Oracle JDBC for Rdb User Guide

Release V7.2-410

March 2006

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. They both support the following standards and features:

- JDK 1.4 / JDBC 3.0
- Same syntax and APIs

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

General information on Java may be found at
http://java.sun.com/reference

General information on JDBC may be found at
http://java.sun.com/products/jdbc/index.html.

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 thin multi-process server
- Oracle Rdb thin pool server

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

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

Top of the Document

Oracle JDBC for Rdb

There are two types of Oracle JDBC for Rdb drivers:

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

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 Client libraries are also available on that same machine. 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.

Oracle Rdb Database 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>
```

where

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

Example:

To connect to MY_DB_DIR:PERSONNEL:

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

Note:

Because the database will reside on an OpenVMS system; the `<database_specification>` should be a valid OpenVMS-style file specification or logical name, for example:

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

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
```

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.

Oracle Rdb Database 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_switches>
```

where:

<node>	Is the node name or IP address of the node that the Rdb JDBC server you wish to connect to is running on.
<port>	Is the port the Rdb thin server you wish to connect to is listening on.
<database_specification>	Is the full file specification of the Rdb database that you wish to connect to.

<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.

Example:

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

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

Note:

Because the database will reside on an OpenVMS system; 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.

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
```

Top of the Document

Connection Options

The Oracle JDBC for Rdb drivers recognize a number of options that may be specified on a connection that specify certain default behavior and settings which will 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 .

The format used to add a connection option to a connection URL is

@<option_name>=<value>

The following connection options may be used

<option_name>	<value>	Description
cli.idleTimeout	Decimal or hex integer	Sets the maximum time (in milliseconds) this client connection may be idle. If no operation is carried out using this connection within the time specified, the connection will be forcibly disconnected. The default is cli.idleTimeout=0 meaning unlimited idle time allowed. See Client connection timeout for more details
lockwait	Decimal or hex integer	Sets the lockwait (in seconds) for transactions. The default is lockwait=-1 See Lockwait and Maxtries for more details.
multiProcess	true or 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. See Executor Sub-process used with the Rdb Native driver for more details.

sqlcache	Decimal or hex integer	If less than or equal to 0, SQL statement cache is disabled. Positive values specify the size of the SQL statement cache. The default is sqlcache=0
ssl*	various	Sets one or more SSL characteristics, see USING SSL for more details on these
tracelevel or tl	Decimal or hex integer	Specifies the default tracelevel for the connection. The default is tracelevel=0
transaction	readonly or readwrite or automatic or oracle or manual	Specifies the default transaction for this connection. The default is transaction=automatic See Default Transaction and Scope of CONNECTION.setReadOnly() for more details
usehints	true or false	If true, the optional JDBC hint methods will be observed. If false, the optional JDBC hint methods will be silently ignored. The default is usehints=true See JDBC Hint Methods for more details.

Example:

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

```
Connection conn = DriverManager.getConnection(
      “jdbc:rdbThin://bravo:1755/my_db_dir: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/my_db_dir:pers”,
      info);
```

Top of the Document

Oracle JDBC for Rdb System Properties

The Oracle Rdb 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.

The format of the system property is:

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

For 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

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 specifed 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 specifed on the command line as a standard configuration option or in a configuration file.

Top of the Document

Oracle JDBC for Rdb Servers

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

There are three 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

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.

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.

Starting a Thin Server

A thin 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.

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.

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:

```
thincontroller> start server serv1
```

Starting a Thin Server from Oracle SQL/Services

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

```
$run sys$system:SQLSRV_MANAGE72
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.

Starting a Thin Server from the Command Line

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

```
$ java –jar rdbthinsrv.jar [-option ]
```

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

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

Instead of placing a number of 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.

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
```

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

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 subprocesses 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.

The majority of the multi-process server operations are carried out in a client thread context within the main server process, handing off control to the clients 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, see Prestarted Executors for details.

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.

Shared Memory Usage

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

The default allocation for shared memeory 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.

Prestarted Executors

With a multi-process server you may also specify the number of executor process that may be prestarted when the server starts running.

In addition you can also specify the maximum number of free executor process that may be kept around while the server is running. This is particulary useful if your system takes a while to start OpenVMS processes and sub-processes due to system load.

By prestarting executor processes you may reduce the overall elapsed time it takes for a client to make its initial database connection.

Executor Naming

By default the name of the executor subprocess has the following format:
First seven (7) characters of server name + eight (8) character hexadecimal id, for example:

```
RDBTHNS00000220
```

Thus 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.

The executor naming format may be changed by using the srv.execPrefix configuration option.

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 still keep the executor name within the
Process name sized expected by OpenVMS.

For example given the srv.execPrefix of “MY_EXECUTOR_” the fourth executor will be named:

```
MY_EXECUTOR_004
```

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 exopect a server to maintain when specify a customized executor name prefix.

Names of servers are not case-sensitive.

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

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.

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.

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:

```
thincontroller> start server Mpserv1
```

Starting a Multi-process Server from Oracle SQL/Services

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

```
$run sys$system:SQLSRV_MANAGE72
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.

Starting a Multi-process Server from the Command Line

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

```
$ java –jar rdbthinsrv.jar [-option ]
```

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

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.

```
$ java –jar rdbthinsrv.jar –port 1755 –type “RdbThinSrvMP”
```

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

Alternatively, you 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.

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
```

Note:

The muli-process server needs to allocate shared global memory for communication with its executors. The default allocation of 1024 KB is only adequate for one or two executors. You can use the sharemem attribute to set the size of this shared memory area.

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.

With a Multi-process server you may also specify the number of executor process that may be prestarted when the server starts running. In addition you can also specify the maximum number of free executor process that may be kept around while the server is running. This is particulary useful if your system takes a while to start OpenVMS processes and sub-processes due to system load. By prestarting executor processes you may reduce the overall elapsed time it takes for a client to make its initial database connection.

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

Top of the Document

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 your 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.

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.

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

Note:

The thin pool server carries out server pooling NOT connection pooling. Connections are created in each connection request and are not reusable.

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.

Starting a Pool Server from the Controller

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

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

```
    thincontroller> start server mypoolserver
```

Starting a Pool Server from Oracle SQL/Services

A pool server may be started from the Oracle SQL/Services manager:

```
$run sys$system:SQLSRV_MANAGE72
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.

Starting a Pool Server from the Command Line

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

```
$ java –jar rdbthinsrvpool.jar [-option ]
```

See Pool Server Configuration Options for a list of valid options.

Each option must have a hyphen – prepended to it, for example:

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

Pool Server Operation

Once it is started, the pool server will scan the list of pooled servers in a round-robin fashion to select the next available server.
You can start and stop the servers in the pool at anytime. If a server is not available, then it will be bypassed by the pool server. The pool server also has the ability to 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 startup 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 have autoStart enabled will also be shut down.

Top of the Document

Server Configuration

There are a number of 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 uses these options within a configuration file.

The following sections detail the available configuration options.

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 and thin multi-process servers. See Pool Server Configuration Options for the options that may be used with thin pool servers.

anonymous	If specified, 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 password and user to specify the default username/password to use on connections. By default, anonymous connections are disabled and the

	client must specify a valid username and password combination to access the Rdb database.
allowDatabase <name=database-name>	Specifies the name of a database that will be allowed to be accessed using this server. This is used in conjunction with the restictAccess option. This option should only be used within an XML formatted configuration file. The named database should also be described in the same configuration file. See Restricting Database Access for more details.
autorestart	If specified, indicates to any pool server that may include this server in its pool of servers to automatically restart this pooled server. This option is only valid in an XML formatted Configuration File. See Oracle JDBC for Rdb Pool Server for more details.
autostart	If specified, indicates to any pool server that may include this server in its pool of servers to automatically start up this pooled server. This option is only valid in an XML formatted Configuration File. See Pool Server Operation for more details.
b or buffersize <send_buffer_size>	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 connection, 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	Specifies that 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 NOBYPASS 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>	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>	Sets the maximum time (in milliseconds) a client connection may be idle. If no operation is carried out using this connection within the time specified, the connection will be forcibly disconnected. The default is 0 that means unlimited idle time allowed. See Client connection timeout for more details.

controlpass <control_password>	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 a password digest value. See Control Password for more information on this password.
fs or fetchsize <default_fetch_size>	Specifies the default fetchsize to use. The fetchsize 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 may improve the network performance by reducing the average network overhead per record retrieved. If not specified, the default fetchsize is set to 100 records.
lockwait <lock_wait>	Specifies the maximum number of seconds to wait on getting a record lock. This switch used in conjunction with *maxtries* specifies how long to keep trying to get a lock on a locked object before issuing a locked object Exception. The default is 1 second.
log or logfile <file_specification>	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.

name <server name >	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.
maxclients <maximum_number_of_clients>	Specifies the maximum number of concurrent clients this server instance may handle. The default is an unlimited number of clients.
maxFreeExecutors <maximum_number_of_free_executors>	Specifies the maximum number of free (unused) executor processes that may be maintained while the server is running. This feature is only applicable to Multi-process servers The default is 0.
maxtries <maximum_number_of_lock_attempts>	Specifies the maximum number of times to try to get a record lock. This switch used in conjunction with *lockwait* specifies how long to keep trying to get a lock on a locked object before issuing a locked object Exception. The default is ten (10) times.
p or port <port_num>	Tells the server to listen on port <port_num>
pw or password <default_user_password>	Used in conjunction with the *user* and *anonymous* switches provides the password to use on an anonymous connection
persona <username>	Specifies the Operating system username which the process running the server will assume. See Persona for more details.

prestartedExecutors <number_of_prestarted_executors>	Specifies the number of executor process to start up when the Multi-process server starts. This feature is only applicable to Multi-process servers. The default is 0.
relay	If specified designates that this server should relay poll requests to all active servers in its network community This feature is currently unavailable.
restrictAccess	Used in conjunction with the allowDatabase option to restrict access to designated databases. This option should only be used within an XML formatted configuration file. See Restricting Database Access for more details.
sharedMem <size_in_KB>	Specifies the amount of global shared memory (in KB) that should be allocated by the server. This feature is only applicable to Multi-process servers. The default is 1024.
srv.bindTimeout <timeout>	Sets the timeout (in miilliseconds) on waiting for a database connection to complete. If the database fails to connect within this time an exception will be raised. The default is 0 that means unlimited timeout.
srv.execPrefix <prefix>	Only valid for multi-process servers. Specifies the prefix to use for executor names. See Executor Naming for more details.

srv.execStartup <file_specification>	Only valid for multi-process servers. Specifies the startup batch or command file that will be used to startup the subprocess for each client connection. See Server Command Procedures for more details.
srv.idleTimeout <timeout>	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 down due to inactivity. The default is 0 that means unlimited idle time allowed. See Server Inactivity Timeout for more details.
srv.mcBasePort <base_port>	Specifies the base port number that will be used for multicast operations. The default is 5517. A value of zero (0) will disable multicast operations.
srv.mcGroupIP <group_ip>	Specifies the multicast IP group that this server will participate in. The default is 239.192.1.1.
srv.MpMaxTries <max_tries>	Only valid for multi-process servers. Specifes the number of times the server should try to synchronize handshake with executor before giving up. The default is 500 times.

srv.MpTryWait <wait_time>	Only valid for multi-process servers. Specifies the time in millseconds to wait between server/executor handshake synchronization tries. The default is 10 milliseconds.
srv.onExecStartCmd <command>	Specifies a DCL command statement that should be executed prior to starting up an executor. See On Start Commands for more details.
srv.onStartCmd <command>	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 up using the controller or by a pool server. See On Start Commands for more details.
srv.startup <file_specification>	Specifies the startup batch or command file that will be used by the controller to startup the process for this server See Server Command Procedures for more details.
tl or tracelevel <trace_level>	Sets the trace level for debugging purposes. See Trace for further information
tracelocal	Specifies that only local server base tracing should be enabled. Tracelevel values specified by a client connection will not affect the trace level of the server components if this option is set.

type <server_type>	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 ■ RdbThinSrvPool pool server ■ RdbThinSrvPoolSSL pool server using SSL The default value is RdbThinSrv
u or user <default_user_name>	Used in conjunction with the *password* and *anonymous* switches provides the username to use on an anonymous connection
url <connection URL>	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>/

As the multi-process server starts up separate OpenVMS subprocesses for each client connection, there are a number of system quota and naming issues that you should be aware of.

The name of the executor subprocess has the following format:

First seven (7) characters of server name + eight (8) character hexadecimal id, for example:

```
RDBTHNS00000220
```

Thus 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.

Names of servers are not case-sensitive.

Top of the Document

Pool Server Configuration Options

The following table lists valid configuration options that may be used with a pool server.

cfg or configfile <configuration_filename>	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.
controlpass <control_password>	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>	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>	Specifies the node on which the thin server number <n> resides. This option is not valid for use in XML-formatted configuration files.
poolserver	Specifies that the server should act as a pool server. This is a mandatory option if used on the command line or a nonXML formatted configuration file
pooledserver <name=server-name>	Specifies the name of a server that will take part in the pool. This option is only available when using an XML-formatted configuration file. The named server should also be described in the same configuration file.
poolsize <pool_size>	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>	Specifies the port for the thin server number <n> in server list. This option is not valid for use in XML-formatted configuration files.
p or port <port_num>	Tells the pool server to listen on port <port_num>.
srv.keepAliveTimer <seconds>	Sets the time (in seconds) of the duration between pool server checks for non-running pooled servers that have autoRestart enabled. The default for this value is 60 seconds. See Oracle JDBC for Rdb Pool Server for more details.

srv.mcBasePort <base_port>	Specifies the base port number that will be used for multicast operations. The default is 5517. A value of zero (0) will disable multicast operations.
srv.mcGroupIP <group_ip>	Specifies the multicast IP group that this server will participate in. The default is 239.192.1.1.
tl or tracelevel <trace_level>	Sets the trace level for debugging purposes. See Trace for further information
url <connection URL>	Specifies the node IP and port this server will run on. This switch overrides any *port* switch.

As there can be a number of servers listed in the server pool it is advised to use the configuration file to specify these options.

The number of servers in the pool is specified by the *poolsize* option, or, in the case of an XML-formatted configuration file, the number of PooledServer subsections.

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.

Top of the Document

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

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.

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

```
$java –jar rbthinsrv.jar –cfg thinsrv.cfg
```

The same server configuration options as specified above can be used but with the following changes:

1. 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.
2. Each keyword must be separated from its value by an equals sign (=)

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 thin pool server should contain information about the list of thin 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
```

Top of the Document

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 its own server section and must be given a unique name. This name is used to get default configuration information about the server on server startup, 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.

```
$java –jar rbthinsrv.jar –cfg rdbjdbccfg.xml
```

The format of the configuration file is XML V1.0.

The same server configuration options as specified above 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 equals 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. Each server must have a unique name.

```
<?xml version = ‘1.0’?>
<!—Configuration file for Rdb Thin JDBC Drivers and Servers à
<config>
  <!—SERVERS à
  <servers>
    <!—DEFAULT server characteristics à
    <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$jdbc_home:rdbjdbc_startsrv.com”
    sharedmem = “0”
    ssl.default=”true”
  />
  <!—DEFAULT Secure socket server à
  <server
    name=”DEFAULTSSL”
    type=”RdbThinSrvSSL”
    ssl.default=”false”
    ssl.context=”TLS”
    ssl.keyManagerFactory=”SunX509”
    ssl.keyStoreType=”jks”
    ssl.keyStore=”rdbjdbcsrv.kst”
    ssl.keyStorePassword=”CHANGETHIS”
    ssl.trustStore=”rdbjdbcsrv.kst”
    ssl.trustStorePassword=”CHANGETHIS”
  />
  <!—now specific servers that will be started up by pool server à
  <server
    name=”srv1forRdb”
    type=”RdbThinSrv”
    url=”//localhost:1701/”
    autoStart=”true”
    autoRestart=”true”
    logfile=”rdb$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 à
    <!–sharedmem is in KB default = 1024 à
    <server
      name=”srvMPforRdb”
      type=”RdbThinSrvMP”
      url=”//localhost:1705/”
      autoStart=”true”
      maxClients=”10”
      maxFreeExecutors=”10”
      prestartedExecutors=”10”
      sharedMem=”10240”
    />
    <!–the pool server à
    <server
      name=”rdbpool”
      type=”RdbThinSrvPool”
      url=”//localhost:1702/” >
      <pooledServer name=”srv1forRdb”/>
      <pooledServer name=”srv2forRdb”/>
      <pooledServer name=”srvMPforRdb”/>
    </server>

    <!–Secure socket server à
    <server
      name=”srvssl1forRdb”
      type=”RdbThinSrvSSL”
      url=”//localhost:1709/”
    />

  </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>
```

Description of the various sections within an XML-formatted configuration file follows.

Config Section

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

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

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 user names that may be used when you start up a controller session. Currently the controller will recognize only one session which must be named DEFAULT.

```
<session
        [ session option ]
/>
```

These session options provide an alternate way of specifying the options other than command line options at controller startup.

The following options are valid within this section:

controlPass	Specifies the password that will be used by default when connecting to an active server for control purposes. Note that this password must be a plain text password as it will be sent to the server to authorize the connection. Do not use an obfuscated password here.
password	Currently this has the same function as controlPass, however if both are present, controlPass will take precedence.

name	Name for this session description; must be DEFAULT
user	User name to use on connection
tracelevel	The sessions default trace level
srv.mcBasePort <base_port>	Specifies the base port number that will be used for multicast operations. The default is 5517.
srv.mcGroupIP <group_ip>	Specifies the multicast IP group that will be used for multicast operations. The default is 239.192.1.1.
ssl.*	Specifies SSL configuration information for the session that may be used to connect to SSL-enabled thin servers. See Using SSL for more information.

For 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:

The session options 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.

Databases Section

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

Specifies one or more database sections.

Database Section

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

Specifies a named database with the given properties.

Each database property comprises of a keyword and value combination:

- 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 equals sign (=)
- All option values must be enclosed in double quotation marks.

Database properties:

name	Is 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	Is the url that may be used to access this database.
driver	Is the class path of the preferred JDBC driver that may be used to access this database.
URLPrefix	Is the prefix that needs to be added to the url above to provide a complete JDBC Connection URL

For 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>
```

Servers Section

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

Specifies one or more server property sections.

Server Section

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

or

```
      <server
       <property=”value”/>
      >
      [ server subsection ]
      </server>
```

Specifies one or more properties to assign to this server.

See Server Configuration for details on the properties that may be set.

For example, 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”
/>
```

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.

> **Note**:
> 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.

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

Pooled Server Subsection

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

This subsection specifies a server that will take part in the parent servers server pool, where the declared server name must reference a server already named in this configuration file.

The subsection is valid only when used within an `RdbThinSrvPool` server declaration.

The set of pooledServers provided will make up the pool of servers that the parent pool server may try to access.

Allowed Database Subsection

```
      <allowDatabase name="pers"/>
```

This subsection specifies the database that clients using the server may access. 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.

The subsection is only valid when used within a server declaration.

See the section Restricting Database Access for more details

Using filenames in the configuration file

A number of 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.

Top of the Document

Using SSL

Secure Sockets Layer (SSL) was developed to provide security for Web traffic. Including confidentiality, message integrity, and authentication. SSL achieves this through the use of cryptography, digital signatures, and certificates.

SSL may be used by Oracle JDBC for Rdb servers and thin clients 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 your 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

Information about SSL connection characteristics must be provided to both the client and server, and in order 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

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 setup.

```
    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/my_db_dir:pers”,
info);
```

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.

```
    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/my_db_dir:pers”,
info);
```

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

Note:

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 your Java Security documentation for more information on certificates.

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

```
<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”
/>
```

If you wish to define a number of 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.

```
<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”
/>
```

See SSL Configuration Options for details of these options.

Note:

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.

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 a XML-formatted configuration file that has the appropriate SSL information in its SESSION section.

```
<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”
/>
```

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.

SSL Configuration Options

ssl.default	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 “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 specifed. 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. The default is false.

ssl.context <ssl context>	Indicates the SSL context to use, for example “TLS”. There is no default value.
ssl.keyManagerFactory <keymanager factory>	Indicates the keymanager factory to use, for example “SunX509”. There is no default value.
ssl.keyStoreType <store type>	Indicates the type of the key store, for example “jks”. There is no default value.
ssl.keyStore <store filename>	Indicates the filename of the keystore . There is no default value.
ssl.keyStorePassword <password>	Indicates the password for the keystore. There is no default value.
ssl.trustStore <trust store filename>	Indicates the filename of the trust store. There is no default value.
ssl.trustStorePassword <password>	Indicates the password of the trust store. There is no default value.

Using Self-Signed Certificates for Testing

The following code is an example that may be used to 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.

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

```
$! 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=Jim Murray, OU=Rdb Engineering, O=Oracle, c=US”
-keypass “CHANGETHIS” –storepass “CHANGETHIS” –KeyStore rdbjdbcsrv.kst
$!
$write sys$output “Exporting the certificate from keystore to external 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 external 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 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
$!
$!-----------------------------------------------------------------
$!
$! Now 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.

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

Once the keystores have been set up, as long as 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.

Top of the Document

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 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.

Running the Controller

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

Note:

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.

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

```
$java –jar rdbthincontrol.jar [<option> | <command_keyword>]
```

where <option> can be one of the following :

-cfg or –configfile <configuration_filename>	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>`	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>`	Specifies the node where the server to be connected to is running.
-oem	Used by OEM to indicate that the return status and messages should be formatted for OEM usage.
-p or –port <port_num>	Specifies the port on which the server to be connected to is listening.
-pw or –password <password>	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.
-srvargs <server_arguments>	Additional arguments to be passed on the connection URL when connecting to the server. For Example @tracelevel=-1

-srv.mcBasePort <base_port>	Specifies the base port number that will be used for multicast operations. The default is 5517.
-srv.mcGroupIP <group_ip>	Specifies the multicast IP group that this server will participate in. The default is 239.192.1.1.
-tl or –tracelevel <trace_level>	Specifies the default tracelevel for the session The default is tracelevel=0 (no tracing)
-u or –user <user_name>	Specifies the username to use for connection to the server.
-url <connection URL>	Specifies the node IP and port of the server to connect to. This switch overrides any *port* and *node* switch specified. The format of the <connection URL> is //<node>:<port>/

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.

<command_keyword> can be one of the following (in order of precedence) :

-poll	Sends a pool request out to locate active servers
-startserver	Starts the server as specified by the other options given on the command line
-openserver	Opens the server as specified by the other options given on the command line

-closeserver	Closes the server as specified by the other options given on the command line
-showserver	Issues the Show Server command which gets server information from the connected server
-showclients	Issues the Show Clients command which gets client information from the connected server.
-stopserver	Stops the server as specified by the other options given on the command line
-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

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.

An example of issuing command keyword to the controller:

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

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.

```
      $java -jar rdbthincontrol.jar
      rdbthincontrol>
```

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.

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.

Once a connection has been established the following commands can be issued:

close server	Closes the currently connected server. See Closing Servers for more details
disconnect	Disconects 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 of 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.
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.

show clients	Show all clients on the currently connected server
stop client <client_id>	Stops the client matching the specified <client_id> on the currently connected server.
stop clients	Stops all clients on the currently connected server.
watch [server]	Send trace logging from connected server to the current console. See Watching Servers for more details

For example:

```
    $java jar 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 : V7.1-300 20040624 B46N
RDB$SERVER_SHR_VERSION : V7.1-300 20040624 B46N
RDB$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
$
```

In addition, a number of 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 command are listed in the following table:

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 logfile for the identified 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 the value OFF then the current logfile will be closed and log messages will no longer be sent to the log file.
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.
show active servers show all servers show server <server_ident>	Show 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>	Opens active servers. See Watching Servers for more details.
quit or exit	Exits the controller application

For example:

```
$java -jar 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 : V7.1-300 20040624 B46N
RDB$SERVER_SHR_VERSION : V7.1-300 20040624 B46N
RDB$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 : V7.2-410 20060314 B63E
RDB$SERVER_SHR_VERSION : V7.2-410 20060314 B63E
RDB$SERVER_PID : 0x0B88(2952)
RDB$ALLOWS_ANON : false
RDB$ALLOWS_BYPASS : false
RDB$NUMBER_OF_CLIENTS : 0
RDB$MAX_CLIENTS : -1
rdbthincontrol>
```

If the provided control password is not recognized by a server, 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>
```

Connecting to Servers

The majority of 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 request 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.

```
<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.

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, 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.

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.

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

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

The following examples use the Connect command:

```
rdbthincontrol> connect //localhost:1701/ jones mypassword

rdbthincontrol> connect server 1701

rdbthincontrol> connect myServer jim xxxxx
```

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.

Implicit Connection

A number of the control commands require a control connection to be established with the target server. If the target server is not currently connected then both explicitly provided connection

information and session connection information is used to attempt to establish a control connection.

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

```
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.

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.

```
rdbthincontrol> stop server myMPServer

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

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.

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$NODE : 192.168.1.100
RDB$PORT : 1701
RDB$STATUS : Idle
RDB$SERVER_NAME : jiserv
RDB$SERVER_TYPE : RdbThinSrv
RDB$SERVER_VERSION : V7.2-410 20060314 B63E
RDB$SERVER_SHR_VERSION : V7.2-410 20060314 B63E
RDB$SERVER_PID : 0x1728(5928)
RDB$ALLOWS_ANON : false
RDB$ALLOWS_BYPASS : false
RDB$NUMBER_OF_CLIENTS : 0
RDB$MAX_CLIENTS : -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 the maxClients to 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.

The following control commands are available:

close active servers close all servers	Closes all responding servers.
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

Only those servers where the control password matches the control session control password will be stopped.

Examples:

```
thincontrol> close server myserv

thincontrol> close server //prod_node:1766/

thincontrol> close server 1701

thincontrol> close active servers

thincontrol> close server myserv george mySecretPassword
```

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.

The following control commands are available:

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

Only those servers where the control password matches the control session control password will be opened.

Examples:

```
thincontrol> open server
```

```
thincontrol> open server myserv

thincontrol> open server //prod_node:1766/

thincontrol> open server 1701

thincontrol> open all servers

thincontrol> open server //prod_node:1766/ fred mypass
```

You may issue a open command to re-open a closed server, which will reestablish the maxClient value for the server back to the value it was prior to closing.

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.

The following control commands are available:

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

Only those servers where the control password matches the control session control password will have information displayed.

Examples:

thincontrol> show server

thincontrol> show server myserv

thincontrol> show server //prod_node:1766/

thincontrol> show server 1701

thincontrol> show active servers

thincontrol> show server //prod_node:1766/ fred mypass

Starting Servers

Servers may be started using the controller.

The following control commands are available:

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.

Examples

thincontrol> start server myserv

thincontrol> start server 1799

Stopping Servers

Active servers may be stopped using the controller. You must provide a valid control password for the server.

The following control commands are available:

stop active servers stop all servers	Stops all responding servers.
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

Only those servers where the control password matches the control session control password will be stopped.

Examples:

```
thincontrol> stop server

thincontrol> stop server myserv

thincontrol> stop server //prod_node:1766/

thincontrol> stop server 1701

thincontrol> stop active servers

thincontrol> stop server //prod_node:1766/ fred mypass
```

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.

Showing Clients

Information about clients within active servers may be displayed using the controller. You must provide a valid control password for the server.

The following control commands are available:

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> show all clients in <database_spec>	Shows all clients currently connected to the specified database on all responding servers
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

Clients will only be displayed for those servers where the control password matches the control session control password.

Examples:

thincontrol> show active clients

thincontrol> show all clients fred

thincontrol> show clients

thincontrol> show clients in disk1:[dbc]pers

thincontrol> show all clients in disk1:[dbc]pers

Stopping Clients

Clients within active servers may be stopped using the controller. You must provide a valid control password for the server.

The following control commands are available:

stop active clients stop all clients	Stops all clients on responding servers.
stop active clients <name> stop all clients <name>	Stops all clients with user name <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 connect server that are currently connected to the specified database.
stop client <client_id>	Stops the specified client on the currently connected 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, then 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 Client output.

Examples:

thincontrol> stop active clients

thincontrol> stop all clients fred

thincontrol> stop clients

thincontrol> stop client 0000000A

thincontrol> stop all clients in disk1:[dbs]pers

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 Show Clients command to see clients that are currently using the server. See Showing Clients for more details.

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.

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 servers log file.

The following control commands are available:

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

Only those servers where the control password matches the control session control password will be watched

Examples:

thincontrol> watch server myserv

thincontrol> watch server //prod_node:1766/

thincontrol> watch server 1701 jack password1

thincontrol> watch

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.

Polling Servers

The poll command uses the multicast information to poll responding Oracle JDBC for Rdb servers:

```
rdbthincontrol> poll
Polling servers ...
myserver(0) //localhost:1711/ (0x0B88<2952>)
rdbthnsrv1(0) //localhost:1701/ (0x0B30<2864>)
rdbthincontrol>
```

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.

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 startup 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 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.

Top of the Document

Oracle SQL/Services and Oracle JDBC for Rdb Servers

The Oracle SQL/Service management command line may be used to start and stop servers.

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.

The format of the JDBC dispatcher creation statement is:

```
      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

For example:

```
$ MCR SQLSRV_MANAGE72
```

```
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:
```

Note:

The existing version of the Oracle SQL/Services Management GUI 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.

Starting a JDBC Dispatcher

Once you have defined a JDBC dispatcher, it can be started like any other Oracle SQL/Services dispatcher:

```
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$SYSROOT:[SYSMGR]SQS_DECRDB_JDBC_DISP06O72.LOG;
Dump File: SYS$SYSROOT:[SYSMGR]SQS_DECRDB_JDBC_DISP06O.DMP;
```

The Oracle SQL/Services monitor will attempt to start the server associated this dispatcher and create a log of the dispatcher events in the SYS$MANAGER directory in a log file named:

SYS$MANAGER:SQS_<nodename>_JDBC_DISP<nnnnn>.LOG

The <nodename> depends on the node the dispatcher is started up on.
The <nnnnn> is the unique id given to this dispatcher instance by Oracle SQL/Services

For example:

SQS_MALIBU_SQLSRV_DIS06010.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 similar to 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 21-JUL-2004 21:52:07.56

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
```

Stopping a JDBC Dispatcher

A running JDBC dispatcher may be stopped by using the standard STOP DISPATCHER statement.

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 your Oracle SQL/Services documentation for more information on the Oracle SQL/Services management console.

Relating 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.

The Port ID specified in the dispatcher creation is the key to this relationship.

The Port ID is used to determine the name of the Oracle JDBC for Rdb server using the following:

- If the logical name RDB$JDBC_SQSNAM_<port> exists then it is translated to provide the server name
- If the logical name does not exist the server name will be SQS<port>

For example, the following dispatcher description:

```
      SQLSRV> CREATE DISPATCHER JDBC_DISP NETWORK_PORT TCPIP PORT_ID 1888
PROTOCOL JDBC;
```

means that during dispatcher start up, Oracle SQL/Services will look for the logical name RDB$JDBC_SQSNAM_1888 to obtain the true server name. If this logical name is not defined, it will use SQS1888 as the server name.

Once the server name is determined, the appropriate configuration file is located using the following precedence:

1. RDB$JDBC_COM:<server name>_CFG.XML
2. RDB$JDBC_COM:SQLSRV_JDBC_SERVER_CFG.XML
3. RDB$JDBC_COM:RDBJDBCCFG.XML

Given the example shown above and assuming that the logical name RDB$JDBC_SQSNAM_1888 does not exist then the configuration file RDB$JDBC_COM:SQS1888_CFG.XML will be searched for first.

The server will then be started up using the configuration information for that named server in the appropriate configuration file.

Command Procedures

Oracle SQL/Services uses the OpenVMS command procedure

SYS$MANAGER:SQLSRV_JDBC_SERVER_STARTUP72.COM

which in turn uses

RDB$JDBC_HOME:RDBJDBC_STARTSRV.COM

to start the server associated with a JDBC dispatcher.

In addition, an OpenVMS command procedure can be defined to set up environmental characteristics required for your system. This SQS_ONSTARTUP 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

This command procedure will be executed just prior to the server being invoked and may contain environmental setup commands, for example:

```
$@sys$share:rdb$setver 72
$@sys$common:[java$142.com]JAVA$142_SETUP.COM
```

Determining Server Type

During the startup of the server associated with the Oracle SQL/Services JDBC dispatcher, the type of the server to startup also needs to be determined.

There are three types of Oracle JDBC for Rdb servers recognized by Oracle SQL/Services:

- POOL a pool server
- MP a multi-process server
- STD a standard thin server

The following precedence is used in determining the type of server

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

The type of server must be set correctly as this determines the appropriate the Oracle JDBC for Rdb server jar file that will be used.

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:

Note:
This example uses the default server naming, default server type of POOL and a standard SQS_ONSTARTUP command procedure. No *RDB$JDBC_SQS** logical names need be set up.

Define an Oracle SQL/Services dispatcher

```
$ MCR SQLSRV_MANAGE72
SQLSRV> CONNECT SERVER;
SQLSRV> CREATE DISPATCHER POOL_DISP NETWORK_PORT TCPIP PORT_ID 1880 PROTOCOL
JDBC;
```

Create a configuration file for this server in RDB$JDBC_COM:SQS1880_CFG.XML

```
<?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:1880/"
                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 specific servers that will be started up 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="SQS1880"
            type="RdbThinSrvPool"
            url="//localhost:1880/" >
            <pooledServer name="SQSrjs1"/>
            <pooledServer name="SQSrjs2"/>
            <pooledServer name="SQSrjs3"/>
        </server>
    </servers>
</config>
```

Create a onStartup command procedure that sets up the appropriate Rdb and Java versions for your system
eg RDB$JDBC_COM:RDBJDBC_SQS_ONSTARTUP.COM may contain

```
$@sys$share:rdb$setver 72
$@sys$common:[java$142.com]JAVA$142_SETUP.COM
```

Start the dispatcher

```
SQLSRV> start dispatcher pool_disp;
```

Note:

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 command procedure pointed to by the srv.onStartCmd switch is used by those servers that the pool server starts up.

The Oracle SQL/Services JDBC dispatcher does not directly use any information from the JDBC XML configuration file.

Top of the Document

Other Features

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 tells 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>
```

For example:

```
$ java -jar rdbthinsrv.jar "-anonymous" "-username" fred "-password" jones
```

Top of the Document

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 entitys privilege at the database is checked. If the entity has been granted **DBADM** it will be allowed to carry out the operation even if it does not have the explicit privilege such as **CREATE**. This privilege is a kind of catch all much 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).

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 users 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 must run is required to run with BYPASS privilege, thus allowing less privileged users access to the database objects, use the -bypass option

Top of the Document

Persona

When an Oracle JDBC for Rdb thin server is running, it assumes the default privileges and indentifiers 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.

Top of the Document

Default Transaction

The type of transaction the Oracle JDBC for Rdb drivers start up when a transaction is required depends on a number of conditions

- Whether autoCommit is enabled
- The verb of the SQL statement to be executed
- The default transaction type specified on connection using the connection switch transaction
- The setting of the transaction types in the connection if changed by methods such as

Connection.setReadOnly() and Connection.setTransactionIsolation().

If no specific behavior 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.

When AUTOCOMMIT is disabled, the type of transaction started will depend on whether the connection has been set read-only and is a default transaction type has been specified on the connection. 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.

Top of the Document

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 ir 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 enviroment 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:my_db_dir:pers@multiprocess=true",
user, pass);
```

Note that this switch is only available when you use the Rdb Native driver.

Top of the Document

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 changed the default FetchSize by using the FetchSize option, the default is 100.

Top of the Document

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 execessive messages being written to the application log file.

If your application does not depend on the statement cancellation to actually 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:my_db_dir:pers@ignoreStatementCancel=true",
user, pass);
```

Top of the Document

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.

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 down. 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.

For 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 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 specifed 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/my_db_dir: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.

Server Inactivity Timeout

You can specify the amount of time that a server may remain idle before being closed down 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 down. 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.

For 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 server is terminated by this timeout the following message will be logged in the server log:

```
Server terminated due to inactivity
2006-02-08 12:28:03.578 : Forced disconnect by Server terminated
due to inactivity @ LOCAL
```

A server inactivity timeout will occur if, for the length of time specified, no new client connection is made to that server. 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.

Thus the timeout value is the minimum time that the server will accept between new connection requests before closing down, but due to current server activity this may be extended until there are no more connections current.

Top of the Document

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()

Top of the Document

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 has to 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, in order to try to minimize the impact of one client thread on another you may specify the period of time 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 switch allows control of the duration of the wait for a lock, the minimum actual wait period being one (1) second, which is the minimum lock wait time supported by Rdb transactions.

A lockwait of 0 is the same as starting up a transaction with NOWAIT. A lockwait of 1 is the same as starting up a transaction with WAIT without specifying a value, which causes the server to wait

indefinitely,

The maxtries switch 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.

Top of the Document

Logging

Oracle JDBC for Rdb drivers and servers can now use the Java Logging utilities to log error messages and trace information.

By default Java Logging is turned off.

See your Java JDK 1.4.1 for information on the Java Logger.

Top of the Document

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 startup.

For example given the following entry in MY_CFG.XML file :

```
<servers>
     <server
           name="myMPServer"
           type="RdbThnSrvMP"
           url="//localhost:1788/"
     />
 </servers>
```

and the following command line statement:

```
$ java -jar rdbthnsrv.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.

Within the XML-formatted configuration, two special server names may be used, DEFAULT and DEFAULTSSL.

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.

For example given the following server entry in MY_CFG.XML file :

```
<servers>
      <server
            name = DEFAULT
            type = "RdbThnSrv"
            url = "//localhost:1755/"
            maxClients="-1"
      />
      <server
            name="myServer"
            maxClients="10"
      />
</servers>
```

and the following command line statement:

```
$ java -jar rdbthnsrv.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 RdbThnSrvSSL type servers.

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.

For example given the following server entry in MY_CFG.XML file :

```
<servers
   <server
          name = "DEFAULT"
          type = "RdbThnSrv"
          url = "//localhost:1799/"
          maxClients=-1
   />
</servers>
```

and the following command line statement:

```
$ java -jar rdbthnsrv.jar -cfg MY_CFG.XML -name "myServer"
```

A thin server with the name *myServe*r will be started up on localhost listening to port 1799 with unlimited maxClients.

Top of the Document

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.

Similar to 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 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.

For 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 database may also be used to restrict database access within the server. See Restricting Database Access for more information on this feature.

Top of the Document

On Start Commands

There are two startup command attributes that may be specified in the XML-formatted configuration file server section: srv.onStartCmd and srv.onExecStartCmd.

These options allow the specification of DCL command that should be executed just prior to the start up of a server or executor.

Note:

The srv.onStartCmd and the srv.onExecStartCmd point to a command that will be execute on startup of the server or executor. If the command is to invoke a DCL command procedure you must also include the DCL invocation symbol @ in the command line.

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.

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 similar to the following example.

Create rdb$jdbc_com:our_setup.com containing

```
$@sys$share:rdb$setver 72
$@sys$common:[java$142.com]JAVA$142_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"
/>
```

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.

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 place within the rdb$jdbc_com directory and the file protection set so that the server can access them.

For example, if your system requires a specific setup to be run to set your Oracle Rdb environment, you may create a command procedure similar to the following example.

Create rdb$jdbc_com:our_exec_setup.com containing

```
$@sys$share:rdb$setver 7.1
```

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"
/>
```

Top of the Document

Password Obfuscation in Server Configuration Files

To prevent an unauthorized user from controlling server operations such as closing down 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 passwords, the server configuration file may contain the server control password in an obfuscated form.

For example, in an XML-formatted server configuration file:

```
<server
        name="RdbThinSrv1707"
        type="RdbThinSrvMP"
        url="//localhost:1707/"
        srv.execStartup="mystartup"
        controlUser="jdbc_user"
        controlPass="0x811B15F866179583EB3C96751585B843"
/>
```

You can obtain an obfuscated password by using the Digest statement in the Rdb Thin Server Controller.

For example:

```
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.

This value must be copied exactly as returned by the digest statement.

Note:

> Obfuscated 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 carry out operations on it using the controller, the control password you use in the connect request must still be in plain text. You cannot use the obfuscated value as a password on connection.

Top of the Document

Restricting Database Access

You may restrict connections made via a server to only those databases specified as allowed databases.

This can 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.

```
<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.

For example 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.

```
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.

```
Connection conn = DriverManager.getConnection(
        "jdbc:rdbThin://bravo:1701/disk1:[databases]customers",user, pass);
```

will be allowed.

```
Connection conn = DriverManager.getConnection(
        "jdbc:rdbThin://bravo:1701/disk1:[databases]customers.rdb",user, pass);
```

will NOT be allowed due to the extra .rdb.

```
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`

Top of the Document

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 consistent semantics with the standard Oracle JDBC Drivers, a value of ORACLE may be specified within the TRANSACTION connection switch.

@transaction=oracle

Using this switch value, default transactions will be READ_WRITE but the transaction type may be changed by issuing the CONNECTION.setReadOnly(true) method call, which will set only the next transaction to READ_ONLY.

Top of the Document

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.

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 two command procedures used for startup, the server startup command procedure:

```
rdb$jdbc_home:rdbjdbc_startsrv.com
```

and the executor startup command procedure:

rdb$jdbc_home:rdbjdbc_startexec.com

Caution:

Do not use `SET VERIFY` within these command procedures. As the method Runtime.exec() is used by the Oracle JDBC for Rdb servers to create processes, the use of SET VERIFY within the command procedure will 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 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 or srv.onExecStartCmd should be considered. See On Start Commands for more details.

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.

For 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 startup 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 attributes instead.

> **Caution:**
> Do not use the `SET VERIFY` command within these command procedures. Because the method Runtime.exec() is used by the servers to create processes, the use of the SET VERIFY command within the command procedure will 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 server startup command procedure is only used by the controller or pool server 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.

Executor Startup Command Procedure

The thin 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.

For example:

```
<server
      name=MPsrv2forRdb
      type=RdbThinSrvMP
      url=//localhost:1788/
      srv.execStartup=rdb$jdbc_com:our_customized_startexec.com
/>
```

The `srv.execStartup` option is only valid within the XML-Formatted configuration file server section for a multi-process server.

Top of the Document

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.

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/clients 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
```

Top of the Document

SET Statements

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.

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 a SQL statement in the following methods

java.sql.Statement.execute

java.sql.Statement.executeUpdate

java.sql.Statement.executeQuery

For example:

```
Statement stmt = conn.createStatement();
stmt.execute(set sqlcache 10);
```

Top of the Document

SQL Statement Cache

When using the thin driver, performance may be improved by enabling SQL statement caching.

Whenever the thin driver need to prepare a SQL statement, the statement must be sent over the network to the server for Oracle Rdb to prepare the statement and 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 the preparation and 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.

- 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);
```

- Append @sqlcache to the database specification part of the connect URL:

  Connection conn = DriverManager.getConnection(
  jdbc:rdbthin://bravo:1701/my_db_dir:personnel@sqlcache=100,user, pass);

In addition a SET SQLCACHE statement can be executed.

```
Stmt.executeUpdate(set sqlcache 100);
```

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 is 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 SQL Statement cache property or using the set sqlcache statement will be silently ignored by the native driver.

Top of the Document

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)
```

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.

Trace of JDBC operations may be enabled in one of the following ways:

Set the tracelevel property of the Properties passed to the DriverManager.getConnection method to the appropriate value:

```
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.

Starting a server with the tracelevel switch

```
$java -jar 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.

Placing the tracelevel option in the server definition within an XML-Formatted configuration file.

```
<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.

Using the Rdb system property **Doracle.rdb.jdbc.tracelevel** when invoking your application or Rdb server

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

See Oracle JDBC for Rdb System Properties for more details.

Using the SET TRACELEVEL command in the ThinController.

```
$java jar rdbthincontrol.jar
rdbthincontrol> connect //localhost:1701/ jones mypassword
```

```
rdbthincontrol> set tracelevel 1
```

See Controller Command Line for more details.

Abbreviated form

The abbreviated form for traceLevel (tl) may also be used in the same manner.

Trace Values

The value passed to trace is actually a 32bit flag mask. Each bit set determines what will be traced, as shown in the following table.

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
29	0x20000000	536870912	memory information
30	0x40000000	1073741824	full provides more details on certain flags
(ALL)	0xFFFFFFFF	-1	trace everything

Top of the Document

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.

Enhanced Blob Handling

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.

To enable limited formatting of data returned from Oracle Rdb segmented strings, a new public method has been added to oracle.jdbc.rdb.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.

```
public void oracle.jdbc.rdb.common.Blob.setSegSeparator(java.lang.String
separator)
```

Specify the separator string to use between segmented string segments:

Parameters:

separator - separator string to use

Returns:

Void

The following code segment shows how to add a newline break between segments.

```
      ResultSet rs = s.executeQuery(
                    "select resume from resumes where employee_id = '00164'");
      rs.next();
      oracle.jdbc.rdb.common.Blob bl =
(oracle.jdbc.rdb.common.Blob)rs.getBlob(1);
                    bl.setSegSeparator("\n");
      byte[] bytes = bl.getBytes(1,9999);
      String st1 = new String(bytes);
    System.out.println("resume : " + st1 );
```

The separator can be cleared by passing either a null object or empty String as the parameter to oracle.jdbc.rdb.common.Blob.setSegSeparator().

ResultSet.getBytes()

The JDBC standard limits the use of the ResultSet.getBytes() methods for access to BINARY, VARBINARY and LONGVARBINARY data. The Oracle JDBC for Rdb drivers relax this limitation and will attempt to return byte arrays for all valid SQL datatypes 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.

Top of the Document

Appendix 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

Top of the Document

Datatype Mapping from Oracle Rdb 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

Top of the Document

Datatype Mapping from java.sql.Types to Oracle Rdb

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

Top of the Document

JDBC Specification SQL to Java Datatype Mappings

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

Top of the Document

JDBC Specification Java to SQL Datatype Mappings

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

Top of the Document