Oracle JDBC for Rdb Release Notes

Release V7.2-410
March 2006

This file provides information about how to install and use Oracle JDBC for Rdb.

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.

Oracle JDBC for Rdb Native Driver

The Oracle JDBC for Rdb native driver is a Type II driver for use with client-server Java applications. This driver requires an Oracle Rdb client installation, and therefore is Oracle Rdb-

specific and not suitable for applets. The driver uses the Oracle Rdb libraries on the client machine on which it is installed.

The Oracle Rdb 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.

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

> **Note**:
> When the Oracle JDBC for Rdb thin driver is used with an applet, the client browser must have the capability to support Java sockets.

Oracle JDBC for Rdb Thin Server

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

The server is multi-threaded, able to handle multiple client requests at the same time. However as the server is maintained as a single OpenVMS process, database access for each of the threads must be 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 JDBC for Rdb thin driver using Java sockets over TCP/IP with the default PORT id 1701.

Oracle JDBC for Rdb Pool Server

The Oracle JDBC for Rdb pool server is a server-side component that accepts connection requests from the Oracle JDBC for Rdb thin driver and redirects the requests to the next available Oracle JDBC for Rdb thin 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 thin server. The pool server selects the next available thin 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 thin 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 thin servers. Rdb need not be present on these nodes, as the pool server does not communicate directly with 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.

Oracle JDBC for Rdb Multi-process Server

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

The Oracle Rdb 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 server operations are carried out in a client thread context within the main server process, handing off control to the client's allocated subprocess only when direct 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.

The allocated subprocess is terminated on client disconnect.

A multi-process 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 thin driver using Java sockets over TCP/IP with the default PORT id 1701.

Oracle JDBC for Rdb Controller

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

Contained in the rdbthincontrol.jar, this application allows remote password-protected server management operations to be carried out on a thin server or pool server, such as showing the clients that are currently connected, stopping client threads and starting and stopping servers.

Top of the Document

System Requirements

Oracle JDBC for Rdb requires the following software products and versions to be installed:

Software	Minimum Version
HP OpenVMS for Alpha Servers	V7.3-2

HP OpenVMS Industry Standard 64 for Integrity Servers	X8.2-AV1
HP *Java*™ SDK/RTE for OpenVMS Alpha	V1.4-1
HP *Java*™ SDK/RTE for OpenVMS I64	V1.4-21
Oracle Rdb	V7.2

On the client side, the use of the Oracle JDBC for Rdb thin driver requires the following software products and versions to be installed:

Software	Minimum Version
Java™ SDK/RTE	V1.4.1

In addition, if you require the ability to start and stop Rdb JDBC for Rdb servers using Oracle SQL/Services the following product and version must be installed.

Software	Minimum Version
Oracle SQL/Services	V7.1.6

Detailed information about installing the HP Java for OpenVMS system may be found at the following web site:
http://www.hp.com/java.

Documentation for HP's Java for OpenVMS system may be found at the following web sites:
http://www.compaq.com/java/documentation/index.html - Java 2.
http://h18012.www1.hp.com/java/documentation/index.html

In line with HP recommendations for Java applications, Oracle recommends the following minimum quota setting on accounts used to start up thin servers, in particular those used to start multi-process servers.

UAF Fillm 4096

Channelcnt 4096

Wsdef	2048
Wsquo	4096
Wsextent and Wsmax	16384
Pgflquo	1500000
bytlm	1000000
biolm	150
diolm	150
tqelm	100

Remember to ensure that you system's quotas are set appropriately to accommodate these process quotas.

See the Java for OpenVMS release notes for more information on OpenVMS quotas and resources required by Java.

Also refer to your Oracle Rdb documentation for recommendations on OpenVMS quotas required for Oracle Rdb.

Top of the Document

New or Changed Features

Client and Server timout feature

You can now specify the amount of time a server or a client connection may remain inactive before the connection will be terminated or the server closed down.

See the Oracle JDBC for Rdb User Guide for details.

Executor name prefix

You can now specify the name prefix for executors started up by the Multi-process server. This may help in easily identifying executor process on your syste,

See the Oracle JDBC for Rdb User Guide for details.

Top of the Document

Installation of The Oracle JDBC for Rdb Kit

Contents of the Oracle JDBC for Rdb Kit

The Oracle JDBC for Rdb kit uses OpenVMS Polycenter to simplify the installation of the product.
Please refer to your OpenVMS documentation on the use of OpenVMS Polycenter.

The Oracle JDBC for Rdb kit Product installation file

ORCL-AXPVMS-RDBJDBC72-x0702-xxxxxx-1.PCSI - for OpenVMS Alpha or
ORCL-I64VMS-RDBJDBC72-x0702-xxxxxx-1.PCSI - for OpenVMS I64

is contained in either the file RDBJDBC72xxxx.ZIP that is an OpenVMS ZIP file if you obtained it from the Web or in the RDBJDBC directory of the Rdb Software distribution CD.

The installation kit is comprised of the following files:

BUILD_CERTS.COM	Command procedure example of building certificates for SSL
RDBJDBCCHECKUP.CLASS	Use to verify the installation of this kit
RDBJDBCCHECKUP.JAVA	Use to verify the installation of this kit
RDBJDBCEXEC72.EXE	Executor image in conjunction with a multi-process server
RDBJDBCCFG.XML	Example XML formatted configuration file
RDBJDBCSHR72.EXE	Shared image required for Oracle Rdb database access
RDBJDBCMPSHR72.EXE	Shared image required for multi-process Oracle Rdb database access
RDBJDBC_EXECCLI.COM	CLI helper command procedure

RDBJDBC_INSTALL.COM	Installation command procedure used by Polycenter during installation
RDBJDBC_STARTEXEC.COM	Command procedure used by Oracle JDBC for Rdb multi-process server to start up an executor process
RDBJDBC_STARTSRV.COM	Command procedure used when Oracle JDBC for Rdb servers are started up from the Oracle JDBC for Rdb controller
RDBNATIVE.JAR	Java Jar file containing the classes for the Oracle JDBC for Rdb native driver
RDBTHIN.JAR	Java Jar file containing the classes for the Oracle JDBC for Rdb thin driver
RDBTHINSRV.JAR	Java Jar file containing the classes for the Oracle JDBC for Rdb servers
RDBTHINCONTROL.JAR	Java Jar file containing the classes for the Oracle JDBC for Rdb controller
RDBTHINSRVPOOL.JAR	Java Jar file containing the classes for the Oracle JDBC for Rdb pool server
RDBJDBC_FAQ.TXT(HTML)	Frequently asked questions
NATIVEDRIVERSANDJDEV.TXT(HTML)	How to change Oracle Developer to handle the Oracle JDBC for Rdb drivers.
RDBJDBC_RELNOTES.TXT(HTML,PDF)	This file
RDBJDBC_USERGUIDE. HTML(PDF)	User guide.
SQLSRV_JDBC_SERVER_STARTUP72.COM	Command procedure used by Oracle SQL/Services to start up an Oracle JDBC for Rdb server

Installation

Follow these steps to install the Oracle JDBC for Rdb kit:

1. If you obtained the kit in ZIP format, restore the kit file to a temporary directory, for example:

   ```
   $ unzip RDBJDBCT72xxxx.ZIP -d MY_DIR
   ```

   This will unzip the Polycenter kit for Oracle JDBC for Rdb

ORCL-AXPVMS-RDBJDBC72-x0702-xxxxxx-1.PCSI – for OpenVMS Alpha or
ORCL-I64VMS-RDBJDBC72-x0702-xxxxxx-1.PCSI – for OpenVMS I64

'x' will be "T" if this is a BETA release otherwise it will be "V" and 'xxxxxx' will be the ECO and build instance of this kit, for example:

ORCL-AXPVMS-RDBJDBC72-V0702-4V0616-1.PCSI

2. Use the Polycenter PRODUCT command to install the kit.

Details of the version of the kit will be displayed and you will be prompted whether you want to proceed.
The following examples assume the kit build instance is '4V0616', and that the directory where the PCSI file can be found is 'MY_DIR'.

```
$ PRODUCT INSTALL RDBJDBC72/SOURCE=MY_DIR

The following product has been selected:
    ORCL AXPVMS RDBJDBC72 V7.2-4V0616        Layered Product [Installed]

Do you want to continue? [YES]

Configuration phase starting ...

You will be asked to choose options, if any, for each selected product and
for
any products that may be installed to satisfy software dependency
requirements.

ORCL AXPVMS RDBJDBC72 V7.2-4V0616: Oracle JDBC for Rdb

    Copyright © 1995, 2006,Oracle Corporation.  All Rights Reserved.

* This product does not have any configuration options.

Execution phase starting ...

The following product will be installed to destination:
    ORCL AXPVMS RDBJDBC72 V7.2-4V0616        DISK$AXPVMSSYS:[VMS$COMMON.]

Portion done: 0%...10%...30%...40%...50%...60%...70%...80%...90%
```

```
Oracle JDBC for Rdb has been successfully installed in :

DISK$AXPVMSSYS:[SYS1.SYSCOMMON.rdb$jdbc.0702-4V0616]

To help you setup the required logical names, a file named
  RDBJDBC_STARTUP.COM has been added to this installation directory

RDBJDBC_STARTUP.COM:
$! Oracle JDBC for Rdb startup command procedure
$!
$ DEFINE/SYSTEM RDB$JDBC_HOME DISK$AXPVMSSYS:[SYS1.SYSCOMMON.rdb$jdbc.0702-
4V0616]
$ DEFINE/SYSTEM RDB$JDBC_LOGS DISK$AXPVMSSYS:[SYS1.SYSCOMMON.rdb$jdbc.logs]
$ DEFINE/SYSTEM RDB$JDBC_COM  DISK$AXPVMSSYS:[SYS1.SYSCOMMON.rdb$jdbc.com]
$ DEFINE/SYSTEM RDBJDBCSHR RDB$JDBC_HOME:RDBJDBCSHR72.EXE
$ DEFINE/SYSTEM RDBJDBCMPSHR RDB$JDBC_HOME:RDBJDBCMPSHR72.EXE
$ DEFINE/SYSTEM RDBJDBCEXEC RDB$JDBC_HOME:RDBJDBCEXEC72.EXE

...100%

The following product has been installed:
ORCL AXPVMS RDBJDBC72 V7.1-4V0616        Layered Product
```

The installation procedure will copy all the kit files to the appropriate Oracle JDBC for Rdb product directory within the sys$common:[rdb$jdbc] directory:

If not already present the installation procedure will create two new directories, sys$common:[rdb$jdbc.logs] and sys$common:[rdb$jdbc.com] and if not already present will copy the rdbjdbccfg.xml found in this installation to the sys$common:[rdb$jdbc] directory.

```
$ dir sys$common:[rdb$jdbc]/col=1

Directory SYS$COMMON:[RDB$JDBC]

0702-4V0616.DIR;1
COM.DIR;1
LOGS.DIR;1
RDBJDBCCFG.XML;1
```

In addition, the command procedure SQLSRV_JDBC_SERVER_STARTUP72.COM will be copied to the system specific SYS$MANAGER directory.

3. Define the required system logical names

RDB$JDBC_HOME to point to the installation home
RDB$JDBC_LOGS to point to the Oracle JDBC for Rdb log directory
RDB$JDBC_COM to point to the Oracle JDBC for Rdb command directory
RDBJDBCSHR to point to the shared image RDBJDBCSHR72.EXE.
RDBJDBCMPSHR to point to the shared image RDBJDBCMPSHR72.EXE.
RDBJDBCEXEC to point to the shared image RDBJDBCEXEC72.EXE.

```
$ define/system RDB$JDBC_HOME –
SYS$COMMON:[RDB$JDBC.0702-4V0616]
$ define/system RDB$JDBC_LOGS –
SYS$COMMON:[RDB$JDBC.LOGS]
$ define/system RDB$JDBC_COM –
SYS$COMMON:[RDB$JDBC.COM]
$ define/system RDBJDBCSHR –
SYS$COMMON:[RDB$JDBC.0702-4V0616]RDBJDBCSHR72.EXE
$ define/system RDBJDBCMPSHR –
SYS$COMMON:[RDB$JDBC.0702-4V0616]RDBJDBCMPSHR72.EXE
$ define/system RDBJDBCEXEC –
SYS$COMMON:[RDB$JDBC.0702-4V0616]RDBJDBCEXEC72.EXE
```

A command procedure named RDBJDBC_STARTUP.COM defining these logical names for this installation can be found in the Oracle JDBC for Rdb product installation directory.

The RDB$JDBC_HOME logical name is required to be defined if you wish to use a thin multi-process server or use the thin controller or pool servers to start server processes.

4. Include the rdbnative.jar and rdbthin.jar files in your Java CLASSPATH by using either the logical names CLASSPATH or JAVA$CLASSPATH or the -classpath option on the Java command line.

```
$ define JAVA$CLASSPATH
[],RDB$JDBC_HOME:RDBNATIVE.JAR,RDB$JDBC_HOME:RDBTHIN.JAR
```

5. Test your installation using the "RdbJdbcCheckup" Java class that can be found in the RDBJDBCCHECKUP.CLASS file. During installation RDBJDBCCHECKUP.CLASS will be copied to RDB$JDBC_HOME.

Copy this file to your default directory and then you can invoke it using Java.

You will be prompted for a username and password and an Oracle Rdb database to test the installation against. If the test succeeds, the text "Your JDBC installation is correct." is

displayed.

```
$ java "RdbJdbcCheckup"
Please enter information to test connection to the database user:
password:
database: my_db_dir:personnel
Connecting to the database...Connecting…
connected.
Hello World
Your JDBC installation is correct.
$
```

Test the thin server by using the following commands:

```
$spawn/nowait/proc=rdbthinsrvtest java -jar rdbthinsrv.jar
$java "RdbJdbcCheckup" "-t"
Please enter information to test connection to the database user:
password:
database: my_db_dir:personnel
Connecting to the database...Connecting...
connected.
Hello World
Your JDBC installation is correct.
$stop rdbthinsrvtest
```

Note:
Since Java is a case-sensitive language it is important to specify class and method names exactly as they are described in the various APIs. By default, the OpenVMS operating system uppercases command line parameters unless you surround them with double quotation marks.

Top of the Document

Known Problems and Limitations

This section describes known problems and limitations in this release.

Known Problems

Using PreparedStatement and Parameter Markers.

problem

During the creation of a prepared statement using the Connection.PrepareStatement() method, the Oracle JDBC for Rdb drivers call Oracle Rdb SQL to compile the Sql statement and describe its select fields and parameter markers.

At this time SQL builds internal message representations of the parameter markers that may be passed to Oracle Rdb when the prepared statement is executed.

The maximum size of character values that may be passed using each parameter marker is fixed by SQL at this stage.

This may cause inconsistent results when the application attempts to use character string values that are longer than the maximum size determined by SQL for that parameter. If the input value is longer, the value will be truncated by SQL prior to being sent to Rdb for processing.

This does not pose any problems if the query selection is equality, however other Boolean comparisons may cause unexpected results.

For example

```
        Statement stmt = conn.createStatement();
        stmt.execute("create table tab (f1 char(3))");
        stmt.execute("insert into tab values ('123')");
        PreparedStatement ps;

         ps = conn.prepareStatement( "select f1 from tab where f1 like ?");
         ps.setString(1, “123”);
         ps.execute();

--- will return the record

         ps.setString(1, “%123”);
         ps.execute();

--- will not return the record
```

The reason the above query fails is that SQL will set the maximum size of the parameter text

string to 3 characters (i.e. the size of field F1) and the input value will be truncated to “%12” before being sent to Oracle Rdb thus it will not match with the record.

A work around is to force SQL to allocate a parameter size that is large enough to hold the input values that the application may use, by altering the input query.

For example:

```
 ps = conn.prepareStatement( "select f1 from tab where cast (f1 as VARCHAR(4))
like ?");
```

will now return the correct value.

We are currently investigating ways of fixing this problem.

Top of the Document

Using Java Fast VM on OpenVMS.

problem

Using Java Fast VM on OpenVMS when starting up thin servers may limit the number of clients a single server may be able to handle concurrently as using the Fast VM drastically reduces the amount of certain system memory that the Oracle Rdb sub-system has access to.

The usual symptom of running out of memory due to this situation is for the server process to issue COSI-VASFULL errors.

Please refer to the OpenVMS Java documentation on using Fast VM for suggestions on how memory usage may be altered.

Heap size used by the Java VM is important in determining how much memory will be pre-allocated by the Java VM.

You can set the size of the heap using the -Xmx option. By default, the Fast VM looks at your quota and the size of physical memory on the system to decide how large a heap to give you. So if both are very large, you may wind up with a larger heap than you really need.

You can use -verbosegc on the command line of the command used to start a server to see the current heap size.

Memory usage may also be altered by using the "-Xglobal" switch.

If the thin servers are getting COSI-VASFULL errors when Fast VM is enabled then we suggest trying the following switch settings as a first pass at rectifying the problem.

```
$ java "-Xmx24m" "-Xglobal120m" –jar rdbthinsrv.jar
```

Top of the Document

Using the Oracle SQL/Services Management GUI and JDBC Dispatchers.

Problem

The existing version of the Oracle SQL/Services Management GUI does not recognize Dispatchers of the type JDBC.

Unfortunately, this means that you will no longer be able to use the GUI once a JDBC dispatcher has been defined.

Removing the JDBC dispatcher from your Oracle SQL/Services definitions will alleviate this problem.

Top of the Document

DEC_KANJI and DEC_HANZI support not yet complete

Problem

Although some support has now been added to V7.1.3 Oracle JDBC for Rdb Drivers for accessing

DEC_KANJI and DEC_HANZI data from Oracle Rdb databases this support is still in the process of being fully tested, especially when used in conjunction with SHIFT_JIS on PC platforms.

Oracle currently advises against using Oracle JDBC for Rdb drivers to access DEC_KANJI and DEC_HANZI data from Oracle Rdb databases.

This limitation will be lifted in a future maintenance release of this product.

Top of the Document

Limitations

- The following JDCB 2.0, JDBC 3.0 and JDK 1.4 methods are not currently supported:

  Blob.setBytes
  Blob.setBinaryStream
  Clob.setString
  Clob.setAsciiStream
  Clob.setCharacterStream
  Clob.truncate
  Connection.setSavepoint
  Connection.rollback(savepoint)
  Connection.releaseSavepoint
  DatabaseMeteData.getSQLKeywords
  PreparedStatement.setRef
  PreparedStatement.setArray
  PreparedStatement.setNull(int,int,String)
  PreparedStatement.setURL(int,URL)
  ResultSet.getRef
  ResultSet.getArray
  ResultSet.updateAsciiStream
  ResultSet.updateBinaryStream
  ResultSet.updateCharacterStream
  ResultSet.updateRef
  ResultSet.updateArray
  ResultSet.rowUpdated

ResultSet.rowInserted
ResultSet.rowDeleted
ResultSet.updateBytes
Statement.cancel
Statement.setQueryTimeout
Statement.getMoreResults
Statement.executeUpdate(String sql, int autoGeneratedKeys)
Statement.executeUpdate(String sql, int[] columnIndexes)
Statement.executeUpdate(String sql, String[] columnNames)
Statement.execute(String sql, int autoGeneratedKeys)
Statement.execute(String sql, int[] columnIndexes)
Statement.execute(String sql, String[] columnNames)

- The following features or datatypes in JDBC 2.0 and JDBC 3.0 are not supported:

Array
Ref
Clob
User Defined datatypes
Scroll cursors
Savepoints

Auto-generated keys

- The total number of markers and fields allowed in a single SQL statement is 250.

- String truncation warnings:
  - The Oracle JDBC for Rdb drivers follow the SQL-92 rules for string truncation that differ depending on whether it is a store or a retrieval.
  - If a string truncation happens during a store operation, Oracle Rdb signals the error RDB$_TRUN_STORE, unless all of the truncated characters are spaces, in which case there is no error. If a string truncation happens during a retrieval, Oracle Rdb signals the SQL warning RDMS$K_SQLCODE_TRUNCWARN.

- Numeric and string functions in JDBC
  - A number of JDBC standard Numeric and String functions are not supported within Oracle Rdb unless you have previously prepared the database for use with Oracle SQL*Net for Rdb using the sql_functions.sql script. Refer to the Oracle SQL/Services documentation for more details on using this script.

- Blobs will only be returned correctly from a SQL join statements for the first table

mentioned in the join set.
For example:

Given the SQL statement

```
Select ta.blob, tb.blob from table1 ta, table2 tb where
ta.name = tb.name
```

ta.blob will be returned correctly as it is from the first table referenced in the join set, trying to access tb.blob may result in the following SQL error:

%SQL-F-BADPREPARE, Cannot use DESCRIBE or EXECUTE on a statement that is not prepared

Top of the Document

Problems Fixed in This Release

Release Notes specify incorrect Installation directory for RDBJDBCCFG.XML

fixed (Build 20060130)

problem

The release notes for the Oracle JDBC for Rdb incorrectly specified that the RDBJDBCCFG.XML file will be copied to SYS$COMMON:[RDB$JDBC] directory.

The RDBJDBCCFG.XML is actually coped to two directories during product installation.

- The product installation directory found under the main JDBC directory, for example, SYS$COMMON:[RDB$JDBC.0701-4V0614]
- The SYS$COMMON:[RDB$JDBC.COM] directory

In addition contary to the documentation the installation procedure incorrectly replaces the RDBJDBCCFG.XML file in the SYS$COMMON:[RDB$JDBC.COM] directory overwriting any already existing file of the same name.

The release notes have been fixed, and the installation procedure will only copy the RDBJDBCCFG.XML file to the SYS$COMMON:[RDB$JDBC.COM] directory if the file does not already exist in that directory.

Top of the Document

Persona not handled correctly by the Multi-Process and Pool servers.

fixed (Build 20060130)

problem

When the Persona feature is used in conjuction with a Multi-Process or Pool server, a problem in the way either the executor processes or the pooled server processes are created prevented the correct Persona identification from being passed to the created processes. This problem may result in the following error being raised.

java.io.IOException: Child creation error: not owner

Due to a restriction in the use of the JAVA System.exec() method that was used by the JDBC servers to start executor sub-processes and pooled servers, the security information and Persona details were not copied across to the newly created process.

The JDBC servers have now changed the method by which processes are started up and now use the OpenVMS system service CREPRC . CREPRC correctly transfers the security information to the new process.

Top of the Document

Multi-Process Server / Executor handhake timeout may be too short on heavily loaded systems

fixed (Build 20060130)

problem

When a Multi_process server talks to an executor it uses a handshake protocol to check that the executor is still alive and accepting direction. By default if the executor has not responded to the servers synchronization request within 5 seconds it will raise the following exception and terminated the connection :

```
Lost connection to executor
```

This synchonization handshake is done after the executor has replied to server that it has completed the task requested and is waiting for the next operation to carry out. So this synchonization failure will not be raised while executor is busy within the database and thus is unaffected by such things as database locks or the duration required to compile or execute queries. It will only occur when the executor is known to be waiting for the next action to carry out.

In heavily loaded systems especially on single-cpu systems it is possible that the executor process may not be schedule for execution within the window of this synchronization handshake and consequently the exception may be raised

In previous version of the drivers in order to carry out this synchronization the server polls the executor upto 500 times with a 10 millisecond delay between each poll request. If no response is found after 500 tries the server will raise the above exception.

This version of the Oracle JDBC for Rdb drivers now allows you to specify, at the server level, the maximum number of poll tries and the delay between each try, so that if you know that the system on which the server is executing could possibly have extended process scheduling delays, that the server will not timeout on the synchronization handshake.

Two new switches have been added to the server definition and startup.

Srv.MPmaxTries
Srv.MPtryWait

See the Oracle JDBC for Rdb User Guide for more information.

Top of the Document

Problems with srv.idleTimeout and srv.bindTimout configuration variables and their use with SSL servers.

fixed (Build 20060208)

problem

The Oracle JDBC for Rdb User Guide for version V7.1-3* incorrectly referred to the srv.idleTimeout as affecting the inactivity timeout for a connection.

This switch actually refers to the timeout period for server inactivity.

In addition this feature was not fully functional in previous versions.

The srv.bindTimeout configuration variable was meant to limit the time the server will wait for an acknowledgement from the client that the database attach should proceed.
The default value is 0 which means that the server will wait indefinitely.

This timeout is useful when deailing with SSL communication , as the server uses it to limit the time it will wait after a new socket connection has been requested for the client to send down an attach request. If the client fails to use SSL secure socket when trying to communicate with a server that has SSL enabled, then the cleint thread within the server will hang as the connection cannot complete. The srv.bindTimeout value specifies how long this wait should be before giving up.

Unfortunately the default for the value was incorrectly set to 1 second.and the srv.bindTimeout server attribute was ignored in the XML configuration file.

This meant that on CPU-bound systems, it was possible that the initial SSL negotiation could take longer than 1 second and thus cause a TIMOUT failure on the new connection request.

These problems have now been fixed.

See the Oracle JDBC for Rdb User Guide for more information.

Top of the Document

IA64 problem causes Array out of bounds exception when handling string indexing

fixed (Build 20060208)

problem

A problem in the way JAVA on IA64 carries out string index operations in association with static final String constants may infrequently cause the following type of exception to be raised

```
Caused by: java.lang.ArrayIndexOutOfBoundsException: 1054649176
at java.lang.String.indexOf(String.java:1266)
at java.lang.String.indexOf(String.java:1236)
at java.lang.String.indexOf(String.java:1218)
at
oracle.rdb.jdbc.common.Statement.getTableName(Statement.java:3148)
```

In all cases the index value as shown after the exception name is very large in the same order of magnitude as seen above.

The getTableName method has now been changed to a mechanisn other than indexOf to carry out its operation. This problem should no longer be seen.

Top of the Document

Comments within SQL text not handled correctly

fixed (Build 20060301)

problem

Executing or preparing a statement that has SQL text containing leading or embedded comments may cause errors during parsing of the statement.

Some third-party products may use comments such as **/* comment** */ in the text they send down to the JDBC drivers for compilation. Comments of this style although handled correctly by Rdb, caused a problem in the determination of statement types during the preliminary parsing of the statement by the JDBC driver.

For example the following SQL text

```
Stmt.Execute("/* This is a comment */ select * from jobs");
```

would cause an SQLException:

```
SQLException: in <rdbjdbcsrv:execute_immediate>
%SQL-F-EXESELSTA, Attempted to EXECUTE a SELECT statement:RR000
```

The JDBC driver could not correctly determine the type of statement and used the wrong underlying SQL operation to attempt to execute it.

The drivers now extracts out comments prior to determining the statement type and sending the native SQL down to Rdb.

The drivers will now correctly parse out C and SQL type comments.

For example:

/* comment */

! this comment will be terminated at the next line break

-- this comment will be terminated at the next line break

// this comment will be terminated at the next line break

Top of the Document

Prepared statements may cause a memory leak with Multi-Process Servers.

fixed (Build 20060301)

`problem`

During the preperation of PreparedStatements, the Multi-Process server has to allocate memory from the Servers global shared memory pool that will hold some information about columns and parameter markers in the statement that is being prepared.

Due to a coding problem, some of this memory was incorrectly allocated each time the propared statement was executed, instead of only once at statement compilation time and this wrongly allocated memory was never freed after use.

Thus executing the same prepared statement multiple times will slowly diminish the shared memory available to the server, eventually causing a problem when the shared memory allocation is all used up.

This has now been fixed.

Top of the Document