-name -node -port -URL -using -in	Shows active servers as well as the server definitions as found in the configuration file used to start the controller.
-showServer	-stored		-name -node -port -URL -using -in	Shows the server definitions as found in the configuration file used to start the controller.

Qualifiers specified in the Ignored Options column are silently ignored if present on the Command line.

Note:

If multiple conflicting keywords are found on the one command line only one action will be taken, and the following precedence is used:

- -all
- -active
- -stored
- specified server

Example

```
$ thincontrol -showServer -url //prod_node:1766/
$ thincontrol -showServer -port 1701 -node localhost
$ thincontrol -showServer -active
$ thincontrol -showServer -all
$ thincontrol -showServer -stored
$ thincontrol -showServer -name myserver
```

6.7.3.3 Information Displayed in Show Server

When the `show server` command is executed, the following information about the server is displayed:

Table 6-13 Show Server display Attributes

Attribute	Description
RDB$NODE	The node the server is running on.
RDB$PORT	The TCP/IP port the server is listening on.
RDB$STATUS	The status of the server.
RDB$SERVER_NAME	The name of the server.
RDB$SERVER_TYPE	The type of server.
RDB$SERVER_VERSION	The version information for the server JAR file used. Format Release release# build# internalCheckId .
RDB$SERVER_SHR_VERSION	The version of the JDBC shared images used. Format Release release# build# internalCheckId .
RDB$SERVER_PID	The process identification of the server process. Format hexadecimal PID (decimal PID) .
RDB$ALLOWS_ANON	If **true ,** the server allows anonymous connections. See Anonymous Usernames.
RDB$ALLOWS_BYPASS	If **true ,** the server will use BYPASS during database connection setup. See BYPASS Privilege.
RDB$NUMBER_OF_CLIENTS	The number of non-control client connections currently using the server.
RDB$MAX_CLIENTS	The maximum number of non-control client connections the server will allow.
RDB$TRACE_LEVEL	The current trace level for the server.
RDB$LOG_FILE	The log file currently being used.
RDB$RESTRICT_ACCESS	If **true** , access is restricted to this server.
RDB$CAST_INFO	Multicast information. Format Group IP:Base port .
RDB$START_TIMESTAMP	Timestamp when server was started.
RDB$CFG_FILE	The server configuration file.

In addition, if the connected server is Multi-process then the following attributes will also be displayed:

Table 6-14 Additional Multi-process Server Attributes

Attribute	Description
RDB$NUMBER_OF_FREE_EXECUTORS	The number of free executor processes.
RDB$MAX_FREE_EXECUTORS	The maximum number of free executor processes that the server will maintain.
RDB$SHARED_MEMORY	The amount of shared memory (bytes) allocated by the server.
RDB$FREE_SHARED_MEMORY	The amount of shared memory (bytes) allocated but not used.
RDB$MEM_SEG_STRINGS	The amount of shared memory (bytes) used by segmented strings (blobs).
RDB$NUMBER_OF_CHUNKS	The number of free chunks of shared memory available for use.
RDB$LARGEST_CHUNK	The amount of memory (bytes) of the largest memory chunk available.
RDB$NUMBER_OF_BUSY_EXECUTORS	The number of executor processes currently allocated to active connections.
RDB$MEM_PAGEFILE_QUOTA	Server process JPI$_PGFLQUOTA value.
RDB$MEM_PAGEFILE_COUNT	Server process JPI$_PAGFILCNT value.
RDB$MEM_GLOBPAGE_COUNT	Server process JPI$_GPGCNT value.
RDB$MEM_WS_COUNT	Server process JPI$_PPGCNT value.
RDB$MEM_PAGETBL_COUNT	Server process JPI$_APTCNT value.
RDB$MEM_PAGE_FAULTS	Server process JPI$_PAGEFLTS value.
RDB$MEM_CPU_TIME	Server process JPI$_CPUTIM value.

Note:

The format and contents of the `show server` display may change in future releases of Oracle JDBC for Rdb.

Example

```
rdbthincontrol> connect 1804
rdbthincontrol> show server
RDB$NODE : nnn.nnn.nnn.nnn
RDB$PORT : 1804
RDB$STATUS : Idle
```

```
RDB$SERVER_NAME : mppool2
RDB$SERVER_TYPE : RdbThinSrvPool
RDB$SERVER_VERSION : Release 7.4.1.0.0 20250113 BP1D
RDB$SERVER_SHR_VERSION : Release 7.4.1.0.0 20250113 BP1D
RDB$SERVER_PID : 0x22A7DF71(581427057)
RDB$ALLOWS_ANON : false
RDB$ALLOWS_BYPASS : false
RDB$NUMBER_OF_CLIENTS : 0
RDB$MAX_CLIENTS : -1
RDB$TRACE_LEVEL : -1
RDB$LOG_FILE : null
RDB$RESTRICT_ACCESS : false
RDB$CAST_INFO : 239.192.1.1:5517
RDB$START_TIMESTAMP : 2025-01-12 9:57:12.531
RDB$CFG_FILE : [myDir]jdbccfg.xml
rdbthincontrol> connect 1881
rdbthincontrol> show server

RDB$NODE : nnn.nnn.nnn.nnn
RDB$PORT : 1881
RDB$STATUS : Idle
RDB$SERVER_NAME : srv1MP
RDB$SERVER_TYPE : RdbThinSrvMP
RDB$SERVER_VERSION : Release 7.4.1.0.0 20250113 BP1D
RDB$SERVER_SHR_VERSION : Release 7.4.1.0.0 20250113 BP1D
RDB$SERVER_PID : 0x22A7DF73(581427059)
RDB$ALLOWS_ANON : false
RDB$ALLOWS_BYPASS : false
RDB$NUMBER_OF_CLIENTS : 0
RDB$MAX_CLIENTS : 10
RDB$TRACE_LEVEL : 0
RDB$LOG_FILE : srv1MP.log
RDB$RESTRICT_ACCESS : false
RDB$CAST_INFO : 239.192.1.1:5517
RDB$START_TIMESTAMP : 2025-01-12 10:01:38.921
RDB$CFG_FILE : [myDir]jdbccfg.xml
RDB$NUMBER_OF_FREE_EXECUTORS : 2
RDB$MAX_FREE_EXECUTORS : 2
RDB$SHARED_MEMORY : 1048576
RDB$FREE_SHARED_MEMORY : 432312
RDB$NUMBER_OF_CHUNKS : 1
RDB$LARGEST_CHUNK : 432292
RDB$NUMBER_OF_BUSY_EXECUTORS : 0
RDB$MEM_PAGEFILE_QUOTA : 0
RDB$MEM_PAGEFILE_COUNT : 9
RDB$MEM_GLOBPAGE_COUNT : 0
RDB$MEM_WS_COUNT : 7
RDB$MEM_PAGETBL_COUNT : 0
RDB$MEM_PAGE_FAULTS : 7105
RDB$MEM_CPU_TIME : 57
rdbthincontrol>
```

6.7.4 Starting Servers

Servers may be started using the controller.

If the server specifies a node or URL that is not the same as the node the controller is running on you must use the VIA command qualifier otherwise an exception will be raised.

6.7.4.1 Interactive Mode

The control commands available to start servers can be seen in the following table:

Table 6-15 Interactive Start Server

Command	Description
`start all servers`	Starts all `autostart` servers found in the XML-formatted configuration file used when invoking the controller. Only those servers that have the `autostart` attribute and are for the local host will be started.
`Start server`	Starts a server of type `RdbThinSrv` on the local host with all default characteristics.
`Start server <port id>`	Starts a server of type `RdbThinSrv` listening on the designated port on the local host with default remaining characteristics.
`Start server <name>`	Starts the server that matches the <name> provided. See XML formatted Configuration File for more information on named server definitions.
`Start server <name> via <server_connection>`	Starts the server that matches the <name> provided, by forwarding the request to the JDBC Manager server specified by `<server_connection>`.

Example

```
rdbthincontrol> start server myserver
rdbthincontrol> start server 1799
rdbthincontrol> start server all
rdbthincontrol> start server serv1 via manag1
```

6.7.4.2 Command Mode

The command mode commands available to start servers can be seen in the following table:

Table 6-16 Command Mode Start Server

Command	Required options	Additional options	Ignored options	Description

`-startServer`		`-name`
		`-node`
		`-port`
		`-URL`
			`-active`
			`-all`
			`-using`
			`-in`
				Starts the server as specified by other command line options.

`-startServer -all`			`-name`
			`-node`
			`-port`
			`-URL`
			`-using`
			`-in`
				Starts all `autostart` servers found in the XML-formatted configuration file used when invoking the controller.

Qualifiers specified in the Ignored Options column are silently ignored if present on the Command line.

Note:
There is no command mode equivalence of the start server via interactive node command.

Example

```
$ thincontrol -startServer -port 1701 -node localhost
$ thincontrol -startServer -name myserver
$ thincontrol -startServer -all
```

6.7.5 Stopping Servers

Active servers may be stopped using the controller. You must provide a valid control password for the server.

Only those servers where the control password matches the control session control password will be stopped.

Note:
Stopping a server will forcibly terminate all database connections on that server and does not wait for client transaction completion.

Consider using the `Close Server` command first, to stop further client connections and then use the `Stop Server` command later when no clients are bound. See Closing Servers for more details.

You may use `Show Server` or `Show Clients` command to see if any clients are currently using the server. See Showing Servers for more details.

6.7.5.1 Interactive Mode

The control commands available to stop servers can be seen in the following table:

Table 6-17 Interactive Stop Server

Command	Description
`stop active servers` `stop all servers`	Stops all responding servers. The keywords `all` and `active` in this context are considered synonyms.
`stop server`	Stops the currently connected server.
`stop server <server_connection>`	Stops the active server specified by the server connection information. See Connecting to servers for more information.

Example

```
rdbthincontrol> stop server
rdbthincontrol> stop server myserv
rdbthincontrol> stop server //prod_node:1766/
rdbthincontrol> stop server 1701
rdbthincontrol> stop active servers
rdbthincontrol> stop server //prod_node:1766/ fred mypass
```

6.7.5.2 Command Mode

The command mode commands available to stop servers can be seen in the following table:

Table 6-18 Command Mode Stop Server

Command	Required options	Additional options	Ignored options	Description

`-stopServer`		`-name`
		`-node`
		`-port`
		`-URL`
			`-active`
			`-all`
			`-using`
			`-in`
				Stops the server as specified by other command line options.

`-stopServer`	`-active or -all`		`-name`
			`-node`
			`-port`
			`-URL`
			`-using`
			`-in`
				Stops all responding servers.

Qualifiers specified in the Ignored Options column are silently ignored if present on the Command line.

Example

```
$ thincontrol -stopServer -url //prod_node:1766/
$ thincontrol -stopServer -port 1701 -node localhost
$ thincontrol -stopServer -active
$ thincontrol -stopServer -name myserver
```

6.7.6 Watching Servers

The trace output for an active server may be displayed on the controller console. You must provide a valid control password for the server to be able to watch its trace.

Only those servers where the control password matches the control session control password will be watched.

When you watch a server, all trace output from that server will also be sent to the current console running the controller.

The display of trace output messages occurs asynchronously with the command line interface. The same trace information will also be sent to the server s log file.

Note:
Because the server uses Java logger to log trace message to remote consoles such as the controller, the output from the server will be buffered prior to being sent across the network to the console. This means that the trace output may be displayed sporadically on the console as the buffer is first filled and then flushed.

6.7.6.1 Interactive Mode

The control commands available to watch servers can be seen in the following table:

Table 6-19 Interactive Watch Server

Command	Description
`watch [server]`	Watch the currently connected server.
`watch server <server_connection>`	Watch the active server specified by the server connection information. See Connecting to servers for more information.

Example

```
rdbthincontrol> watch server myserv
rdbthincontrol> watch server //prod_node:1766/
rdbthincontrol> watch server 1701 jack password1
rdbthincontrol> watch
```

6.7.6.2 Command Mode

The command mode commands available to watch servers can be seen in the following table:

Table 6-20 Command Mode Watch Server

Command	Required options	Additional options	Ignored options	Description

`-watch`		`-name`
		`-node`
		`-port`
		`-URL`
			`-active`
			`-all`
			`-using`
			`-in`
				Watches the server as specified by other command line options.

Example

```
$ thincontrol -port 1701 -node localhost -watch
$ thincontrol -cfg my_cfg.xml -name myserver -watch
```

6.7.7 Watching Events

Since release 7.3.1

Logged event output for an active server may be displayed on the controller console. You must provide a valid control password for the server to be able to watch for events.

When you watch for events on a server, all Events logged by that server will also be sent to the current console running the controller. The logged event output is in the form of XML-formatted messages.

The display of event output messages occurs asynchronously with the command line interface.

Note:
Because the server uses Java logger to log trace message to remote consoles such as the controller, the output from the server will be buffered prior to being sent across the network to the console. This means that the trace output may be displayed sporadically on the console as the buffer is first filled and then flushed.

See also Event Logging and Notification.

6.7.7.1 Interactive Mode

The control commands available to watch events can be seen in the following table:

Table 6-21 Interactive Watch Events

Command	Description
`watch events`	Watch for events on the currently connected server.
`watch events [on] <server_connection>`	Watch for events on the active server specified by the server connection information. The keyword ON is optional. See Connecting to servers for more information.

Example

```
rdbthincontrol> watch events myserv
rdbthincontrol> watch events //prod_node:1766/
rdbthincontrol> watch events on 1701 jack password1
rdbthincontrol> watch events
```

6.7.7.2 Command Mode

The command mode commands available to watch events can be seen in the following table:

Table 6-22 Command Mode Watch Events

Command	Required options	Additional options	Ignored options	Description

`-watchEvents`		`-name`
		`-node`
		`-port`
		`-URL`
			`-active`
			`-all`
			`-using`
			`-in`
				Watches events logged by the server as specified by other command line options.

Example

```
$ thincontrol -port 1701 -node localhost -watchEvents
$ thincontrol -cfg my_cfg.xml -name myserver -watchEvents
```

6.7.8 Polling Servers

The poll command uses the multicast information to poll responding Oracle JDBC for Rdb servers.

Each available server will respond with information about which node and port it is listening on. In addition the poll response identifies the Process ID the server is using on that node.

A control password is not required to use the poll command.

As the operation is a poll, the thin controller will wait a prescribed amount of time before displaying the responses from the poll request. The `pollTimeout` attribute specifies the amount of time in milliseconds that the controller should wait for a reply on the receiving socket. If no reply is received within that period the poll operation is deemed complete and the responding servers will be sorted and listed.

The default value for the `pollTimeout` attribute is 2000 milliseconds (2 seconds).

If the network or the servers are very busy it is possible that a server may not respond within the default time. If this is the case, the timeout may be increased by specifying a larger value for the `pollTimeout` attribute in either the command line or the session section of the configuration file used when invoking the controller.

The `pollTimeout` attribute may also be set within a controller session using the `Set` command on the controller command line.

6.7.8.1 Interactive Mode

The control commands available to poll servers can be seen in the following table:

Table 6-23 Interactive Poll Server

Command	Description
`poll`	Poll active servers.

Example

```
rdbthincontrol> poll
Polling servers ...
i73spregtestsrv(5) //nnn.nnn.nnn.nnn:2505/ (0x23E3B538<602125624>) node = alfred
i73sslregtstsrv(2) //nnn.nnn.nnn.nnn:2503/ (0x2400FF7E<604045182>) node = victoria
i73sslregtstsrv(2) //nnn.nnn.nnn.nnn:2503/ (0x23E3CC2A<602131498>) node = alfred
regtestsrv_a71(0) //nnn.nnn.nnn.nnn:1850/ (0x2026C201<539410945>) node = spencer
rdbthincontrol>
```

6.7.8.2 Command Mode

The command mode commands available to poll servers can be seen in following table:

Table 6-24 Command Mode Poll Server

Command	Required options	Additional options	Ignored options	Description

`-poll`			`-active`
			`-all`
			`-name`
			`-node`
			`-port`
			`-URL`
			`-using`
			`-in`
				Poll active servers.

Example

```
$ thincontrol -poll
```

6.7.9 Poll Sub-commands

Starting with release 7.3.0.1, the controller will allow subcommands to be issued during a POLL request.

When a listening server receives a POLL request it may optionally carry-out a sub-command sent to it within the POLL request message.
There is currently only one recognized POLL sub-command:

- Reopenlogs

Note:
The POLL sub-command functionality is only available when using the thin controller JAR from release 7.3.0.1 (or above). Only release 7.3.0.1 (or above) servers will accept a POLL sub-command.

6.7.9.1 Reopenlogs Sub-command

The `reopenlogs` subcommand tells the listening server to re-open its log files allowing the prior version of the files to be read or copied.

Due to a restriction within Java on OpenVMS, log files opened by a JDBC server cannot be read or copied while the log files are currently being used by the server thus, to see the contents of these files, they must first be closed by the server process.

On receiving this sub-command, the server will flush out its log stream, and then close its currently opened log files. It will then create a new version of the log files using the same file names.

6.7.9.2 Interactive Mode

The control commands available to poll servers can be seen in the following table:

Table 6-25 Interactive Poll reopenlogs

Command	Description
`poll reopenlogs`	Poll active servers and request the log files to be re-opened.

Example

```
rdbthincontrol> poll reopenlogs
Polling servers ...
i73spregtestsrv(5) //nnn.nnn.nnn.nnn:2505/ (0x23E3B538<602125624>) node = alfred
i73sslregtstsrv(2) //nnn.nnn.nnn.nnn:2503/ (0x2400FF7E<604045182>) node = victoria
i73sslregtstsrv(2) //nnn.nnn.nnn.nnn:2503/ (0x23E3CC2A<602131498>) node = george
regtestsrv_a71(0) //nnn.nnn.nnn.nnn:1850/ (0x2026C201<539410945>) node = spencer
rdbthincontrol> poll reopenlogs #name:regtestsrv_a71
Polling servers (using qualifiers : #name:regtestsrv_a71) ...
regtestsrv_a71(0) //nnn.nnn.nnn.nnn:1850/ (0x2026C201<539410945>) node = spencer
```

6.7.9.3 Command Mode

The command mode commands available to poll servers can be seen in following table:

Table 6-26 Command Mode Poll reopenlogs

Command	Required options	Additional options	Ignored options	Description

`-poll reopenlogs`			`-active`
			`-all`
			`-name`
			`-node`
			`-port`
				Poll active servers and request the log files to be re-opened.

Command	Required options	Additional options	Ignored options	Description
			-URL -using -in	

Example

```
$ thincontrol -poll -reopenlogs
$ thincontrol -poll -reopenlogs #name:MY_MP_SRV
```

6.7.10 Showing Executors

Since release 7.3.2

Information about executors within the currently connected Multi-process server may be displayed using the controller.

6.7.10.1 Interactive Mode

The control commands available to show executors can be seen in following table:

Table 6-27 Interactive Show Executors

Command	Description
show executors	Shows all executors used by the currently connected server.

Example

```
rdbthincontrol> show executors
```

6.7.10.2 Command Mode

The command mode commands available to show executors can be seen in following table:

Table 6-28 Command Mode Show Executors

Command	Additional options	Ignored options	Description
-showexecutors		-active -all	Show executors on the currently connected server.
	-name -node -port -URL		If present will be used to establish a connection to a server.

Qualifiers specified in the Ignored Options column are silently ignored if present on the Command line.

Example

```
$ thincontrol -showexecutors  -port 1701 -node mynode
```

6.7.10.3 Information Displayed in Show Executors

When the `show executors` command is executed, information about each executor is displayed as a list of attributes:

Table 6-29 Show Executors Display Attributes

Attribute	Description
RDB$EXECUTOR_NAME	Internal unique JDBC executor name allocated to each new executor.
RDB$PID	Show the OpenVMS process ID of the executor process.
RDB$STATE	Current state of the executor.
RDB$NUM_STMTS	The number of open and compiled SQL statements.
RDB$NUM_CURSORS	The number of cursors allocated.
RDB$MEM_SQLDAS	The amount of shared memory (bytes) used by SQLDA structures.
RDB$MEM_BUFFERS	The amount of shared memory (bytes) used by buffers.
RDB$MEM_SEG_STRINGS	The amount of shared memory (bytes) used by segmented strings (blobs).
RDB$MEM_NUM_FREE_CHUNKS	The number of free chunks of shared memory available for use.
RDB$MEM_LARGEST_CHUNK	The amount of memory (bytes) of the largest memory chunk available.
RDB$MEM_TOTAL_FREE	The amount of shared memory (bytes) allocated to this executor and still free.
RDB$MEM_PAGEFILE_QUOTA	Executor process JPI$_PGFLQUOTA value.
RDB$MEM_PAGEFILE_COUNT	Executor process JPI$_PAGFILCNT value.
RDB$MEM_GLOBPAGE_COUNT	Executor process JPI$_GPGCNT value.
RDB$MEM_WS_COUNT	Executor process JPI$_PPGCNT value.
RDB$MEM_PAGETBL_COUNT	Executor process JPI$_APTCNT value.
RDB$MEM_PAGE_FAULTS	Executor process JPI$_PAGEFLTS value.
RDB$MEM_CPU_TIME	Executor process JPI$_CPUTIM value.

Note:

The format and contents of the `show executors` display may change in future releases of Oracle JDBC for Rdb.

Example

```
rdbthincontrol> show executors
//localhost:1799/ is currently running 1 executor

RDB$EXECUTOR_NAME : JDBCDEV00000001
RDB$PID : 0x22A7DF71(581427057)
RDB$STATE : Client 00000004 connected d1:[jdbc.74]mf_personnel as user1
RDB$NUM_STMTS : 1
RDB$NUM_CURSORS : 0
RDB$MEM_SQLDAS : 139496
RDB$MEM_BUFFERS : 0
RDB$MEM_SEG_STRINGS : 0
RDB$MEM_NUM_FREE_CHUNKS : 2
RDB$MEM_LARGEST_CHUNK : 165008
RDB$MEM_TOTAL_FREE : 165800
RDB$MEM_PAGEFILE_QUOTA : 718750
RDB$MEM_PAGEFILE_COUNT : 576751
RDB$MEM_GLOBPAGE_COUNT : 945
RDB$MEM_WS_COUNT : 2070
RDB$MEM_PAGETBL_COUNT : 0
RDB$MEM_PAGE_FAULTS : 3987
RDB$MEM_CPU_TIME : 55
rdbthincontrol>
```

6.7.11 Showing Server Pool

Information about pooled servers being maintained by the currently connected pool server may be displayed using the controller.

6.7.11.1 Interactive Mode

The control commands available to show pooled servers can be seen in following table:

Table 6-30 Interactive Show ServerPool

Command	Description
`show serverpool`	Shows all pooled servers maintained by the currently connected pool server.

Example

```
rdbthincontrol> show serverpool
```

6.7.11.2 Command Mode

The command mode commands available to show server pools can be seen in following table:

Table 6-31 Command Mode Show Clients

Command	Additional options	Ignored options	Description
`-showserverpool`		`-active` `-all`	Show pooled servers maintained by the currently connected server.

	`-name`
	`-node`
	`-port`
	`-URL`
			If present will be used to establish a connection to a server.

Qualifiers specified in the Ignored Options column are silently ignored if present on the Command line.

Example

```
$ thincontrol -showserverpool  -port 1701 -node mynode
```

6.7.11.3 Information Displayed in Show ServerPool

When the `show serverpool` command is executed, information about each pooled server is displayed as a list of attributes:

Table 6-32 Show Executors Display Attributes

Attribute	Description
`RDB$SERVER_NAME`	Server name.
`RDB$URL`	URL of the server. Format //node:port/
`RDB$PID`	The process identification of the server process. Format hexadecimal PID (decimal PID) .
`RDB$VERSION`	The version information for the server JAR file used. Format Release release# build# .
`RDB$MAX_CLIENTS`	Maximum number of non-control client connections allowed by server.
`RDB$NUM_CLIENTS`	Current number of non-control client connections.
`RDB$STATE`	Current state of the pooled server. Possible values: RUNNING pooled server available. NOT RESPONDING pooled server not responding to poll server status request.

Note:

The format and contents of the `show serverpool` display may change in future releases of Oracle JDBC for Rdb.

Example

```
rdbthincontrol> connect 1804
rdbthincontrol> show serverpool
Server Pool controlled by //localhost:1804/ contains 3 servers:

RDB$SERVER_NAME : srv1MP
RDB$URL : //localhost:1881/
RDB$PID : 5804
RDB$VERSION : 74100:20250107
RDB$MAX_CLIENTS : 10
RDB$NUM_CLIENTS : 0
RDB$STATE : RUNNING

RDB$SERVER_NAME : srv2MP
RDB$URL : //localhost:1882/
RDB$PID : 5032
RDB$VERSION : 74100:20250107
RDB$MAX_CLIENTS : 10
RDB$NUM_CLIENTS : 0
RDB$STATE : RUNNING

RDB$SERVER_NAME : srv3MP
RDB$URL : //localhost:1883/
RDB$PID : 5960
RDB$VERSION : 74100:20250107
RDB$MAX_CLIENTS : 10
RDB$NUM_CLIENTS : 0
RDB$STATE : RUNNING
rdbthincontrol>
```

Contents

6.8 Client Operations

6.8.1 Showing Clients

Information about clients within active servers may be displayed using the controller. You must provide a valid control password for the server.

Clients will only be displayed for those servers where the control password matches the control session control password.

6.8.1.1 Interactive Mode

The control commands available to show clients can be seen in following table:

Table 6-33 Interactive Show Clients

Command	Description
`show active clients` `show all clients`	Shows all clients on responding servers.
`show active clients <name>` `show all clients <name>`	Shows all clients with username `<name>` on responding servers.
`show active clients in <database_spec>`	Shows all clients currently connected to the specified database on all responding servers.

Command	Description
`show all clients in <database_spec>`	
`show clients`	Shows all clients in the currently connected server.
`show clients in <database_spec>`	Shows all clients currently connected to the specified database on the currently connected server.

Example

```
rdbthincontrol> show active clients
rdbthincontrol> show all clients fred
rdbthincontrol> show clients
rdbthincontrol> show clients in disk1:[dbc]pers
rdbthincontrol> show all clients in disk1:[dbc]pers
```

6.8.1.2 Command Mode

The command mode commands available to show clients can be seen in following table:

Table 6-34 Command Mode Show Clients

Command	Required options	Additional options	Ignored options	Description
`-showClient`	`<client id>`		`-active` `-all`	Show specified client on the currently connected server.

		`-name`
		`-node`
		`-port`
		`-URL`
				If present will be used to establish a connection to a server.

		`-using` `<user>`		If present specifies that only users using the username <user> should be shown.
`-showClients`				Show all clients on the currently connected server.

		`-name`
		`-node`
		`-port`
		`-URL`
				If present will be used to establish a connection to a server.

		`-using` `<user>`		If present specifies that only users using the username <user> should be shown.
		`-in` `<database_spec>`		If present specifies that only users connected to <database_spec> should be shown.

`-showClients`	`-all or active`		`-name`
			`-node`
			`-port`
			`-URL`
				Show all clients on all responding servers.

Command	Required options	Additional options	Ignored options	Description
		-using <user>		If present specifies that only users using the username <user> should be shown.
		-in <database_spec>		If present specifies that only users connected to <database_spec> should be shown.

Qualifiers specified in the Ignored Options column are silently ignored if present on the Command line.

Example

```
$ thincontrol -showclients -all
$ thincontrol -showclients -port 1701 -node mynode
$ thincontrol -showclients -all -in db_dir:personnel
$ thincontrol -showclients -all -using joeblogs
```

6.8.1.3 Information Displayed in Show Client

When the show client command is executed, information about each client that matches any selection criteria provided is displayed as a list of attributes:

Table 6-35 Show Clients display attributes

Attribute	Description
RDB$CLIENT_ID	Internal unique JDBC client id allocated to each new connection request. An asterisk (*) suffix indicates your current control connection.
RDB$URL	The database specification component of the URL provided by the client connection. If this is a control connection the following string will be displayed: <CONTROL CONNECTION> .
RDB$USER	The username used by this client. In addition, if an application name has been provided for this connection (using the @app connection option) it will be displayed in parenthesis after the username. See Connection Options for more information on the use of the @app option. If this is a control connection the following string will be displayed: <control> .
RDB$IP	The IP of the source of the connection request. Format node:port .

Attribute	Description
RDB$PID	The identification of the thread this client is using on the server. Format hexadecimal PID:THREAD_ID (decimal PID:THREAD_ID) . If this is a control connection only the PID portion will be displayed.
RDB$PID_AT_EXECUTOR	The process identification of the executor this client is using. Format hexadecimal PID (decimal PID) . If the server is not multi-process, this attribute indicates the process identification of the server and the unique connection number allocated to this connection. Format hexadecimal PID:CONNECT_ID (decimal PID:CONNECT_ID) .
RDB$LAST_SQL	The last SQL statement (if any) issued by the client.
RDB$LAST_ACTION	The last action issued by client. Format timestamp : action .
RDB$LAST_EXCEPTION	The last exception (if any) raised on this client request within the server.
RDB$TIME_SINCE_LAST_ACTION	The amount of time elapsed since the last action recorded for this client on the server. Format days hours:minutes .
RDB$MINUTES_SINCE_LAST_ACTION	The amount of time elapsed in minutes since the last action recorded for this client on the server. Format minutes .
RDB$LAST_OPEN	Timestamp of the last connection made by this client.
RDB$PID_AT_DB	***Since release 7.3.2.0.2*** The process identification of the internal Rdb Stream this client is using. Format hexadecimal PID: decimal STREAM_ID . This is the same PID format used by other Rdb facilities such as RMU to uniquely identify client connections within the database. The Rdb stream information is only available if the client has connected to a database within an Rdb environment of Oracle Rdb release 7.2.5 or above, and the JDBC server used is release 7.3.2.0.2 or above.

In addition, if the connected server is Multi-process, then the following attributes will also be displayed for each client s executor process (control connections will have all values set to zero (0)):

Table 6-36 Additional Multi-process Server Attributes

Attribute	Description
RDB$NUM_STMTS	The number of open and compiled SQL statements.
RDB$NUM_CURSORS	The number of cursors allocated.
RDB$MEM_SQLDAS	The amount of shared memory (bytes) used by SQLDA structures.
RDB$MEM_BUFFERS	The amount of shared memory (bytes) used by buffers.
RDB$MEM_SEG_STRINGS	The amount of shared memory (bytes) used by segmented strings (blobs).
RDB$MEM_NUM_FREE_CHUNKS	The number of free chunks of shared memory available for use.
RDB$MEM_LARGEST_CHUNK	The amount of memory (bytes) of the largest memory chunk available.
RDB$MEM_TOTAL_FREE	The amount of shared memory (bytes) allocated to this executor and still free.
RDB$MEM_PAGEFILE_QUOTA	Executor process JPI$_PGFLQUOTA value.
RDB$MEM_PAGEFILE_COUNT	Executor process JPI$_PAGFILCNT value.
RDB$MEM_GLOBPAGE_COUNT	Executor process JPI$_GPGCNT value.
RDB$MEM_WS_COUNT	Executor process JPI$_PPGCNT value.
RDB$MEM_PAGETBL_COUNT	Executor process JPI$_APTCNT value.
RDB$MEM_PAGE_FAULTS	Executor process JPI$_PAGEFLTS value.
RDB$MEM_CPU_TIME	Executor process JPI$_CPUTIM value.

Note:

The format and contents of the `show clients` display may change in future releases of Oracle JDBC for Rdb.

Example

```
rdbthincontrol> show clients
RDB$CLIENT_ID : 00000002*
RDB$URL : <CONTROL CONNECTION>
RDB$USER : <control>
```

```
RDB$IP : 127.0.0.1:62799
RDB$PID : 0x22A57736(581269302)
RDB$PID_AT_EXECUTOR : 0x22A51437:1(581243959:1)
RDB$LAST_SQL :
RDB$LAST_ACTION : 2025-01-12 18:43:56.416 : INIT_CONTROL
RDB$LAST_EXCEPTION :
RDB$TIME_SINCE_LAST_ACTION : 0 01:46
RDB$MINUTES_SINCE_LAST_ACTION : 106
RDB$LAST_OPEN :
RDB$PID_AT_DB :
RDB$NUM_STMTS : 0
RDB$NUM_CURSORS : 0
RDB$MEM_SQLDAS : 0
RDB$MEM_BUFFERS : 0
RDB$MEM_SEG_STRINGS : 0
RDB$MEM_NUM_FREE_CHUNKS : 0
RDB$MEM_LARGEST_CHUNK : 0
RDB$MEM_TOTAL_FREE : 0
RDB$MEM_PAGEFILE_QUOTA : 0
RDB$MEM_PAGEFILE_COUNT : 0
RDB$MEM_GLOBPAGE_COUNT : 0
RDB$MEM_WS_COUNT : 0
RDB$MEM_PAGETBL_COUNT : 0
RDB$MEM_PAGE_FAULTS : 0
RDB$MEM_CPU_TIME : 0

RDB$CLIENT_ID : 00000005
RDB$URL : d1:[jdbc.74]mf_personnel
RDB$USER : user1 [rSQL]
RDB$IP : 127.0.0.1:62850
RDB$PID : 0x22A57736:2(581269302:2)
RDB$PID_AT_EXECUTOR : 0x22A51437:1(581243959:1)
RDB$LAST_SQL :
RDB$LAST_ACTION : 2025-01-12 20:29:49.919 : CONNECT_SECURE_V73
RDB$LAST_EXCEPTION :
RDB$TIME_SINCE_LAST_ACTION : 0 00:00
RDB$MINUTES_SINCE_LAST_ACTION : 0
RDB$LAST_OPEN : 2025-01-12 20:29:50.044
RDB$PID_AT_DB : 22A57736:3
RDB$NUM_STMTS : 1
RDB$NUM_CURSORS : 0
RDB$MEM_SQLDAS : 139496
RDB$MEM_BUFFERS : 0
RDB$MEM_SEG_STRINGS : 0
RDB$MEM_NUM_FREE_CHUNKS : 2
RDB$MEM_LARGEST_CHUNK : 165008
RDB$MEM_TOTAL_FREE : 165800
RDB$MEM_PAGEFILE_QUOTA : 718750
RDB$MEM_PAGEFILE_COUNT : 576763
RDB$MEM_GLOBPAGE_COUNT : 1200
RDB$MEM_WS_COUNT : 2071
RDB$MEM_PAGETBL_COUNT : 0
RDB$MEM_PAGE_FAULTS : 3608
RDB$MEM_CPU_TIME : 47
rdbthincontrol>
```

6.8.2 Stopping Clients

Clients within active servers may be stopped using the controller. You must provide a valid control password for the server.

Clients will only be stopped in those servers where the control password matches the control session control password.

If a database file specification is used, only those clients current connected to that database will be stopped. The database file specification must match exactly, ignoring character case, to that shown in the `show clients` output.

Note:

Stopping a client will forcibly terminate all database connections on that server for that client and does not wait for client transaction completion.

You may use the `Show Clients` command to see clients that are currently using the server. See Showing Clients for more details.

In the following command, if `<client_id>` is provided, it must match a client id returned by the show clients command. Leading zeroes (0) may be left off the `<client_id>`.

6.8.2.1 Interactive Mode

The control commands available to stop clients can be seen in the following table:

Table 6-37 Interactive Stop Clients

Command	Description
`stop active clients` `stop all clients`	Stops all clients on responding servers.
`stop active clients <name>` `stop all clients <name>`	Stops all clients with username `<name>` on responding servers.
`stop active clients in<database_spec>` `stop all clients in <database_spec>`	Stops all clients currently connected to the specified database on all responding servers.
`stop clients`	Stops all clients in the currently connected server.
`stop clients in<database_spec>`	Stops all clients on the currently connected server that are currently connected to the specified database.
`stop client <client_id>`	Stops the specified client on the currently connected server.

Example

```
rdbthincontrol> stop active clients
rdbthincontrol> stop all clients fred
rdbthincontrol> stop clients
rdbthincontrol> stop client 0000000A
rdbthincontrol> stop all clients in disk1:[dbs]pers
```

6.8.2.2 Command Mode

The command mode commands available to stop clients can be seen in the following table:

Table 6-38 Command Mode Stop Clients

Command	Required options	Additional options	Ignored options	Description
-stopClient	<client id>		-active -all	Stops specified client on the currently connected server.
		-name -node -port -URL		If present will be used to establish a connection to a server.
		-using <user>		If present specifies that only users using the username <user> should be stopped.
-stopClients				Stops all clients on the currently connected server.
		-name -node -port -URL		If present will be used to establish a connection to a server.
		-using <user>		If present specifies that only users using the username <user> should be stopped.
		-in <database_spec>		If present specifies that only users connected to <database_spec> should be stopped.
-stopClients	-all or active		-name -node -port -URL	Stops all clients on all responding servers.
		-using <user>		If present specifies that only users using the username <user> should be stopped.
		-in <database_spec>		If present specifies that only users connected to <database_spec> should be stopped.

Qualifiers specified in the **Ignored Options** column are silently ignored if present on the Command line.

Example

```
$ thincontrol -stopClient 0000000A
$ thincontrol -stopClients -all
$ thincontrol -stopClients -active -in db_dir:mf_personnel
$ thincontrol -stopClients -all -using joeblogs
$ thincontrol -stopClients -port 1701 -node mynode -using joeblogs
```

6.9 Other Commands

The Controller has several commands that are neither server nor client operations:

- digest
- obfuscate

6.9.1 Digest

`Digest` will create a non-reversible obfuscated control password to be used in configuration files. See obfuscating Control Passwords for more details.

6.9.1.1 Interactive Mode

The control command available to obfuscate control passwords can be seen in following table:

Table 6-39 Interactive Mode Digest

Command	Description
`digest <plain text pwd>`	Obfuscates the control password using digest.

Example

```
rdbthincontrol> digest thisismypassword
digest : 0x31435008693CE6976F45DEDC5532E2C1
```

6.9.1.2 Command Mode

The command mode commands available to obfuscate user passwords can be seen in following table:

Table 6-40 Command Mode Digest

Command	Description
`-digest <plain text pwd>`	Obfuscates the control password using digest.

Example

```
$ java -jar rdbthincontrol.jar -digest "MySecretPassword"
digest : 0x7315A012ECAD1059A3634F8BE1347846
```

Note:

If double quotation characters are not used to surround the plain text password DCL may, depending on your environment, force the value to all lower case or all upper case which may differ from the original.

6.9.2 Obfuscate

`Obfuscate` will create a reversible obfuscated user password to be used in configuration files. See obfuscating User Passwords for more details.

6.9.2.1 Interactive Mode

The control command available to obfuscate user passwords can be seen in following table:

Table 6-41 Interactive Mode Obfuscate

Command	Description
`obfuscate <plain text pwd>`	Obfuscates the user password.

Example

```
rdbthincontrol> obfuscate mypassword
obfuscation : ##016BA4158E5884C8D6EAFE71697D4DC9483417DA0BA1
```

6.9.2.2 Command Mode

The command mode commands available to obfuscate user passwords can be seen in following table:

Table 6-42 Command Mode Obfuscate

Command	Description
`-obfuscate <plain text pwd>`	Obfuscates the user password.

Example

```
$ thincontrol -obfuscate "mypassword"
obfuscation : ##0145A4158E5884C8D6EAFE71697D4DC9483417DA0BA1
```

Note:

If double quotation characters are not used to surround the plain text password DCL may, depending on your environment, force the value to all lower case or all upper case which may differ from the original.

Contents

Chapter 7
Oracle SQL/Services and Oracle JDBC for Rdb Servers

The Oracle SQL/Services management command line may be used to start and stop servers using the new dispatcher protocol called JDBC available in Oracle SQL/Services V7.1.6 and later.

Currently the Oracle SQL/Services interface to Oracle JDBC for Rdb Servers is minimal and may only be used to start and stop a JDBC dispatcher which in turn will start or stop the associated Oracle JDBC for Rdb server.

Starting an Oracle JDBC for Rdb server using Oracle SQL/Services involves the following steps:

1. Create an SQL/Services Dispatcher with the protocol JDBC.
   See Creating an Oracle SQL/Services JDBC Dispatcher.
2. Associate the JDBC Dispatcher with an Oracle JDBC for Rdb server.
   See Associating an Oracle SQL/Services JDBC Dispatcher to a Server.
3. Start the JDBC dispatcher.
   See Starting a JDBC Dispatcher.

For the dispatcher to start a server, the dispatcher must determine the name and type of the server, as well as the command procedures and configuration files to use during startup.

The following sections show how these determinations are carried out.
A1.3 Sample Setup, Starting an Oracle JDBC for Rdb Thin server from Oracle SQL/Services. provides a working example on creating a JDBC dispatcher and its server associations.

Contents

7.1 JDBC Dispatcher

A new SQL/Services dispatcher protocol of JDBC was introduced in release 7.1.6 of Oracle SQL/Services. This dispatcher type allows you to create JDBC dispatchers that may be associated with Oracle JDBC for Rdb servers.

7.1.1 Creating an Oracle SQL/Services JDBC Dispatcher

To be able to start and stop Oracle JDBC for Rdb servers using Oracle SQL/Services, a dispatcher with protocol JDBC must be defined using the Oracle SQL/Services management console.

You must provide the new dispatcher with a unique name and network port. It is important to ensure that the use of the PORT_ID is unique as the port provided will be used by the associated Oracle JDBC for Rdb server and only one server at a time may listen on a single TCPIP port.

Format

```
CREATE DISPATCHER <dispatcher name> NETWORK_PORT TCPIP PORT_ID <port> PROTOCOL JDBC;
```

Where:

- <dispatcher name> is a unique name for this dispatcher instance.
- <port> is the port number the associated server will listen on.

Example

```
$ MCR SQLSRV_MANAGE74
SQLSRV> CONNECT SERVER;
SQLSRV> CREATE DISPATCHER JDBC_DISP NETWORK_PORT TCPIP PORT_ID 1880 PROTOCOL JDBC;
SQLSRV> SHOW DISPATCHER;
Dispatcher JDBC_DISP
    State:                    UNKNOWN
    Autostart:                on
    Max connects:             100 clients
    Idle User Timeout:        <none>
    Max client buffer size:   5000 bytes
    Network Ports:                      (State)  (Protocol)
      TCP/IP  port     1880             Unknown   JDBC clients
    Log path:             SYS$MANAGER:
    Dump path:            SYS$MANAGER:
```

Caution:

The Oracle SQL/Services Management GUI (unsupported software) does not recognize dispatchers of the type JDBC. This means that you will no longer be able to use the GUI once a JDBC dispatcher has been defined.

7.1.2 Associating an Oracle SQL/Services JDBC Dispatcher to a Server

Each Oracle SQL/Services JDBC dispatcher must be associated with an Oracle JDBC for Rdb server.

Since release 7.3.5

The simplest way to associate a JDBC server to a dispatcher, is to provide a JDBC server entry in the default JDBC configuration file named the same as the SQL/Services dispatcher.
An alternate mechanism is to tie the SQL/Services dispatcher to the JDBC server using the dispatcher s PORT_ID. See Legacy Behaviour Using the PORT_ID for more details.

7.1.2.1 Associating Server Characteristics using Dispatcher Name

Since release 7.3.5

The name chosen for the JDBC Dispatcher may be used as the name of the JDBC server that is started when the dispatcher is started.

By default, on dispatcher startup, SQL/Services will look in the default JDBC configuration file, RDB$JDBC_COM:RDBJDBCCFG.XML for a JDBC server named the same as the dispatcher s name. If found this will be the server that the dispatcher will startup. For example, given the dispatcher s name JDBC_DISP, you might create the following RDB$JDBC_COM:RDBJDBCCFG.XML file:

```
<?xml version = '1.0'?>
<!-- Configuration file for Rdb Thin JDBC Drivers/Servers -->
```

```
<config>
<!-- SERVERS -->
<servers>
<!-- DEFAULT server characteristics-->
  <server
    name="DEFAULT"
    type="RdbThinSrv"
    url="//localhost:1701/"
    maxClients="-1"
    srv.bindTimeout="0"
    srv.idleTimeout="0"
    srv.mcBasePort="5520"
    srv.mcGroupIP="239.192.1.10"
    autoStart="false"
    controlUser="jdbc_user"
    controlPass="0x811B15F866179583EB3C96751585B843"
    cfg="rdb$jdbc_com:rdbjdbccfg.xml"
    srv.startup="rdb$jdbc_home:rdbjdbc_startsrv.com"
    srv.onStartCmd="@rdb$jdbc_com:rdbjdbc_sqs_onstartup.com"
  />

  <! JDBC Dispatcher associated Server -->
  <server
    name="JDBC_DISP"
    type="RdbThinSrv"
    url="//localhost:1880/" >
  </server>
</servers>
</config>
```

If you choose not to use the default configuration file, you may create a server specific configuration file `RDB$JDBC_COM:<dispatcher name>_CFG.XML` , for example, `RDB$JDBC_COM:JDBC_DISP_CFG.XML`.

Note:
To ensure that SQL/Services and JDBC correctly associates server usage, you should make sure that the URL and PORT characteristics of the server in the configuration file match up with the PORT_ID chosen for the JDBC dispatcher.

If your system relies on legacy behaviour and expects that the associated JDBC server be named *SQS<port>*, you may disable the use of the dispatcher s name association by deassigning the `RDB$JDBC_SQS_USE_DISPNAME` system logical name that will be setup if you executed the JDBC startup command procedure suggested during JDBC installation.

By default, this is a system/exec based logical name so to deassign the logical name use:

```
$ deassign/system/exec RDB$JDBC_SQS_USE_DISPNAME
```

7.1.3 Legacy Behaviour Using the PORT_ID

The `PORT_ID` specifies the TCPIP port that can be used by the Oracle JDBC for Rdb server and is used by the dispatcher start up procedures to determine information about the associated server.

In addition to which port the server will listen on, the `PORT_ID` may be used by the dispatcher to determine:

- What type of Oracle JDBC for Rdb server to start.

- The name that will be given to this server.
- What configuration file to use for this server.
- Any DCL command to run during the server startup procedure.

The overloading of the use of the PORT_ID by the JDBC dispatcher is necessary as the amount of information stored for a JDBC dispatcher is minimal keeping it in line with the information stored for other SQL/Services Dispatcher types.

In the process of determining the server attributes the dispatcher may try to translate the following logical names:

- RDB$JDBC_SQSNAM_<port>
- RDB$JDBC_SQSCFG_<port>
- RDB$JDBC_SQSCMD_<port>
- RDB$JDBC_SQSTYPE_<port>

In the above logical names, the <port> will be substituted by the PORT_ID of the JDBC dispatcher prior to logical name translation.
If no such logical names exist, the dispatcher will then use alternate methods to provide the server with a name and will to try to locate a suitable command procedure and configuration file. The following sections detail how these determinations are carried out.

When determining the server information required to correctly start the associated Oracle JDBC for Rdb server, the dispatcher will carry out the following steps in the order specified:

1. First the dispatcher will create a name for the server.
2. Any DCL command required to be executed during server start up is then determined.
3. The file specification of the configuration file to provide to the server is then determined.
4. The server type for the server is then determined.

If you have not set up specific criteria for the JDBC server as detailed in the following sections, these defaults will be used:

1. The server s name will be SQS<port>.
2. The standard JDBC server startup command procedure, RDB$JDBC_HOME:RDBJDBC_STARTSRV.COM will be executed.
3. The configuration file RDB$JDBC_COM:RDBJDBCCFG.XML will be used.
4. The server type will be designated as POOL.
5. If Java is not set up system-wide, then Java Release 8.0 will be selected.
6. If Rdb has not been set up for the server process, then Rdb V7.3 will be selected.
7. If running on OpenVMS IA-64 and Java 8.0 or later is available, then 64 Bit JDBC servers will be used.

7.1.3.1 Determining the Server s Name

A server name is required as it may be used by the server start up procedure to locate properties from its configuration file. The name used will determine various characteristics of the started server.

In addition, the server s name will be used as the OpenVMS process name and will determine the naming of any associated executors if the server is a Multi-process server.

The server s name is also used in creating log and temporary files during the running of the server.

The PORT_ID is used to determine the name of the Oracle JDBC for Rdb server using the following precedence:

1. If the logical name RDB$JDBC_SQSNAM_<port> exists, then it is translated to provide the server s name.
2. If the logical name does not exist, the server s name will be SQS<port>.

Example 1

Logical name not defined:

```
$ show log RDB$JDBC_SQSNAM_1888
%SHOW-S-NOTRAN, no translation for logical name RDB$JDBC_SQSNAM_1888
$ MCR SQLSRV_MANAGE74
SQLSRV> CONNECT SERVER;
SQLSRV> CREATE DISPATCHER JDBC_DISP NETWORK_PORT TCPIP PORT_ID 1888 PROTOCOL JDBC;
```

This will create a server named ***SQS1888.***

Example 2

Logical name defined:

```
$ DEFINE/SYSTEM RDB$JDBC_SQSNAM_1888 MY_POOL_SRV
$ MCR SQLSRV_MANAGE74
SQLSRV> CONNECT SERVER;
SQLSRV> CREATE DISPATCHER JDBC_DISP NETWORK_PORT TCPIP PORT_ID 1888 PROTOCOL JDBC;
```

This will create a server named ***MY_POOL_SRV.***

7.1.3.2 Determining Extra DCL Commands for use During Start-up

During the invocation of a JDBC server, the following DCL command procedure is executed:

```
RDB$JDBC_HOME:RDBJDBC_STARTSRV.COM
```

This is the standard startup command procedure used by Oracle JDBC for Rdb and was created for you during the installation of the Oracle JDBC for Rdb product.

This command procedure will set up some environmental elements and then execute a Java command to start the server. A discrete dispatcher process will be set up by the SQL/Services START DISPATCHER command and the Java command will be run under this process context.

The RDBJDBC_STARTSRV command procedure will try to locate and execute any specific setup command procedures you may have designated for its use. This is done prior to the procedure executing the Java command that will ultimately start the server instance.

The PORT_ID is used to determine the name of an Open VMS DCL command procedure that may be invoked containing your system and environmental setup procedures. The file specification of the command procedure is determined using the following precedence:

1. If the logical name RDB$JDBC_SQSCMD_<port> exists, then it is translated to provide the command procedure file specification.

2. If the logical name does not exist, the dispatcher will try to locate and execute the file `rdb$jdbc_com:rdbjdbc_sqs_onStartup.com`.

3. If this file does not exist, the dispatcher will try to locate and execute the file `rdb$jdbc_home:rdbjdbc_sqs_onStartup.com`.

Example 1

Logical name not defined and file rdb$jdbc_com:rdbjdbc_sqs_onStartup.com **does** exist:

```
$ show log RDB$JDBC_SQSCMD_1888
%SHOW-S-NOTRAN, no translation for logical name RDB$JDBC_SQSCMD_1888
$ MCR SQLSRV_MANAGE74
SQLSRV> CONNECT SERVER;
SQLSRV> CREATE DISPATCHER JDBC_DISP NETWORK_PORT TCPIP PORT_ID 1888 PROTOCOL JDBC;
```

The file RDB$JDBC_COM:RDBJDBC_SQS_ONSTARTUP.COM will be executed.

Example 2

Logical name defined:

```
$ DEFINE/SYSTEM RDB$JDBC_SQSCMD_1888 RDB$JDBC_COM:MY_SRV1888_ONSTART.COM
$ MCR SQLSRV_MANAGE74
SQLSRV> CONNECT SERVER;
SQLSRV> CREATE DISPATCHER JDBC_DISP NETWORK_PORT TCPIP PORT_ID 1888 PROTOCOL JDBC;
```

The file RDB$JDBC_COM:MY_SRV1888_ONSTART.COM will be executed.

7.1.3.3 Determining the Server Configuration File

The PORT_ID is also used to determine the configuration file to use on server startup. This file can be a CFG or an XML-formatted configuration file and is used to provide information to the server about what characteristics it should use when running. See Configuration Files for more details on the use of configuration files.

You may choose to provide a separate configuration file for the server associated with each JDBC dispatcher, or you may choose to use a single XML-formatted configuration file containing the server attributes for all your servers.

The appropriate configuration file is determined by the dispatcher by trying to translate the logical name `RDB$JDBC_SQSCFG_<port>` where `PORT_ID` is substituted for `<port>` prior to logical name translation. If the logical name is not there, the dispatcher will try to use a configuration file from the JDBC system directories.

The following is the precedence for this file search:

1. The file pointed to by the `RDB$JDBC_SQSCFG_<port>` logical name if it exists.

2. `RDB$JDBC_COM:<server name>_CFG.XML` where the server s name as determined in previous steps is substituted for `<server_name>`.

3. `RDB$JDBC_COM:SQLSRV_JDBC_SERVER_CFG.XML`.

4. `RDB$JDBC_COM:RDBJDBCCFG.XML`.

Example 1

Logical name not defined and file RDB$JDBC_COM:SQLSRV_JDBC_SERVER_CFG.XML **does** exist:

```
$ show log RDB$JDBC_SQSCFG_1888
%SHOW-S-NOTRAN, no translation for logical name RDB$JDBC_SQSCFG_1888
$ MCR SQLSRV_MANAGE74
SQLSRV> CONNECT SERVER;
SQLSRV> CREATE DISPATCHER JDBC_DISP NETWORK_PORT TCPIP PORT_ID 1888 PROTOCOL JDBC;
```

The file RDB$JDBC_COM:SQLSRV_JDBC_SERVER_CFG.XML will be used.

Example 2

Logical name defined:

```
$ DEFINE/SYSTEM RDB$JDBC_SQSCFG_1888 RDB$JDBC_COM:MY_SRV1888_CFG.XML
$ MCR SQLSRV_MANAGE74
SQLSRV> CONNECT SERVER;
SQLSRV> CREATE DISPATCHER JDBC_DISP NETWORK_PORT TCPIP PORT_ID 1888 PROTOCOL JDBC;
```

The file RDB$JDBC_COM:MY_SRV1888_CFG.XML will be used.

7.1.3.4 Determining Server Type

During the start of the server associated with the Oracle SQL/Services JDBC dispatcher, the type of the server to startup also needs to be determined.

The server type will be used by the dispatcher to determine the appropriate JDBC JAR file to use when invoking the server. The server type will also be used to determine other server attributes that have to be set for a successful instantiation of a server process.

The dispatcher will use the PORT_ID to try to identify the appropriate JDBC server type to start.
There are four types of Oracle JDBC for Rdb servers recognized by Oracle SQL/Services:

- POOL - a Pool server i.e., type="RdbThinSrvPool".
- MP - a Multi-process server i.e., type="RdbThinSrvMP".
- STD - a standard thin server i.e., type="RdbThinSrv".
- MAN - a Manager server i.e., type="RdbManSrv".

When the dispatcher determines the server type, the following steps are used:

1. If the logical name RDB$JDBC_SQSTYPE_<port> exists, it is translated to provide the server type. The translated logical name must be one of the valid server types as shown above.
2. If the logical name does not exist, the server type will be POOL.

Note:

As the dispatcher cannot currently use the server s name to determine the server type, it is important that this logical name be correctly setup if the type of the server to start is not a POOL server i.e., type="RdbThinSrvPool". If this is not correctly set the wrong JDBC JAR file may be used and the server may fail to start correctly. The log files associated with the

server, usually written to the directory RDB$JDBC_LOGS will show the start-up failure and the reason for the failure.

Example 1

Logical name not defined:

```
$ show log RDB$JDBC_SQSTYPE_1888
%SHOW-S-NOTRAN, no translation for logical name RDB$JDBC_SQSTYPE_1888
$ MCR SQLSRV_MANAGE74
SQLSRV> CONNECT SERVER;
SQLSRV> CREATE DISPATCHER JDBC_DISP NETWORK_PORT TCPIP PORT_ID 1888 PROTOCOL JDBC;
```

This will create a server with type ***RdbThinSrvPool.***

Example 2

Logical name defined:

```
$ DEFINE/SYSTEM RDB$JDBC_SQSTYPE_1888 MP
$ MCR SQLSRV_MANAGE74
SQLSRV> CONNECT SERVER;
SQLSRV> CREATE DISPATCHER JDBC_DISP NETWORK_PORT TCPIP PORT_ID 1888 PROTOCOL JDBC;
```

This will create a server with type ***RdbThinSrvMP.***

Example 3

Logical name defined:

```
$ DEFINE/SYSTEM RDB$JDBC_SQSTYPE_2070 MAN
$ MCR SQLSRV_MANAGE74
SQLSRV> CONNECT SERVER;
SQLSRV> CREATE DISPATCHER JDBC_DISP NETWORK_PORT TCPIP PORT_ID 2070 PROTOCOL JDBC;
```

This will create a server with type ***RdbManSrv.***

Contents

7.1.4 Starting a JDBC Dispatcher

Once you have defined a JDBC dispatcher, it can be started like any other Oracle SQL/Services dispatcher:

Example

```
SQLSRV> start dispatcher jdbc_disp;
SQLSRV> show disp jdbc_disp;
Dispatcher JDBC_DISP
State: STARTING
Autostart: on
Max connects: 100 clients
Idle User Timeout: <none>
Max client buffer size: 5000 bytes
Network Ports: (State) (Protocol)
TCP/IP port 1880 Inactive JDBC clients
Log path: SYS$MANAGER:
Dump path: SYS$MANAGER:
```

```
SQLSRV> show disp jdbc_disp;
Dispatcher JDBC_DISP
State: RUNNING
Autostart: on
Max connects: 100 clients
Idle User Timeout: <none>
Max client buffer size: 5000 bytes
Network Ports: (State) (Protocol)
TCP/IP port 1880 Inactive JDBC clients
Log path: SYS$MANAGER:
Dump path: SYS$MANAGER:
  Log File: SYS$SYSROOT:[SYSMGR]SQS_MYNODE_JDBC_DISP06O71.LOG;
Dump File: SYS$SYSROOT:[SYSMGR]SQS_MYNODE_JDBC_DISP06O.DMP;
```

The Oracle SQL/Services monitor will attempt to start the server associated with this dispatcher.
If the dispatcher fails to start, the SQL/Services server should contain an error entry for this event in the server s log file:

```
SQLSRV> show server;
Server Version: 7.3
Server Platform: HP OpenVMS IA64
.
.
.

Log File: SYS$SYSROOT:[SYSMGR]SQS_MYNODE_SQLSRV_MON_0073.LOG;
Dump File: SYS$SYSROOT:[SYSMGR]SQS_MYNODE_SQLSRV_73.DMP;


$ type SYS$SYSROOT:[SYSMGR]SQS_MYNODE_SQLSRV_MON_0073.LOG;
.
.
.
---EVENT BEG: EVENT_LOG ------------------------ Tue Jan 24 16:36:18.880 2017---
%SQLSRV-I-EVENT_LOG, event logged at line 8704 in file CMD.C;1
%SQLSRV-E-FLOBDISPUNAVAIL, Dispatcher process JDBC_DISP00I failed
%SQLSRV-I-LOGFILE, Log file : SYS$SYSROOT:[SYSMGR]SQS_MYNODE_JDBC_DISP00I73.LOG;
---EVENT END: EVENT_LOG ------------------------ Tue Jan 24 16:36:18.880 2017---
```

The dispatcher events are written to the SYS$MANAGER directory in a log file named:

```
SYS$MANAGER:SQS_<nodename>_<dispnam><nnnnn>.LOG
```

The <nodename> depends on the node the dispatcher is started up on.
The <dispname> depends on the name of the dispatcher.
The <nnnnn> is the unique id given to this dispatcher instance by Oracle SQL/Services.

For example:

```
SQS_MYNODE_JDBC_DISP00I73.LOG
```

This log can be useful in determining why a dispatcher did not start up properly. For example, if appropriate logical names have not been setup as specified in the installation of Oracle JDBC Drivers for Rdb then a message like the following may be found at the end of the log file:

```
.
.
.
$ @rdb$jdbc_home:rdbjdbc_startsrv SQS1880 "SQS"
```

```
%DCL-E-OPENIN, error opening RDB$JDBC_HOME:[SYSMGR]RDBJDBC_STARTSRV.COM; as input
-RMS-F-DEV, error in device name or inappropriate device type for operation
SYSTEM job terminated at 12-JAN-2025 16:36:18.880

Accounting information:
Buffered I/O count: 37 Peak working set size: 2272
Direct I/O count: 14 Peak virtual size: 173072
Page faults: 192 Mounted volumes: 0
Charged CPU time: 0 00:00:00.04 Elapsed time: 0 00:00:00.21
```

If it is not clear from the dispatcher log what may have caused the error, you can define the logical name SQLSRV_JDBC_SERVER_STARTUP_VERIFY:

```
$ define/sys SQLSRV_JDBC_SERVER_STARTUP_VERIFY "true"
```

This will enable DCL verify for the SQL/Services JDBC startup command procedure.

Once this logical name is defined try to start the dispatcher again. The dispatcher log should now show the DCL verify for the startup command procedure.

Check this trace information for the names of the configuration file and command procedures and other setup criteria used during the startup attempt.

7.1.5 Stopping a JDBC Dispatcher

The `STOP DISPATCHER` statement may be used to stop a running JDBC dispatcher.

Example

```
SQLSRV> STOP DISPATCHER JDBC_DISP
```

This will also stop the associated Oracle JDBC for Rdb server.

If you have associated the dispatcher with a Pool server, and the pooled servers have autoStart enabled, then these pooled servers will also be shut down at this time.

See the Oracle SQL/Services documentation for more information on the Oracle SQL/Services management console.

Contents

7.2 Command Procedures used by Oracle SQL/Services

When a JDBC dispatcher is started, Oracle SQL/Services will use the OpenVMS command procedure:

```
SYS$MANAGER:SQLSRV_JDBC_SERVER_STARTUP<version>.COM
```

Since release 7.3.5

and

```
RDB$JDBC_HOME:SQLSRV_JDBC_SERVER_STARTUP_COMMON.COM
```

to start the server associated with a JDBC dispatcher.

As multiple versions of SQL/Services may be present on your system, the Oracle JDBC for Rdb installation provides multiple versions of the SQLSRV_JDBC_SERVER_STARTUP command procedure. The <version> of the command procedure determines the version of SQL/Services it is associated with, thus:

```
SYS$MANAGER:SQLSRV_JDBC_SERVER_STARTUP73.COM
```

will be the command procedure used by version 7.3 SQL/Services during the JDBC dispatcher startup.

These command procedures in turn execute the following command procedure:

```
RDB$JDBC_HOME:RDBJDBC_STARTSRV.COM
```

This enables you to have multiple versions of the Oracle JDBC for Rdb on your systems, each with potentially different startup requirements specified in the RDBJDBC_STARTUP.COM. The logical name RDB$JDBC_HOME in your SQL/Services environment may be used to select the specific version of Oracle JDBC for Rdb it will use.

Note:

As the releases of Oracle JDBC for Rdb are independent of the releases of Oracle SQL/Services, the currently installed version of Oracle JDBC for Rdb may not have installed an appropriate SQL/Services JDBC Server command procedure for all SQL/Services versions installed on your system.

If this is the case, JDBC dispatchers will not start up correctly for the installed SQL/Services version.

To fix this problem you can simply copy an existing SQL/Services JDBC Server command procedure within SYS$MANAGER: and alter the version number of its filename to reflect the SQL/Services version you are using.

7.2.1 JDBC Dispatcher Setup Procedure

An additional OpenVMS command procedure can be defined to set up environmental characteristics required for your system. This command procedure is located for use with this server using the following precedence:

1. The file pointed to by the logical name RDB$JDBC_SQSCMD_<port>, if defined.
2. RDB$JDBC_COM:RDBJDBC_SQS_ONSTARTUP.COM
3. RDB$JDBC_HOME:RDBJDBC_SQS_ONSTARTUP.COM

If a command procedure is found on your system using this search list, this command procedure will be executed just prior to the server being invoked. You may use this command procedure to setup environmental conditions for the server execution, for example:

```
$ @sys$share:rdb$setver 74
$ @sys$common:[java$60.com]JAVA$60_SETUP.COM
```

7.3 Using Pool Servers

Each JDBC dispatcher defined is related only to a single server. Use a Pool server if you require more than one server to be started for a single dispatcher.

By defining a pool of servers that the Pool server can use and enabling `autoStart` on each of these servers, a whole pool of servers can be started by starting a single dispatcher. See Pool Server Operation for more information on Pool servers.

The following example shows how you can define a dispatcher to start up a Pool server that will automatically start up three standard thin servers as part of its pool.

Since release 7.3.5

Note:

This example uses the default server naming, server type of `POOL` and a standard `SQS_ONSTARTUP` command procedure. No *`RDB$JDBC_SQS*`* logical names need be set up.

---

1. Define an Oracle SQL/Services dispatcher.

```
$ MCR SQLSRV_MANAGE74
SQLSRV> CONNECT SERVER;
SQLSRV> CREATE DISPATCHER POOL_DISP NETWORK_PORT TCPIP PORT_ID 1880 PROTOCOL JDBC;
```

2. Create a configuration file for this server in RDB$JDBC_COM:POOL_DISP_CFG.XML.

```
<?xml version = '1.0'?>
<!-- Configuration file for Rdb Thin JDBC Drivers/Servers -->
<config>
<!-- SERVERS -->
<servers>
<!-- DEFAULT server characteristics-->
  <server
    name="DEFAULT"
    type="RdbThinSrv"
    url="//localhost:1701/"
    maxClients="-1"
    srv.bindTimeout="0"
    srv.idleTimeout="0"
    srv.mcBasePort="5520"
    srv.mcGroupIP="239.192.1.10"
    autoStart="false"
    controlUser="jdbc_user"
    controlPass="0x811B15F866179583EB3C96751585B843"
    cfg="rdb$jdbc_com:sqlsrv_jdbc_server_cfg.xml"
    srv.startup="rdb$jdbc_home:rdbjdbc_startsrv.com"
    srv.onStartCmd="@rdb$jdbc_com:rdbjdbc_sqs_onstartup.com"
  />
  <!-- now the servers that will be started by pool server -->
  <server
    name="SQSrjs1"
    type="RdbThinSrv"
    url="//localhost:1891/"
    autoStart="true"
    maxClients="10"
```

```
  />
  <server
    name="SQSrjs2"
    type="RdbThinSrv"
    url="//localhost:1892/"
    autoStart="true"
    maxClients="10"
  />
  <server
    name="SQSrjs3"
    type="RdbThinSrv"
    url="//localhost:1893/"
    autoStart="true"
    maxClients="10"
  />

  <!-- Pool Server -->
  <server
    name="POOL_DISP"
    type="RdbThinSrvPool"
    url="//localhost:1880/" >
    <pooledServer name="SQSrjs1"/>
    <pooledServer name="SQSrjs2"/>
    <pooledServer name="SQSrjs3"/>
  </server>
</servers>
</config>
```

3. **Create an onStartup command procedure that sets up the appropriate Rdb and Java versions for your system:**

For example, RDB$JDBC_COM:RDBJDBC_SQS_ONSTARTUP.COM may contain:

```
$ @sys$share:rdb$setver 74
$ @sys$common:[java$60.com]JAVA$60_SETUP.COM
```

4. **Start the dispatcher.**

```
SQLSRV> start dispatcher pool_disp;
```

Remarks

In this example the command procedure pointed to by default srv.onStartCmd in the XML configuration file happens to be the same as the one created as the `SQS_ONSTARTUP` command procedure. These do not have to be the same command procedure.

The Oracle SQL/Services JDBC dispatcher `SQS_ONSTARTUP` command procedure is used during the startup of the associated Pool server. The pooled servers that the Pool server starts use the command procedure pointed to by the srv.onStartCmd switch.

Contents

Chapter 8
Performance

The overall performance of application access to an underlying relational database depends on several factors including:

- Database performance including:
  - Speed of query compilation.
  - Efficiency of query optimization.
  - Efficiency of record lookup using indexes.
  - Efficiency of record retrieval.
  - Performance of the underlying operating system and hardware.

- JDBC performance including:
  - Efficiency of object creation and disposal.
  - Efficiency of internal message protocols.
  - Degree of buffering of data and metadata.
  - Efficiency of the underlying subsystem used by the drivers and servers including the Java VM, operating system, and hardware.

- Network performance including:
  - The number of client /server message roundtrips.
  - The network distance between the client and server machines, the more hops taken between the two nodes, the longer the round-trip time.
  - Size of network buffers and flush times.
  - Overall performance of the network.

- Application performance including:
  - Effective utilization of database and operating system resources.
  - SQL statement re-use utilizing PreparedStatements.
  - Use of data buffering by utilizing appropriate Fetchsize.

Details on performance considerations for the underlying Rdb database system may be found in your Oracle Rdb documentation.

Details about performance and your network may be found in the appropriate documentation provided by your hardware and operating system vendors.

Details about Java VM and operating system performance may be found in documentation provided by your operating system vendors.

Details about performance consideration related to the use of Oracle JDBC for Rdb drivers and servers may be found in the following sub-sections and elsewhere in this document and the Oracle JDBC for Rdb Release Notes.

Contents

8.1 Performance Features

There are several features available in Oracle JDBC for Rdb to help improve the overall performance of your applications using the JDBC drivers and the efficiency and performance of the JDBC servers:

- Fetchsize may be used to improve the overall performance of record retrieval by reducing the number of network roundtrips used to retrieve records.
- Lockwait and Maxtries may help overall concurrency and performance when using thin servers.
- Inactivity timeouts may be used to limit the number of resources tied-up by unused servers and inactive connections.
- SQL statement caching may be used to reduce the compilation and setup time of frequently used queries.
- Results caching may be used to improve record retrieval times by caching frequently used query results.

8.1.1 FetchSize

The `SetFetchSize` methods in `Statement` and `ResultSet` allow you to set the record fetch size for server record retrieval. The `FetchSize` gives a hint to the server as to how many records to batch up and send over the network at one time.

Network I/O is very expensive, so the more data you can send in a single I/O the better the performance. If you do not explicitly change the default `FetchSize` by using the `FetchSize` option, the default is 100.

8.1.2 Lockwait and Maxtries

The standard thin server is a multi-threaded server that allows concurrent access to Oracle Rdb by many client processes. Within a single OpenVMS process, Oracle Rdb is single-threaded, thus the thin server must synchronize client database activity.

Because database actions must be serialized, any action that might take a prolonged length of time may seriously impact the overall throughput of the server.

By default, the server will wait indefinitely for a lock, however, to try to minimize the impact of one client thread on another you may specify the period the server should wait for a lock.

If this wait is not indefinite, any thread will wait for the specified amount of time trying to get a lock. If the lock is not granted, control is returned to the server. By default, the server will then try to get a lock ten (10) times, waiting for the specified amount of time each time, before raising a locking exception.

Specifying a short wait duration, for example one (1) second, may help reduce the impact that one thread may have on another sibling thread.

The `lockwait` connection option or server option allows control of the duration of the wait for a lock. The minimum actual wait period is one (1) second, which is the minimum lock wait time supported by Rdb transactions.

A `lockwait` of zero (0) is the same as starting up a transaction with `NOWAIT`.
A `lockwait` of minus one (-1) is the same as starting up a transaction with `WAIT` without specifying a value, which causes the server to wait indefinitely.

The `maxtries` connection option or server option allows you to specify the maximum number of times the server will try to get a lock before giving up. The default `maxtries` value is 10.

The higher the value you assign to the lockwait switch, the more likely that a locked object may slow down all clients, so it is preferable to keep the lockwait at a minimum but increase the number of lock attempts appropriately.

8.1.2.1 Lockwait Precedence

As well as being able to specify the lockwait either at the server level or at the connection level as shown above, Oracle Rdb allows you to specify a maximum lock wait for the process by using the RDM$BIND_LOCK_TIMEOUT_INTERVAL logical name. In addition, a database-wide lock timeout value may be established using the LOCK TIMEOUT INTERVAL clause of the SQL CREATE DATABASE and SQL ALTER DATABASE statements.

The following describes the order of precedence observed when lockwait has been specified in more than one way:

1. A connection lockwait value as specified explicitly on the connection string will take precedence over the server lockwait value but only for that one connection.
2. An explicit lockwait set on either the server or connection will take precedence over the value set by the RDM$BIND_LOCK_TIMEOUT_INTERVAL logical name.
3. The database-wide lock timeout interval, if specified, will place an upper limit on the interval specified by the RDM$BIND_LOCK_TIMEOUT_INTERVAL logical name or the lockwait on both the server and connection.

Example 1

```
RDM$BIND_LOCK_TIMEOUT_INTERVAL = 10
server LOCKWAIT = 20
connection LOCKWAIT = 30
LOCK TIMEOUT INTERVAL not specified
```

Results in a lockwait of 30.

Example 2

```
RDM$BIND_LOCK_TIMEOUT_INTERVAL = 10
server LOCKWAIT = 20
connection LOCKWAIT = 30
LOCK TIMEOUT INTERVAL = 25
```

Results in a lockwait of 25.

Example 3

```
RDM$BIND_LOCK_TIMEOUT_INTERVAL = 10
server LOCKWAIT = 20
connection LOCKWAIT = 30
LOCK TIMEOUT INTERVAL = 35
```

Results in a lockwait of 30.

Example 4

```
RDM$BIND_LOCK_TIMEOUT_INTERVAL = 10
server LOCKWAIT = 20
connection LOCKWAIT not specified
```

```
LOCK TIMEOUT INTERVAL not specified
```

Results in a lockwait of 20.

Example 5

```
RDM$BIND_LOCK_TIMEOUT_INTERVAL = 10
server LOCKWAIT not specified
connection LOCKWAIT not specified
LOCK TIMEOUT INTERVAL = 25
```

Results in a lockwait of 10.

See the Oracle Rdb Documentation for more information on the use of the `RDM$BIND_LOCK_TIMEOUT_INTERVAL` logical name and the `LOCK TIMEOUT INTERVAL` clause.

8.1.3 Inactivity Timeouts

The amount of time either a client connection or a server may remain inactive before being forcibly terminated may be set using server and connection switches.

8.1.3.1 Client Connection Timeout

The `cli.idleTimeout` switch may be used to specify the amount of time in milliseconds that a connection may remain inactive before being closed. The default value of 0 specifies that the time is indefinite, i.e., the connection will not timeout.

You may specify the client idle timeout as a server configuration option either in the server definition within an XML-formatted configuration file or as a command-line switch when starting a server.

Example

```
$ java -jar rdbthinsrv.jar port 1701 cli.idleTimeout 3600000
```

Specifies that any client connection may remain idle for 1 hour before being terminated.

Or you can place this option in the XML-formatted configuration file:

```
<server
name="srv2forRdb"
type="RdbThinSrv"
url="//localhost:1708/"
cli.idleTimeout="3600000"
/>
```

When a client is forcibly terminated by this timeout the following message will be logged in the server log:

```
oracle.rdb.jdbc.common.RdbException:
Client terminated due to inactivity
```

When specified as a server switch, the timeout will apply to all clients connected using that server.

You may also specify the client timeout as a qualifier on the connection string on the client-side application.

```
Connection conn = DriverManager.getConnection(
"jdbc:rdbthin://bravo:1701/myDir:personnel"+
"@cli.idleTimeout=3600000", user, pass);
```

When specified this way the timeout will only apply to this one connection.

If a non-zero `cli.idleTimeout` is specified in both the server configuration and as a connection qualifier, the lesser of the two values will be used for that connection.

Inactivity is determined by the lack of activity on the socket the server is listening to the client on, if no request is sent from the client for the specified amount of time, a timeout is deemed to have occurred.

If a client inactivity timeout occurs on a connection that is using a Multi-process server executor, that executor will be terminated. Even though the connection will be correctly closed after the timeout event, as it is unknown why there was no activity seen on the connection, the executor sub-process is deemed "unsafe" and consequently is terminated.

8.1.3.2 Server Inactivity Timeout

You can specify the amount of time that a server may remain idle before being closed due to inactivity.

The `srv.idleTimeout` switch may be used to specify the amount of time in milliseconds that a server may remain inactive before being closed. The default value of 0 specifies that the time is indefinite, i.e., the server will not timeout.

You may specify the server idle timeout as a server configuration option either in the server definition within an XML-formatted configuration file or as a command-line switch when starting a server.

Example

```
$ java -jar rdbthinsrv.jar port 1701 srv.idleTimeout 3600000
```

Specifies that the server may remain idle for 1 hour before being terminated.

Or in the XML-formatted configuration file:

```
<server
name="srv2forRdb"
type="RdbThinSrv"
url="//localhost:1708/"
srv.idleTimeout="3600000"
/>
```

When a server is terminated by this timeout the following message will be logged in the server log:

```
Server terminated due to inactivity
2025-01-08 12:28:03.578 : Forced disconnect by Server terminated due to inactivity @ LOCAL
```

A server inactivity timeout will occur if, after the length of time specified, no new client connection is made to that server, and there are no currently connected clients.

In other words, the timeout period is started after each new connection. If the timeout expires and there are current connections still using the server, the timeout period will be reset to start again.

8.1.4 SQL Statement Cache

When using the thin driver, performance may be improved by enabling SQL statement caching.

Whenever the thin driver needs to prepare a SQL statement, the statement must be sent over the network to the server for Oracle Rdb to prepare the statement and to send back a list of columns or parameters that the statement references.

If the same SQL statement is prepared repeatedly during a single connection, without SQL statement caching the statement will be prepared and column information sent back each time. This can be time consuming because it requires network traffic, the preparation of the statement, and getting the column and parameter information. These steps can be a substantial part of the network I/O and performance cost of the queries.

To help reduce this cost, the thin driver allows you to cache SQL statements so that if the exact same SQL string is prepared more than once during a single connected session, the cost for retrieving column information is only incurred once.

SQL statement caching can be enabled by using the `sqlcache` switch when you request a connection either by placing the switch in the connection URL or using the information block that is passed in the connect request.

Example

Set the `sqlcache` property of the Properties passed to the `DriverManager.getConnection` method:

```
Properties info = new Properties();
info.put("user", user);
info.put("password", pw);
info.put("sqlcache", 100);
conn = DriverManager.getConnection (connStr, info);
```

Or append `@sqlcache` to the database specification part of the connect URL:

```
Connection conn = DriverManager.getConnection(
"jdbc:rdbthin://bravo:1701/"+
"myDir:personnel@sqlcache=100",user, pass);
```

In addition, a `SET SQLCACHE` statement can be executed:

```
Stmt.executeUpdate("set sqlcache 100");
```

Remarks

The value specified with the sqlcache switch tells the thin driver how many SQL statements it can hold concurrently in its cache. A value of 0 (the default) specifies that SQL statement caching be disabled.

Once the SQL statement cache is full for a given connection, the storing of a new statement will remove the least commonly used statement from the cache.

Because SQL statements may be held in cache even after the user has closed the containing java.sql.Statement, the query will still be registered as current by Oracle Rdb and may prevent actions such as `DROP TABLE` from being done. In addition, each concurrent statement that is held in cache may take up memory on both the server and client side of the connection.

You can clean out the connection SQL cache by issuing a `SET SQLCACHE` statement with value 0 and then issuing another `SET SQLCACHE` statement to reset the cache to the desired size.

Currently you cannot specify the removal of a specific SQL statement from cache.

Note:

SQL statement caching is a client-side action and is disabled by default. This feature is only applicable to the thin driver. Using the `sqlcache` property or using the `set sqlcache` statement will be silently ignored by the native driver.

8.1.4.1 Caching Statement Handles

In addition to saving the network cost of retrieving column information, enabling SQL statement handle caching may also improve application performance when used in conjunction with SQL statement caching.

Similar to the use of PreparedStatements, enabling statement handle caching allows the Thin driver to re-use compiled Rdb statements which may improve the overall performance of retrieving results as the statement does not have to be compiled again or the column information retrieved from the server.

SQL statement handle caching works for both Statements and PreparedStatements. If the exact SQL text is recognized as being prepared previously in the same connection context, and that Statement is no longer in use (i.e., the Statement or PreparedStatement has been closed) then, instead of sending down a request to the server to compile the query again, the driver will re-use the statement handle compiled by the previous request.

This is particularly effective where applications may be using connection pooling. As it cannot be guaranteed that the query that they wish to use is available within the connection context of the pooled connection allocated to the connection request, the same PreparedStatements may have to be issued repeatedly within the same actual Rdb connection context. This redundant query compilation may be costly in terms of network traffic.

If SQL statement handle caching is enabled, PreparedStatements may be effectively re-used across serial re-use of a pooled connection, thus saving expensive network IO required for query recompilation.
Statement handle caching can be enabled by using the `sqlcachePS` switch when you request a connection either by placing the switch in the connection URL or using the information block that is passed in the connect request.

Example

Set the `sqlcacheps` property of the Properties passed to the `DriverManager.getConnection` method:

```
Properties info = new Properties();
info.put("user", user);
info.put("password", pw);
info.put("sqlcacheps", "true");
conn = DriverManager.getConnection (connStr, info);
```

Or append `@sqlcacheps` to the database specification part of the connect URL

```
Connection conn = DriverManager.getConnection(
"jdbc:rdbthin://bravo:1701/"+
```

```
"myDir:personnel@sqlcacheps=true", user, pass);
```

In addition, a `SET SQLCACHEPS` statement can be executed:

```
   Stmt.executeUpdate("set sqlcacheps true");
```

Remarks

The value **true** specified for the `sqlcacheps` switch tells the thin driver to keep hold of Rdb statement handles and other statement information to re-use if the same Statement SQL text is recognized. A value of **false** specifies that SQL statement handle caching be disabled.

SQL Statement caching must be enabled for SQL statement handle caching to take place; if SQL statement caching is disabled (i.e., sqlcache having the value 0), the `sqlcacheps` switch is ignored.

Enabling SQL statement handle caching by executing a `SET SQLCACHEPS = TRUE` statement will automatically clear out any of the existing SQL statements that may already be cached to ensure that handles are being maintained for all cached statements.

Disabling SQL statement handle caching by executing a `SET SQLCACHEPS = FALSE` statement will prevent any further statement handles being saved. Existing cached statements will still be available for reuse for query compilation, but the associated statement handles will not be reused.

To release all the resources associated with holding statement handles in cache you must clear the SQL cache by issuing a `set sqlcache 0` statement.

Note:

SQL statement handle caching is a client-side action and is disabled by default. This feature is only applicable to the thin driver. The `sqlcacheps` switch and the `set sqlcacheps` statement will be silently ignored by the native driver, or if SQL statement caching is not enabled.

8.1.5 Results Cache

When using the thin driver, performance may be improved by enabling Results caching.

Results caching will maintain ResultSet context across the life of a connection, allowing frequently used data to be cached and reused by subsequent identical queries within the same connection.

Results cache effectively takes a snapshot of the query results the first time a particular SQL query is executed within a connection.

Results caching can be enabled by using the `resultscache` switch when you request a connection either by placing the switch in the connection URL or using the information block that is passed in the connect request.

Example

Set the `resultscache` property of the Properties passed to the `DriverManager.getConnection` method:

```
Properties info = new Properties();
info.put("user", user);
info.put("password", pw);
info.put("resultscache", 10);
conn = DriverManager.getConnection (connStr, info);
```

Or append `@resultscache` to the database specification part of the connect URL:

```
Connection conn = DriverManager.getConnection(
"jdbc:rdbthin://bravo:1701/"+
"myDir:personnel@resultscache=10", user, pass);
```

In addition, a `SET RESULTSCACHE` statement can be executed:

```
Stmt.executeUpdate("set resultscache 10");
```

Remarks

The value specified with the `resultscache` switch tells the thin driver how many ResultSet contexts it can hold concurrently in its cache. A value of 0 (the default) specifies that results caching be disabled.

The Results cache may be cleared by issuing a `SET RESULTSCACHE` statement with the value 0 and then issuing another `SET RESULTSCACHE` statement to reset the cache to the desired size.

Currently you cannot specify the removal of a specific ResultSet from cache.

Note:

ResultSet caching is a client-side action and is disabled by default. This feature is only applicable to the thin driver. Using the `resultscache` property or using the `set resultscache` statement will be silently ignored by the native driver.

Contents

Chapter 9
Event Logging and Notification

Since release 7.3.1

9.1 Event Logging

Oracle Rdb for JDBC servers may be set up to log specified events to a special event logger that can be monitored by either the Oracle JDBC for Rdb controller or a special application created by you or a third party. JDBC servers use the standard Java logging facility to log enabled server events, and these events may be then monitored by local or remote applications that have enlisted with the server as log watchers.

The controller may be used to watch event notifications by using the `WATCH EVENTS` command. Alternatively, you may choose to write your own application to handle event notifications, details on how this may be achieved may be found in the following sections.

An example on how you may use the controller to watch for events from thin servers may be seen in the section Watch Events using the Controller. The Controller subsection Watching Events covers the syntax of the `WATCH EVENTS` command.

Notification of events using a sample event watcher application is covered in WatchEvent Sample Application.

But before events can be monitored, they need to be defined in the XML configuration file used by the server, and then enabled for each server by using enableEvent elements. See the following sections for more details.

Contents

9.2 Defining and Enabling Events.

Rdb Events are special events that are defined in the Events section of the XML configuration file. Once defined these events may be enabled on a server-by-server basis by designating one or more enableEvent elements within the server configuration section of the XML configuration file.

Enabled Events specified in a default server configuration will be enabled for all servers that inherit the default server characteristics.
Servers will only log events that have been enabled for them either explicitly in the server s configuration or in a default server configuration inherited by the server.

The following is an example configuration file that defines and enables events:

```
<?xml version = '1.0'?>
<!-- Configuration file for Rdb Thin JDBC Drivers/Servers -->
<!-- NOTE: all values (including numbers) must be quoted -->
<!--server names and database names are not case sensitive -->

<config>

<connections>
<!-- SERVERS -->
<servers>
<!-- DEFAULT server characteristics-->
  <server
    name="DEFAULT"
```

```
    type="RdbThinSrv"
    url="//localhost:1701/"
    maxClients="-1"
    srv.bindTimeout="0"
    srv.idleTimeout="0"
    srv.networkKeepAlive="true"
    srv.mcBasePort="5517"
    srv.mcGroupIP="239.192.1.1"
    autoStart="false"
    controlUser="jdbc_user"
    controlPass="0x811B15F866179583EB3C96751585B843"
    cfg="my.xml"
    >
    <allowPrivUser name = "owner"/>
    <enableEvent name = "denied_sql"/>
    <enableEvent name = "db_error"/>
    <enableEvent name = "syntax_error"/>

  </server>

  <server
    name = "rdbthnsrv8"
    description = "My test server"
    type = "RdbThinSrv"
    url = "//localhost:1708/"
    allowAccessToCL = "true"
    allowAccessToRMU = "true"
    controluser = "jdbc_user"
    restrictAccess = "true"
    restrictSQL = "SELECT,SHOW"
    >
    <deny SQL = "(?i).*select.*jobs.*"/>
    <allowUser name = "jdbc_user"/>
    <allowPrivUser name = "jdbc_user"/>
    <enableEvent name = "near max"/>
    <enableEvent name = "connect event"/>
    <enableEvent name = "Atleast2"/>
  </server>
</servers>
<!-- events -->
<events>
  <event
    name = "ALL_DENIALS"
    type = "DENIAL"
    message =
    "The operation you have attempted has been denied by this server - please contact
your database administrator"
  />
  <event
    name = "DENIED_USER"
    type = "DENIAL"
    watch = "USER"
    message =
    "You are not allowed to access this server - please contact your database
administrator"
  />
  <event
    name = "DENIED_SQL"
    type = "DENIAL"
    watch = "SQL"
    message =
    "This sql operation is not allowed on this server - please contact your database
administrator"
  />
  <event
    name = "DB_ERROR"
    type = "EXCEPTION"
```

```
    watch = "DBERROR"
    message =
    "Problem with db connection - please contact your database administrator"
  />
  <event
    name = "SYNTAX_ERROR"
    type = "EXCEPTION"
    watch = "SYSERROR"
    pattern = ".*SYNTAX.*"
    message = "Problem with syntax "
  />
  <event
    name = "Near Max"
    type = "THRESHOLD"
    watch = "NUMUSERS"
    deviation= "-3"
    message =
    "The number of users nearing maximum allowed"
  />
  <event
    name = "Atleast2"
    type = "THRESHOLD"
    watch = "NUMUSERS"
    threshold="2"
    message = "The number of users is at least 2"
  />
  <event
    name = "Connect event"
    type = "EVENT"
    watch = "CONN_DISCON"
    message = "Connect or Disconnect occurred "
  />

</events>
</config>
```

The following sections explain how to define and enable events.

9.2.1 Defining Events

Events are described by an event section within the XML configuration file which comprises of several attributes as seen in the following table:

Table 9-1 Event criteria

Name	Description
`deviation`	The deviation criteria allowed. This attribute is only valid for *threshold* type events. See Deviation for more details.
`message`	A customized message to display in the event log when this event occurs. This attribute is valid for all event types.
`name`	The unique name of this event, this attribute must be provided for each event.
`pattern`	Pattern that should be matched, the pattern value should be in the form of a Java regular expression.

Name	Description
	See the Java documentation for more information on Regular Expressions. Details of how the pattern attribute may be used to filter events can be found under each event type subsection within the Event Types section below.
testFor	Test criteria to test for, see the testFor table below.
threshold	This is the actual value that must be met, exceeded, or near to trigger a threshold event. See Threshold Event Threshold attribute for more information.
type	Type or category of event; valid types are specified in Event Types.
watch	The specific element to monitor, this will depend on the type of the event. Details of valid values to watch can be found under the appropriate event type subsection within the Event Types section.

Example

```
<event
  name = "NO_TABLE"
  type = "EXCEPTION"
  watch = "SYSERROR"
  pattern = ".*RELNOTDEF.*"
  message = "Problem with table "
/>
```

The above event will be triggered when an exception is raised that contains the string RELNOTDEF in its message. Details of the format of the event configuration section may be found in Events Section.

Each event must have a unique name, and be valid type as specified in Event Types.
The addition event criteria will depend on the type of the event and is covered under each event type in the following sections.

Contents

9.3 Event Types

Events fall into the following categories or types:

- DENIAL triggered when the server denies an operation.
- EXCEPTION triggered when an exception is raised by the server or the underlying database system.
- THRESHOLD triggered when a threshold has been reached or about to be reached.
- EVENT general purpose event notification.

Depending on the type of the Event, additional criteria may be required within the event description to characterize the event that should be logged. The following sections detail the different event types and attributes used to define the event criteria.

9.3.1 Denial Events

A Denial event, that is an event with the event type `DENIAL` , is triggered when an operation is denied by the server.

The following event criteria may be used with this event type:

- `Name`
- `Message`
- `Watch`

Example

```
<event
  name = "DENIED_USER"
  type = "DENIAL"
  watch = "USER"
  message = "You are not allowed to access this server"
/>
```

The above event will be triggered when a user attempts to login but is not in the `allowUser` list for this server.

9.3.1.1 Denial Event Watch Attribute

The specific operation to monitor is specified by the `WATCH` attribute of the event and include:

- USER will be triggered when the server raises a denied user exception when the user attempting to use the server is not found in the server s allowedUser list.
- DB will be triggered when the server raises a denied database exception when the user attempts to connect to a database that is not found in the server s allowedDB list.
- IP will be triggered when the server raises a denied IP exception when the user attempts to connect to a database from an IP that is not found in the server s allowed IP list.
- AUTH will be triggered if the username/password provided is refused by the underlying database system.
- SQL will be triggered if an attempt to use a SQL statement that is either not in the `restrictSQL` list of the server or is found to match the `deny SQL` attribute of the server configuration. See Restricting SQL Statements for more details.

If no `WATCH` attribute is provided, then the event will be triggered on the occurrence of any of the denial events listed above.

9.3.2 Exception Events

An Exception event, that is an event with the event type `EXCEPTION` , is triggered when an exception is raised by the server or the underlying database system.

The following event criteria may be used with this event type:

- `Name`
- `Message`
- `Watch`
- `Pattern`

Example

```
<event
  name = "NO_TABLE"
  type = "EXCEPTION"
  watch = "SYSERROR"
  pattern = ".*RELNOTDEF.*"
  message = "Problem with table "
/>
```

The above event will be triggered when an exception is raised that contains the string RELNOTDEF in its message.

9.3.2.1 Exception Event Watch Attribute

Exception types are specified by the `WATCH` attribute of the event and include:

- SYSERROR - any exception thrown.

In addition, a pattern may be specified that filters the exceptions to only those that match the given regular expression pattern. The matching pattern is specified using the `PATTERN` attribute.

9.3.2.2 Exception Event Pattern Attribute

A regular expression pattern may be used to filter the exceptions that should be notified as an event. If a pattern is specified for the event, then only those exceptions where the exception message matches the specified pattern will be logged as an event.

If no pattern is specified all exceptions will be logged for this exception event.

See the Java documentation for more information on Regular Expressions.

9.3.3 Threshold Events

A Threshold event, that is an event with the event type `THRESHOLD` , is triggered when the specified threshold is met or exceeded or is within or not within the deviation specified.

The following event criteria may be used with this event type:

- `Name`
- `Message`
- `Watch`

- Threshold
- TestFor
- Deviation

The value that will be tested to see if a threshold has been triggered is specified in the Threshold attribute of the event. The threshold comparison style is determined by the testFor attribute. If a Deviation attribute is also specified, it is used in conjunction with the Threshold attribute to determine a triggering range or limit.

Note:

The testing for threshold events is only conducted during certain standard server operations, such as during the connection of a new client or during the disconnect of an existing connection. During the normal execution cycle of a server, the various thresholds may be exceeded, but an event is only raised if the triggering criteria are met at the time of the trigger test.

See Threshold Event Watch attribute for more information.

Example

```
<event
  name = "Atleast2"
  type = "THRESHOLD"
  watch = "NUMUSERS"
  threshold = "2"
  message = "The number of users of this server is >= 2"
/>
```

The above event will be triggered when the number of users is equal to or exceeds 2.

9.3.3.1 Threshold Event Watch attribute

Threshold types are specified by the WATCH attribute of the event and include:

Table 9-2 Threshold Types for Watch

Literal	Tested during	Max determined by
NumUsers	Connect Disconnect	The number of current clients on the server.
FreeSharedMem	Connect Disconnect	The amount of **free** global shared memory in bytes.
NumFreeExecs	Connect Disconnect	The number of free executors.

The value that will be tested to see if a threshold has been triggered is specified in the Threshold attribute of the event. The threshold comparison style is determined by the testFor attribute. If a Deviation attribute is also specified it is used in conjunction with the Threshold attribute to determine a triggering range or limit.

An event is only raised if the triggering threshold is met at the time specified by the Tested during column in the above table. At all other times the watched criteria may come within trigger range without triggering an

event. So, for example, `FreeSharedMem` events will be only raised if the triggering criteria is met during the Connect or Disconnect phase of a user connection.

9.3.3.2 Threshold Event Threshold attribute

The `Threshold` attribute can be an integer value or may be the literal `Min` or may start with the text literal `Max` .

If a text literal with the value `Min` or starting with `Max` is used, then the threshold value is predetermined by the server attribute that is appropriate:

Table 9-3 Threshold Attributes

Literal	Watch	Threshold determined by
`MaxFreeExecs`	<any>	The server s `maxfreeexecutors` attribute.
`MaxUsers`	<any>	The server s `maxClients` attribute.
`MaxSharedMemory`	<any>	The server s `sharedmem` attribute converted to the number of bytes.
`Max`	`NumUsers`	The server s `maxClients` attribute.
	`NumFreeExecs`	The server s `maxfreeexecutors` attribute.
	`FreeSharedMem`	The server s `sharedmem` attribute converted to the number of bytes.
`Min`	`NumUsers`	The server s `maxClients` attribute.
	`NumFreeExecs`	The server s `maxfreeexecutors` attribute.
	`FreeSharedMem`	The server s `sharedmem` attribute converted to the number of bytes. (**Note**: See description below.)

The text literals shown above are not case sensitive.

If the `Threshold` attribute is not specified, then `Max` is assumed.

By default, `Min` implies the value 0 and will be evaluated as such when used as a threshold check, however when used with the `Watch` values as specified in the table above, the literal `Min` takes on a special meaning:

1. If a % deviation is present, then the absolute deviation is determined by applying the % deviation value against the appropriate threshold attribute as shown in the table above. In this case the positive value of the % deviation is always used, any sign (+/-) is discarded.
2. The threshold range is then determined as being from 0 to the value calculated in point 1 above.

Example

```
<event
  name = "Nearly Exhausted"
  type = "THRESHOLD"
  watch = "FreeSharedMem"
  threshold = "Min"
  deviation = "10%"
  message = " We have nearly used all free memory !"
/>
```

In this example, if the server s sharedmem is set to 100000 bytes, this event will trigger when the amount of free shared memory is in the range of 0 through 10000 bytes.

9.3.3.3 Threshold Event TestFor attribute

The `testFor` attribute determines the test used to set the threshold trigger:

Table 9-4 TestFor Attributes

testFor	Alternatives	Trigger when watch is
`EQL`	`=, ==, EQUAL`	Equal to threshold.
`NEQ`	`!=, <>, NOT EQUAL`	Not Equal to threshold.
`LSS`	`<, LESS`	Less than threshold.
`GTR`	`>, GREATER`	Greater than threshold.
`LEQ`	`<=, LESS OR EQUAL`	Less than or equal to threshold.
`GEQ`	`>=, GREATER OR EQUAL`	Greater than or equal to threshold.
`WITHIN`		Within the deviation of threshold.
`NOT WITHIN`		Not within the deviation of threshold.

The value of the `testFor` attribute may be the keyword as shown in the ***testFor*** column above, or one of the alternate literals as shown in the ***Alternative*** column.

If neither a `testFor` nor a `Deviation` value is specified in the Event attribute list the `testFor` will default to **GEQ**.

If a `Deviation` value is specified but no `testFor` the `testFor` will default to **WITHIN**.

9.3.3.4 Threshold Event Deviation attribute

If a `Deviation` value is specified and a `testFor` value other than **WITHIN** or **NOT WITHIN** is specified, then the `Deviation` is used to modify the `Threshold` value.

If a `testFor` value is not specified or the `testFor` value of **WITHIN** or **NOT WITHIN** is specified, the deviation is used to produce a value range to be tested. The range is inclusive, that is, the range limit values will be considered as being included in the testing criteria.

The `Deviation` value may be signed or unsigned and may be an absolute or a percentage value using the following format:

Format

[+ | -] value [%]

If a sign operator prefix is used the value will be used to modify the `Threshold` value appropriately.

If a `testFor` value is not specified or the value of **WITHIN** or **NOT WITHIN** is specified, then if a plus operator (+) is present the trigger range will be assumed to be from the threshold value to the threshold incremented by the value. If a minus operator (-) is present the range will be assumed to be from the threshold minus the value to the threshold value. If neither a plus operator (+) nor a minus operator (-) is present, then the range will be the threshold +/- the value;

If a percent (%) operator is present a percentage value rather than an absolute value will be used to modify the `Threshold` value.

Example1

```
<event
  name = "NearExhausted"
  type = "THRESHOLD"
  watch = "FREESHAREDMEM"
  threshold = "MAX"
  deviation = "-90%"
  testFor = "LSS"
  message = " Global shared memory is almost Exhausted !"
/>
```

This event will trigger when the amount of free shared memory is less than 10% of the maximum allocated by the server.

Example2

```
<event
  name = "NearExhausted"
  type = "THRESHOLD"
  watch = "FREESHAREDMEM"
  threshold = "MIN"
  deviation = "10%"
  message = " Global shared memory is almost Exhausted !"
/>
```

This event will trigger when the free shared memory is less than or equal to 10% of the maximum allocated by the server.

Note:

This is a range test for values falling in the range 0% to 10% of the maximum free shared memory, inclusive.

Example3

```
<event
  name = "Check Max Free Executors"
  type = "THRESHOLD"
  watch = "NUMUSERS"
  threshold= "MAXFREEEXECS"
  deviation= "2"
  testFor = "WITHIN"
  message = " Consider increasing number of free executors"
/>
<event
  name = "Emphatic Check Max Free Executors"
```

```
  type = "THRESHOLD"
  watch = "NUMUSERS"
  threshold = "MAXFREEEXECS"
  deviation = "+2"
  testFor = "GTR"
  message = " INCREASE the number of free executors NOW!"
/>
```

Event *Check Max Free Executors* will trigger if the current number of users changes to be within 2 of the maximum number of free executors set for the server.

If the number of users is more than 2 above the max free executors, the event *Emphatic Check Max Free Executors* will be triggered instead.

9.3.4 General Events

A general event, that is an event with the event type `EVENT` , is triggered when the specified event or operation is noted by the server.
The following event criteria may be used with this event type:

- `Name`
- `Message`
- `Watch`

Example

```
<event
  name = "Connect event"
  type = "EVENT"
  watch = "CONN_DISCON"
  message = "Connect or Disconnect occurred "
/>
```

The above event will be triggered whenever a connection is made or is disconnected.

9.3.4.1 General Event Watch Attribute

The server event types are specified by the `WATCH` attribute of the event and include:

- `Conn_Discon` log whenever a connection is made or is disconnected.

Contents

9.4 Watch Events Using the Controller

The controller may be used to watch for events happening on a JDBC server.

The `Watch Events` command tells the controller to enlist as an event listener with the specified server. All events then logged by the server will be displayed by the thin controller as XML-formatted events:

```
rdbthincontrol> watch events 1708
rdbthincontrol>
logger:1540:// nnn.nnn.nnn.nnn:1708/>rdbthnsrv8 : 14/01/2025 12:29:29
oracle.rdb.jdbc.srv.RdbSrv logEvent
logger:1540:// nnn.nnn.nnn.nnn:1708/>INFO:
<event>
  <seq>0</seq>
  <timestamp>2025-01-14 12:29:29.120</timestamp>
  <type>13</type>
  <name>CONNECT</name>
  <source>
    <jdbcServer>
      <name>rdbthnsrv8</name>
      <type>0</type>
      <typeName>RdbThinSrv</typeName>
      <ip>//nnn.nnn.nnn.nnn:1708/</ip>
      <pid>4212</pid>
      <pidHex>0x1074</pidHex>
    </jdbcServer>
  </source>
  <context>
    <trigger>
      <eventTrigger> <name>Connect event</name> <type>EVENT</type>
        <message>Connect or Disconnect occurred </message>
      </eventTrigger>
      <reason>
        <type>CONNECT</type>
        <info>
          <![CDATA[127.0.0.1:1542]]>
        </info>
      </reason>
    </trigger>
    <client>
      <id>00000002</id>
      <name>joeblogs</name>
      <db>e:\regtest\mf_personnel</db>
      <ip>127.0.0.1:1542</ip>
    </client>
  </context>
</event>

logger:1540:// nnn.nnn.nnn.nnn:1708/>rdbthnsrv8 : 11/01/2025 12:29:30
oracle.rdb.jdbc.srv.RdbSrv logEvent
logger:1540:// nnn.nnn.nnn.nnn:1708/>INFO:
<event>
  <seq>1</seq>
  <timestamp>2025-01-11 12:29:30.573</timestamp>
  <type>14</type>
  <name>DISCONNECT</name>
  <source>
    <jdbcServer>
      <name>rdbthnsrv8</name>
      <type>0</type>
      <typeName>RdbThinSrv</typeName>
      <ip>//nnn.nnn.nnn.nnn:1708/</ip>
      <pid>4212</pid>
```

```
      <pidHex>0x1074</pidHex>
    </jdbcServer>
  </source> <context>
  <trigger>
    <eventTrigger> <name>Connect event</name>
      <type>EVENT</type>
      <message>Connect or Disconnect occurred </message>
    </eventTrigger>
    <reason>
      <type>DISCONNECT</type>
      <info>
         <![CDATA[127.0.0.1:1542]]>
      </info>
    </reason>
  </trigger>
  <client>
    <id>00000002</id>
    <name>joeblogs</name>
    <db>e:\regtest\mf_personnel</db>
    <ip>127.0.0.1:1542</ip>
  </client>
</context>
</event>
```

See Watching Events for the syntax of the Controller `Watch Events` command.

Contents

9.5 WatchEvent Sample Application

During installation the WatchEvent sample application will be installed in your JDBC installation directory. This application, deployed as a Java JAR, can be run to watch for Rdb events being raised by the JDBC servers.

This sample application may be used as a template to help you develop your own Event notification application.
If you have not used this application before, please carry out the steps described in the following sections.

9.5.1 WatchEvent Sample Application Setup

The `WatchEvent.jar` file contains a sample application that may be modified to carry out your own operations when a JDBC server event is received.

You may run the application unchanged on systems that run Java and have Java/Swing enabled by carrying out the following steps:

Step 1 Copy the `WatchEvent.jar` file to your desired system host.

The `WatchEvent.jar` is a Java/Swing based application JAR file that may be run on any system where JRE Version 1.6 or higher is running and that has Java/Swing enabled.
During installation of JDBC the `WatchEvent.jar` file will be copied to the JDBC installation directory. If you have conducted the advised post installation steps this directory will be pointed to by the logical name `RDB$JDBC_HOME`.

Copy this JAR file to the system you wish to run on. Make sure that any file copy application you use copies the file in binary format otherwise it may not work correctly on the destination system.

As well as containing the Java application, the JAR file contains the following source files that may be extracted and used:

- `WatchEvents.java` the source for this application
- `WatchEvents.cfg` a sample configuration file
- `Event.xsl` a sample XSLT stylesheet

Various JAR and ZIP file explorer applications are available that allow the extraction of components from Java JAR files, please consult the operating system documentation or the internet for a tool you may use on your chosen host system.

Inline comments within the `WatchEvents.java` source file should help you use the source as an example of how to write your own JDBC notification application.

These three source files may be tailored or modified to suit your own requirements.

How these files may be modified and used is discussed in the following sections.

Step 2 Optionally create or modify the configuration file.

The JAR file contains a sample configuration file, `WatchEvent.cfg`, that may be extracted and modified to suit your specific requirements. This configuration file is formatted as a standard Java properties file and contains the application configuration properties that are used to determine the behaviour of the application.

The default configuration file contains the following:

```
# WatchEvents configuration file
user = jdbc_user
pw = jdbc_user
server = localhost:1701
loggerport = 1555
XSL = Event.xsl
display = 0
```

The following properties may be used:

Table 9-5 WatchEvent Configuration Properties

Property	Default	Description
`display`	0	The type of display to use: 0 = display each message within its own individual modal dialog box. 1 = append formatted event information to the WatchEvent application s main dialog box. -1 = display each message on the console running the WatchEvent application.
`loggerport`	1555	The port to be used for the remote logging.

Property	Default	Description
pw	jdbc_user	The control password for the server you will be watching.
server	localhost:1701	The url (host:port) of the server.
user	jdbc_user	The control username for the server you will be watching.
XSL	Event.xsl	The XSLT stylesheet file used to format the event XML input message received. See below for more details.

If you use a tailored configuration file, you should make sure that the application can use it when it is invoked.

By default, the application will look for the configuration file named WatchEvents.cfg in the same directory that the application JAR file is invoked from.

If you wish to change the configuration file name or change its location you will have to provide the configuration switch -CFG on the command line used to invoke the application, for example:

```
java jar WatchEvents.jar cfg c:\myDir\my_own_config.cfg
```

If no configuration file is found, or if a property is not specified in the configuration file, then the default settings described in the table above will be used.

Step 3 Optionally create or modify the event XSL stylesheet file.

The XSL property of the configuration file specifies an XSLT stylesheet file that should be used to format the event messages received from the server.

The event messages are XML documents that contain information about the event raised.

You may change how this information is displayed by changing the default XSL file contents or by specifying your own stylesheet. See XSLT for more information on how to use stylesheets.

The sample XSLT stylesheet `Event.xsl` may be extracted from the application JAR and modified to suit your own setup.

The application will try to locate a valid XSL stylesheet using the following search precedence:

1. the file described by the XSL property switch if provided on the command line.
2. the file described by the XSL property switch if provided in the configuration file.
3. the file `Event.xsl` if it exists in the same directory containing the application JAR file.
4. the file `xmlout.xsl` if it exists in the same directory containing the application JAR file.
5. `Event.xsl` if it exists inside `WatchEvents.jar`.

If no stylesheet is found, then the raw XML information will be displayed.

9.5.2 Invoking the WatchEvent Sample Application

Once correctly set up, the application can be invoked using the standard Java executable JAR invocation:

```
java jar WatchEvents.jar
```

The invocation command line may also contain configuration switches using the same configuration properties as shown in the *WatchEvent Configuration Properties* table above by preceding the configuration property name with a hyphen `(-)` and adding the property value after it separated by a blank space, for example:

```
java jar WatchEvents.jar pw mysecret XSL demo.xsl
```

Any property not specified will be set to the default value as specified in the *WatchEvent Configuration Properties* table.

Contents

Chapter 10
Other Features

10.1 Anonymous Usernames

By default, the thin driver disallows blank usernames to be passed to it during database connection. A valid username for that database must be used. If the client attempts to connect to the database using a blank username the following exception will be raised:

```
rdb.RdbException: Io exception : Io exception :
in <rdbjdbcsrv:connect>
%RDB-E-AUTH_FAIL, authentication failed for user .Anonymous.
```

The following server configuration option can be used to change this behavior:

- `anonymous`

Use this option to allow anonymous connections (that is, where the username is blank) to the Oracle JDBC for Rdb Thin server, for example:

```
$ java -jar rdbthinsrv.jar -anonymous
```

In addition, if anonymous connections are allowed, you can specify the default username and password to use on an anonymous connection by using the following options:

- `username <username>`
- `password <password>`

Example

```
$ java -jar rdbthinsrv.jar -anonymous
-username fred -password "jones"
```

Contents

10.2 BYPASS Privilege

Privilege checking on Oracle Rdb uses the layered method. Sometimes it is not obvious how privilege checking obtains its results:

- The first pass at privilege checking occurs at an object identifier level, asking if this entity has the right to do this action to this object. If access is denied at this level a series of cascading attempts are made to try to get the privilege.

- After the object protection is checked, the entity's privilege at the database is checked. If the entity has been granted **DBADM** it will be allowed to conduct the operation even if it does not have the explicit privilege such as **CREATE**. This privilege is a type of override privilege like BYPASS on OpenVMS.

- If the entity still has not been granted the privilege at the database level, the OpenVMS privileges for the OpenVMS user that the application is running under are checked.

- If that user has the appropriate level of privilege, they are then granted the action on the object.

This means that privilege checking within Oracle JDBC for Rdb server not only depends on the privilege assigned to the connection user within the database, but also on the privilege of the OpenVMS user that started the server application (the Executor).

Note:

The Executor is the standard term used for the OpenVMS user under which the application is executing. This should not be confused with the "executor" processes used in conjunction with JDBC Multi-process servers.

This allows you to set up a privileged server that has access to data that the user may not have. In other words, you can restrict user access to data in the database if and only if they come through the Oracle JDBC for Rdb server; they do not have access directly.

If you wish restricted access, grant restricted access only to the Executor and set minimum privileges. Then grant the appropriate rights to connection users so that they will have the required access. If they do not have the rights and the Executor does not have the rights, access is denied. If the user does have the right even though the Executor does not, access is allowed.

Within the thin server the **BYPASS** and **SYSPRV** privileges are disabled by default. The user will only get the privileges he has been granted and will not inherit privileges from the executor.

If the server is required to run with BYPASS privilege, thus allowing less privileged users access to the database objects, you should use the `-bypass` option.

10.2.1 BYPASS and Multi-process Servers

When you use a JDBC Multi-process server a separate executor process is used to carry out the database operations. This executor process inherits the privileges and authorization characteristics from the server process that started it.

Thus, the information as described above applies to the executor processes in the same manner as described for the server process.

10.2.2 Rights Identifiers

When using OpenVMS rights identifiers to restrict access to a database and its objects you should ensure that the database security option has been set on the Rdb database to enable PERSONA checking.

By default, Rdb databases created or restored prior to Oracle Rdb release 7.3.1 will not have the security PERSONA feature enabled by default. If this feature is not enabled, Rdb will not consider any OpenVMS rights identifiers of the connecting username when determining access and authorization.

See the `ALTER DATABASE SECURITY IS EXTERNAL (PERSONA SUPPORT IS ENABLED)` option in the ***Oracle Rdb SQL Reference Manual Volume 2*** for more details on how to change your older databases to enable PERSONA for Rdb security checking.

Note:

Rdb PERSONA checking is different to the **Persona** option used by JDBC servers. Please see your Oracle Rdb documentation for more information on PERSONA.

Contents

10.3 Persona

When an Oracle JDBC for Rdb Thin server is running, it assumes the default privileges and identifiers of the user that started the server process. Similarly, when a SQL/Services JDBC Dispatcher starts a server, the server will inherit the privileges and identifiers of the SQL/Services dispatcher process.

You can change this behaviour by specifying a persona value in the server definition for the server in the XML-formatted configuration file, or by using the persona switch on the command line when starting up the server.

When started with a persona, the server process will inherit its privileges and identifiers from the named persona.

BYPASS and **SYSPRV** privileges are still disabled by default, see BYPASS Privilege for more details.

To start a server with a specific persona, you will need to be logged into an account that has `IMPERSONATE` privilege and read access to the system authorization database.

The persona value associated with the server must be a valid OpenVMS persona on the system you are running the server on.

See Server Configuration Options for the format of the Persona option.

10.3.1 Persona and Server Operations

When persona is used with a server, you should ensure that the persona used has appropriate access to the JDBC command procedures and JDBC log directories. This is especially important if you use persona with a Pool server or a Multi-process server.

Before a server carries out any other operation it will assume the persona provided and then by default disable BYPASS (see BYPASS Privilege). From that time on the server is operating under the persona supplied and will be restricted to the rights and authorization given to that persona.

When persona is used both the Multi-process server and the Pool server will need to have read/execute/write access to the `RDB$JDBC_COM` directory and read/write access to the `RDB$JDBC_LOG` directory.

By default, the installation of the JDBC drivers will create these directories on your installation destination directory and set the access to both these directories to give WORLD read/execute access (W:RE). You will have to alter the file protection on these directories and grant WRITE access to the persona.

If you have redirected these logical names to another directory, you must ensure that the persona has the read/write access to these directories.

See File and Directory access Requirements for more details.

Contents

10.4 Default Transaction

The type of transaction the Oracle JDBC for Rdb drivers start up when a transaction is required depends on several conditions:

- Whether `AUTOCOMMIT` is enabled.
- The verb of the SQL statement to be executed.
- The default transaction type specified on connection using the connection option `transaction`.
- The setting of the transaction types in the connection if changed by methods such as `Connection.setReadOnly()` and `Connection.setTransactionIsolation()`.

If no specific behaviour has been specified, by default the Oracle JDBC for Rdb drivers will start in `AUTOCOMMIT` mode and will start up a `READ_WRITE SERIALIZABLE` transaction if the SQL statement requires a read-write transaction, for example, `INSERT` or `UPDATE`. If the statement does not require a read-write transaction, a `READ_ONLY` transaction is started.

Immediately after the connection has been made, JDBC will start the default transaction in anticipation of subsequent operations on the database. By default, this will be a `READ_ONLY` transaction. If you do not wish to have a transaction started at this stage, you may defer the start of the transaction by using either `READONLYDEFER` or `READWRITEDEFER` on the connection string `TRANSACTION` option. See Connection Options for more details.

When `AUTOCOMMIT` is disabled, the type of transaction started will depend on whether the connection has been set read-only and whether a default transaction type has been specified on the connection using a connection switch. By default, a `READ_WRITE SERIALIZABLE` transaction will be started if `AUTOCOMMIT` is turned off and no other method has been called to change the default transaction type.

If the setting of the transaction type in the connection is `MANUAL` this default behaviour changes. Setting transactions to `MANUAL` indicates that the client will take responsibility for the starting of transactions. The drivers will no longer start transactions. However, if `AUTOCOMMIT` is enabled, the driver will still commit transactions appropriately.

When transactions are set to `MANUAL`, and the first operation after a connection or after a transaction termination is not `SET TRANSACTION`, Oracle Rdb will start a transaction on behalf of the client. Please see the Oracle Rdb documentation for information on the default transaction mechanism provided by SQL.

10.4.1 Autocommit Transaction Duration

Since release 7.3.4.0.4

When `AUTOCOMMIT` is enabled, JDBC will automatically commit transactions at the end of the execution of a statement.

Sometimes during long-duration client-side read operations it may be advantageous to commit the current transaction more frequently, to prevent the read-only transaction from locking out other database operations.

For example, when using *SQLDeveloper* to read through Rdb records using Oracle JDBC for Rdb, records are delivered to the client application in groups, where the number of records in a record group depends on the `FETCHSIZE` specified for the statement or session. As you scroll down a list of records, the application may request the next group of records from the server. This may mean that if you have not finished scrolling through all the records in the retrieval set, there may still be a transaction outstanding on the server waiting for the rest of the records to be requested. If you leave this browser window open for an extended duration, this could cause read-only transaction lockouts on the database.

JDBC allows you to specify that transactions should be auto-committed each time the server sends a `FETCHSIZE` number of records back to the client. This means that while the client is browsing the current group of records, that no transaction will be outstanding on the server.

The `TRANSACTION` option `AUTOFETCH` may be used to inform JDBC that auto-commit is required each time the server sends `FETCHSIZE` groups of records. In addition, the `TRANSACTION` option `AUTOREADFETCH` not only causes an auto-commit to be done on each record group send, but also tells JDBC to start a Read-Only transaction by default.

See the general JDBC documentation on the role of `FETCHSIZE` in JDBC operations and how to specify the number of records to retrieve during client/server fetch operations. The application you are using may also have documentation on how to specify the `FETCHSIZE` when using JDBC drivers.

Contents

10.5 Executor Sub-process Used with the Rdb Native Driver

To improve multi-threaded concurrent access to the database while using the Rdb Native driver, you may specify that separate sub-process executors should be started for each connection request.

By default, all database operations within a standard Rdb Native driver instance are carried out synchronously, within a single OpenVMS process. This synchronization is required as Rdb will only let one thread carry out a database operation at a time. This may limit the general concurrency that may be seen if you are using the Rdb Native driver within a multi-threaded environment.

To improve concurrency in a multi-threaded environment you can request the Rdb Native driver to start-up a separate executor for the database connection.

To start a separate executor for the connection you need to specify the `multiprocess` switch on connection URL you use for your database connection:

```
Connection conn = DriverManager.getConnection(
   "jdbc:rdbNative:myDir:pers@multiprocess=true", user, pass);
```

Since release 7.3.4.2.3

Alternatively, you could set the `multiprocess` option to the value `spawn` to achieve the same results:

```
Connection conn = DriverManager.getConnection(
   "jdbc:rdbNative:myDir:pers@multiprocess=spawn", user, pass);
```

Note that this switch is only available when you use the Rdb Native driver.

As a separate sub-process is created for each connection made, output written by the executor process to SYS$OUTPUT and SYS$ERROR will be redirected to log files specific to that sub-process. You should ensure that your process has write access to the log directory RDB$JDBC_LOGS.

10.5.1 Starting a Detached Executor Process

Since release 7.3.4.2.3

By default, when requested, the native driver will start a separate sub-process to run the executor image within. If you wish the driver to use a detached process instead of a sub-process, you may specify this by giving the multiprocess connection string switch the value **detached** :

```
Connection conn = DriverManager.getConnection(
   "jdbc:rdbNative:myDir:pers@multiprocess=detached", user, pass);
```

This tells the driver to create a separate process that will run the standard ***loginout.exe*** on startup.

10.5.2 Setting an Alternative ExecStartup Command Procedure

Like the MP server, after the executor process is created, to start the executor image the native driver uses an OpenVMS DCL command procedure. By default, this is:

```
   rdb$jdbc_home:rdbjdbc_startexec.com
```

As the executor process does have its own environment it may be necessary for you to include environmental setup commands specific to your database environment. For example, if you run multiple versions of Rdb on your system you may have to setup the environment to choose the correct Rdb version to use prior to the executor image being invoked.

You may choose to add these environment setup commands to the default JDBC executor startup command procedure but, if you do so, you should be reminded that this command procedure may be replaced if you reinstall JDBC on your system.
Thus, Oracle recommends that you do not alter the standard command procedure. Instead, you should create a copy of it, make your changes and rename this copy for your own use. To tell the driver to use this modified command procedure add a srv.execStartup specification to the connection string:

```
Connection conn = DriverManager.getConnection(
    "jdbc:rdbNative:myDir:pers@multiprocess=true" +
"@srv.execStartup=myDir:my_startexec.com", user, pass);
```

10.5.3 Setting Maximum Handshake Tries and Wait Duration

When the main process starts an executor process a handshake protocol is established between the two processes to allow them to conduct subsequent inter-process communication.

The main process will attempt 100 times in quick succession to establish the handshake and then, by default, will try 500 more times with a delay of 10 milliseconds between each try.

On some systems where the workload is heavy and particularly on single-CPU systems it is possible that after the sub-process is created the main process may attempt to establish the communication unsuccessfully.

Depending on process and thread scheduling it is possible that the maximum number of attempts to establish handshake may occur before the sub-process is scheduled for execution.

On these systems you may wish to increase the number of attempts at handshake or the duration to wait between handshake attempts to prevent the premature aborting of the driver-executor connection. You may use the `handshakeTries` and `handshakeWait` options on the connection string to change these values.

See Connection Options for more details.

Contents

10.6 JDBC Hint Methods

Several methods in the JDBC classes are considered to provide hints to the drivers or underlying database system and do not have to be strictly observed. Many existing drivers silently ignore these methods.

To allow compatibility with other drivers, you may specify that optional hint methods be ignored by using the `usehints` connection switch:

```
@usehints=false
```

This setting tells the Oracle JDBC for Rdb drivers to ignore hint methods.

By default, the Oracle JDBC for Rdb drivers will observe hint methods.

The following methods are perceived as non-mandatory hints:

- `Connection.setReadOnly()`
- `ResultSet.setFetchDirection()`
- `ResultSet.setFetchSize()`
- `Statement.setFetchDirection()`
- `Statement.setFetchSize()`

Contents

10.7 Logging and Tracing

Exceptions, informational and warning messages are written to the server s log file, if one has been specified.

In addition, when a server is started with a non-zero ***tracelevel***, trace messages are also written to the server s logfile.

The server s log file is generally found within the RDB$JDBC_LOGS directory, but server configuration attributes may be used to point to any directory that the server process has write access.

If an exception is raised within the context of an executor process, then the exception message will be logged to one of the executor s log files, usually also found in the RDB$JDBC_LOGS directory.

If no log file is specified for a server, these messages are written to the `System.out` channel of the process within which the server is running.

The logfile server configuration attribute is generally an absolute or relative file specification of where to write log messages. If the file specification is relative, then it will be relative to the default directory from which the server process was invoked.

The logfile specification may also be a special pattern, in which case certain key character sequences may be used in conjunction with normal ASCII filename characters to specify the automatic creation of the log file name. See Logfile Pattern for details on log filename pattern characters.
In addition to logging to a log file, certain events and other messages can also be logged using the standard Java Logging facilities.
By default, Java Logging is turned off.

See the Java JDK 1.4.1 (or later) documentation for information on the Java Logger.

10.7.1 *Logfile Pattern*

Since release 7.3.2

The following table shows the pattern character sequences that may be used in logfile patterns:

Table 10-1 Logfile patterns sequences

Pattern	Description
"/"	Replace with the local directory separator.
"%c"	Replace with the current client id.
"%d"	Replace with a compressed date/time value.
"%g"	Replace with a generation number to distinguish rotated logs. If no "%g" field has been specified and the file count is greater than one, then a generation number will be added to the end of the generated filename.
"%l"	Replace with the standard log directory, rdb$jdbc_logs: .
"%m"	***Since release 7.3.5.2*** Replace with the node name.
"%n"	Replace with the server s name.
"%p"	Replace with the current process id (PID) of the server.
"%u"	Replace with a unique number to resolve conflicts.
"%%"	Translates to a single percent sign "%".

Example

Given the following entries in the configuration file:

```
<server
name="DEFAULT"
type="RdbThinSrv"
url="//localhost:1701/"
```

```
log="%l%n_%d.log"
/>

<server
name="TESTSRV"
tracelevel="-1"
port = "1899"
/>
```

When TESTSRV server is started on the date January 14, 2025 it will log trace messages and exceptions to the file:

```
RDB$JDBC_LOGS:TESTSRV_P1EL2LC.LOG
```

Contents

10.8 Ignoring Statement.cancel() Method Calls

Currently the method `Statement.cancel()` is not supported in the Oracle JDBC for Rdb drivers. If an application calls this method the driver will raise the following Exception:

```
oracle.rdb.jdbc.common.RdbException: Unsupported feature <Statement.cancel>
```

In applications and application servers that expect this feature to be present, the raising of this exception may cause problems with the application functionality or may lead to excessive messages being written to the application log file.

If your application does not depend on the statement cancellation to take effect, and that failure to cancel can be safely ignored, you may specify the `ignoreStatementCancel` switch of the connection URL:

```
Connection conn = DriverManager.getConnection(
"jdbc:rdbNative:myDir:pers@ignoreStatementCancel=true",
user, pass);
```

Contents

10.9 Server Name

Each server may be given its own name that may be used to identify a server within the controller and to look up configuration information. The name of a server may be used to identify configuration setting within an XML-formatted configuration file on server start up.

Example 1

Given the following entry in `MY_CFG.XML` file:

```
<servers>
<server
name="myMPServer"
type="RdbThinSrvMP"
url="//localhost:1788/"
/>
```

```
</servers>
```

And the following command line statement:

```
$ java -jar rdbthinsrv.jar -cfg MY_CFG.XML -name myMPServer
```

A Multi-process server with the name *myMPServer* will be started up on localhost listening to port 1788.

Names of servers within an XML-formatted configuration file must be unique as it is by name alone that server characteristics are searched for within the configuration file. Note that on OpenVMS character case is not significant in name matching.

The following two special server names may be used, DEFAULT and DEFAULTSSL, within the XML-formatted configuration file.

The server characteristics defined in the DEFAULT server will be used to provide the base configuration information for all servers, but any of these characteristics can be over-ridden either by command line switches or by characteristics defined within the specified named server in the configuration file.

Example 2

Given the following server entry in MY_CFG.XML file:

```
<servers>
<server
name = "DEFAULT"
type = "RdbThinSrv"
url = "//localhost:1755/"
maxClients="-1"
/>
<server
name="myServer"
maxClients="10"
/>
</servers>
```

And the following command line statement:

```
$ java -jar rdbthinsrv.jar -cfg MY_CFG.XML -name "myServer"
```

A thin server with the name *myServer* will be started up on localhost listening to port 1755 with maxClients =10.

The server characteristics within the DEFAULTSSL server will be used to provide base SSL information for RdbThinSrvSSL type servers.

Example 3

If an XML-formatted configuration file is used, a server is not found that matches the name provided on the command line and a DEFAULT server definition is provided, then the DEFAULT server characteristics will be used for that server.

Given the following server entry in MY_CFG.XML file:

```
<servers
<server
name = "DEFAULT"
```

```
type = "RdbThinSrv"
url = "//localhost:1799/"
maxClients=-1
/>
</servers>
```

And the following command line statement:

```
$ java -jar rdbthinsrv.jar -cfg MY_CFG.XML -name "myServer"
```

A thin server with the name *myServer* will be started up on localhost listening to port 1799 with unlimited maxClients.

Contents

10.10 Named Databases

The XML-formatted configuration file allows the specification of known named databases, allowing the Oracle JDBC for Rdb servers the ability to recognize alternate names for databases served on the node the server is running on.

Like logical names and JNDI name spaces, the use of alternate names allows the separation of the name the client uses for the database and the actual file specification of the database.

Before requesting Oracle Rdb to connect to a database, the thin server will check its list of known databases for a match against the file specification portion on the given Connection URL. If one matches, then the file specification portion of the URL property of the named database will be used to provide the connection database specification.

Example

Given the following named database:

```
<database
name="mf_pers"
url="//localhost:1701/disk1:[databases]mf_personnel"
driver="oracle.rdb.jdbc.rdbThin.Driver"
URLPrefix="jdbc:rdbThin:"
/>
```

And the following connection statement:

```
   Connection conn = DriverManager.getConnection(
"jdbc:rdbThin://bravo:1701/mf_pers",user, pass);
```

The client will be connected to the Oracle Rdb database:

```
disk1:[databases]mf_personnel.rdb
```

During the translation of the named database, the node and port part of the URL within the named database definition is discarded.

Named databases may also be used to restrict database access within the server. See Restricting Database Access for more information on this feature.

The list of named databases may be made available for client application access if the server configuration option `allowShowDatabases` is set to **true** . See Getting a List of Known Databases from Server for more details.

Contents

10.11 On Start Commands

There are three startup command attributes that may be specified in the XML-formatted configuration file server section: `srv.onStartCmd` , `srv.onExecStartCmd` and `srv.onCliStartCmd`.

These options allow the specification of DCL commands that should be executed just prior to the startup of a server or executor.

Note:

The `srv.onStartCmd`, `srv.onExecStartCmd` and the `srv.onCliStartCmd` point to a command that will be executed on startup of the server, executor or CLI invocation. If the command is to invoke a DCL command procedure you must also include the DCL invocation symbol @ in the command line.

10.11.1 srv.onStartCmd

This option specifies a DCL command to be executed prior to the invocation of the specified thin server. It must be a valid OpenVMS DCL command and must be valid within the context of the server process created by the controller or Pool server.

If multiple DCL commands are required then they should be placed within a DCL command procedure, which in turn should be made available to the environment under which the controller or Pool server runs. Oracle recommends that these command procedures be placed within the `rdb$jdbc_com` directory and the file protection set to allow the controller or Pool server execute access.

Example 1

For example, if your system requires a specific setup to be run to set your Java environment and Oracle Rdb environment, you may create a command procedure like the following example.

Create `rdb$jdbc_com:our_setup.com` containing:

```
$ @sys$share:rdb$setver 74
$ @sys$common:[java$80.com]JAVA$80_SETUP.COM
```

And provide a pointer to this command procedure in the `srv.onStartCmd` option:

```
<server
name="srv2forRdb"
type="RdbThinSrv"
url="//localhost:1708/"
srv.onStartCmd="@rdb$jdbc_com:our_setup.com"
/>
```

Care should be taken when providing commands for the server process to execute using this property. These commands will be executed prior to the invocation of the Java statement that starts the actual server instance. As detached OpenVMS processes will be used to run the server you must ensure that all the necessary symbols and logical names are available for the server's use within the detached process.

If you redefine the standard `RDB$JDBC_*` logical names within your set-up to use a private version of Oracle JDBC for Rdb, you must ensure that appropriate JAR files and images are available and executable within the detached process server environment.

Example 2

For example, care should be taken in how the logical names are specified. The following redefinition may appear to point the logical name to the current default directory:

```
$ define rdb$jdbc_home []
```

However, this logical name will be translated during the creation of the temporary command procedure that will be used to start the server, which in this case as only the directory has been specified, the disk or device will default to the current device of the login directory of the detached process, which might not be the same device as you expected. This may prevent the server process from correctly starting.

If you need to redefine a logical name to the current default directory, you can use the `f$environment` DCL lexical function:

```
$define rdb$jdbc_home 'f$environment("DEFAULT")
```

This will set both the default device and directory.

If problems are found with starting a server process, you can look for new log files in the `RDB$JDBC_LOGS` directory which may provide some information on any errors found.

Caution:

Do not use the SET VERIFY command within these command procedures, as the method `Runtime.exec()` may be used by the servers to create processes. The use of the SET VERIFY command within the command procedure may hang the server. This is a documented limitation of using `Runtime.exec()` on Open VMS Java. The logical name JAVA$EXEC_TRACE is available to help debug `Runtime.exec()` calls on OpenVMS. Refer to the OpenVMS Java documentation for more details.

Note:

The `srv.onStartCmd` command is only used by the controller or Pool server to start a server. If the server is started by any other means, neither the server startup command procedure nor any commands in the `srv.onStartCmd` server attribute will be executed.

10.11.2 srv.onExecStartCmd

This option specifies a DCL command to be executed prior to the invocation of an executor by a Multi-process server. It must be a valid OpenVMS DCL command and must be valid within the context of the Multi-process server process.

If multiple DCL commands are required, then they should be placed within a DCL command procedure, which in turn should be made available to the environment under which the server runs. It is recommended that these

command procedures should be placed within the `rdb$jdbc_com` directory and the file protection set so that the server can access them.

Example

If your system requires a specific setup to be run to set your Oracle Rdb environment, you may create a command procedure like the following example.

Create `rdb$jdbc_com:our_exec_setup.com` containing:

```
$ @sys$share:rdb$setver 7.4
```

And provide a pointer to this command procedure in the `srv.onExecStartCmd` option:

```
<server
name="MPsrv2forRdb"
type="RdbThinSrvMP"
url="//localhost:1788/"
srv.onExecStartCmd="@rdb$jdbc_com:our_exec_setup.com"
/>
```

Caution:

Do not use the SET VERIFY command within these command procedures, as the method `Runtime.exec()` may be used by the servers to create processes. The use of the SET VERIFY command within the command procedure may hang the server. This is a documented limitation of using `Runtime.exec()` on Open VMS Java. The logical name JAVA$EXEC_TRACE is available to help debug `Runtime.exec()` calls on OpenVMS. Refer to the OpenVMS Java documentation for more details.

10.11.3 srv.onCliStartCmd

This option specifies a DCL command to be executed prior to the invocation of a CLI statement by a JDBC server. It must be a valid OpenVMS DCL command and must be valid within the context of the server process.

If multiple DCL commands are required, then they should be placed within a DCL command procedure, which in turn should be made available to the environment under which the server runs. It is recommended that these command procedures should be placed within the `rdb$jdbc_com` directory and the file protection set so that the server can access them.

Example

If your system requires a specific setup to be run to set your Oracle Rdb environment, you may create a command procedure like the following example.

Create rdb$jdbc_com:our_cli_setup.com containing:

```
$ @sys$share:rdb$setver 7.4
```

And provide a pointer to this command procedure in the `srv.onCliStartCmd` option:

```
<server
name="MPsrv2forRdb"
type="RdbThinSrvMP"
url="//localhost:1788/"
srv.onCliStartCmd="@rdb$jdbc_com:our_cli_setup.com"
/>
```

Caution:

Do not use the SET VERIFY command within these command procedures, as the method `Runtime.exec()` may be used by the servers to create processes. The use of the SET VERIFY command within the command procedure may hang the server. This is a documented limitation of using `Runtime.exec()` on Open VMS Java. The logical name JAVA$EXEC_TRACE is available to help debug `Runtime.exec()` calls on OpenVMS. Refer to the OpenVMS Java documentation for more details.

Contents

10.12 Password Obfuscation in Server Configuration Files

There are two types of passwords that may be stored in the server configuration files:

- control passwords
- user passwords

In addition, two types of obfuscated passwords are allowed in server configuration file:

- Obfuscations produced by the `digest` command which is an allowed form for control passwords.
- Obfuscations produced by the `obfuscate` command which is an allowed form obfuscation of user passwords.

The main difference in the obfuscation produced by these two commands is that `digest` uses one-way algorithms where-as `obfuscate` uses reversible algorithms.

10.12.1 Control Passwords

To help prevent an unauthorized user from controlling server operations such as closing a running server, a control password should be assigned to each server on startup. This password must be used whenever server control operations are carried out using the Oracle JDBC for Rdb Controller interface.

To ensure better security of these control passwords, the server configuration file may contain the server control password in an obfuscated form. You can obtain an obfuscated password for a control password by using the `digest` statement in the Controller.

Example 1

```
rdbthincontrol> digest thisismypassword
digest : 0x31435008693CE6976F45DEDC5532E2C1
```

The value can then be used in the configuration file where you would have normally provided a plain text control password.

Example 2

```
<server
name="RdbThinSrv1707"
type="RdbThinSrvMP"
url="//localhost:1707/"
srv.execStartup="mystartup"
controlUser="jdbc_user"
```

```
controlPass="0x811B15F866179583EB3C96751585B843"
/>
```

This value must be copied exactly as returned by the `digest` command.

The plain text password conversion to its obfuscated form is case-sensitive, so the same word or phrase but with different character casing will produce a different digested form.

Passwords are case sensitive so you must ensure that the value of the password used in plain text and in its digested form match exactly character by character including case.

This is particularly important if a password is used on the DCL command line. If double quotation characters are not used to surround the plain text password DCL may, depending on your environment, force the value to all lower case or all uppercase which may differ from the original.

Example 3

When *`-digest`* is used in command mode make sure the value is enclosed in double quotations:

```
$ java -jar rdbthincontrol.jar -digest "MySecretPassword"
digest : 0x7315A012ECAD1059A3634F8BE1347846
$ java -jar rdbthincontrol.jar -digest MySecretPassword
digest : 0x4CAB2A2DB6A3C31B01D804DEF28276E6
```

Note:

Obfuscated control passwords are only valid when used in conjunction with a server definition in a configuration file or as a server start up command line configuration option. To connect to the server as a control user to conduct operations on it using the controller, the control password you use in the connect request must still be in plain text. For security reasons, you cannot use the obfuscated value as a password on connection.

See also: Digest in the section Oracle JDBC for Rdb Controller.

10.12.2 User Passwords

User passwords may be stored in the server configuration file, however storing these passwords in plain text form may leave your system vulnerable to anyone who can read the configuration file. To help improve security, user passwords may be stored in the configuration file in obfuscated form.

As user passwords must be passed to SQL and Oracle Rdb in their plain-text form, any obfuscation of these passwords must be reversible. The `obfuscate` command of the Controller may be used to create a reversible obfuscation of a password.

Example

Create an obfuscated password:

```
rdbthincontrol> obfuscate mypassword
obfuscation : ##016BA4158E5884C8D6EAFE71697D4DC9483417DA0BA1
```

The value can then be used in the configuration file where you would have normally provided a plain text user password.

```
<server
name="RdbThinSrv1701"
type="RdbThinSrv"
url="//localhost:1701/"
anonymous = "true"
User="jdbc_user"
Password="##016BA4158E5884C8D6EAFE71697D4DC9483417DA0BA1"
/>
```

This value must be copied exactly as returned by the `obfuscate` command.

The plain-text password conversion to its obfuscated form is case-sensitive, so the same word or phrase but with different character casing will produce a different obfuscated form.

Passwords are case sensitive so you must ensure that the value of the password used in plain text and in it digested form match exactly character by character including case.

This is particularly important if a password is used on the DCL command line. If double quotation characters are not used to surround the plain text password DCL may, depending on your environment, force the value to all lower case or all uppercase which may differ from the original.

The value of the obfuscated form of the password will change every time the obfuscate command is used:

```
rdbthincontrol> obfuscate mypassword
obfuscation : ##01114E48372901FAADFF86A79B1304CCBC9F51872FAF
rdbthincontrol> obfuscate mypassword
obfuscation : ##01329A04611A8C6DAC388BBA0DD369C20C2E4DFCB801
rdbthincontrol>
```

Note:

Obfuscated user passwords are only valid when used in conjunction with a session or server definition in a configuration file or as a server start up command line configuration option. Any user password used in a connection statement must be in plain text form.

See also: Obfuscate in the section Oracle JDBC for Rdb Controller.

Contents

10.13 Restricting Server, Database and Operational Access

In addition to the standard Rdb authorization checking that is carried out during the connection to a database using a thin server, the databases accessed, the operations attempted, and the usernames allowed may be restricted at the server level.

The following sub-sections detail how access to a thin server and its served databases may be intentionally restricted.

10.13.1 Restricting Database Access

You may restrict connections made via a server to only those databases specified as allowed databases.

This may be done by setting the `restrictAccess` property for the server in the configuration file and then providing a list of databases that may be accessed using `allowDatabase` subsections.

Examples

```
<server
name="srv2restrict"
type="RdbThinSrv"
url="//localhost:1701/"
restrictAccess="true">
<allowDatabase name="mf_pers"/>
<allowDatabase name="disk1:[databases]customers"/>
</server>
```

The name value of an `allowDatabase` subsection may be either the name of a database already declared within the same configuration file, or the database file specification portion of a connection URL.

If a client is using a server with restricted access, then the file specification portion of the JDBC Connection URL used must match one of the names within the allowed database subsections. No file expansions or logical name translations are done on the Connection URL before the server checks these names against the allowed databases, so it is important that, apart from the variations in case, the names be exactly as specified in the allowed database subsections.

If the server `restrictAccess` property is **true** and there is at least one `allowDatabase` subsection specified, then the server will allow access to only those databases specified.

If the server `restrictAccess` property is **false** or not specified or if no `allowDatabase` subsection is specified for the server then no database restrictions will be applied.

The `allowDatabase` subsections are also inherited from both the DEFAULT and, if appropriate, the DEFAULTSSL server definition.

Thus, the list of allowed databases will be the combination of the allowed database list present in the DEFAULT and the list within the specific server s definition. If the server uses SSL, the database list within DEFAULTSSL is also included.

Example 1

Given the above server description of a server running on the node **bravo**:

```
Connection conn = DriverManager.getConnection(
"jdbc:rdbThin://bravo:1701/mf_pers",user, pass);
```

will be allowed.

Example 2

```
Connection conn = DriverManager.getConnection(
"jdbc:rdbThin://bravo:1701/MF_Pers",user, pass);
```

will be allowed because character case in the database specification is not significant.

Example 3

```
Connection conn = DriverManager.getConnection(
"jdbc:rdbThin://bravo:1701/disk1:[databases]customers",user, pass);
```

will be allowed.

Example 4

```
Connection conn = DriverManager.getConnection(
"jdbc:rdbThin://bravo:1701/disk1:[databases]customers.rdb",
user, pass);
```

will NOT be allowed due to the extra `".rdb".`

Example 5

```
Connection conn = DriverManager.getConnection(
"jdbc:rdbThin://bravo:1701/cust",user, pass);
```

will NOT be allowed even though `cust` may be a logical name that translates to:

```
disk1:[databases]customers
```

10.13.2 Restricting User Access

When using a thin server, Rdb authorization checking is carried out during the connection to the database. Rdb will check the username and password provided to determine the authorization access for the given user.
In addition, you may further restrict which users may use the server by setting the `restrictAccess` property for the server in the configuration file and then providing a list of usernames that will be restricted using `allow` or `deny User` subsections.

Example

```
<server
name="srv2restrict"
type="RdbThinSrv"
url="//localhost:1701/"
restrictAccess="true">
<allow user="jdbc_user"/>
<allow user="APP.*"/>
<deny user="APP_M1"/>
</server>
```

The name value of an `allow` or `deny User` subsection must be a valid Rdb username, the keyword owner or a be a valid Java regular expression **.**

If a client is using a server with restricted access, the username used by the originating client request will be checked against the ***allow*** and/or ***deny*** lists to see if the server will accept the connection.

If the server `restrictAccess` property is **true** and there is at least one *`allow user`* or *`deny`* `user` subsection specified then the server will allow access to only those users that match any of the *`allow user`* values specified but do not match any `deny user` specified for that server.

If the server `restrictAccess` property is **true** and there are no *`allow user`* subsections specified but there is at least one *`deny`* `user` subsection specified, then the server will deny access to usernames that match any of the *`deny`* `users` specified for that server.

If the server `restrictAccess` property is **false** or not specified, or if no *`allow user`* or *`deny`* `user` subsections are specified for the server then no username restrictions will be applied.

The *`allow user`* and *`deny user`* subsections are also inherited from both the DEFAULT and, if appropriate, the DEFAULTSSL server definition. Thus, the list of usernames to be considered will be the combination of the username list present in the DEFAULT and the list within the specific server s definition. If the server uses SSL, the username list within DEFAULTSSL is also included.

As any username within the restriction lists may be a standard username value or a Java regular expression, the server determines the style of match to apply to each list entry by looking at the user pattern of that entry.

If the username pattern contains any standard Java regular expression control characters, other than asterisk (*), it will be considered a regular expression.

If the username is not a Java regular expression, username matching is not case-sensitive.

If you plan to use Java regular expression for matching usernames, it is important to note that prior to restriction checking, the username provided for the server connection attempt will have leading, trailing, and embedded space characters removed and remaining characters converted to lowercase.

Since release 7.3.4

The keyword `owner` informs JDBC to use the OpenVMS username of the account under which the server was started.

10.13.3 Restricting IP Access

Since release 7.3.2

You may restrict client access to a JDBC server by specifying sets of IP addresses that indicate who will be allowed or denied access to make a connection using the server.

This may be done by setting the `restrictAccess` property for the server in the configuration file and then providing one or more IPs using `allow` or `deny` subsections.

Example 1

```
<server
name="srv2restrict"
type="RdbThinSrv"
url="//localhost:1701/"
restrictAccess="true">
<allow IP = "192\.168\.1\.(1[0-9][0-9]|[1-9][0-9]|[0-9])"/>
<deny IP = "192.168.1.3"/>
</server>
```

The value of an `allow` or `deny IP` subsection may be either a numerical IPv4 address presented in the standard *dot-decimal* notation or may be a Java regular expression based on the same standard IPv4 *dot-decimal* notation.

Note:

IPv4 addresses are 32-bit values that are canonically represented in a dot-decimal notation, which consists of 4 decimal numbers (segments), each ranging from 0 through 255, separated by dots. For example: `192.168.1.3`.

If a client is using a server with restricted access, the IP of the originating client request will be checked against the ***allow*** and/or ***deny*** lists to see if the server will accept the connection.

If the server `restrictAccess` property is **true** and there is at least one *`allow IP`* or *`deny`* `IP` subsection specified then the server will allow access to only those IPs that match any of the *`allow IP`* values specified but do not match any `deny IP` specified for that server.

If the server `restrictAccess` property is **true** and there are no *`allow IP`* subsections specified but there is at least one *`deny`* `IP` subsection specified, then the server will deny access to IPs that match any of the *`deny`* `IPs` specified for that server.

If the server `restrictAccess` property is **false** or not specified, or if no *`allow IP`* or *`deny`* `IP` subsections are specified for the server then no IP restrictions will be applied.

The *`allow IP`* and *`deny IP`* subsections are also inherited from both the DEFAULT and, if appropriate, the DEFAULTSSL server definition. Thus, the list of IPs to be considered will be the combination of the IP list present in the DEFAULT and the list within the specific server s definition. If the server uses SSL, the IP list within DEFAULTSSL is also included.

As any IP within the restriction lists may be a full or partial IPv4 *dot-decimal* value or a Java regular expression, the server determines the style of match to apply to each list entry by looking at the IP pattern of that entry.

If the IP pattern contains any standard Java regular expression control characters, other than period (.) or asterisk (*), it will be considered a regular expression. Remember, if you are using a regular expression, you should escape any dot separators present in the IP pattern otherwise they will be used as character wildcards.

If only numeric characters and periods are found in the pattern, the IP will be matched using standard string equality matching.

Example 2

Take care when applying restrictions to client connections made from the same host as the JDBC server is running. You should provide a local network IP value:

```
<server
name="srv2restrict"
type="RdbThinSrv"
url="//192.168.1.1:1701/"
restrictAccess="true">
<allow IP = "192.168.1.1"/>
<allow IP = "127.0.0.1"/>
</server>
```

Only client applications running on the same node as the server will be allowed access.

Note:

If the client connection is made from the same node the server is running on, the network socket connection may be using the local network range `127.0.*.*`. This may depend on how your network has been setup.

If this is the case, you may have to provide both the local network IP and the full network IP in the *`allow`* or *`deny IP`* lists.

The IP masks used in the restriction lists are matched as simple strings, even when Java regular expressions are used. No translation of IPs or conversion to/from canonical form is done. Thus, in **Example 2** above, even though `"192.168.1.1" and "127.0.0.1"` may be two designations of the same network node they are considered discrete IP values when the JDBC server applies IP restriction testing. Hence, both designations should be provided in the restriction lists.

Example 3

Groups of IPs may be specified by providing partial IP values specifying from 1 to 3 leading IP segments. The missing segments will be considered as matching the full range of the segment values i.e., 0 through 255. In this way groups of IPS may be represented by a single partial IP value, for example:

```
<allow IP = "192.168.1" />
```

Tells the server to allow IP values 192.168.1.0 through 192.168.1.255.

Example 4

```
<allow IP = "192.0" />
```

Tells the server to allow IP values 192.0.0.0 through 192.0.255.255.

Example 5

Groups of IPs may be specified by providing from 1 to 3 leading IP segments and using asterisk wildcard characters, .* , in the remaining trailing segments. The wildcard segments will be considered as matching the full range of the segment values i.e., 0 through 255. In this way groups of IP values may be represented by a wildcard IP value, for example:

```
<allow IP = "192.3.*.*" />
```

Tells the server to allow IP values 192.3.0.0 through 192.3.255.255.

Example 6

Alternatively, IP groups may be specified by providing an appropriate Java regular expression:

```
<deny IP = "192\.0\.0\.(25[0-5]|2[0-4][0-9]|1[0-9][0-9]|[1-9][0-9]|[0-9])"/>
```

Tells the server to deny any IP in the range 192.0.0.0 through 192.0.0.255

Note:

IPv4 addresses may be subject to change during rerouting of connections through your TCP/IP network. Care should be taken to determine the appropriate IP address or address ranges that client applications may be connecting from, considering any IP translations or reassignments. Because IP addresses may be subject to change, Oracle recommends that you should not rely solely on IP addresses for server access restrictions. You should also consider, additionally using other restriction criteria such as user restrictions or server passwords to ensure tighter access security.

Server restrictions do not remove the need to ensure correct authorization and control access on each of the databases that may be accessed. Server restrictions are meant to enhance the security provided by the underlying standard Rdb database authorization and access control security measures and not replace them.

10.13.4 Privileged Users Access

Users may be granted a Privileged User status when accessing a JDBC server. Privileged users may be allowed to carry out operations on the server or the server s host that would not normally be granted, for example, access to the command line.

A user is designated privileged by having their username specified within an `allowPrivUser` configuration option for that server, for example:

Example

```
<server
name="srv2"
type="RdbThinSrv"
url="//localhost:1701/">
<allowPrivUser name="jdbc_user"/>
<allowPrivUser name="owner"/>
</server>
```

The name value of an `allowPrivUser` subsection must be a valid Rdb username or the keyword `owner` .

The keyword `owner` informs JDBC to accept the OpenVMS username of the account under which the server was started as a valid privileged user.

Normal OpenVMS authorization and privilege checking is still carried out on all operations executed by a privileged user. The username/password provided for the database connection will be used for authorization checking by OpenVMS.

Note:

Control users that have successfully connected to the server using a control connection are automatically considered Privileged Users .

10.13.5 Access to the Command Line

Users may be granted access to execute command line operations on the host that the server is executing on.

If the user has been granted access, the server may execute OpenVMS DCL commands on the server s host system on behalf of the user. The commands are executed in a separate loginout session established specifically for command line access.

During the execution of the DCL command, messages that are written to either SYS$OUTPUT or SYS$ERROR will be relayed back to the client application.

Enabling command line access for a server requires two server configuration options:

1. The server must have Command Line access enabled by having the server configuration option `allowAccessToCL` set to true . See Server Configuration Options for more details.
2. The user must be a designated Privileged User . See Privileged Users for more details.

The following example shows that both `jdbc_user` and `smith` will be allowed command line access when using `srv1`.

Example

```
<server
name = "srv1"
type = "RdbThinSrv"
url = "//localhost:1701/"
autoStart="true"
allowAccessToCL = "true">
<allowPrivUser name = "jdbc_user"/>
<allowPrivUser name = "smith"/>
</server>
```

Command line access is carried out by the server within a separate process using special command procedures. See CLI Startup Command Procedure for more details.

Command line access is only available when using certain applications such as the *SQLDeveloper Addin for Rdb* and is used to provide the ability to execute RMU and other operations required by the *SQLDeveloper* application.

This command line access feature is currently not available for general application use.

10.13.6 Access to the Server Root

Since release 7.3.4

Users may be granted access to the server root to carry out operations that do not require a bind to an Rdb database.

If the user has been granted access, the server may execute commands such as CREATE and DROP database on behalf of the user.

Server root access is enabled by setting the server configuration option `allowAccessToRoot` to either of the following values:

- **true** - allows all users access to server root.
- **priv** - allows only privilege users access to server root.

By default, server root access is disabled. Access may also be disabled by explicitly setting the `allowAccessToRoot` server configuration option to **false** .

If `allowAccessToRoot` is set to **priv** then the user must be a designated Privileged User before root access is allowed, see Privileged Users for more details.

Example

```
<server
name = "srv1"
type = "RdbThinSrv"
url = "//localhost:1701/"
allowAccessToRoot = "priv">
<allowPrivUser name = "jdbc_user"/>
<allowPrivUser name = "smith"/>
</server>
```

In this example only the users **jdbc_use**r and **smith** will be allowed access to the server root.

10.13.7 Create and Drop Database Entitlement

Since release 7.3.4

Users may be granted the ability to create or drop databases on the host that the server is executing on.

If the user has been granted access, the server may execute CREATE and DROP database commands on behalf of the user.

Enabling CREATE and DROP database access for a server requires the setting of the following server configuration options:

- `allowAccessToRoot`
- `allowCreateDatabase`
- `allowDropDatabase`

Normal Oracle Rdb and OpenVMS authorization and privilege checking are still carried out on all operations executed by the user after being granted create and drop database access by the JDBC server. The username/password provided for the server root connection will be used for authorization checking.

Note:
Control users that have successfully connected to the server using a control connection are automatically considered Privileged Users .

The following sections provide more detail on the granting of CREATE and DROP database access.

10.13.7.1 Create Database Entitlement

New database creation access is enabled by setting the server configuration option `allowCreateDatabase` to either of the following values:

- **true** - allows all users access to create databases.
- **priv** - allows only privileged users access to create databases.

In addition, access to the server root must also be enabled. Thus, for a user to be able to create new databases, the server must have the following configuration options set:

1. The server must have server root access enabled by having the server configuration option `allowAccessToRoot` set to either the value **true** or the value **priv** . See Access to the Server Root for more details.
2. The server must have database create access enabled by having the server configuration option `allowCreateDatabase` set to the value **true** or the value **priv** if the creation of new databases should be allowed.
3. If either server configuration option `allowAccessToRoot` or `allowCreateDatabase` is set to the value **priv** , then the user must be a designated Privileged User before being allowed to carry out the operation. See Privileged Users for more details.

Example

The following specifies that both **jdbc_user** and **smith** will be allowed to create new databases when using `srv1`:

```
<server
name = "srv1"
type = "RdbThinSrv"
url = "//localhost:1701/"
allowAccessToRoot = "true"
allowCreateDatabase = "priv">
<allowPrivUser name = "jdbc_user"/>
<allowPrivUser name = "smith"/>
</server>
```

Once granted access to the operation by the server, normal Oracle Rdb and OpenVMS authorization and privilege checking will still be carried out at the time of the command execution.

See the section Create Database for more information.

10.13.7.2 Drop Database Entitlement

The dropping of existing databases is enabled by setting the server configuration option `allowDropDatabase` to either of the following values:

- **true** - allows all users access to drop databases.
- **priv** - allows only privileged users access to drop databases.

In addition, access to the server root must also be enabled. Thus, for a user to be able to drop a database, the server must have the following configuration options set:

1. The server must have server root access enabled by having the server configuration option `allowAccessToRoot` set to either the value **true** or the value **priv** . See Access to the Server Root for more details.
2. The server must have database drop access enabled by having the server configuration option `allowDropDatabase` set to the value **true** or the value **priv** if the dropping of existing databases should be allowed.

3. If either server configuration option `allowAccessToRoot` or `allowDropDatabase` is set to the value **priv** , then the user must be a designated Privileged User before being allowed to carry out the operation. See Privileged Users for more details.

Example

The following specifies that both **jdbc_user** and **jones** will be allowed to use drop database when served by `srv1`:

```
<server
name = "srv1"
type = "RdbThinSrv"
url = "//localhost:1701/"
allowAccessToRoot = "priv"
allowDropDatabase = "priv">
<allowPrivUser name = "jdbc_user"/>
<allowPrivUser name = "jones"/>
</server>
```

Once granted access to the operation by the server, normal Oracle Rdb and OpenVMS authorization and privilege checking will still be carried out at the time of the command execution.

10.13.8 Further Server Access Protection

In addition to restricting the databases accessed and the users allowed to use the server, a server may also be protected using a server password.

This may be done by setting the `srv.password` property for the server in the configuration file. This password may be either a plain text password or an obfuscated password value.

Oracle recommends not to store passwords in your configuration file. However, if you choose to store them then an obfuscated form should be used. You may use the `digest` or `obfuscate` functions within the Controller application to generate an obfuscated password that is suitable to use with the `srv.password` property. See Password Obfuscation in Server Configuration Files for more details.

To make a successful connection to a database using a password-protected server, the client connection properties must also provide either the plain text value of the password or an obfuscated value produced using the Controller application `obfuscate` command on the client connection request.

The connection attribute `srv.password` is used to provide the password on the connection request. This attribute may be specified as part of the connection string or within the info properties list provided to the `DriverManager.getConnection` method.

Note:

If the plain text password contains the character **@** then you must use an obfuscated password as the **@** may cause problems with internal JDBC parsing.

Example

```
<server
name="srv2restrict"
type="RdbThinSrv"
url="//localhost:1701/"
srv.password="0x811B15F866179583EB3C96751585B843"
   />
```

In this example, an obfuscated password is used which matches the plain text password **jdbc_user** .

To connect to a database using this server the client must provide a `@srv.password` value on the connection request and the password can be a plain text password that matched the one specified for the server:

```
Connection conn = DriverManager.getConnection(
    "jdbc:rdbThin://bravo:1755/"+
"myDir:pers@srv.password=jdbc_user",user, pass);
```

Alternatively, and preferred, you may use an obfuscated password:

```
Connection conn = DriverManager.getConnection(
       "jdbc:rdbThin://bravo:1755/"+
           "myDir:pers@srv.password="+
"##013F5615A41C8E4384E7D6FEFE71696B4DCC",
user, pass);
```

10.13.9 Restricting SQL Statements

Since release 7.3.1

You may restrict SQL statements executed via a server to only those specified by providing a list of restricted verbs and/or by providing a denied SQL statement pattern.

This may be done by providing a list of SQL verbs that may be allowed using the server s `restrictSQL` attribute. Alternatively, or in addition to, you may provide a Java Regular Expression (REGEX) pattern in a `deny SQL` attribute that will be used to determine if the SQL statement should be denied.

The `restrictSQL` attribute takes a comma separated list of verbs that will be accepted by the server. If the SQL statement text starts with a verb that is not in this list it will be denied. This verb check is not case-sensitive.

The `deny SQL` attribute of the server is also used to determine if the SQL statement attempted will be allowed. If the SQL statement text matches the specified regular expression pattern it will be denied.

The `restrictSQL` attribute, if present, overrides any `restrictSQL` attribute that may be inherited from the default server definition.

However, the `deny SQL` attributes may also be inherited from the default server specification. The resultant denial list is the combination of the attributes from both the default server and the specific server definition.

Note:

A Java regular expression is case-sensitive by default. However, you may use the in-line modifier **`(?i)`** to make the pattern case-insensitive. See the Java documentation for more information on using Java regular expressions.

Example

```
<server
```

```
name="srv2restrict"
type="RdbThinSrv"
url="//localhost:1701/"
restrictSQL = "SELECT,SHOW"
>

<deny SQL = "(?i).*select.*jobs.*"/>
</server>
```

In the above example, allowable SQL statements must start with either **SELECT** or **SHOW** and cannot contain the combination of **SELECT** and **JOBS** , otherwise the SQL statement will be denied by the server. In this example the tests for verb restriction and statement denial are both case-insensitive.

The `restrictSQL` or `deny SQL` attributes of the server will be used to determine if the SQL statement will be allowed.

If no `restrictSQL` or `deny SQL` attributes are specified, then no SQL restrictions will be applied by the server.

Note:

Even if the server allows the SQL statement to proceed, the underlying database system may still restrict the statement depending on ACLs and other database specific criteria.

The `restrictSQL` or `deny SQL` attributes allow some level of checking to be placed on queries being processed by JDBC, however they are not a substitute for correct security procedures and controls being in-place on the underlying databases.

You should refer to the Oracle Rdb documentation on database security and restricting access to tables and contents.

Contents

10.14 Scope of CONNECTION.setReadOnly()

By default, the scope of the `CONNECTION.setReadOnly()` method is session. That is, if the method `CONNECTION.setReadOnly(true)` is called, the default transactions for the rest of the connected session will be `READ_ONLY` unless changed by another call to `CONNECTION.setReadOnly()`.

However, the standard Oracle JDBC Drivers have a different scope for `CONNECTION.setReadOnly()`. If the method `CONNECTION.setReadOnly(true)` is called, only the next transaction will be `READ ONLY`. Once that transaction has ended, the default transaction will resort back to `READ WRITE`.

To provide semantics consistent with the standard Oracle JDBC Drivers, a value of ORACLE may be specified within the `TRANSACTION` connection option.

Format

```
@transaction=oracle
```

The default transaction will be `READ_WRITE` when this switch is used, but this transaction type may be changed by issuing the `CONNECTION.setReadOnly(true)` method call. This will set **only** the next transaction to `READ_ONLY`.

Contents

10.15 Server Command Procedures

OpenVMS DCL command procedures are used in the creation of processes in which a thin server is started using the controller and when a Multi-process server starts up an executor process. A command procedure is also used whenever the server must execute a CLI statement on behalf of a client.

These command procedures may be tailored for your system environment so that operation such as software version setup and re-direction of output may be customized.

There are three command procedures used for startup.

1. The server startup command procedure:

```
   rdb$jdbc_home:rdbjdbc_startsrv.com
```

2. The executor startup command procedure:

```
rdb$jdbc_home:rdbjdbc_startexec.com
```

3. The CLI startup command procedure:

```
rdb$jdbc_com:rdbjdbc_execcli.com
```

Caution:

Do not use the SET VERIFY command within these command procedures, as the method `Runtime.exec()` may be used by the servers to create processes. The use of the SET VERIFY command within the command procedure may hang the server. This is a documented limitation of using `Runtime.exec()` on Open VMS Java. The logical name JAVA$EXEC_TRACE is available to help debug `Runtime.exec()` calls on OpenVMS. Refer to the OpenVMS Java documentation for more details.

Note:

If the only changes required are environmental setup, Oracle recommends that instead of altering the start-up command procedures, the server attribute `srv.onStartCmd`, `srv.onExecStartCmd` or `srv.onCliStartCmd` should be considered. See On Start Commands for more details.

10.15.1 Server Startup Command Procedure

The controller uses the server startup command procedure to start a thin server.

The srv.startup option within the server section of an XML-formatted configuration file may be used to specify the file specification of the command procedure that should be used to start that server.

Example

```
<server
name="srv2forRdb"
type="RdbThinSrv"
url="//localhost:1708/"
autoStart="true"
logfile="rdb$jdbc_logs:srv2forRdb.log"
srv.startup="rdb$jdbc_com:our_customized_startsrv.com"
/>
```

During the driver kit installation, the command procedure rdbjdbc_startsrv.com is placed in the rdb$jdbc_home directory. This file will be used by default for server start up using the controller and Pool servers.

The DEFAULT server provided in the default configuration file RDBJDBCCFG.XML specifies this command procedure:

```
srv.startup="rdb$jdbc_home:rdbjdbc_startsrv.com"
```

You can choose to change this default command procedure to customize for your system settings, or you can create a new customized procedure and change the configuration file so that servers use this new file. However, Oracle recommends that you use the srv.onStartCmd server attribute instead. See srv.onStartCmd for more information.

Caution:

Do not use the SET VERIFY command within these command procedures, as the method Runtime.exec() may be used by the servers to create processes. The use of the SET VERIFY command within the command procedure may hang the server. This is a documented limitation of using Runtime.exec() on Open VMS Java. The logical name JAVA$EXEC_TRACE is available to help debug Runtime.exec() calls on OpenVMS. Refer to the OpenVMS Java documentation for more details.

Note:

The controller or Pool server only use the server startup command procedure to start a thin server. If the server is started by any other means neither the server startup command procedure nor any commands in the srv.onStartCmd server attribute will be executed.

10.15.2 Executor Startup Command Procedure

The Multi-process server uses the executor startup command procedure to start an executor process for a client connection.

You can use the srv.execStartup option to specify the file specification of the command procedure that should be used to start executors by a Multi-process server.

Example

```
<server
name="MPsrv2forRdb"
type="RdbThinSrvMP"
url="//localhost:1788/"
srv.execStartup="rdb$jdbc_com:our_customized_startexec.com"
/>
```

You can choose to change this default command procedure to customize for your system settings, or you can create a new customized procedure and change the configuration file so that servers use this new file. However, Oracle recommends that you use the `srv.onExecStartCmd` server attribute instead. See `srv.onExecStartCmd` for more information.

Caution:

Do not use the SET VERIFY command within these command procedures, as the method `Runtime.exec()` may be used by the servers to create processes. The use of the SET VERIFY command within the command procedure may hang the server. This is a documented limitation of using `Runtime.exec()` on Open VMS Java. The logical name JAVA$EXEC_TRACE is available to help debug `Runtime.exec()` calls on OpenVMS. Refer to the OpenVMS Java documentation for more details.

The `srv.execStartup` and `srv.onExecStartCmd` options are only valid within the XML-Formatted configuration file server section for a Multi-process server.

10.15.3 CLI Startup Command Procedure

The JDBC server uses the CLI startup command procedure to execute any CLI statements it is required to issue on behalf of a client.

If the server attribute `allowAccessToCL` is set to **true** , clients may issue CLI statements to execute OpenVMS DCL commands in the context of the running server.

You can use the `srv.cliStartup` option to specify the file specification of the command procedure that should be used to execute the CLI commands.

Example

```
<server
name="Srv2forRdb"
type="RdbThinSrv"
url="//localhost:1777/"
srv.cliStartup="rdb$jdbc_com:our_customized_cli.com"
/>
```

You can choose to change this default command procedure to customize for your system settings, or you can create a new customized procedure and change the configuration file so that servers use this new file. However, Oracle recommends that you use the `srv.onCliStartCmd` server attribute instead. See `srv.onCliStartCmd` for more information.

Caution:

Do not use the SET VERIFY command within these command procedures, as the method `Runtime.exec()` may be used by the servers to create processes. The use of the SET VERIFY command within the command procedure may hang the server. This is a documented limitation of using `Runtime.exec()` on Open VMS Java. The logical name JAVA$EXEC_TRACE is available to help debug `Runtime.exec()` calls on OpenVMS. Refer to the OpenVMS Java documentation for more details.

The `srv.execStartup` and srv.onExecStartCmd options are only valid within the XML-Formatted configuration file server section for a Multi-process server.

Contents

10.16 Server/Client Protocol Checking

To ensure that the protocol between the Oracle JDBC for Rdb thin driver and servers correctly align, the Oracle JDBC for Rdb servers check versioning information transmitted by the client. This allows the quick trapping of problems that may occur because of a mismatch between the server instance and the thin driver.

Example

The following is an example of the type of error message that will be seen if the client and server mismatch:

```
oracle.rdb.jdbc.common.RdbException: Io exception :
Io exception : Server Protocol error : received 1 :
expected 2 @rdb.Client.FetchBlobSeg
```

To prevent these protocol errors, all the Oracle JDBC for Rdb driver JAR files should be replaced at the same time whenever a new kit is installed.

To check that the server and client instances match, enable `@tracelevel=-1` on the connection URL for your client application. See Trace for more details.

Near the start of the log there will be messages indicating the instance values for both the client and the server. If these two numbers do not match, then protocol errors are likely.

An example of the log messages showing Instance information:

```
>> main ThinConnect@3.setTraceLevel msg : Rdb nativeInstance=20030508
>> main ThinConnect@3.setTraceLevel msg : Rdb serverInstance=20030508
```

Contents

10.17 Using OpenVMS failSAFE IP

OpenVMS failSAFE IP may be used in conjunction with Oracle JDBC for Rdb thin driver and servers. During failover, failSAFE IP will redirect the existing Oracle JDBC for Rdb client/server IP connections to the standby service.

If the failover service exists on the same node as the failed service, the connections should continue to be viable transparently.

If, however, the failover service is on another node, then as Rdb connections cannot be transferred between processes, the failover will not be transparent. The thin driver should receive a socket exception on the failed TCP/IP port, as the original service is no longer available.

Note that server socket exceptions will only be raised on a connection if there is a network read or write outstanding. If the driver is currently idle and not carrying out a read or write on the socket to the server, no exception will be raised. Subsequent operations on that connection by the driver will however raise the socket exception.

The socket exception will be passed through to the application wrapped in an SQLException. It is then up to the application to catch the exception and to clean up its environment and, if applicable, establish a new Connection to the driver.

Depending on where the client is running it is possible that the client operating system may not raise a `SocketException` even if a read or write is pending. On these systems it is possible for the client connection to be held in limbo waiting for a read or write to complete.

To help reduce the impact of possible hangs due to the failure of the socket subsystem to raise the correct socket exception, a timeout may be placed on network read/writes. If the read/write does not complete within the designated time an exception will be raised.

Care should be taken in setting this timeout value as longer-duration database operations such as statement compilation may delay the server sending back its results.

The client-side will have a socket read waiting on the return of the results, which could timeout if this duration is set too short in relation to the performance of your system and database software. Oracle recommends that, if used, this timeout value should be set to a large value (in the order of several minutes) if you suspect that query operations on the server side may take some time.
See `networkTimeout` in Connection Options for more details on network read and write timeout.

Contents

10.18 Attaching to Multiple Databases in the Same Connection

Oracle Rdb allows the application programmer to attach to multiple databases using the same connection context. Starting with Version 7.3 of Oracle JDBC for Rdb, this feature is also available to developers using the Oracle JDBC for Rdb drivers.

Both the Native and Thin Driver classes have Oracle Rdb-specific extensions allowing the developer to attach more databases to an existing `Connection`.

Example

```
.
.
.
String dbUrl = "jdbc:rdbThin://localhost:1701/db_dir:DB_one";
String user = "jdbc_user";
String pwd = "jdbc_user";
Connection dbConnection =
DriverManager.getConnection(dbUrl, user, pwd);
dbConnection.setAutoCommit(false);
oracle.rdb.jdbc.rdbThin.Driver d =
(oracle.rdb.jdbc.rdbThin.Driver)DriverManager.getDriver(
dbUrl);

dbConnection2 = d.attach("db_dir:DBtwo","MYDB2",
```

```
user, pwd, dbConnection);
Statement s1 = dbConnection.createStatement();

// first declare a transaction

s1.execute("declare transaction read only");

.
.
.
s1.execute("commit");
.
.
.
dbConnection2.close();
dbConnection.close();
```

The `attach()` methods in the Oracle JDBC for Rdb drivers allow the developer to attach another database to an existing Connection. The following conditions should be noted:

- Autocommit must be disabled prior to attaching a second database and should remain disabled until the connections have been closed.
- Each attach must provide a unique `alias` for Oracle Rdb to use.
- Transactions must be handled manually:
  - Transactions must be declared or set manually prior to executing any SQL statement.
  - Transactions must be finalized manually prior to disconnecting from the databases.
- Rdb transaction handling when multiple databases are attached is more complex than single database connections. See the Oracle Rdb documentation for other limitations and conditions applying to multiple database attaches.

See attach() Public Method in the Driver class for more information on `attach()`.

Contents

10.19 Shutdown Thread

During the normal shutdown of a multi-threaded application on OpenVMS, the shutdown will wait on all children threads of the application to terminate before the shutdown will be finalized.

If for any reason a thread remains running, the shutdown of the application will stall indefinitely until the thread terminates.

Normal Java class finalizers are not run during application shutdown and so class finalizer cannot be used to ensure that subordinate threads are correctly terminated during shutdown.

When the Multi-process option is used with the JDBC Native Driver, each connection made by the user application will create its own thread to execute under. This thread is correctly terminated when the connection Disconnect is called.

If one or more application connections are not closed prior to shutting down the application, the application will hang during shutdown waiting for these connections to terminate. Thus, the developer should ensure that all connections made using the Multi-process Native Driver during the application are correctly disconnected prior to terminating the application.

Starting with release 7.3 the Oracle JDBC for Rdb drivers will create a shutdown thread when they are initially invoked. The purpose of this thread is to use the shutdown-hook feature provided by OpenVMS that will execute the thread at shutdown. The shutdown thread will ensure that any connection left open by the application will be correctly disconnected prior to the application shutdown proceeding, thus preventing application hangs at shutdown.

The application developer does not need to change any code for this shutdown feature to be enabled. If the application is using release 7.3 (or later) JDBC driver libraries the shutdown hook will be in place.

Contents

10.20 Getting a List of Known Databases from the Server

Databases known to servers may be listed within the databases section of the server configuration file.

This list of known databases is made available to the client application utilizing the Oracle JDBC for Rdb thin driver by using either:

- Show Databases SQL statement or the
- getDatabases() method

A list of known databases will be returned to the caller only if the server configuration option `allowShowDatabases` has been set to **true** for the connected server. See Server Configuration Options for more details on this option.

10.20.1 Show Databases SQL Statement

The Oracle JDBC for Rdb thin driver will allow the following SQL syntax extension in the SQL text of a Statement or PreparedStatement:

```
SHOW DATABASES
```

This syntax is specific to the Oracle JDBC for Rdb thin driver and is not passed through to the underlying database system. On recognizing this SQL statement, the driver will create a ResultSet that will contain a list of databases known to the connected thin server.

Example

```
.
.
.
Connection conn = DriverManager.getConnection(dbUrl,user,pwd );

Statement sc = conn.createStatement();
ResultSet rs = sc.executeQuery("show databases");
while (rs.next())
{
System.out.println(
rs.getString("RDB$DATABASE_NAME") + " : " +
rs.getString("RDB$DESCRIPTION"));
}
rs.close();
sc.close();
```

```
conn.close();
.
.
.
```

See Extended SQL Syntax SHOW DATABASES for more information.

10.20.2 getDatabases()

The Oracle JDBC for Rdb thin driver has a JDBC extension method, `getDatabases()` that may be used to return a list of databases known to a server.

Unlike the `SHOW DATABASES` SQL statement, the `getDatabases()` method does not require a connection to a database prior to being called. The `getDatabases()` takes a single parameter, the URL to the server that will be interrogated.

Example

```
.
.
.
Hashtable h = oracle.rdb.jdbc.rdbThin.Driver.getDatabases(
"//localhost:1701/");

if (h != null)
{
Enumeration e = h.keys();
while (e.hasMoreElements())
{
String key =(String)e.nextElement();
log( key + " : " + h.get(key));
}
}
.
.
.
```

See the Driver class extension getDatabases() Public Static Method for more information.

Contents

10.21 Create and Drop Database

Since release 7.3.4

The Oracle JDBC for Rdb drivers and servers allow the creation of new Rdb databases and the dropping of existing Rdb databases.

A special JDBC connection is required for CREATE/DROP database operations, this connection, called a ***server root*** connection, is made to the JDBC server itself, rather than to a database instance.

Rdb databases may be created or dropped by executing a JDBC Statement against the server root connection. The JDBC Statement SQL text may contain the standard Oracle Rdb SQL create or drop database syntax. As with other SQL statements when using the JDBC drivers, all syntactic keywords must not be abbreviated.

10.21.1 Server Root

Normally, JDBC thin connections are made to database instances using the JDBC server. That is, you must provide the details of both the server instance and the database file specification when specifying the connection URL.

However, when creating or dropping databases, the connection needs to be made to the server itself rather than having the server bind to an existing database. To facilitate this, the JDBC thin drivers will allow connections to be made directly to a JDBC server; the term ***server root*** is used for such a connection.

To connect to the server root, instead of providing a database file specification in the connection URL you must use the following URL clause:

```
@ROOT=TRUE
```

Example

```
Properties info = new Properties();
info.put("user", user);
info.put("password", pw);
info.put("tracelevel", -1);
String connStr = "//mynode:1701/@root=true";
conn = DriverManager.getConnection (connStr, info);
```

You must still provide a valid username and password to the **getConnection()** method, this information will be used by the server to:

- validate that you have access to the server root.
- be used in the underlying SQL CREATE/DROP statements sent to Rdb.

10.21.2 Server Configuration Requirements

By default, JDBC servers will not allow access to server root. To allow user access you must enable root access in the server configuration section of the XML-based configuration file used by the server.

The server configuration option **allowAccessToRoot** is used to control root access. If the server s **allowAccessToRoot** option is not specified it will be inherited from default server and if not specified for the default server, access will be denied.

See Access to the Server Root and Create and Drop Database Entitlement for more details.

10.21.3 Create Database

The create database operation is controlled by the connected server as well as Rdb and OpenVMS. In addition to having server root access, you will require **create database** access to be granted by the server. The username used in the server root connection will need to have the appropriate Rdb and OpenVMS authorizations necessary for database creation. See Create and Drop Database Entitlement for more information on create database entitlement and see the Oracle Rdb documentation for **create database** authorization requirements.

Once connected to the server root, and you have been allowed CREATE database access, new databases may be created by executing JDBC Statements containing valid Rdb SQL **create database** syntax.

Example 1

```
Statement stmt = conn.createStatement();
stmt.execute("create database filename 'my_db'");
```

Full file specifications may be used when declaring the database filename. If a directory or device specification is not provided then the database will be created relative to the default directory of the connected server if using the JDBC thin driver, or relative to your current default directory if using the JDBC native driver.

You may provide an ALIAS for the database in the create statement, however the alias value **RDB$DBHANDLE** is specifically disallowed and JDBC will throw an exception if this value is used.

You may also provide a username and password within the SQL text to establish the ownership of the database.

Example 2

```
Statement stmt = conn.createStatement();
stmt.execute(
"create database filename 'my_db' " +
"user 'my_username' using 'my_password' ");
```

If you do not provide a username directly in the SQL text, the username and password provided in the server root connection will be used.

10.21.3.1 Overwriting Existing Databases

As OpenVMS allows multiple versions of files, by default, the execution of the *create database* statement will create a new version of the database if one already exists for the file specification provided. JDBC will not throw an exception if the database already exists.

You may specify that JDBC should only create the database if it does not already exist. In addition to the standard Oracle Rdb SQL syntax, JDBC will also accept the following clause:

```
   IF NOT EXISTS
```

Example

```
Statement stmt = conn.createStatement();
stmt.execute(
"create database filename 'my_db' if not exists");
```

If a database already exists matching the provided file specification, the create statement will be silently ignored. This clause may be inserted anywhere after the **database** keyword**;**

Examples

```
"create database filename 'my_db' if not exists"
"create database if not exists filename 'my_db'"
"create database alias 'mydb if not exists filename 'my_db'"
```

10.21.4 Drop Database

The drop database operation is controlled by the connected server as well as Rdb and OpenVMS. In addition to having server root access, you will require **drop database** access to be granted by the server. The username used will need to have the appropriate Rdb and OpenVMS authorizations necessary for database creation.

See Create and Drop Database Entitlement for more information on drop database entitlement and see the Oracle Rdb documentation for **drop database** authorization requirements.

Once connected to the server root, and you have been allowed **drop database** access, existing databases may be dropped by executing JDBC Statements containing valid Rdb SQL **drop database** syntax.

Example

```
Statement stmt = conn.createStatement();
stmt.execute("drop database filename 'my_db'");
```

Full file specifications may be used when declaring the database filename. If a directory or device specification is not provided then the database will be searched for relative to the default directory of the connected server if using the JDBC thin driver, or relative to your current default directory if using the JDBC native driver.

Note:

Rdb will not allow a database to be dropped if there are any users currently bound to the database. You should ensure that no active connections are currently bound to the database prior to issuing the drop database statement.

10.21.4.1 If Exists Clause

If you issue a drop database on a database file specification for a file that does not exist, Rdb will raise an exception.

To prevent an exception being raised, in addition to the standard Oracle Rdb SQL syntax you may use the following clause:

```
   IF EXISTS
```

If a database does not exist matching the provided file specification, the drop statement will be silently ignored. This clause may be inserted anywhere after the **database** keyword**:**

Examples

```
"drop database filename 'my_db' if exists"
"drop database if exists filename 'my_db'"
```

Contents

10.22 Trace

Trace provides tracing of method calls and other debug information within the Oracle JDBC for Rdb drivers and servers. See Trace Values for valid trace level values.

The trace level value may be a signed decimal or a Java-style hexadecimal literal.

By default, trace output is written to the normal JDBC `DriverManager.PrintWriter`. You can override the default by using one of the following settings:

- `rdb.Debug.setLogStream(PrintStream ps)`
- `rdb.Debug.setLogWriter(PrintWriter pw)`

Example

The following example shows how to override the default:

```
rdb.Debug.setLogStream(new PrintStream(
new FileOutputStream("mylog.log")));
```

If trace is enabled and the `DriverManager.PrintWriter` is not currently defined a `PrintWriter` for `System.out` is defined for you.

10.22.1 Setting Tracelevel

Trace of JDBC operations may be enabled using one of the following methods:

- tracelevel property
- tracelevel switch
- tracelevel option
- Doracle.rdb.jdbc.tracelevel system option
- Set tracelevel

Details of these methods can be found in the following sections.

10.22.1.1 Tracelevel Property

Tracing can be enabled by setting the `tracelevel` property of the `Properties` passed to the `DriverManager.getConnection` method to the appropriate value.

Example

The following example shows how to set tracelevel to -1:

```
Properties info = new Properties();
info.put("user", user);
info.put("password", pw);
info.put("tracelevel", -1);
conn = DriverManager.getConnection (connStr, info);
```

See Connection Options for more details.

10.22.1.2 Tracelevel Switch

Using the `tracelevel` switch when starting a server can enable tracing:

Example

```
$ java -jar rdb$jdbc_home:rdbthinsrv.jar -cfg thinsrv.cfg -tracelevel -1
```

See Starting a Thin Server from the Command Line, Starting a Multi-process Server from the Command Line and Starting a Pool Server from the Command Line for more details.

10.22.1.3 Tracelevel Option

Placing the `tracelevel` option in the server definition within an XML-Formatted configuration file can enable tracing.

Example

```
<server
name="mypoolserver"
type="RdbThinSrvPool"
traceLevel="-1"
url="//localhost:1702/" >
<pooledServer name="srv1forRdb"/>
<pooledServer name="srv2forRdb"/>
<pooledServer name="srvMPforRdb"/>
</server>
```

See Server Configuration for more details.

10.22.1.4 Tracelevel System Property

Using the Rdb system property `Doracle.rdb.jdbc.tracelevel` when invoking your application or Rdb server can enable tracing.

Example

```
$ java Doracle.rdb.jdbc.tracelevel=-1 my_application
```

See Oracle JDBC for Rdb System Properties for more details.

10.22.1.5 Set Tracelevel Statement

Using the `SET TRACELEVEL` command in the controller can enable tracing.

Example

```
$ java -jar rdb$jdbc_home:rdbthincontrol.jar
rdbthincontrol> connect //localhost:1701/ jones mypassword
rdbthincontrol> set tracelevel 1
```

See Controller Command Line for more details.

10.22.2 Abbreviated form of Tracelevel

The abbreviated form for the traceLevel keyword, "**tl**", may also be used in the same manner.

10.22.3 Trace Values

The value passed to trace is a 32-bit flag mask. Each bit set determines what will be traced, as shown in the following table.

Table 10-2 Trace Values

Bit	Hexadecimal Value	Decimal Value	Traces
0	0x00000001	1	Standard JDBC methods.
1	0x00000002	2	Standard JDBC class create/finalize.
2	0x00000004	4	SQL statements.
4	0x00000010	16	Non-standard JDBC methods.
5	0x00000020	32	Non-standard JDBC class create/finalize.
6	0x00000040	64	Garbage collection.
7	0x00000080	128	SQL statement cache information.
8	0x00000100	256	Rdb JNI calls.
9	0x00000200	512	Network sends.
10	0x00000400	1024	Server actions.
11	0x00000800	2048	Performance information.
12	0x00001000	4096	Trace create/release of handles.
14	0x00004000	16384	Dump SQLDA information.
29	0x20000000	536870912	Memory information.
30	0x40000000	1073741824	Full provides more details on certain flags.
(ALL)	0xFFFFFFFF	-1	Trace everything.

Contents

10.23 File and Directory Access Requirements

There are certain file and directory access requirements that must be met to successfully use Oracle JDBC for Rdb servers, drivers, and the controller.

The controller and servers require access to the directories pointed to by the following logical names:

- `RDB$JDBC_HOME`
- `RDB$JDBC_COM`
- `RDB$JDBC_LOGS`

During installation a command procedure will be created for you that can be used to set up these logical names pointing to the installation directory. It is your decision whether to add these logical names to your startup

command procedure or require some other mechanism such as a login setup command procedure to set these up for JDBC users.

The logical names may be placed in any of your logical name tables. The normal OpenVMS logical translation precedence will be followed when any of the JDBC components try to access files using these logical names. This allows you to have system-wide, group level or private copies of the JDBC kits each using their own set of directories.

It is important that the appropriate access be granted to users that require to startup servers or use the JDBC JAR files. During installation the three directories will be created under the installation directory and be given the following protection.

```
(S:RWE, O:RWE, G:RE, W:RE)
```

This allows the world read/execute access to all the directories and contents. If this does not comply with your organizational requirements, then you should alter these protections appropriately.

Users of the controller, or those that startup servers manually will also require WRITE access to both the RDB$JDBC_COM and RDB$JDBC_LOGS directory to successfully startup servers, as the server process needs to be able to write log and temporary files to these directories.

If a server is started up using the SQL/Services JDBC dispatcher, then the account under which the dispatcher runs needs WRITE access to these directories.
If you redirect these logical names to other directories, you must ensure that the file and directory protections comply with the above requirements.

If persona is used with servers, then you must ensure that the persona has the appropriate access rights as described above.

Contents

Chapter 11
JDBC Extensions for Oracle Rdb

The following sections provide information on features that are extensions to the JDBC standard provided by Oracle JDBC for Rdb.

The following Oracle JDBC for Rdb classes have been enhanced:

- Blob Class
- Driver Class
- ResultSet Class

In addition to enhancements made to classes, developers using Oracle JDBC for Rdb drivers may use extended SQL syntax within Statement and PreparedStatement SQL text:

- SET
- SHOW DATABASES

Contents

11.1 Blob Class

Class

```
oracle.rdb.jdbc.common.Blob
```

An additional public method has been added to Blob:

- setSegSeparator()

Note:

The maximum size of a blob segment supported by Oracle Rdb today is 65535. The Oracle JDBC for Rdb drivers will correctly handle segments up to this maximum size.

There is no limit on the number of segments that can be stored for a single Blob, however, as the drivers materialize the blob into internal byte arrays. The correct handling of very large blobs in this version of the Oracle JDBC for Rdb drivers is limited to the free memory that is available to the Java environment.

11.1.1 setSegSeparator() Public Method

Declaration

```
// Additional method
```

```
public void setSegSeparator(java.lang.String separator)
```

Parameters

- *separator*
  The separator string to use between segments. A null or empty string will clear the separator value.

Remarks

To enable limited formatting of data returned from Oracle Rdb segmented strings, an additional public method has been added to `oracle.rdb.jdbc.common.Blob` that allows the specification of a separator string value to be inserted between segments when the segmented string is converted to a JDBC blob object.

The separator can be cleared by passing either a null object or empty String as the parameter to `setSegSeparator()`.

Example

The following code segment shows how to add a newline break between segments:

```
.
.
.
import oracle.rdb.jdbc.common.Blob;
.
.
.
ResultSet rs = s.executeQuery(
   "select resume from resumes where employee_id = '00164'");
rs.next();
Blob bl = (Blob)rs.getBlob(1);
bl.setSegSeparator("\n");
byte[] bytes = bl.getBytes(1,9999);
String st1 = new String(bytes);
System.out.println("resume : " + st1 );
.
.
.
```

Contents

11.2 Driver Class

Class

```
oracle.rdb.jdbc.rdbThin.Driver
```

or

```
oracle.rdb.jdbc.rdbNative.Driver
```

An additional public method has been added to both Oracle JDBC for Rdb driver classes:

- attach() Public Method (overloaded)

An additional public static method has been added to the Oracle JDBC for Rdb thin driver class:

- getDatabase() Public Static Method

11.2.1 attach() Public Method

The attach() method allows another database to be attached within the same Connection context. Databases that are attached within the same connection may take part in the same SQL compound statement.

Overload List:

- attach(String, java.util.Properties. java.sql.Connection)
  Attach the database using the provided properties to the given connection.
- attach(String, String, String, String, java.sql.Connection)
  Attach the database using the provided alias, username, and password to the given connection.

11.2.1.1 attach(String, java.util.Properties, java.sql.Connection)

Declaration

```
// Additional method
public void attach(
String url,
java.util.Properties info,
java.sql.Connection parent )
```

Parameters

- *url*
  The URL of the database to attach.
- *info*
  Properties for the new database attach.
- *parent*
  The parent connection the database should be attached to.

Remarks

The `url` string must contain the specification of the database to attach to. Any driver prefix, for example, **jdbc:rdbThin:** , node and/or port supplied in the `url` string will be disregarded.

See Oracle Rdb Database URL Specification Used with the Oracle JDBC for Rdb native driver and Oracle Rdb Database URL Specification Used with the Oracle Rdb thin driver for more information on specifying a URL.

The `info` properties object should contain at least the alias, username and password used for the database attach. See Connection Options for more details on using the connection properties object.

In addition:

- Autocommit must be disabled prior to attaching a second database and should remain disabled until the connections have been closed.
- Each attach must provide a unique `alias` for Oracle Rdb to use.
- See the Oracle Rdb documentation for other limitations and conditions applying to multiple database attaches.

Example

```
.
.
.
String dbUrl = "jdbc:rdbThin://localhost:1701/db_dir:DB_one";
Properties info = new Properties();
info.setProperty("user", "jdbc_user");
info.setProperty("password", "jdbc_user");

Connection dbConnection = DriverManager.getConnection(
  dbUrl, info);
info.setProperty("alias", "MYDB2");

dbConnection.setAutoCommit(false);
oracle.rdb.jdbc.rdbThin.Driver d =
  (oracle.rdb.jdbc.rdbThin.Driver)DriverManager.getDriver(dbUrl);

dbConnection2 = d.attach("db_dir:DBtwo",info, dbConnection);
.
.
.
```

11.2.1.2 attach(String, String, String, String, java.sql.Connection)

Declaration

```
// Additional method
public void attach(
  String url,
  String alias,
  String username,
  String password,
  oracle.rdb.jdbc.common.Connection parent)
```

Parameters

- *url*
  The URL of the database to attach.
- *alias*
  The alias to use for the database.
- *username*
  The username to use to attach.
- *password*
  The password to use to attach.
- *parent*
  The parent connection the database should be attached to.

Remarks

The `url` string must contain the specification of the database to attach to. Any driver prefix, for example **jdbc:rdbThin:** , server and/or port supplied in the `url` string will be disregarded.

See Oracle Rdb Database URL Specification Used with the Oracle JDBC for Rdb native driver and Oracle Rdb Database URL Specification Used with the Oracle Rdb thin driver for more information on specifying a URL.

In addition:

- Autocommit must be disabled prior to attaching a second database and should remain disabled until the connections have been closed.
- Each attach must provide a unique `alias` for Oracle Rdb to use.
- See the Oracle Rdb documentation for other limitations and conditions applying to multiple database attaches.

Example

```
.
.
.
String dbUrl = "jdbc:rdbThin://localhost:1701/db_dir:DB_one";
String user = "jdbc_user";
String pwd = "jdbc_user";
Connection dbConnection = DriverManager.getConnection(
  dbUrl, user, pwd);
dbConnection.setAutoCommit(false);
oracle.rdb.jdbc.rdbThin.Driver d =
  (oracle.rdb.jdbc.rdbThin.Driver)DriverManager.getDriver(dbUrl);

dbConnection2 = d.attach("db_dir:DBtwo","MYDB2",
user, pwd, dbConnection);
.
.
.
```

11.2.2 getDatabases() Public Static Method

The Oracle JDBC for Rdb thin driver static method getDatabases() returns a list of databases known to the specified thin server.

Declaration

```
// Additional method
public static Hashtable getDatabases(
String serverUrl
)
```

Parameters

- *`serverUrl`* a string containing a partial connection URL containing just the node and port components of the server using the format: **//<node>:<port>/**

This static method is available only in the Oracle JDBC for Rdb thin driver. It returns a Hashtable containing the list of databases known to the specified server.

Each key in this Hashtable contains the name of a database as found in the Databases section of the configuration file used during the server's invocation. The value associated with the Hashtable key contains the description of the database.

Example

```
.
.
.
Hashtable h = oracle.rdb.jdbc.rdbThin.Driver.getDatabases("//localhost:1701/");
if (h != null)
{
```

```
  Enumeration e = h.keys();
  while (e.hasMoreElements())
  {
    String key =(String)e.nextElement();
    log( key + " : " + h.get(key));
  }
}
.
.
.
```

Contents

11.3 ResultSet Class

Class

```
oracle.rdb.jdbc.common.ResultSet
```

A semantic enhancement has been made to an existing public method:

- getBytes() all overloaded methods.

11.3.1 getBytes() Public Method

The JDBC standard limits the use of all the overloaded `getBytes()` methods for access to `BINARY`, `VARBINARY` and `LONGVARBINARY` data only. The Oracle JDBC for Rdb drivers relax this limitation and will attempt to return byte arrays for all valid SQL data types using these methods.

Using getBytes() on:

- `CHAR` and `VARCHAR` columns will return the raw data as returned by Rdb to the driver.
- Numeric columns will be returned in their Rdb native format as a big-endian array of bytes.
- `DATE` and `TIME` will be returned as 64-bit big-endian array of bytes.

Contents

11.4 Extended SQL Syntax - SET

In addition to the standard SQL SET statements allowable in dynamic SQL, the Oracle JDBC for Rdb drivers will recognize driver specific SET statements as specified below.

Format

SET TRACELEVEL <trace_level>	Sets the trace level, see Trace for more information.
SET SQLCACHE <sqlcache_size>	Sets the SQL statement cache size to the specified value. A value of 0 disables SQL statement caching.

The `SET` statements can be issued as an SQL statement in the following methods:

- java.sql.Statement.execute
- java.sql.Statement.executeUpdate
- java.sql.Statement.executeQuery

These SET statements will not be sent down to the underlying database system.

Example

```
Statement stmt = conn.createStatement();
stmt.execute("set sqlcache 10");
```

Contents

11.5 Extended SQL Syntax SHOW DATABASES

The Oracle JDBC for Rdb thin driver allows the developer to retrieve the list of known databases from a connected server.

Format

```
SHOW DATABASES
```

The SHOW DATABASES statement is captured by the Thin driver, which will return a ResultSet containing a row for each database that is known to the connected thin server.

A known database is any database specified in the server XML-formatted configuration file Databases Section of the configuration file used during startup of the server.

As this is an SQL statement, a valid Connection to a database on the server is required before the SHOW DATABASES statement can be executed.

The SHOW DATABASES statement will not be sent down to the underlying database system.

The ResultSet returned by the SHOW DATABASES statement contains one row for each database known, and each row contains the following columns:

Table 11-1 SHOW DATABASES Columns

Column Name	Datatype	Description
RDB$DATABASE_NAME	string	The name given to the database in the server configuration file.
RDB$DESCRIPTION	string	Description of the database in the server configuration file.

The value of the RDB$DATABASE_NAME column may be used as the database file specification component of a URL string for Connections made to this server.

Example

```
.
.
.
Connection conn = DriverManager.getConnection(
```

```
  dbUrl,user,pwd );

Statement sc = conn.createStatement();
ResultSet rs = sc.executeQuery("show databases");
while (rs.next())
{
  String dbnam = rs.getString("RDB$DATABASE_NAME");
  String desc = rs.getString("RDB$DESCRIPTION"));
  System.out.println( dbnam + " : " + desc);
  Connection conn2 = DriverManager.getConnection(
    "jdbc:rdbThin://localhost:1701/"+dbnam, user,pwd );
  System.out.println (" version : " +
  conn2.getMetaData().getDatabaseProductVersion();
  conn2.close();
}
rs.close();
sc.close();
conn.close();
.
.
.
```

Contents

11.6 Extended SQL Syntax {nodbk} escape

During the preconditioning of SQL queries for use within JDBC, the JDBC drivers may add extra SQL syntax to the provided query to return dbkeys for future potential record access by the driver.

Although JDBC attempts to ensure that the additional text is valid for the statement, there may be some rare cases where the retrieval of dbkeys from the query is not possible, in which case JDBC may raise an SQL syntax exception during statement processing.

You may explicitly tell JDBC not to add this additional code by using the **{nodbk}** escape to the start of the SQL text of your statement.

Example

```
PreparedStatement ps = connection.prepareStatement(
"{nodbk}SELECT EMPLOYEES.EMPLOYEE_ID, S1.SALARY_START, " +
" S2.SALARY_START " + " FROM EMPLOYEES, " +
" SALARY_HISTORY S1, " + " SALARY_HISTORY S2
```

If JDBC sees this escape value, no additional dbkey clause(s) will be added.

A side-effect of this is that JDBC will not be able to carry out internal operations on the retrieved data that require JDBC knowing the dbkey of the records involved. An example of such an operation is the retrieval of BLOB (Segmented string) data from a table which requires JDBC to obtain the dbkey of each record to re-establish context of the record when it executes the internal segmented string retrieval subqueries.

You may also turn off the addition of the extra dbkey clause(s) to all queries connection-wide by using the **@suppressAddingDbkey=true** connection switch. See Connection Options for more details.

Contents

Chapter 12
Other Information

12.1 Disallowed Dynamic SQL Statements

Because JDBC has its own connection protocol, the following dynamic SQL statements will raise an exception if they are executed from a Statement or PreparedStatement:

- SET CONNECT
- CONNECT
- DISCONNECT

Contents

12.2 Sample Setup, Starting and Using an Oracle JDBC for Rdb Thin Server.

This section describes step by step how you can start a simple JDBC server and use it to access a database on your system.

1. Install Oracle JDBC for Rdb on the database server.
2. Decide on the versions of Rdb and Java you wish to use on the server.
3. Setup server-side configuration files and command procedures.
4. Start the Oracle JDBC for Rdb Thin server.
5. Install Oracle JDBC for Rdb thin driver on your client machine.
6. Write your application using the JDBC API.
7. Run your applications.

You may choose to start-up a server by:

- Invoking the rdbthinsrv JAR directly at the DCL command line.
  See Starting a Thin Server from the Command Line.
- Or creating and starting a JDBC Dispatcher in SQL/Services.
  See Starting a Thin Server from Oracle SQL/Services.
- Or using the Oracle JDBC for Rdb controller.
  See Starting a Thin Server from the Oracle JDBC for Rdb controller.

In this walk-through we will use the controller to maintain the servers. It is important that the command procedures used during the start-up of a server from the controller be correctly specified thus details of the appropriate command procedures will be provided below.

Step 1 Install Oracle JDBC for Rdb.

The Oracle JDBC for Rdb Release Notes describe the steps required to install Oracle JDBC for Rdb. These steps should be followed to install the product on the OpenVMS node that will be used as the server for your Oracle Rdb database.

The server machine requires Java to be installed prior to installing the Oracle JDBC for Rdb kit.

Once you have installed the kit you must set up your system so that it can use the JDBC kit. Several configuration files may have to be created or altered. Details of these steps follow.

Step 2 Decide on the versions of Rdb and Java.

The Oracle JDBC for Rdb Release Notes will tell you the minimum versions of Rdb and Java supported by Oracle JDBC for Rdb. You may however have several versions of both Rdb and Java on your server, which meet the minimum requirements.

When the thin servers run, they will need to have the environment they are running within set up so that the correct version of Rdb and Java will be used depending on your organization requirements, and which Oracle Rdb databases you wish the thin servers to access.

If you do have multiple versions of Rdb on your system, it is important that the server runs within the correct version of Rdb for the databases it will access. The Rdb environment is set up at the process level and cannot be changed for that server while the server is running. This means that a single running instance of a thin server may only be able to access databases for a single Rdb version.

If you try to connect to a database that does not match the version of Rdb you have set up for the thin server execution instance you will get an exception like:

```
SQLException: Failed to connect : in
<rdbjdbcsrv:connect failure>
%RDB-F-WRONG_ODS, the on-disk structure of the database file is not supported by this
version
-RDMS-F-ROOTMAJVER, database format 73.0 is not compatible with software version 74.1
```

You may have different versions of Java on your system as well. In addition, you can choose different Java VMs to run under. The VM version and type must be decided for a single thin server instance as once Java has been invoked and the server is running it cannot be changed for that server instance.

Both Rdb and Java provide mechanisms by which you can set up your environment for a specific version or variant. The Rdb version set up and the Java VM set up may be conducted manually by you prior to invoking a thin server from the DCL command line.

Alternatively, there are ways of providing the appropriate set up during the thin server start-up when you choose to start the server using an SQL/Services JDBC Dispatcher or by using the controller. An example of this type of set up can be seen in the steps that follow.
For this walkthrough, we will assume the following:

- Integrity server machine running OpenVMS
- Oracle Rdb release 7.4.x.x.x
- Java 6.0 (1.6.0)

Step 3 Set up server-side configuration files and command procedures.

Oracle JDBC for Rdb uses various command procedures to conduct server operations and set up its environment. These command procedures may have to be altered to suit your organizational and operational needs.

You may be required to:

- Modify `RDBJDBC_STARTUP.COM`.

- Add an invocation RDBJDBC_STARTUP.COM from your system startup procedure.
- Create an XML-formatted configuration file for your server definitions.
- Create a server set up command procedure.

In addition, Oracle recommends that XML-Formatted configuration files should be used to maintain the server and other information. These configuration files will have to be created by you. An example of a server configuration file may be found below and in Sample configuration file MY_SERVERS.XML.

RDBJDBC_STARTUP.COM

During product installation a file called RDBJDBC_STARTUP.COM will be created in the installation directory. This command procedure may be used to set up the required system wide logical names for Oracle JDBC for Rdb to function correctly. You may choose to use this command procedure with or without changes to set up the JDBC environment.

If the JDBC servers and drivers are to be used system wide, then system logical names should be used. In this case it may be appropriate to add the RDBJDBC_STARTUP.COM command procedure to your system startup procedure.

If you only require private use then JOB level logical names should be used, in which case the RDBJDBC_STARTUP.COM may be copied and/or modified to change the logical names to JOB level. Each user of the Oracle JDBC for Rdb on your server system will then need to invoke this startup procedure prior to conducting operations such as controller actions, starting,stopping or accessing the thin servers.

The RDBJDBC_STARTUP.COM file provides logical names that will be used by the Oracle JDBC for Rdb components to locate JARs, images, and command procedures and where to write log and temporary files. See the Oracle JDBC for Rdb Release Notes for more information on this command procedure and the JDBC specific logical names.

The steps that follow assume that the appropriate logical names have been set up and are available for use by you and Oracle JDBC for Rdb.

Server Configuration File

Server session and connection options may be added individually on the DCL command line when you invoke a server or the controller, but it may be more convenient to place these options in a configuration file and then use this configuration file when you conduct server operations.

See Configuration Files for more information on what may be contained in the configuration files and the format of the data within the file. Oracle recommends using the XML-formatted form of the configuration files as it does allow greater flexibility of option specification and allows more than one server definition to be defined in a single configuration file.

Since release 7.3.3

During product installation a generic configuration file RDBJDBCCFG_TEMPLATE.XML will be copied to the RDB$JDBC_HOME directory. If the RDB$JDBC_COM directory does not already contain a file named RDBJDBCCFG.XML, the contents of the configuration template file will be used to create this file.

The RDBJDBCCFG_TEMPLATE.XML found in the RDB$JDBC_HOME directory will be replaced each time you install Oracle JDBC for Rdb, however, any existing RDBJDBCCFG.XML file found in the

RDB$JDBC_COM directory will not be replaced. Oracle recommends using RDB$JDBC_HOME:RDBJDBCCFG_TEMPLATE.XML only as a template file and not to use this file in production.

You may use the configuration template file as a basis of your server configuration file. The configuration file provides information to Oracle JDBC for Rdb about the various servers you may be running. In addition, it provides session information for users of the controller.

Note:

The Oracle SQL/Services JDBC dispatcher uses a search list to locate the configuration file to use on server start up. If no other appropriate configuration file is found, the file RDBJDBCCFG.XML in RDB$JDBC_COM will be used.

For this walkthrough we have decided to create the definition for a thin server called ***MY_SRV*** listening on port **1888**. The configuration template file was copied and changed to add this information.

We have also chosen to place configuration and any other site-specific files in the RDB$JDBC_COM directory, as this is a standard Oracle JDBC for Rdb directory, and the logical name should be already set up during kit installation. The files may be placed anywhere on your system if the controller and server processes can access them. Remember that a server process will be started up like a normal login to the system, so it is important that any logical names used in the file specification be available to that process. The easiest way to ensure this is to have OpenVMS system-wide logical names.

In addition, a control password, ***MySecretPassword*** , has been chosen for control access to the servers.

Although the ***controlpass*** can be stored in its plain text form in the configuration file, Oracle recommends that you use the obfuscated form in the server characteristics section. But make sure that you are consistent with the casing of the password as passwords are case-sensitive.

The controller may be used to provide this obfuscated password, but make sure that you keep the casing correct by placing double quotations around the password phrase if you use the controller in command mode.

Example

```
$ java -jar rdbthincontrol.jar -digest "MySecretPassword"
digest : 0x7315A012ECAD1059A3634F8BE1347846
```

See Password Obfuscation in Server Configuration Files for more details.

The new configuration file called MY_CFG.XML :

```
<?xml version = '1.0'?>
<!-- Configuration file for MY servers -->
<config>
<!-- SESSION -->
<session
  name="DEFAULT"
  tracelevel="0"
  srv.mcBasePort="5517"
  srv.mcGroupIP="239.192.1.1"
/>
```

```
<!-- SERVERS -->
<servers>
<!-- DEFAULT server characteristics-->
  <server
          name="DEFAULT"
          type="RdbThinSrv"
          url="//localhost:1701/"
          maxClients="-1"
          srv.bindTimeout="1000"
          srv.idleTimeout="0"
          srv.mcBasePort="5517"
          srv.mcGroupIP="239.192.1.1"
          tracelevel = "0"
          autostart = "false"
          autorestart = "false"
          restrictAccess = "false"
          anonymous = "false"
          bypass = "false"
          tracelocal = "false"
          relay = "false"
          srv.startup="rdb$jdbc_home:rdbjdbc_startsrv.com"
  />
   <! My new server -->
   <server
          name="MY_SRV"
          controlUser="GROUND_CONTROL"
          controlPass="0x7315A012ECAD1059A3634F8BE1347846"
          type="RdbThinSrv"
          url="//localhost:1888/"
          cfg="rdb$jdbc_com:my_cfg.xml"
          srv.onStartCmd="@rdb$jdbc_com:my_setup.com"
  />
</servers>
</config>
```

Note:

The server definition for ***MY_SRV*** is minimal allowing most of the ***DEFAULT*** characteristics to be inherited. In addition, the session section is used to ensure that the broadcast IP the controller will check will be the same as the server uses.

RDB$JDBC_HOME:RDBJDBC_STARTSRV.COM

The default server properties in `MY_CFG.XML` sets the server configuration file used by the server by using the srv.startup property:

```
srv.startup="rdb$jdbc_home:rdbjdbc_startsrv.com"
```

This file is used by the controller during the start-up of a detached OpenVMS process that the server will run within. In most situations the default command procedure, `rdb$jdbc_home:rdbjdbc_startsrv.com` created during installation, can be used without change.

Server Setup Command Procedure

During server start-up any DCL command specified on `srv.onStartCmd` for the server will be executed prior to the server class being invoked. This a good place to conduct system specific and version specific set up procedures:

```
srv.onStartCmd="@rdb$jdbc_com:my_setup.com"
```

Note that, as this property should contain an executable DCL command, the `@` character is required to ensure the command procedure is correctly invoked.

Example

my_setup.com:

```
$ @SYS$LIBRARY:RDB$SETVER 74
$ @sys$common:[java$60.com]JAVA$60_SETUP.COM
$ define/job MYDIR sys$common:[DBS]
```

These commands ensure that the environment is correct for the server process to access a V7.4 Oracle Rdb database using Java 6.0.

Step 4 Start the Oracle JDBC for Rdb Thin server.

Now that set up and configuration files are created in place the controller may be used to start the server. The configuration file containing the server definitions is used as a parameter to the DCL command line invoking the controller. In the example we use command mode `startServer` to start the server.

Example

```
$ JAVA -JAR RDBTHINCONTROL.JAR -CFG RDB$JDBC_COM:MY_CFG.XML
CONTROLPASS "MySecretPassword" STARTSERVER NAME MY_SRV

RDB$NODE : nnn.nnn.nnn.nnn
RDB$PORT : 1888
RDB$STATUS : Idle
RDB$SERVER_NAME : srv1
RDB$SERVER_TYPE : RdbThinSrv
RDB$SERVER_VERSION : V7.4-100 20250113 BP1D
RDB$SERVER_SHR_VERSION : V7.4-100 20250113 BP1D
RDB$SERVER_PID : 0x2030DA4D(540072525)
RDB$ALLOWS_ANON : false
RDB$ALLOWS_BYPASS : false
RDB$NUMBER_OF_CLIENTS : 0
RDB$MAX_CLIENTS : -1
RDB$TRACE_LEVEL : 0
RDB$RESTRICT_ACCESS : false
```

In the example we provided both the configuration file to use and the control password. The ***controlpass*** could have been set in plain text in the configuration session section, but Oracle does not recommend placing plain text passwords in plain text files. Note also that the password is enclosed in double quotation marks to prevent case changing.

Step 5 Install the Oracle JDBC for Rdb thin driver on your client machine.

Once the Oracle JDBC for Rdb kit is installed on your OpenVMS server machine you must copy the thin driver component to the machine on which you will be running your application. This machine will also need to have Java installed.

The client-side components of the thin driver are contained in the ***RDBTHIN.JAR*** file.

A file transfer program such as FTP may be used to copy this JAR file to your client machine. Remember to ensure that a binary mode transfer is done as JARs are binary files.

You should place the JAR in an appropriate directory on your client machine. This may depend on how you will use the JDBC drivers and on the application and development systems you will be using on your client machine. See your application or development environment documentation on where JDBC drivers should be placed.

You should ensure that the `RDBTHIN` driver JAR is part of your CLASSPATH so that Java will be able to load it when your application requests it.

Starting with Oracle JDBC for Rdb release 7.3.5 you must choose the appropriate driver JAR to use depending on the Java version you will be using and what JDBC features you require to use.

Depending on the client system there will be methods by which you can include the driver JAR as part of the Java command when running your application, in which case the JAR does not have to added to the CLASSPATH environmental variable.

Example

For example, in MSDOS Java allows the use of the `cp` switch to specify classpath elements:

```
dos> java cp .;rdbThin.jar my_app
```

Note:

JAR files are binary files so you should ensure that the transfer utility copies the JAR file in binary mode.

Step 6 Write your application using the JDBC API.

The following is a simple application that tests that you have installed JDBC and carried out any set up correctly. This example is based on `RdbJdbcCheckup.java` from the installation and assumes that the Rdb server node has an IP of 555.1.14.91 and that the thin server we will use, the one we started earlier, is listening on port 1888.

Example

```
/*
* This sample can be used to check the JDBC installation.
* Just run it and provide the connect information. It will
* select "Hello World" from the database.
*/

// You need to import the java.sql package to use JDBC
```

```
import java.sql.*;

// We import java.io to be able to read from the command line
import java.io.*;

class my_app
{
  static BufferedReader in;
  public static void main(String args[])
  throws SQLException, IOException, Exception
  {
    String driverConStr = "jdbc:rdbThin://555.1.14.91:1888/";
    in = new BufferedReader(
    new InputStreamReader(System.in));
    Class.forName ("oracle.rdb.jdbc.rdbThin.Driver");

    // Prompt the user for connect information
    System.out.println(
      "Please enter information to test connection"+
      " to the database");
    String user;
    String password;
    String database;

    user = readEntry("user: ");
    int slash_index = user.indexOf('/');
    if (slash_index != -1)
    {
        password = user.substring(slash_index + 1);
       user = user.substring(0, slash_index);
     }
     else
        password = readEntry("password: ");

    database = readEntry("database: ");
    System.out.print("Connecting to the database...");
    System.out.flush();

    System.out.println("Connecting...");
    Connection conn = DriverManager.getConnection(
      driverConStr + database, user, password);

    System.out.println("connected.");
    // Create a statement
    Statement stmt = conn.createStatement();
    // Do the SQL "Hello World" thing
    ResultSet rset = stmt.executeQuery(
    "select 'Hello World' from rdb$database");

    while (rset.next())
       System.out.println(rset.getString(1));

    // close the result set, the statement and connect
    rset.close();
    stmt.close();
    conn.close();
  }

  // Utility function to read a line from standard input
  static String readEntry(String prompt)
  {
     try
     {
        StringBuffer buffer = new StringBuffer();
        System.out.print(prompt);
        System.out.flush();
        return in.readLine();
```

```
    }
    catch(IOException e)
    {
       return "";
    }
  }
}
```

Step 7 Run your application.

With the server started you can run the sample application and provide the thin server connection information.

Example
The following example assumes an Oracle Rdb database `personnel` in `MYDIR`:

```
$java cp .;rdb$jdbc_home:rdbThin.jar "my_app"
Please enter information to test connection to the database
user: my_name
password: my_password
database: myDir:personnel
Connecting to the database...Connecting...
connected.
Hello World
Your JDBC installation is correct.
```

Contents

12.3 Sample Setup, Starting an Oracle JDBC for Rdb Thin Server from Oracle SQL/Services

The following sections describe step-by-step how you can set up and start a simple JDBC server using Oracle SQL/Services.

Basically, you must:

1. Decide on the versions of Rdb and Java you wish to use on the server.
2. Setup server-side configuration files and command procedures.
3. Create a JDBC dispatcher in SQL/Services.
4. Associate configuration and setup files.
5. Start the JDBC dispatcher.

See Chapter 7 Oracle SQL/Services and Oracle JDBC for Rdb Servers for more information on these operations.

Step 1 Decide on the versions of Rdb and Java

This step is basically the same as Step 2 Decide on the versions of Rdb and Java, as covered in Sample Setup, Starting and Using an Oracle JDBC for Rdb Thin server.

Step 2 Setup server-side configuration files and command procedures

For the server to start correctly, a command procedure and a configuration file must be created.

The following two files must be created:

- The Server Configuration file
- The Server Setup file

You may use an XML configuration file to store the server definitions for your server. In addition, you should provide a command procedure to set up the Rdb and Java environments correctly for this server. This environment setup may also be done as part of the setup of the dispatcher environment in SQL/Services, but for the purpose of this example, we shall create our own setup procedure.

Server Configuration File

As limited information can be passed to the server at the command line, most of the server characteristics for a JDBC Dispatcher server can be placed in a configuration file.

See Configuration Files for more information on what may be contained in the configuration files and the format of the data within the file. Oracle recommends using the XML-formatted form of the configuration files as it does allow greater flexibility of option specification and allows more than one server definition to be defined in a single configuration file.

During installation a generic configuration file `RDBJDBCCFG_TEMPLATE.XML` will be copied to the `RDB$JDBC_HOME` directory. If the `RDB$JDBC_COM` directory does not already contain a file named `RDBJDBCCFG.XML`, the contents of the configuration template file will be used to create this file.

Note:

The `RDBJDBCCFG_TEMPLATE.XML` found in the `RDB$JDBC_HOME` directory will be replaced each time you install Oracle JDBC for Rdb, however, any existing `RDBJDBCCFG.XML` file found in the `RDB$JDBC_COM` directory will not be replaced. Oracle recommends using `RDB$JDBC_HOME:RDBJDBCCFG_TEMPLATE.XML` only as a template file and not to use this file in production.

You may use the configuration template file as a basis of your server configuration file. The configuration file provides information to Oracle JDBC for Rdb about the various servers you may be running. In addition, it provides session information for users of the controller.

Note:

The Oracle SQL/Services JDBC dispatcher uses a search list to locate the configuration file to use on server start up. If no other appropriate configuration file is found, the file `RDBJDBCCFG.XML` in `RDB$JDBC_COM` will be used.

For this walkthrough we have decided to create the definition for a thin server called ***SQS1888*** listening on port **1888**. The generic configuration file was copied and changed to add this information.

We have also chosen to place configuration and any other site-specific files in the `RDB$JDBC_COM` directory, mainly as this is a standard Oracle JDBC for Rdb directory and the logical name should be already set up for us at the system level. The files may be placed anywhere on your system, but you should ensure that the controller and server processes can access them. Remember that a server process will be started up in much the same way as a normal login to the system, so it is important that any logical

names used in the file specification be available to that process. The easiest way to ensure this is to have system wide logical names.

In addition, a control password, ***MySecretPassword*** has been chosen for control access to the servers.

Although the ***controlpass*** can be stored in its plain text form in the configuration file, Oracle recommends that you use the obfuscated form in the server characteristics section. But make sure that you are consistent with the casing of the password as passwords are case-sensitive.

The controller may be used to provide this obfuscated password, but keep the casing correct by placing double quotations around the password phrase if you use the controller in command mode.

Example

```
$ java -jar rdbthincontrol.jar -digest "MySecretPassword"
digest : 0x7315A012ECAD1059A3634F8BE1347846
```

See Password Obfuscation in Server Configuration Files for more details.

We have chosen to create a configuration file using one of the standard file specifications used by the dispatcher when searching for configuration files. See Determining the server configuration file on how the dispatcher locates a configuration file to use.

As the port used by the server will be 1888, we will create a new configuration file called `SQS1888_CFG.XML` and place it in the `RDB$JDBC_COM` directory:

```
$ type RDB$JDBC_COM:SQS1888_CFG.XML

<?xml version = '1.0'?>
<!-- Configuration file for MY servers -->
<config>
   <!-- SESSION -->
   <session
     name="DEFAULT"
     tracelevel="0"
     srv.mcBasePort="5517"
     srv.mcGroupIP="239.192.1.1"
   />

   <!-- SERVERS -->
   <servers>
   <!-- DEFAULT server characteristics-->
   <server
     name="DEFAULT"
     type="RdbThinSrv"
     url="//localhost:1701/"
     maxClients="-1"
     srv.bindTimeout="1000"
     srv.idleTimeout="0"
     srv.mcBasePort="5517"
     srv.mcGroupIP="239.192.1.1"
     tracelevel = "0"
     autostart = "false"
     autorestart = "false"
     restrictAccess = "false"
     anonymous = "false"
     bypass = "false"
     tracelocal = "false"
     relay = "false"
     srv.startup="rdb$jdbc_home:rdbjdbc_startsrv.com"
   />
```

```
   <! My new server -->
   <server
     name="SQS1888"
     controlUser="SQS_CONTROL"
     controlPass="0x7315A012ECAD1059A3634F8BE1347846"
     type="RdbThinSrv"
     url="//localhost:1888/"
    />
</servers>
</config>
```

Note:

The server definition for ***SQS1888*** is minimal allowing most of the ***DEFAULT*** characteristics to be inherited. Also, the session section is used to ensure that the broadcast IP the controller will check will be the same as the server uses.

Server Setup File

The JDBC dispatcher may require environmental setup for Java and the correct Oracle Rdb version to run. This setup can be done in a command procedure that will be executed just prior to starting the actual server image.

As the setup is generic, we have decided to create the file `RDBJDBC_SQS_ONSTARTUP.COM` and place it in the `RDB$JDBC_COM` directory. By default, this file will be used by the dispatcher whenever a server must be started. JDBC Dispatcher Setup Procedure describes the use of a setup command procedure for the dispatcher.

Example

```
$ type RDB$JDBC_COM:RDBJDBC_SQS_ONSTARTUP.COM

$ @SYS$LIBRARY:RDB$SETVER 74
$ @sys$common:[java$60.com]JAVA$60_SETUP.COM
$ define/job MYDIR sys$common:[DBS]
```

These commands ensure that the environment is correct for the server process to access a V7.4 Oracle Rdb database using Java Hotspot VM, which is the default in Java 6.0.

In addition, we have added a JOB level logical name that may be used in database file specifications on the JDBC Connection URL.

Step 3. Create a JDBC dispatcher in SQL/Services

Now that the configuration file and setup procedure have been created and moved to the appropriate directory we can now create a JDBC Dispatcher. We will use 1888 as the PORT_ID as this will be the key value used by the dispatcher to locate the necessary files for server start-up.

```
$ MCR SQLSRV_MANAGE74
SQLSRV> CONNECT SERVER;
SQLSRV> CREATE DISPATCHER MY_JDBC_DISP NETWORK_PORT TCPIP
```

```
PORT_ID 1888 PROTOCOL JDBC;
SQLSRV> SHOW DISPATCHER;
Dispatcher MY_JDBC_DISP
    State:                    UNKNOWN
    Autostart:                on
    Max connects:             100 clients
    Idle User Timeout:        <none>
    Max client buffer size:   5000 bytes
    Network Ports:                      (State)   (Protocol)
      TCP/IP  port     1888             Unknown   JDBC clients
    Log path:           SYS$MANAGER:
    Dump path:          SYS$MANAGER:
```

Step 4. Associate configuration and setup files

Next, we must associate the server configuration and setup files with this dispatcher.

As we chose to use standard configuration file names, the dispatcher will make the following associations automatically and we need take no further action to make this happen.

Given the PORT_ID of `1888`:

- server name = `SQS1888`
- configuration file = `RDB$JDBC_COM:SQS1888_CFG.XML`
- setup file = `RDB$JDBC_COM:RDBJDBC_SQS_ONSTARTUP.COM`

If we had chosen not to use standard naming, then we would have had to set up logical names to point to the appropriate files. See Associating an Oracle SQL/Services JDBC Dispatcher to a Server for more details.

However, we still need to tell the dispatcher what type of server it will be starting so we must create the appropriate logical name. For simplicity we shall place this logical name in the SYSTEM logical name table. See Determining Server Type for information on server type associations.

```
$ DEFINE/SYSTEM RDB$JDBC_SQSTYPE_1888 STD
```

If we had chosen to start up a Pool server, we would not have needed to create this logical name as this is the default server type used by the JDBC dispatcher, but as the server type is a normal thin server, we must inform the dispatcher of this fact using the logical name.

Step 5 Start the JDBC dispatcher

Now that the configuration files are in place and any logical names used by the dispatcher have been defined, we can now use the SQL/Services manager to start the JDBC dispatcher.

```
SQLSRV> start dispatcher my_jdbc_disp;
SQLSRV> show disp my_jdbc_disp;
Dispatcher MY_JDBC_DISP
    State:                    STARTING
    Autostart:                on
    Max connects:             100 clients
    Idle User Timeout:        <none>
    Max client buffer size:   5000 bytes
```

```
    Network Ports:                      (State)    (Protocol)
      TCP/IP  port     1888             Unknown    JDBC clients
    Log path:             SYS$MANAGER:
    Dump path:            SYS$MANAGER:

SQLSRV> show disp my_jdbc_disp;
Dispatcher MY_JDBC_DISP
    State:                      RUNNING
    Autostart:                  on
    Max connects:               100 clients
    Idle User Timeout:          <none>
    Max client buffer size:     5000 bytes
    Network Ports:                      (State)    (Protocol)
      TCP/IP  port     1888             Unknown    JDBC clients
    Log path:             SYS$MANAGER:
    Dump path:            SYS$MANAGER:
Log File: SYS$SYSROOT:[SYSMGR]SQS_DECRDB_JDBC_DISP08091.LOG;
Dump File: SYS$SYSROOT:[SYSMGR]SQS_DECRDB_JDBC_DISP080.DMP;
```

```
SQLSRV> show disp my_jdbc_disp;
Dispatcher MY_JDBC_DISP
State: RUNNING
Autostart: on
Max connects: 100 clients
Idle User Timeout: <none>
Max client buffer size: 5000 bytes
Network Ports: (State) (Protocol)
TCP/IP port 1888 Inactive JDBC clients
Log path: SYS$MANAGER:
Dump path: SYS$MANAGER:
Log File: SYS$SYSROOT:[SYSMGR]SQS_DECRDB_JDBC_DISP08O91.LOG;
Dump File: SYS$SYSROOT:[SYSMGR]SQS_DECRDB_JDBC_DISP08O.DMP;
```

See the Oracle SQL/Services documentation and Starting a JDBC Dispatcher for more details on starting a dispatcher.

If the server starts up correctly you should be able to use the server from any JDBC client using the Oracle JDBC for Rdb thin driver.

You may also use the controller to check that the server is running:

```
$ java -jar rdb$jdbc_home:rdbthincontrol.jar
-cfg RDB$JDBC_COM:SQS1888_CFG.XML -controlpass "MySecretPassword"
-name SQS1888 -showServer
```

Contents

12.4 Sample configuration file MY_SERVERS.XML

```
<?xml version = '1.0'?>
<! Configuration file for Rdb Thin JDBC Drivers/Servers -->
<config>
<! SESSION -->

<session
     name="fred"
     user="jdcb_user"
     tracelevel="0"
```

```
     srv.mcBasePort="5517"
     srv.mcGroupIP="239.192.1.1"
/>

<! SERVERS -->
<servers>
  <! DEFAULT server characteristics -->
  <!-- NOTE control password is the
       obfuscated form of "MySecretPassword" -->
  <server
     name="DEFAULT"
     type="RdbThinSrv"
     url="//localhost:1701/"
     maxClients="-1"
     srv.bindTimeout="1000"
     srv.idleTimeout="0"
     srv.mcBasePort="5517"
     srv.mcGroupIP="239.192.1.1"
     tracelevel = "0"
     autostart = "false"
     autorestart = "false"
     restrictAccess = "false"
     anonymous = "false"
     bypass = "false"
     tracelocal = "false"
     relay = "false"
     controlUser="control_user"
     controlPass="0x7315A012ECAD1059A3634F8BE1347846"
     cfg="rdb$jdbc_com:rdbjdbccfg.xml"
     srv.execStartup="rdb$jdbc_home:rdbjdbc_startexec.com"
     srv.startup="rdb$jdbc_home:rdbjdbc_startsrv.com"
     sharedmem = "0"
     ssl.default="true"
  />
  <! DEFAULT Secure socket server -->
  <server
     name="DEFAULTSSL"
     type="RdbThinSrvSSL"
     ssl.default="false"
     ssl.context="TLS"
     ssl.keyManagerFactory="SunX509"
     ssl.keyStoreType="jks"
     ssl.keyStore="rdbjdbcsrv.kst"
     ssl.keyStorePassword="CHANGETHIS"
     ssl.trustStore="rdbjdbcsrv.kst"
     ssl.trustStorePassword="CHANGETHIS"
  />
  <! now specific servers that will be started by pool server -->
  <server
     name="srv1forRdb"
     type="RdbThinSrv"
     url="//localhost:1701/"
     autoStart="true"
     autoRestart="true"
     logfile="rdb$jdbc_logs:srv1forRdb.log"
     tracelevel="-1"
     maxClients=1
  />
  <server
     name="srv2forRdb"
     type="RdbThinSrv"
     url="//localhost:1708/"
     autoStart="true"
     logfile="rdb$jdbc_logs:srv2forRdb.log"
  />

  <server
```

```
     name="myserver"
     type="RdbThinSrv"
     url="//localhost:1788/"
  />

  <! MP server -->
  <! sharedmem is in KB default = 1024 -->
  <server
     name="srvMPforRdb"
     type="RdbThinSrvMP"
     url="//localhost:1705/"
     autoStart="true"
     maxClients="10"
     maxFreeExecutors="10"
     prestartedExecutors="10"
     sharedMem="10240"
  />
  <! the pool server -->
  <server
     name="rdbpool"
     type="RdbThinSrvPool"
     url="//localhost:1702/" >
     <pooledServer name="srv1forRdb"/>
     <pooledServer name="srv2forRdb"/>
     <pooledServer name="srvMPforRdb"/>
  </server>

  <! Secure socket server -->
  <server
     name="srvssl1forRdb"
     type="RdbThinSrvSSL"
     url="//localhost:1709/"
  />

</servers>

<!-DATABASES -->
<databases>
  <database
     name="mf_pers"
     url="//localhost:1701/mydisk:[databases]mf_personnel"
     driver="oracle.rdb.jdbc.rdbThin.Driver"
     URLPrefix="jdbc:rdbThin:"
    />
  <database
     name="pers"
     url="//localhost:1702/mydisk:[databases]personnel"
     driver="oracle.rdb.jdbc.rdbThin.Driver"
     URLPrefix="jdbc:rdbThin:"
  />
</databases>

</config>
```

12.5 Datatype Mapping from Oracle Rdb to java.sql.Types

Table 12-1 Datatype Mappings from OracleRdb to java.sql.Types

Rdb SQL datatype	java.sql.Types
CHAR(n)	CHAR
NCHAR(n)	CHAR
VARCHAR(n)	VARCHAR
NCHAR VARYING	VARCHAR
FLOAT[(n)]	If n > 24 then DOUBLE else FLOAT
REAL	FLOAT
DOUBLE PRECISION	DOUBLE
DECIMAL[(n[,n])]	DECIMAL
INTEGER[(n)]	If n == 0 then INTEGER else NUMERIC
SMALLINT[(n)]	If n == 0 then SMALLINT else NUMERIC
TINYINT[(n)]	If n == 0 then TINYINT else NUMERIC
BIGINT[(n)]	If n == 0 then BIGINT else NUMERIC
QUADWORD[(n)]	If n == 0 then BIGINT else NUMERIC
DATE ANSI	DATE
DATE VMS	TIMESTAMP
TIME	TIME
TIMESTAMP	TIMESTAMP
INTERVAL	BIGINT
BYTE VARYING	VARBINARY
LIST OF BYTE VARYING	BLOB

Contents

12.6 Datatype Mapping from java.sql.Types to Oracle Rdb

Table 12-2 Datatype Mappings from java.sql.Types to OracleRdb

SQL Type (from java.sql.Types)	Rdb SQL datatype
CHAR	CHAR(n)
NCHAR	NCHAR(n)
VARCHAR	VARCHAR(n)
FLOAT	REAL
DOUBLE	DOUBLE PRECISION
DECIMAL	DECIMAL[(n[,n])]
INTEGER	INTEGER
SMALLINT	SMALLINT
TINYINT	TINYINT
BIGINT	BIGINT
NUMERIC	BIGINT(n)

DATE	DATE ANSI
TIMESTAMP	TIMESTAMP
TIME	TIME
BIGINT	INTERVAL
VARBINARY	BYTE VARYING
BLOB	LIST OF BYTE VARYING
CLOB	LIST OF BYTE VARYING

Contents

12.7 JDBC Specification SQL to Java Datatype Mappings

Table 12-3 Datatype Mappings from SQL to Java

SQL Type (from java.sql.Types)	Java Type
BIT	boolean
TINYINT	byte
SMALLINT	short
INTEGER	int
BIGINT	long
REAL	float
FLOAT	double
DOUBLE	double
DECIMAL	java.math.BigDecimal
NUMERIC	java.math.BigDecimal
CHAR	java.lang.String
VARCHAR	java.lang.String
LONGVARCHAR	java.lang.String
DATE	java.sql.Date
TIME	java.sql.Time
TIMESTAMP	java.sql.Timestamp
BINARY	byte[]
VARBINARY	byte[]
BLOB	java.sql.Blob
CLOB	java.sql.Clob

Contents

12.8 JDBC Specification Java to SQL Datatype Mappings

Table 12-4 Datatype Mappings from Java to SQL

Java Type	SQL Type (from java.sql.Types)
boolean	BIT
byte	TINYINT
short	SMALLINT
int	INTEGER
long	BIGINT
float	REAL
double	DOUBLE
java.math.BigDecimal	NUMERIC
java.lang.String	VARCHAR or LONGVARCHAR
byte[]	VARBINARY or LONGVARBINARY
java.sql.Date	DATE
java.sql.Time	TIME
java.sql.Timestamp	TIMESTAMP
java.sql.Blob	BLOB
java.sql.Clob	CLOB

Contents