Mobicents JAIN SLEE JDBC Resource Adaptor User Guide

by Eduardo Martins

1. Document Conventions

This manual uses several conventions to highlight certain words and phrases and draw attention to specific pieces of information.

In PDF and paper editions, this manual uses typefaces drawn from the *Liberation Fonts* [https://fedorahosted.org/liberation-fonts/] set. The Liberation Fonts set is also used in HTML editions if the set is installed on your system. If not, alternative but equivalent typefaces are displayed. Note: Red Hat Enterprise Linux 5 and later includes the Liberation Fonts set by default.

1.1. Typographic Conventions

Four typographic conventions are used to call attention to specific words and phrases. These conventions, and the circumstances they apply to, are as follows.

`Mono-spaced Bold`

Used to highlight system input, including shell commands, file names and paths. Also used to highlight key caps and key-combinations. For example:

> To see the contents of the file `my_next_bestselling_novel` in your current working directory, enter the `cat my_next_bestselling_novel` command at the shell prompt and press **Enter** to execute the command.

The above includes a file name, a shell command and a key cap, all presented in Mono-spaced Bold and all distinguishable thanks to context.

Key-combinations can be distinguished from key caps by the hyphen connecting each part of a key-combination. For example:

> Press **Enter** to execute the command.
>
> Press **Ctrl**+**Alt**+**F1** to switch to the first virtual terminal. Press **Ctrl**+**Alt**+**F7** to return to your X-Windows session.

The first sentence highlights the particular key cap to press. The second highlights two sets of three key caps, each set pressed simultaneously.

If source code is discussed, class names, methods, functions, variable names and returned values mentioned within a paragraph will be presented as above, in `Mono-spaced Bold`. For example:

> File-related classes include `filesystem` for file systems, `file` for files, and `dir` for directories. Each class has its own associated set of permissions.

Proportional Bold

This denotes words or phrases encountered on a system, including application names; dialogue box text; labelled buttons; check-box and radio button labels; menu titles and sub-menu titles. For example:

> Choose **System > Preferences > Mouse** from the main menu bar to launch **Mouse Preferences**. In the **Buttons** tab, click the **Left-handed mouse** check box and click **Close** to switch the primary mouse button from the left to the right (making the mouse suitable for use in the left hand).
>
> To insert a special character into a **gedit** file, choose **Applications > Accessories > Character Map** from the main menu bar. Next, choose **Search > Find** from the **Character Map** menu bar, type the name of the character in the **Search** field and click **Next**. The character you sought will be highlighted in the **Character Table**. Double-click this highlighted character to place it in the **Text to copy** field and then click the **Copy** button. Now switch back to your document and choose **Edit > Paste** from the **gedit** menu bar.

The above text includes application names; system-wide menu names and items; application-specific menu names; and buttons and text found within a GUI interface, all presented in Proportional Bold and all distinguishable by context.

Note the **>** shorthand used to indicate traversal through a menu and its sub-menus. This is to avoid the difficult-to-follow 'Select **Mouse** from the **Preferences** sub-menu in the **System** menu of the main menu bar' approach.

`Mono-spaced Bold Italic` or ***Proportional Bold Italic***

Whether Mono-spaced Bold or Proportional Bold, the addition of Italics indicates replaceable or variable text. Italics denotes text you do not input literally or displayed text that changes depending on circumstance. For example:

> To connect to a remote machine using ssh, type `ssh` *`username@domain.name`* at a shell prompt. If the remote machine is `example.com` and your username on that machine is john, type `ssh john@example.com`.
>
> The `mount -o remount` *`file-system`* command remounts the named file system. For example, to remount the `/home` file system, the command is `mount -o remount /home`.
>
> To see the version of a currently installed package, use the `rpm -q` *`package`* command. It will return a result as follows: *`package-version-release`*.

Note the words in bold italics above username, domain.name, file-system, package, version and release. Each word is a placeholder, either for text you enter when issuing a command or for text displayed by the system.

Aside from standard usage for presenting the title of a work, italics denotes the first use of a new and important term. For example:

> When the Apache HTTP Server accepts requests, it dispatches child processes or threads to handle them. This group of child processes or threads is known as

a *server-pool*. Under Apache HTTP Server 2.0, the responsibility for creating and maintaining these server-pools has been abstracted to a group of modules called *Multi-Processing Modules* (*MPMs*). Unlike other modules, only one module from the MPM group can be loaded by the Apache HTTP Server.

1.2. Pull-quote Conventions

Two, commonly multi-line, data types are set off visually from the surrounding text.

Output sent to a terminal is set in `Mono-spaced Roman` and presented thus:

```
books        Desktop   documentation  drafts  mss    photos   stuff  svn
books_tests  Desktop1  downloads      images  notes  scripts  svgs
```

Source-code listings are also set in `Mono-spaced Roman` but are presented and highlighted as follows:

```
package org.jboss.book.jca.ex1;

import javax.naming.InitialContext;

public class ExClient
{
   public static void main(String args[])
      throws Exception
   {
      InitialContext iniCtx = new InitialContext();
      Object         ref    = iniCtx.lookup("EchoBean");
      EchoHome       home   = (EchoHome) ref;
      Echo           echo   = home.create();

      System.out.println("Created Echo");

      System.out.println("Echo.echo('Hello') = " + echo.echo("Hello"));
   }

}
```

1.3. Notes and Warnings

Finally, we use three visual styles to draw attention to information that might otherwise be overlooked.

Note

A note is a tip or shortcut or alternative approach to the task at hand. Ignoring a note should have no negative consequences, but you might miss out on a trick that makes your life easier.

Important

Important boxes detail things that are easily missed: configuration changes that only apply to the current session, or services that need restarting before an update will apply. Ignoring Important boxes won't cause data loss but may cause irritation and frustration.

Warning

A Warning should not be ignored. Ignoring warnings will most likely cause data loss.

2. Provide feedback to the authors!

If you find a typographical error in this manual, or if you have thought of a way to make this manual better, we would love to hear from you! Please submit a report in the the *Issue Tracker* [http://code.google.com/p/mobicents/issues/list], against the product **Mobicents JAIN SLEE JDBC Resource Adaptor**, or contact the authors.

When submitting a bug report, be sure to mention the manual's identifier: JAIN_SLEE_JDBC_RA_User_Guide

If you have a suggestion for improving the documentation, try to be as specific as possible when describing it. If you have found an error, please include the section number and some of the surrounding text so we can find it easily.

Introduction to Mobicents JAIN SLEE JDBC Resource Adaptor

The JDBC Resource Adaptor adapts JDBC Datasources to JAIN SLEE domain, providing means to execute JDBC statements in asynchronous fashion. JDCB statements are executed in the RA runtime resources, freeing the JAIN SLEE Event Router from the burden of having its executors (threads) resources blocked by interactions with JDBC Datasources, and results are provided to applications through JAIN SLEE events. The JAIN SLEE application is also completely free from having to manage connection closings.

Resource Adaptor Type

The Resource Adaptor Type is the interface which defines the contract between the RA implementations, the SLEE container, and the Applications running in it.

The name of the RA Type is `JDBCResourceAdaptorType`, its vendor is `org.mobicents` and its version is `1.0`.

2.1. Activities

The single activity object for JDBC Resource Adaptor is the `org.mobicents.slee.resource.jdbc.JdbcActivity` interface. Through the activity an SBB can execute multiple JDBC statements, and receive the related responses asynchronously through events on it. Due to the nature of SLEE activities, this RA activity acts like a queue of requests, allowing the processing of their responses - the events- in a serialized way

An activity starts on demand by an SBB, through the RA SBB Interface, and it ends when an SBB invokes its `endActivity()` method.

The activity interface is defined as follows:

```
package org.mobicents.slee.resource.jdbc;

import java.sql.PreparedStatement;
import java.sql.Statement;

public interface JdbcActivity {

    void execute(Statement statement, String sql);

    void execute(Statement statement, String sql, int autoGeneratedKeys);

    void execute(Statement statement, String sql, int columnIndexes[]);

    void execute(Statement statement, String sql, String columnNames[]);

    void executeQuery(Statement statement, String sql);

    void executeUpdate(Statement statement, String sql);

    void executeUpdate(Statement statement, String sql, int autoGeneratedKeys);
```

```
    void executeUpdate(Statement statement, String sql, int columnIndexes[]);

    void executeUpdate(Statement statement, String sql, String columnNames[]);

    void execute(PreparedStatement preparedStatement);

    void executeQuery(PreparedStatement preparedStatement);

    void executeUpdate(PreparedStatement preparedStatement);

    public void endActivity();

}
```

The `execute(Statement, String)` method:
: Asynchronous execution of statement with unknown result type, which is provided in an event, fired in the activity. Details about parameters can be seen in the javadoc for `java.sql.Statement` method with same signature. This method should only be used when the application has no idea of what is the SQL to be executed. If the execution throws an exception, it will be provided as an event too.

The `execute(Statement, String, int)` method:
: Asynchronous execution of statement with unknown result type, which is provided in an event, fired in the activity. Details about parameters can be seen in the javadoc for `java.sql.Statement` method with same signature. This method should only be used when the application has no idea of what is the SQL to be executed. If the execution throws an exception, it will be provided as an event too.

The `execute(Statement, String, int[])` method:
: Asynchronous execution of statement with unknown result type, which is provided in an event, fired in the activity. Details about parameters can be seen in the javadoc for `java.sql.Statement` method with same signature. This method should only be used when the application has no idea of what is the SQL to be executed. If the execution throws an exception, it will be provided as an event too.

The `execute(Statement, String, String[])` method:
: Asynchronous execution of statement with unknown result type, which is provided in an event, fired in the activity. Details about parameters can be seen in the javadoc for `java.sql.Statement` method with same signature. This method should only be used when the application has no idea of what is the SQL to be executed. If the execution throws an exception, it will be provided as an event too.

The `executeQuery(Statement, String)` method:
: Asynchronous execution of statement with `result set` result type, which is provided in an event, fired in the activity. Details about parameters can be seen in the javadoc for

`java.sql.Statement` method with same signature. This method should be used when the application knows that the SQL to execute is a read query. If the execution throws an exception, it will be provided as an event too.

The `executeUpdate(Statement, String)` method:

Asynchronous execution of statement with `update count` result type, which is provided in an event, fired in the activity. Details about parameters can be seen in the javadoc for `java.sql.Statement` method with same signature. This method should be used when the application knows that the SQL to execute is a write query. If the execution throws an exception, it will be provided as an event too.

The `executeUpdate(Statement, String, int)` method:

Asynchronous execution of statement with `update count` result type, which is provided in an event, fired in the activity. Details about parameters can be seen in the javadoc for `java.sql.Statement` method with same signature. This method should be used when the application knows that the SQL to execute is a write query. If the execution throws an exception, it will be provided as an event too.

The `executeUpdate(Statement, String, int[])` method:

Asynchronous execution of statement with `update count` result type, which is provided in an event, fired in the activity. Details about parameters can be seen in the javadoc for `java.sql.Statement` method with same signature. This method should be used when the application knows that the SQL to execute is a write query. If the execution throws an exception, it will be provided as an event too.

The `executeUpdate(Statement, String, String[])` method:

Asynchronous execution of statement with `update count` result type, which is provided in an event, fired in the activity. Details about parameters can be seen in the javadoc for `java.sql.Statement` method with same signature. This method should be used when the application knows that the SQL to execute is a write query. If the execution throws an exception, it will be provided as an event too.

The `execute(PreparedStatement)` method:

Asynchronous execution of a prepared statement with `unknown` result type, which is provided in an event, fired in the activity. PreparedStatements are an effective way to improve performance, since the JDBC driver may cache and pool the SQL. Details about parameters can be seen in the javadoc for `java.sql.PreparedStatement` method with same signature. This method should only be used when the application has no idea of what is the SQL to be executed. If the execution throws an exception, it will be provided as an event too.

The `executeQuery(PreparedStatement)` method:

Asynchronous execution of a prepared statement with `result set` result type, which is provided in an event, fired in the activity. PreparedStatements are an effective way to improve performance, since the JDBC driver may cache and pool the SQL. This method should be used when the application knows that the SQL to execute is a read query. Details about parameters can be seen in the javadoc for `java.sql.PreparedStatement` method with same

signature. This method should only be used when the application has no idea of what is the SQL to be executed. If the execution throws an exception, it will be provided as an event too.

The `executeUpdate(PreparedStatement)` method:

Asynchronous execution of a prepared statement with `update count` result type, which is provided in an event, fired in the activity. PreparedStatements are an effective way to improve performance, since the JDBC driver may cache and pool the SQL. This method should be used when the application knows that the SQL to execute is a write query. If the execution throws an exception, it will be provided as an event too.

The `endActivity()` method:

Ends the activity and its related Activity Context.

2.2. Events

There are eight event types fired by JDBC Resource Adaptor, which provides applications the result of each kind of interaction with the `Datasource` - result set, update count, unknown result and exception - from executing a `Statement` or `PreparedStatement`.

Table 2.1. Events which provide results of `Statement` execution

Name	Vendor	Version	Event Class	Description
Statement ResultSet Event	org.mobicents	1.0	org.mobicents.slee.resource.jdbc.event.StatementResultSetEvent	Provides the result set from a successful execution of a statement, requested through the activity method `executeQuery(Statement statement, String sql)`.
Statement UpdateCount Event	org.mobicents	1.0	org.mobicents.slee.resource.jdbc.event.StatementUpdateCountEvent	Provides the update count from a successful execution of a statement, requested through the activity methods `executeUpdate(Statement statement, ...)`.
Statement UnknownResult Event	org.mobicents	1.0	org.mobicents.slee.resource.jdbc.event.StatementUnknownResultEvent	Provides the unknown type result from a successful execution of a statement, requested through the activity methods `execute(Statement statement, ...)`.

Name	Vendor	Version	Event Class	Description
Statement SQLException Event	org.mobicents	1.0	org.mobicents. slee.resource. jdbc.event. Statement SQLException Event	Provides the exception thrown from a unsuccessful execution of a statement.

Important

Spaces where introduced in the `Name` and `Event Class` column values, to correctly render the table. Please remove them when using copy/paste.

Table 2.2. Events which provide results of `PreparedStatement` execution

Name	Vendor	Version	Event Class	Description
PreparedStatement ResultSet Event	org.mobicents	1.0	org.mobicents. slee.resource. jdbc.event. Prepared Statement ResultSet Event	Provides the result set from a successful execution of a prepared statement, requested through the activity method `executeQuery( PreparedStatement preparedStatement)`.
PreparedStatement UpdateCount Event	org.mobicents	1.0	org.mobicents. slee.resource. jdbc.event. Prepared Statement UpdateCount Event	Provides the update count from a successful execution of a prepared statement, requested through the activity method `executeUpdate( PreparedStatement preparedStatement )`.
PreparedStatement UnknownResult Event	org.mobicents	1.0	org.mobicents. slee.resource. jdbc.event. Prepared Statement UnknownResult Event	Provides the unknown type result from a successful execution of a prepared statement, requested through the activity method `execute( Prepared`

Name	Vendor	Version	Event Class	Description
				`Statement preparedStatement ).`
PreparedStatement SQLException Event	org.mobicents	1.0	org.mobicents. slee.resource. jdbc.event. Prepared Statement SQLException Event	Provides the exception thrown from a unsuccessful execution of a prepared statement.

Important

Spaces where introduced in the `Name` and `Event Class` column values, to correctly render the table. Please remove them when using copy/paste.

2.3. Activity Context Interface Factory

The Resource Adaptor's Activity Context Interface Factory is of type `org.mobicents.slee.resource.jdbc.JdbcActivityContextInterfaceFactory`, it allows the SBB to retrieve the `ActivityContextInterface` related with a specific `JdbcActivity` instance. The interface is defined as follows:

```
package org.mobicents.slee.resource.jdbc;

import javax.slee.ActivityContextInterface;
import javax.slee.FactoryException;
import javax.slee.UnrecognizedActivityException;
import javax.slee.resource.ResourceAdaptorTypeID;

public interface JdbcActivityContextInterfaceFactory {

    public static final ResourceAdaptorTypeID RATYPE_ID;

    public ActivityContextInterface getActivityContextInterface(
            JdbcActivity activity) throws UnrecognizedActivityException,
            FactoryException;

}
```

The Resource Adaptor's Activity Context Interface Factory exposes a static RATYPE_ID field, containing the `ResourceAdaptorTypeID` of the Resource Adaptor Type it belongs, which may be used to retrieve the factory instance using the `SbbContextExt` JAIN SLEE 1.1 extension.

2.4. Resource Adaptor Interface

The JDBC Resource Adaptor interface, of type `org.mobicents.slee.resource.jdbc.JdbcResourceAdaptorSbbInterface`, may be used by applications to create RA activities, and retrieve JDBC Connections, its interface is defined as follows:

```
package org.mobicents.slee.resource.jdbc;

import java.sql.Connection;
import java.sql.SQLException;

import javax.slee.resource.ResourceAdaptorTypeID;

public interface JdbcResourceAdaptorSbbInterface {

   public static final ResourceAdaptorTypeID RATYPE_ID;

   public JdbcActivity createActivity();

   Connection getConnection() throws SQLException;

   Connection getConnection(String username, String password)
         throws SQLException;

}
```

The `createActivity()` method:
Creates a new `JdbcActivity` instance.

The `getConnection()` method:
Retrieves a JDBC Connection, which may then be used to create statements (prepared or not). Note that the connection is closed automatically after each statement execution done

through a JDBC activity, thus applications must not do it (unless the connection is retrieved but not used).

The `getConnection(String, String)` method:

Retrieves a JDBC Connection using username and password authentication, which may then be used to create statements (prepared or not). Note that the connection is closed automatically after each statement execution done through a JDBC activity, thus applications must not do it (unless the connection is retrieved but not used).

The JDBC Resource Adaptor interface also exposes a static RATYPE_ID field, containing the `ResourceAdaptorTypeID` of the Resource Adaptor Type it belongs, which may be used to retrieve the factory instance using the `SbbContextExt` JAIN SLEE 1.1 extension.

2.5. Restrictions

The JDBC Resource Adaptor Type does not defines any restriction when using object instances provided, which means an application may use the provided JDBC connection, and the statements it creates, for any its functionalities (including the synchronous execution of statements through its interface).

2.6. Sbb Code Examples

The following code examples shows how to use the Resource Adaptor Type for common functionalities

2.6.1. Retrieving the RA Interface and ACI Factory

The following code examples the retrieval of the RA's SBB Interface and ACI Factory, usually done in the Sbb's `setSbbContext(SbbContext)`:

```
    /**
 * the SBB object context
 */
private SbbContextExt contextExt;

/**
 * the JDBC RA SBB Interface
 */
private JdbcResourceAdaptorSbbInterface jdbcRA;

/**
 * the JDBC RA {@link ActivityContextInterface} factory
 */
private JdbcActivityContextInterfaceFactory jdbcACIF;
```

```
@Override
public void setSbbContext(SbbContext context) {
    this.contextExt = (SbbContextExt) context;
    this.jdbcRA = (JdbcResourceAdaptorSbbInterface) contextExt
            .getResourceAdaptorInterface(
                    JdbcResourceAdaptorSbbInterface.RATYPE_ID, raEntityLinkName);
    this.jdbcACIF = (JdbcActivityContextInterfaceFactory) contextExt
            .getActivityContextInterfaceFactory(JdbcActivityContextInterfaceFactory.RATYPE_ID);
}
```

The `raEntityLinkName` is the link name of the RA entity to use. The link to the default RA entity, use the link name `JDBCRA`..

2.6.2. Create and Attach to RA Activities

The following code examples the creation of `JdbcActivity`, and the attachment to its `ActivityContextInterface`:

```
// create activity using the RA sbb interface
JdbcActivity jdbcActivity = jdbcRA.createActivity();
// get its aci from the RA ACI factory
ActivityContextInterface jdbcACI = jdbcACIF
        .getActivityContextInterface(jdbcActivity);
// attach the sbb entity
jdbcACI.attach(contextExt.getSbbLocalObject());
```

2.6.3. Execute a Statement

The following code examples the creation of a `Statement` and the execution of SQL on a `JdbcActivity`:

```
// get connection and create statement
Statement statement = jdbcRA.getConnection().createStatement();
// execute SQL in the jdbc activity
jdbcActivity.executeQuery(statement,
```

```
            "CREATE TABLE TestTable (Name VARCHAR(30));");
```

2.6.4. Handling Events and Ending an Activity

The following code examples the handling of events, for both `PreparedStatement` and `Statement` execution, following the service logic execution. It also shows the explicit ending of the activity:

```
/**
 * Event handler for {@link StatementResultSetEvent}.
 *
 * @param event
 * @param aci
 */
public void onStatementResultSetEvent(StatementResultSetEvent event,
        ActivityContextInterface aci) {
    tracer.info("Received a StatementResultSetEvent, as result of executed SQL "
            + event.getSQL());
    tracer.info("Result: " + event.getResultSet());
    try {
        PreparedStatement preparedStatement = jdbcRA.getConnection()
                .prepareStatement("INSERT INTO TestTable VALUES(?)");
        preparedStatement.setString(1, "Mobicents");
        tracer.info("Created prepared statement, executing...");
        ((JdbcActivity) aci.getActivity()).executeUpdate(preparedStatement);
    } catch (Throwable e) {
        tracer.severe("failed to create statement", e);
    }
}

/**
 * Event handler for {@link PreparedStatementUpdateCountEvent}.
 *
 * @param event
 * @param aci
 */
public void onPreparedStatementUpdateCountEvent(
        PreparedStatementUpdateCountEvent event,
        ActivityContextInterface aci) {
    tracer.info("Received a PreparedStatementUpdateCountEvent.");
    tracer.info("Update Count: " + event.getUpdateCount());
```

```
        try {
            Statement anotherStatement = jdbcRA.getConnection()
                    .createStatement();
            tracer.info("Created statement, executing query...");
            ((JdbcActivity) aci.getActivity()).executeUpdate(anotherStatement,
                    "DROP TABLE TestTable;");
        } catch (Throwable e) {
            tracer.severe("failed to create statement", e);
        }
    }

    /**
     * Event handler for {@link StatementUpdateCountEvent}.
     *
     * @param event
     * @param aci
     */
    public void onStatementUpdateCountEvent(StatementUpdateCountEvent event,
            ActivityContextInterface aci) {
        tracer.info("Received a StatementUpdateCountEvent, as result of executed SQL "
                + event.getSQL());
        tracer.info("Update Count: " + event.getUpdateCount());
        tracer.info("Ending JDBC Activity...");
        ((JdbcActivity) aci.getActivity()).endActivity();
    }
```

The SBB XML descriptor code to declare the handling of such events:

```
    <event event-direction="Receive" initial-event="False">
        <event-name>StatementResultSetEvent</event-name>
        <event-type-ref>
            <event-type-name>StatementResultSetEvent</event-type-name>
            <event-type-vendor>org.mobicents</event-type-vendor>
            <event-type-version>1.0</event-type-version>
        </event-type-ref>
    </event>

    <event event-direction="Receive" initial-event="False">
        <event-name>PreparedStatementUpdateCountEvent</event-name>
        <event-type-ref>
            <event-type-name>PreparedStatementUpdateCountEvent</event-type-name>
```

```
            <event-type-vendor>org.mobicents</event-type-vendor>
            <event-type-version>1.0</event-type-version>
        </event-type-ref>
    </event>

    <event event-direction="Receive" initial-event="False">
        <event-name>StatementUpdateCountEvent</event-name>
        <event-type-ref>
            <event-type-name>StatementUpdateCountEvent</event-type-name>
            <event-type-vendor>org.mobicents</event-type-vendor>
            <event-type-version>1.0</event-type-version>
        </event-type-ref>
    </event>
```

Resource Adaptor Implementation

This chapter documents the JDBC Resource Adaptor Implementation details, such as the configuration properties, the default Resource Adaptor entities, and the JAIN SLEE 1.1 Tracers and Alarms used.

The name of the RA is `JDBCResourceAdaptor`, its vendor is `org.mobicents` and its version is `1.0`.

3.1. Configuration

The Resource Adaptor supports configuration only at Resource Adaptor Entity creation time. The following table enumerates the configuration properties:

Table 3.1. Resource Adaptor's Configuration Properties

Property Name	Description	Property Type	Default Value
DATASOURCE_ JNDI _NAME	the JNDI name used to retrieve the Datasource	java.lang.String	java:DefaultDS
EXECUTOR_ SERVICE _THREADS	the number of threads executing statements	java.lang.Integer	4

Important

Spaces where introduced in the `Property Name` column values, to correctly render the table. Please remove them when using copy/paste.

3.2. Default Resource Adaptor Entities

There is a single Resource Adaptor Entity created when deploying the Resource Adaptor, named `JDBCRA`.

The `JDBCRA` entity is also bound to Resource Adaptor Link Name `JDBCRA`, to use it in an Sbb add the following XML to its descriptor:

```
<resource-adaptor-type-binding>
   <resource-adaptor-type-ref>
      <resource-adaptor-type-name>
         JDBCResourceAdaptorType
      </resource-adaptor-type-name>
```

```
            <resource-adaptor-type-vendor>
               org.mobicents
            </resource-adaptor-type-vendor>
            <resource-adaptor-type-version>
               1.0
            </resource-adaptor-type-version>
         </resource-adaptor-type-ref>
         <activity-context-interface-factory-name>
            slee/ra/jdbc/1.0/acifactory
         </activity-context-interface-factory-name>
         <resource-adaptor-entity-binding>
            <resource-adaptor-object-name>
               slee/ra/jdbc/1.0/sbbinterface
            </resource-adaptor-object-name>
            <resource-adaptor-entity-link>
               JDBCRA
            </resource-adaptor-entity-link>
         </resource-adaptor-entity-binding>
      </resource-adaptor-type-binding>
```

3.3. Traces and Alarms

3.3.1. Tracers

Each Resource Adaptor Entity uses a single JAIN SLEE 1.1 Tracer, named `JdbcResourceAdaptor`. The related Log4j Logger category, which can be used to change the Tracer level from Log4j configuration, is `javax.slee.RAEntityNotification[entity=JDBCRA]`

3.3.2. Alarms

No alarms are set by this Resource Adaptor.

Setup

4.1. Pre-Install Requirements and Prerequisites

Ensure that the following requirements have been met before continuing with the install.

4.1.1. Hardware Requirements

The RA hardware requirements don't differ from the underlying Mobicents JAIN SLEE requirements, refer to its documentation for further information.

4.1.2. Software Prerequisites

The RA requires Mobicents JAIN SLEE properly set.

4.2. Mobicents JAIN SLEE JDBC Resource Adaptor Source Code

4.2.1. Release Source Code Building

1. **Downloading the source code**

Important

Subversion is used to manage its source code. Instructions for using Subversion, including install, can be found at *http://svnbook.red-bean.com*

Use SVN to checkout a specific release source, the base URL is http://mobicents.googlecode.com/svn/tags/servers/jain-slee/2.x.y/resources/jdbc, then add the specific release version, lets consider 1.0.0.BETA1.

```
[usr]$ svn co http://mobicents.googlecode.com/svn/tags/servers/jain-slee/2.x.y/resources/jdbc/1.0.0.BETA1 slee-ra-jdbc-1.0.0.BETA1
```

2. **Building the source code**

Important

Maven 2.2.1 (or higher) is used to build the release. Instructions for using Maven2, including install, can be found at *http://maven.apache.org*

Use Maven to build the deployable unit binary.

```
[usr]$ cd slee-ra-jdbc-1.0.0.BETA1
[usr]$ mvn install
```

Once the process finishes you should have the `deployable-unit` jar file in the `target` directory, if Mobicents JAIN SLEE is installed and environment variable JBOSS_HOME is pointing to its underlying JBoss Application Server directory, then the deployable unit jar will also be deployed in the container.

4.2.2. Development Trunk Source Building

Similar process as for *Section 4.2.1, "Release Source Code Building"*, the only change is the SVN source code URL, which is http://mobicents.googlecode.com/svn/trunk/servers/jain-slee/resources/jdbc.

4.3. Installing Mobicents JAIN SLEE JDBC Resource Adaptor

To install the Resource Adaptor simply execute provided ant script `build.xml` default target:

```
[usr]$ ant
```

The script will copy the RA deployable unit jar to the `default` Mobicents JAIN SLEE server profile deploy directory, to deploy to another server profile use the argument `-Dnode=`.

4.4. Uninstalling Mobicents JAIN SLEE JDBC Resource Adaptor

To uninstall the Resource Adaptor simply execute provided ant script `build.xml undeploy` target:

```
[usr]$ ant undeploy
```

The script will delete the RA deployable unit jar from the `default` Mobicents JAIN SLEE server profile deploy directory, to undeploy from another server profile use the argument `-Dnode=`.

Clustering

The JDBC Resource Adaptor is cluster aware, it supports Activity replication, which means that any application instance may retrieve and interact with any JDBC Activity, in any node in a Mobicents SLEE cluster. The RA defines no failover mechanisms.

Appendix A. Revision History

Revision History

Revision 1.0 Wed Apr 20 2011 EduardoMartins

Creation of the Mobicents JAIN SLEE JDBC RA User Guide.

Index

F