User Guide to Restcomm JAIN-SLEE 2.8.0-SNAPSHOT

Table of Contents

Preface

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 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.

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 to switch to the first virtual terminal. Press 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 menu:>[] shorthand used to indicate traversal through a menu and its sub-menus. This is to avoid the difficult-to-follow 'Select from the **Preferences** | **] sub-menu in the menu: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.

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"));
   }

}
```

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.

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 {this-issue.tracker.ur}, against the product Restcomm JAIN-SLEE` `, or contact the authors.

When submitting a bug report, be sure to mention the manual's identifier: Restcomm JAIN-SLEE

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.

Chapter 1. Introduction to Restcomm JAIN SLEE

JAIN SLEE is the specification for a Java Service Logic and Execution Environment (SLEE) architecture, created in the Java Community Process (JCP) by several individuals and companies, including Red Hat. A SLEE is an application server, or service container, which defines a component model for structuring the logic of communications services as a collection of reusable components, and for combining these components into even more sophisticated services. This model was designed and optimized for event-driven applications.

In addition to the service component model, the SLEE also defines management interfaces used to administer the container and the service components executing within, and a set of standard facilities, which provide common features, such as timers, traces and alarms, to JAIN SLEE components.

Restcomm JAIN SLEE is the first and only open source platform certified for JAIN SLEE 1.1 compliance, providing a highly scalable, event-driven application server with a robust component model and a fault tolerant execution environment.

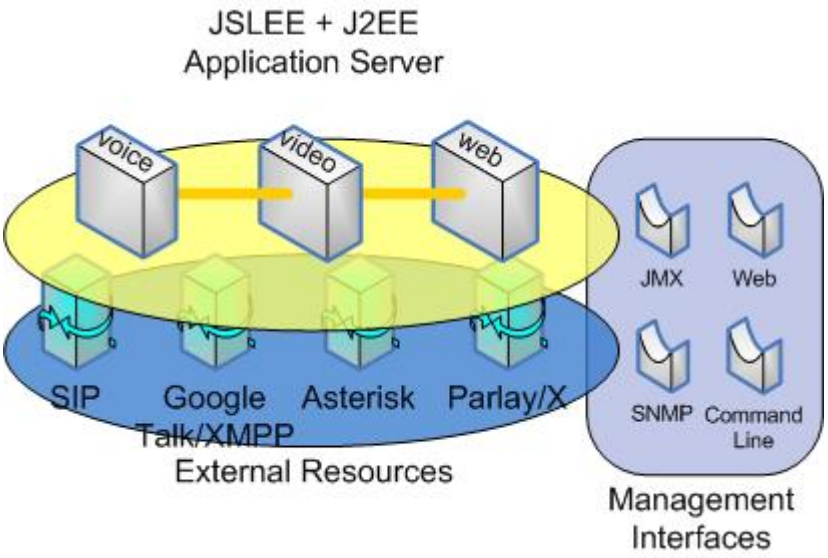

Figure 1. Overview of JAIN SLEE

Restcomm is built on top of the open source award winning JBoss Application Server , which means that Restcomm complements with Java Enterprise () 5 container features, allowing strong convergence of different application models, for even feature richer communication services, for instance, the Web and can be combined to achieve a more sophisticated and natural user experience. Restcomm inherits quality management features and tools from JBoss Application Server , such as the JMX Console, Jopr Plugins and SNMP Adaptor.

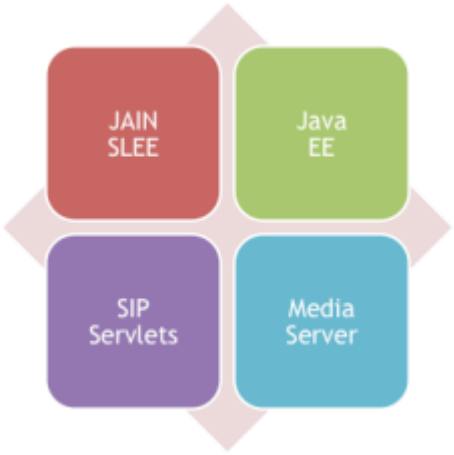

Figure 2. Restcomm Platform

Restcomm JAIN SLEE can also be complemented with Restcomm SIP Servlets and Restcomm Media Server, providing unique value and integration features not found elsewhere.

Chapter 2. Installing Restcomm JAIN SLEE

2.1. Pre-Install Requirements and Prerequisites

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

2.1.1. Hardware Requirements

Sufficient Disk Space

Once unzipped, the Restcomm JAIN SLEE binary release requires *at least* 500MB of free disk space. Keep in mind that disk space requirements may change from release to release.

Anything Java Itself Will Run On

The Restcomm JAIN SLEE container, and bundled JAIN SLEE components are 100% java. The JAIN SLEE will run on the same hardware that the JBoss Application Server runs on, but it is recommended at least 2GB or 4GB of RAM memory, for 32 or 64 bit Operating Systems.

2.1.2. Software Prerequisites

JDK 6

A working installation of the Java Development Kit () version 6 is required in order to run the Restcomm JAIN SLEE. Note that the JBoss Application Server is a runtime dependency, but comes bundled with the binary distribution. For instructions on how to install the JDK, refer to Java Development Kit (): Installing, Configuring and Running

2.2. Install Alternatives

Binary Release

The binary release is a zip file containing an already built binary release of Restcomm JAIN SLEE

Release Source Building

As an alternative to the binary release, it is possible to download a specific release source code and build a binary from it.

Master Source Building

Similar as the Release Source Building, but done on the master (current development) source code.

Binary Release Snapshot

The binary release snapshot is a build of the master source code done daily and upoaded to a public web site.

2.2.1. Binary Release

You can download the Binary zip files from https://mobicents.ci.cloudbees.com/job/Restcomm-JAIN-SLEE-Release/lastSuccessfulBuild/artifact/release/.

In this form of installation, simply unzip the downloaded zip file to the directory of your choice on any operating system that supports the zip format.

1. Unzip the release file

   Unzip the file to extract the archive contents into the location of your choice. You can do this using the JDK jar tool (or any other ZIP extraction tool). In the example below we are assuming you downloaded the zip file was named restcomm-jainslee-2.8.0-SNAPSHOT .zip to the /restcomm directory.

   ```
   [usr]$ cd /restcomm
   [usr]$ jar -xvf restcomm-jainslee.zip
   ```

2. Setting up JBOSS_HOME Environment Variable

   You should now have a directory called restcomm-jainslee-2.8.0-SNAPSHOT . Next you need to set your JBOSS_HOME environment variables. This is discussed in Setting the JBOSS_HOME Environment Variable.

2.2.2. Binary Release Snapshot

You can download the Binary Snapshot zip files from https://github.com/RestComm/jain-slee/releases/latest. The installation is similar to the [_binary_release] one.

2.2.3. Release Source Building

1. Downloading the source code

   

   Git is used to manage Restcomm JAIN SLEE source code. Instructions for downloading, installing and using Git can be found at http://git-scm.com/

   Use Git to checkout the specific release source, the Git repository URL is https://github.com/RestComm/jain-slee/ , then switch to the specific release version, lets consider 2.8.0-SNAPSHOT .

   ```
   [usr]$ git clone https://github.com/RestComm/jain-slee restcomm-jain-slee-release
   [usr]$ cd restcomm-jain-slee-release
   [usr]$ git checkout tags/
   [usr]$ cd release
   ```

2. Building the source code

   

   Apache Ant 1.6 (or higher) and Maven 2.0.9 (or higher) is used to build the release. Instructions for using Ant and Maven2, including install, can be found at http://ant.apache.org and http://maven.apache.org

   Use Ant to build the binary.

```
[usr]$ ant
```

Once the process finishes you should have a restcomm-jainslee-2.8.0-SNAPSHOT.zip file, installation is the same as for [_binary_release].

2.2.4. 2.x Branch Source Building

Similar process as for [_release_source_building], the only change is the Git reference should be the `2.x`. The `git checkout tags/2.8.0-SNAPSHOT` command should not be performed. If already performed, the following should be used in order to switch back to the 2.x branch:

```
[usr]$ git checkout 2.x
```

2.2.5. Master Source Building

Similar process as for [_release_source_building], the only change is the Git reference should be the `master`. The `git checkout tags/2.8.0-SNAPSHOT` command should not be performed. If already performed, the following should be used in order to switch back to the master:

```
[usr]$ git checkout master
```

2.3. Uninstall Restcomm JAIN SLEE

To uninstall simply delete the directory containing Restcomm JAIN SLEE.

Chapter 3. Configuring and Running Restcomm JAIN SLEE

3.1. Server Profiles

Restcomm JAIN SLEE reuses &JEE.PLATFORM;server profiles to expose different configurations for different needs:

Default Profile

The `default` profile is proper for standalone or pure high availability. It provides the best performance per cluster node, with linear scaling, but there is no state replication in the cluster, which means that there is no support for failover, neither there is any kind of state gravitation (one node sending state so another node continues its work.

All Profile

The `all` profile is proper for more flexible high availability and failover support. Performance per node decreases but the cluster does state replication or gravitation.

Profiles can be selected when starting the server, see Running Restcomm JAIN SLEE for detailed instructions.

3.2. Running Restcomm JAIN SLEE

Starting or stopping Restcomm JAIN SLEE is no different than starting or stopping &JEE.PLATFORM;

3.2.1. Starting

Once installed, you can run server(s) by executing the run.sh (Unix) or run.bat (Microsoft Windows) startup scripts in the <*install_directory*>*/bin* directory (on Unix or Windows).

Starting Parameters

Server Profile

To specify the server profile use `-c profile_name`, for instance, to use the `all` profile then start the server with `-c all`

If not specified the default profile is used.

IP / Host

To specify the IP/Host which the server binds, use `-b IP`, for instance, to use the 192.168.0.1 IP then start the server with `-b 192.168.0.1`

If not specified then 127.0.0.1 is used.

3.2.2. Stopping

You can shut down the server(s) by executing the `shutdown.sh -s` (Unix) or `shutdown.bat -s` (Microsoft Windows) scripts in the *<install_directory>/bin* directory (on Unix or Windows). Note that if you properly stop the server, you will see the following three lines as the last output in the Unix terminal or Command Prompt:

```
[Server] Shutdown complete
Shutdown complete
Halting VM
```

3.3. Configuring Restcomm JAIN SLEE

JAIN SLEE is configured through an XML descriptor for each Server Profiles. The XML file is named *jboss-beans.xml* and is located at *$JBOSS_HOME/server/profile_name/deploy/restcomm-slee/META-INF*, where `profile_name` is the server profile name.

This configuration greatly affects performance or correctness of the container behavior. This is for advanced users that know the internals of the container.

3.4. EventContext Factory Configuration

The EventContext Factory is responsible for managing all EventContexts in the SLEE Container, and its behavior is configurable.

The EventContext Factory configuration can be changed through an XML file and a JMX MBean.

3.4.1. EventContext Factory Persistent Configuration

Configuration is done through an XML descriptor for each Restcomm Cluster. The XML file is named *jboss-beans.xml* and is located at *{JBOSS_HOME}/server/{profile_name}/deploy/restcomm-slee/META-INF*

The configuration is exposed a JBoss Microcontainer Bean:

```
<bean name="Mobicents.JAINSLEE.EventContextFactoryConfiguration"

class="org.mobicents.slee.container.management.jmx.EventContextFactoryConfiguration">
    <annotation>@org.jboss.aop.microcontainer.aspects.jmx.JMX(name=
        "org.mobicents.slee:name=EventContextFactoryConfiguration",exposedInterface=

org.mobicents.slee.container.management.jmx.EventContextFactoryConfigurationMBean.clas
s,
        registerDirectly=true)</annotation>
    <property name="defaultEventContextSuspensionTimeout">60000</property>
</bean>
```

Table 1. EventContext Factory Bean Configuration

Property Name	Property Type	Description
defaultEventContextSuspension Timeout	int	defines the default timeout applied when suspending delivery of an EventContext

3.4.2. EventContext Factory JMX Configuration

Through JMX the EventContext Factory module configuration can be changed with the container running. Note that such configuration changes are not persisted.

The JMX MBean which can be used to change the EventContext Factory configuration is named `org.mobicents.slee:name=EventContextFactoryConfiguration`, and provides getters and setters to change each property defined in the persistent configuration. The JMX Console can be used to use this MBean, see JMX Console.

3.5. Event Router Statistics and Configuration

The JAIN SLEE Event Router is the module responsible for creating new service instances and delivering events to all interested parties. It is capable of doing the routing of several events in parallel, through the usage of multiple executors, each bound to a different thread.

The Event Router is also able to account performance and load statistics, indicating the number of activities being assigned or several timings regarding event routing, globally or for each individual executor/thread. Statistics are turned on by default and may be retrieved through the JMX MBean `org.mobicents.slee:name=EventRouterStatistics`.

An important sub-module of the Event Router is the Executor Mapper, which is responsible for assigning activities to the available executors. JAIN SLEE includes two different Executor Mappers. The default one takes into account the hashcode of the activity handle when distributing, while the alternative uses a round robin algorithm.

In the case of advanced performance tuning, it is advised to try the different implementations available, or even provide a custom one.

The Executor Mapper is nothing more than an interface: `org.mobicents.slee.container.eventrouter.EventRouterExecutorMapper`. To deploy a custom implementation, drop the implementation class or classes, packed in a jar file, in the server profile */deploy* directory.

The whole Event Router is a critical component with respect to the container's performance. Its configuration can be tuned, through an XML file and a JMX MBean.

3.5.1. Event Router Persistent Configuration

Configuration is done through an XML descriptor for each Server Profiles. The XML file is named *jboss-beans.xml* and is located at *$JBOSS_HOME/server/profile_name/deploy/restcomm-slee/META-*

INF, where `profile_name` is the server profile name.

The configuration is exposed a JBoss Microcontainer Bean:

```
<bean name="Mobicents.JAINSLEE.EventRouterConfiguration"
    class="org.mobicents.slee.container.management.jmx.EventRouterConfiguration">
    <annotation>@org.jboss.aop.microcontainer.aspects.jmx.JMX(name=
        "org.mobicents.slee:name=EventRouterConfiguration", exposedInterface=

org.mobicents.slee.container.management.jmx.EventRouterConfigurationMBean.class,
        registerDirectly=true)</annotation>
    <property name="eventRouterThreads">8</property>
    <property name="collectStats">true</property>
    <property name="executorMapperClassName">

org.mobicents.slee.runtime.eventrouter.mapping.ActivityHashingEventRouterExecutorMappe
r
    </property>
</bean>
```

Table 2. JAIN SLEE Event Router Bean Configuration

Property Name	Property Type	Description
eventRouterThreads	int	defines how many executors should be used by the Event Router, each bounds to a different thread
collectStats	boolean	defines if performance and load statistics should be collected, turning this feature off will increase performance
confirmSbbEntityAttachement	boolean	defines if the event router should reconfirm that sbb entities are attached to activity context, before delivering event, this will avoid that a sbb entity handles concurrent events after it detachs, turning this feature off will increase performance

Property Name	Property Type	Description
executorMapperClassName	Class	This property defines the implementation class of Executor Mapper used by the Event Router, the one above and default uses the activity handle hashcode to do the mapping, an alternative is org.mobicents.slee.runtime.eventrouter .mapping.RoundRobinEvent RouterExecutorMapper, which uses Round Robin algorithm.

3.5.2. Event Router JMX Configuration

Through JMX, the Event Router module configuration can be changed while the container is running. These configuration changes are not persisted.

The JMX MBean that can be used to change the Event Router configuration is named `org.mobicents.slee:name=EventRouterConfiguration`, and provides getters and setters to change each property defined in the persistent configuration. See JMX Console for how the JMX Console can be used to use this MBean.

3.6. Timer Facility Configuration

The JAIN SLEE Timer Facility is the module responsible for managing SLEE timers, and the number of threads it uses is configurable.

The Timer Facility configuration can be changed through an XML file and a JMX MBean.

3.6.1. Timer Facility Persistent Configuration

Configuration is done through an XML descriptor for each Restcomm Cluster. The XML file is named *jboss-beans.xml* and is located at *{JBOSS_HOME}/server/{profile_name}/deploy/restcomm-slee/META-INF*

The configuration is exposed a JBoss Microcontainer Bean:

```
<bean name="Mobicents.JAINSLEE.TimerFacilityConfiguration"
    class="org.mobicents.slee.container.management.jmx.TimerFacilityConfiguration">
    <annotation>@org.jboss.aop.microcontainer.aspects.jmx.JMX(name=
        "org.mobicents.slee:name=TimerFacilityConfiguration",exposedInterface=

org.mobicents.slee.container.management.jmx.TimerFacilityConfigurationMBean.class,
        registerDirectly=true)</annotation>
    <property name="timerThreads">4</property>
</bean>
```

Table 3. JAIN SLEE Timer Facility Bean Configuration

Property Name	Property Type	Description
timerThreads	int	defines how many threads should be used by the Timer Facility
purgePeriod	int	defines the period (in minutes) of purging canceled tasks from the Timer Facility. Use 0 for no purge at all.

3.6.2. Timer Facility JMX Configuration

Through JMX the Timer Facility module configuration can be changed with the container running. Note that such configuration changes are not persisted.

The JMX MBean which can be used to change the Timer Facility configuration is named `org.mobicents.slee:name=TimerFacilityConfiguration`, and provides getters and setters to change each property defined in the persistent configuration. The JMX Console can be used to use this MBean, see JMX Console.

3.6.3. Configuring JAIN SLEE Profiles

JAIN SLEE Profiles is a component used to store data, usually related with a user and/or service profile. JAIN SLEE maps JAIN SLEE Profiles to a Java Persistence API () Datasource, through Hibernate.

There are two configurations for JAIN SLEE Profiles provided as JBoss Microcontainer Beans:

```
<bean name="Mobicents.JAINSLEE.Profiles.JPA.HSQLDBConfig"
    class="org.mobicents.slee.container.deployment.profile.jpa.Configuration">
    <property name="persistProfiles">true</property>
    <property name="clusteredProfiles">false</property>
    <property name="hibernateDatasource">java:/DefaultDS</property>
    <property name="hibernateDialect">org.hibernate.dialect.HSQLDialect</property>
    <depends>jboss.jca:service=DataSourceBinding,name=DefaultDS</depends>
</bean>
<bean name="Mobicents.JAINSLEE.Profiles.JPA.PostgreSQLConfig"
    class="org.mobicents.slee.container.deployment.profile.jpa.Configuration">
    <property name="persistProfiles">true</property>
    <property name="clusteredProfiles">true</property>
    <property name="hibernateDatasource">java:/PostgresDS</property>
    <property name="hibernateDialect">
org.hibernate.dialect.PostgreSQLDialect</property>
</bean>
```

Configurations can be changed, or new ones can be added. For new ones, ensure that the name attribute of the bean element is unique.

Table 4. JAIN SLEE Profiles Bean Configuration

Property Name	Property Type	Description
persistProfiles	boolean	If true, profile changes are persisted into the data source.
clusteredProfiles	boolean	If true, then the container is aware there is a shared data source and that updates may be done by other nodes (for example, deletion of a JAIN SLEE profile table).
hibernateDatasource	String	The name of the Java Datasource deployed in the JBoss Application Server.
hibernateDialect	String	The java class name of the hibernate dialect to use, related with the selected datasource.

To switch the active configuration simply change the parameter injected in the bean named `Mobicents.JAINSLEE.Container`.

3.6.4. Other Configurations

Other JAIN SLEE runtime configuration is done through the following JBoss Microcontainer Bean:

```
<bean name="Mobicents.JAINSLEE.MobicentsManagement"
    class="org.mobicents.slee.container.management.jmx.MobicentsManagement">
    <annotation>@org.jboss.aop.microcontainer.aspects.jmx.JMX(
        name="org.mobicents.slee:service=MobicentsManagement",
        exposedInterface=org.mobicents.slee.container.management.
        jmx.MobicentsManagementMBean.class,
        registerDirectly=true)</annotation>
    <property name="entitiesRemovalDelay">1</property>
    <property name="timerThreads">8</property>
    <property name="loadClassesFirstFromAS">true</property>
    <property name="initializeReferenceDataTypesWithNull">true</property>
</bean>
```

Table 5. Other JAIN SLEE Configurations

Property Name	Property Type	Description
entitiesRemovalDelay	int	The number of minutes before the container forces the ending of SBB entities from a service being deactivated.
timerThreads	int	The number of threads used by the timer facility.

Property Name	Property Type	Description
initializeReferenceDataTypesWithNull	boolean	The flag for initializing SBB CMP fields with Numeric Reference Data types to 0 (false) or null (true).

This configuration can be changed with the container running with JMX. Note that such configuration changes are not persisted.

To change the configuraton, use the JMX MBean named `org.mobicents.slee:service=MobicentsManagement`, which provides getters and setters to change each property defined in the persistent configuration that is configurable with the container running. The JMX Console can be used to use this MBean, as described in JMX Console.

3.6.5. Logging Configuration

Logging configuration is documented in section Simplified Global Log4j Configuration

3.6.6. Congestion Control Configuration

Congesture Control feature configuration is documented in section Congestion Control Configuration

Chapter 4. Managing Restcomm JAIN SLEE

4.1. JAIN SLEE JMX Agent

The JMX Agent exposes all MBeans running in the server, including the ones mandated by the JAIN SLEE 1.1 specification.

The operations done through the JMX Agent are not kept once the server is shutdown. For instance, if a deployable unit is installed through JMX, and the server is shutdown, once the server is started again the deployable unit will not be installed.

By default, the JMX Agent listens to port 1099, and is only available at the host/ip which the server is bound.

4.2. SNMP Agent

JBoss Application Server provides an SNMP Agent, which can be used to ...

TODO

4.3. Managing JAIN SLEE Components

4.4. Persistent Deployable Unit Management

JAIN SLEE provides a file deployer that greatly simplifies the management of JAIN SLEE deployable unit jars. The deployer:

- Handles the installation of enclosed JAIN SLEE components.
- Automatically activates and deactivates services contained.
- Handles the creation, removal, activation, deactivation, link binding and link unbinding of all Resource Adapter Entities.

All operations are persistent, which means that unlike management done through JMX, these survive server shutdowns.

4.4.1. Persistent Deployable Unit Install

To install a deployable unit jar simply copy the file to *$JBOSS_HOME/server/profile_name/deploy/*, where `profile_name` is the server profile name. Child directories can be used.

4.4.2. Persistent Deployable Unit Uninstall

To uninstall a deployable unit jar simply delete the file.

4.4.3. Beyond Deployable Unit (Un)Install

The file deployer provides additional behavior then simply (un)install deployable unit jars, as done through the JAIN SLEE 1.1 DeploymentMBean:

Service (De)Activation

All services contained in the deployable unit jar are activated after the install, and deactivated prior to uninstall. On service activation, if there is an active service in the SLEE, with same service name and vendor, then it is considered an older version, and the SLEE deactivates it. The deactivation of the old, and activation of the new, is done smoothly in a single operation, allowing service upgrades with no down time.

Dependencies Management

The deployer puts the installation process on hold until all of the component's dependencies are installed and activated. When uninstalling, it waits for all of the components which depend on components inside the deployable unit to be uninstalled. After an install or uninstall, the deployer evaluates all operations on hold.

4.4.4. Deploy-Config Extension

A deployable unit jar may include a deploy-config.xml file in its *META-INF/* directory. This file provides additional management actions for persistent install/uninstall operations:

Resource Adaptor Entity Management

It is possible to specify RA entities, and the container will create and activate the RA entities after the deployable unit is installed. During the uninstall process, the container will deactivate and remove those RA entities.

Resource Adaptor Links Management

It is possible to specify RA links, and the container will bind those after the deployable unit is installed. When uninstalled, the container will unbind those RA links as well. The links are set for resource adapter entities created in the *deploy-config.xml* file.

This file should comply with the following schema:

```
<?xml version="1.0" encoding="UTF-8" ?>

<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
  <xs:element name="deploy-config">
    <xs:complexType>
      <xs:sequence>
        <xs:element ref="ra-entity" maxOccurs="unbounded" minOccurs="0"/>
      </xs:sequence>
    </xs:complexType>
  </xs:element>

  <xs:element name="property">
    <xs:complexType>
      <xs:attribute name="name" type="xs:string" use="required" />
      <xs:attribute name="type" type="xs:string" use="required" />
      <xs:attribute name="value" type="xs:string" use="required" />
    </xs:complexType>
  </xs:element>

  <xs:element name="properties">
    <xs:complexType>
      <xs:sequence>
        <xs:element ref="property" maxOccurs="unbounded" minOccurs="0"/>
      </xs:sequence>
      <xs:attribute name="file" type="xs:string" use="optional" />
    </xs:complexType>
  </xs:element>

  <xs:element name="ra-entity">
    <xs:complexType>
      <xs:sequence>
        <xs:element ref="properties" maxOccurs="1" minOccurs="0"/>
        <xs:element ref="ra-link" maxOccurs="unbounded" minOccurs="0"/>
      </xs:sequence>
      <xs:attribute name="resource-adaptor-id" type="xs:string" use="required" />
      <xs:attribute name="entity-name" type="xs:string" use="required" />
    </xs:complexType>
  </xs:element>

  <xs:element name="ra-link">
    <xs:complexType>
      <xs:attribute name="name" type="xs:string" use="required" />
    </xs:complexType>
  </xs:element>

</xs:schema>
```

```
<ra-entity
    resource-adaptor-
id="ResourceAdaptorID[name=JainSipResourceAdaptor,vendor=net.java.slee.sip,version=1.2
]"
    entity-name="SipRA">
    <properties>
        <property name="javax.sip.PORT" type="java.lang.Integer" value="5060" />
    </properties>
    <ra-link name="SipRA" />
</ra-entity>
```

The deploy-config.xml example above defines a resource adaptor entity named `SipRa`, to be created for the resource adaptor with id `ResourceAdaptorID[name=JainSipResourceAdaptor, vendor=net.java.slee.sip, version=1.2]`, and with a single config property named `javax.sip.PORT` of type `java.lang.Integer` and with value `5060`. Additionally, a resource adaptor link named `SipRa` should be bound to the resource adaptor entity.

After the deployable unit is installed, the resource adaptor entity is created, activated and the resource adaptor link is bound. Before the deployable unit is uninstalled, or the server is shutdown, the link is unbound, then the resource adaptor entity is deactivated, and finally the same resource adaptor entity is removed.

The SLEE includes a deploy-config.xml file at *$JBOSS_HOME/server/profile_name/deploy/restcomm-slee*, where `profile_name` is the server profile name, and that file can be used to specify RA entities and links too. If an RA is installed using the persistent deployer, then SLEE reads its own deploy-config.xml file, and if there are RA entities and/or links specified with same RA ID, then the operations to create/activate/deactivate/remove these are executed too, as if these were specified in the Deployable Unit's own deploy-config.xml.

4.5. Ant Tasks

JAIN SLEE includes some tasks for Apache Ant, which can be used for common management tasks done through , when the container is running. The tasks come bundled in *$JBOSS_HOME/server/default/deploy/restcomm-slee/lib/ant-tasks.jar*. To use these, the Ant script must include the following code:

```
<property environment="system" />
<property name="node" value="default" />
<property name="jboss.deploy" value="${system.JBOSS_HOME}/server/${node}/deploy" />

<path id="project.classpath">
    <fileset dir="${jboss.deploy}/restcomm-slee/lib">
        <include name="*.jar" />
    </fileset>
    <fileset dir="${system.JBOSS_HOME}/client">
        <include name="*.jar" />
    </fileset>
</path>

<property name="project.classpath" refid="project.classpath" />

<property name="jnpHost" value="127.0.0.1" />
<property name="jnpPort" value="1099" />

<taskdef name="slee-management"
    classname="org.mobicents.ant.MobicentsManagementAntTask"
    classpath="${project.classpath}" />
```

It is important to understand some properties set by the code above:

`node`
: This property defines the server configuration profile to be used, for further information about those refer to Server Profiles.

`jnpHost`
: The host/ip which Restcomm JAIN SLEE is bound.

`jnpPort`
: The port which the JMX Agent is listening.

The property values can be overridden when invoking the Ant script. To do this, use the parameter `-DpropertyName=propertyValue`. For example, the server profile can be changed from `default` to `all` using `-Dnode=all`.

4.5.1. SLEE Management Task

The `slee-management` task allows a script to manage JAIN SLEE deployable units, services and resource adapters. The operations, or sub-tasks, are done through .

```
<slee-management jnpport="${jnpPort}" host="${jnpHost}">
    <!-- sub-tasks -->
</slee-management>
```

The attributes have the same meaning as the properties listed in the script code to import the tasks

here: Ant Tasks.

Install Deployable Unit Sub-Task

The `install` sub-task installs JAIN SLEE the deployable unit jar pointed by the value of attribute `url`. Example of usage:

```
<slee-management jnpport="${jnpPort}" host="${jnpHost}">
    <install url="file:///tmp/my-deployable-unit.jar" />
</slee-management>
```

Uninstall Deployable Unit Sub-Task

The `uninstall` sub-task uninstalls JAIN SLEE the deployable unit jar which was installed from the value of attribute `url`. Example of usage:

```
<slee-management jnpport="${jnpPort}" host="${jnpHost}">
    <uninstall url="file:///tmp/my-deployable-unit.jar" />
</slee-management>
```

Activate Service Sub-Task

The `activateservice` sub-task activates an already installed JAIN SLEE service, with the id specified by the value of attribute `componentid`. Example of usage:

```
<slee-management host="${jnpHost}" jnpport="${jnpPort}">
    <activateservice
        componentid="ServiceID[name=FooService,vendor=org.mobicents,version=1.0]" />
</slee-management>
```

Deactivate Service Sub-Task

The `deactivateservice` sub-task deactivates an already installed JAIN SLEE service, with the id specified by the value of attribute `componentid`. Example of usage:

```
<slee-management host="${jnpHost}" jnpport="${jnpPort}">
    <deactivateservice
        componentid="ServiceID[name=FooService,vendor=org.mobicents,version=1.0]" />
</slee-management>
```

Create Resource Adaptor Entity Sub-Task

The `createraentity` sub-task creates the resource adaptor entity with the name specified by the value of attribute `entityname`, for an already installed JAIN SLEE resource adaptor, with the id specified by the value of attribute `componentid`. Example of usage:

```
<slee-management host="${jnpHost}" jnpport="${jnpPort}">
    <createraentity
        componentid="ResourceAdaptorID[name=FooRA,vendor=org.mobicents,version=1.0]"
        entityname="FooRA" />
</slee-management>
```

Remove Resource Adaptor Entity Sub-Task

The `removeraentity` sub-task removes the resource adaptor entity with the name specified by the value of attribute `entityname`. Example of usage:

```
<slee-management host="${jnpHost}" jnpport="${jnpPort}">
    <removeraentity entityname="FooRA" />
</slee-management>
```

Activate Resource Adaptor Entity Sub-Task

The `activateraentity` sub-task activates the resource adaptor entity with the name specified by the value of attribute `entityname`. Example of usage:

```
<slee-management host="${jnpHost}" jnpport="${jnpPort}">
    <activateraentity entityname="FooRA" />
</slee-management>
```

Deactivate Resource Adaptor Entity Sub-Task

The `deactivateraentity` sub-task deactivates the resource adaptor entity with the name specified by the value of attribute `entityname`. Example of usage:

```
<slee-management host="${jnpHost}" jnpport="${jnpPort}">
    <deactivateraentity entityname="FooRA" />
</slee-management>
```

Bind Resource Adaptor Link Sub-Task

The `bindralinkname` sub-task binds the resource adaptor link with the name specified by the value of attribute `linkname`, for an already active JAIN SLEE resource adaptor entity, with the name specified by the value of attribute `entityname`. Example of usage:

```
<slee-management host="${jnpHost}" jnpport="${jnpPort}">
    <bindralinkname entityname="FooRA"
        linkname="FooRA" />
</slee-management>
```

Unbind Resource Adaptor Link Sub-Task

The `unbindralinkname` sub-task unbinds the resource adaptor link with the name specified by the value of attribute `linkname`. Example of usage:

```
<slee-management host="${jnpHost}" jnpport="${jnpPort}">
    <unbindralinkname linkname="FooRA" />
</slee-management>
```

4.6. Management Consoles

4.6.1. JMX Console

JBoss Application Server provides a simple web console that gives quick access to all MBeans registered in the server, which includes the ones defined by the JAIN SLEE 1.1 specification.

To access the JMX console once the server is running, point a web browser to `http://ip:8080/jmx-console`, where `ip` is the IP/Host the container is bound. Unless set during start up, the IP/Host will be `127.0.0.1/localhost` by default.

MBeans in the domain `javax.slee` are all standard JAIN SLEE 1.1 MBeans, while the ones in the domain `org.mobicents.slee` are proprietary to Restcomm JAIN SLEE. The following ones are of particular interest:

`org.mobicents.slee:service=MobicentsManagement`
: the MBean which can be used to make non persistent changes to the server configuration, in runtime. The operation `dumpContainerState` displays a textual snapshot of the server's state, which can be used to quickly look for memory leaks or other debug/profiling related tasks.

`org.mobicents.slee:name=DeployerMBean`
: the MBean allows interaction with the persistent deployable unit deployer. The operation `showStatus` displays a textual snapshot of the deployers's state, which can be used to quickly find out if there is any deployable unit deployment pending, for instance, due to missing dependencies.

`org.mobicents.slee:name=CongestionControlConfiguration`
: the MBean allows changing or retrieving the Congestion Control feature, with the container running. Details are provided in section Congestion Control Configuration.

For further information about JAIN SLEE 1.1 MBeans and their operations refer to the JAIN SLEE 1.1 Specification, all are covered with great detail.

4.6.2. SLEE Management Console

The JMX Console is simple but the MBeans operations were made considering its invocation by management clients, not people using browsers. The SLEE Management Console is a web application that provides high level management functionality for the SLEE, and comes pre-deployed in SLEE binary releases. To access this console point a web browser to

`http://ip:8080/slee-management-console`, where `ip` is the IP/Host the container is bound. Unless set during start up, the IP/Host will be `127.0.0.1/localhost` by default.

Full documentation for this management tool can be found in *docs/tools/slee-management-console* directory.

4.6.3. Jopr Console

Joprwas developed to become Red Hat's Middleware Administration Tool, providing an unified interface and extensible model, to be used mainly to control and monitor servers individually, or clusters.

Restcomm JAIN SLEE binary release includes a JoprConsole in *tools/jopr-plugin*, with standalone documentation on same path, but inside *docs* directory.

4.6.4. TWIDDLE CLI

Both, Console and Jopr Console, are graphic(web) based tools. Some deployments may require command line access to Restcomm . To aid such cases, Restcomm offers `TWIDDLE` based CLI. It allows to manage single instance (remote or local) of Restcomm server.

Restcomm JAIN SLEE binary release includes a TWIDDLE CLI in *tools/twiddle*, with standalone documentation on same path, but inside *docs* directory.

Chapter 5. Logging, Traces and Alarms

5.1. Log4j Logging Service

In Restcomm JAIN SLEE `Apache log4j` is used for logging. If you are not familiar with the log4j package and would like to use it in your applications, you can read more about it at the Jakarta web site.

Logging is controlled from a central *conf/jboss-log4j.xml* file, in each server configuration profile. This file defines a set of appenders specifying the log files, what categories of messages should go there, the message format and the level of filtering. By default, in Restcomm produces output to both the console and a log file (*log/server.log*).

There are 6 basic log levels used: TRACE, DEBUG, INFO, WARN, ERROR and FATAL.

Logging is organized in categories and appenders. Appenders control destination of log entries. Different appenders differ in configuration, however each supports threshold. Threshold filters log entries based on their level. Threshold set to WARN will allow log entry to pass into appender if its level is WARN, ERROR or FATAL, other entries will be discarded. For more details on appender configuration please refer to its documentation or java doc.

The logging threshold on the console is INFO, by default. In contrast, there is no threshold set for the server.log file, so all generated logging messages are logged there.

Categories control level for loggers and its children, for details please refer to log4j manual.

By default Restcomm JAIN SLEE inherits level of INFO from root logger. To make platform add more detailed logs, file *conf/jboss-log4j.xml* has to be altered. Explicit category definition for Restcomm JAIN SLEE looks like:

```
<category name="org.mobicents.slee">
    <priority value="INFO"/>
</category>
```

This limits the level of logging to INFO for all Restcomm JAIN SLEE classes. It is possible to declare more categories with different level, to provide logs with greater detail.

For instance, to provide detailed information on Restcomm JAIN SLEE transaction engine in separate log file(*txmanager.log*), file *conf/jboss-log4j.xml* should contain entries as follows:

```
<appender name="TXMANAGER" class="org.jboss.logging.appender.RollingFileAppender">
    <errorHandler class="org.jboss.logging.util.OnlyOnceErrorHandler"/>
    <param name="File" value="${jboss.server.home.dir}/log/txmanager.log"/>
    <param name="Append" value="false"/>
    <param name="MaxFileSize" value="500KB"/>
    <param name="MaxBackupIndex" value="1"/>

    <layout class="org.apache.log4j.PatternLayout">
        <param name="ConversionPattern" value="%d %-5p [%c] %m%n"/>
    </layout>
</appender>

<category name="org.mobicents.slee.runtime.transaction">
    <priority value="DEBUG" />
    <appender-ref ref="TXMANAGER"/>
</category>
```

This creates a new file appender and specifies that it should be used by the logger (or category) for the package org.mobicents.slee.runtime.transaction.

The file appender is set up to produce a new log file every day rather than producing a new one every time you restart the server or writing to a single file indefinitely. The current log file is txmanager.log. Older files have the date they were written added to their filenames.

5.1.1. Simplified Global Log4j Configuration

Besides manual logging configuration, described previously, Restcomm JAIN SLEE also exposes management operations that greatly simplify such configuration, allowing the administrator to select through predefined and complete logging configuration presets. Such operations are available in MBean named *org.mobicents.slee%3Aservice%3DMobicentsManagement*, and the available presets are:

Level

The available management operations are:

- DEFAULT: Regular logging, at INFO level, displaying most user-related messages;
- DEBUG: More verbose logging, mostly using DEBUG/TRACE level, displaying message of interest for developers;
- PRODUCTION: Low verbosity and async logging, mostly in WARN level, for systems in production so that logging does impact performance.

JMX Operation

- getLoggingConfiguration: retrieves what is the current logging configuration;
- switchLoggingConfiguration: allows switching to a different configuration preset;
- setLoggingConfiguration: used to upload a complete logging configuration.

Custom presets can be easily deployed in the application server too. Simply name the configuration

file as *jboss-log4j.xml.PRESET_NAME*, where `PRESET_NAME` should be unique preset name, and copy it to directory *$JBOSS_HOME/server/profile_name/deploy/restcomm-slee/log4j-templates*, where `profile_name` is the server profile name.

These procedures changes the whole JBoss Application Server Platform logging configuration, so it will affect also logging for other running applications besides the JAIN SLEE container.

5.2. Alarm Facility

The `JAIN SLEE Alarm Facility` is used by SBBs, Resource Adaptors, and Profiles to request the SLEE to raise or clear alarms. If a request is made to raise an alarm and the identified alarm has not already been raised, the alarm is raised and a corresponding alarm notification is generated by the `AlarmMBean`. If a request is made to clear an alarm and the identified alarm is currently raised, the alarm is cleared and a corresponding alarm notification is generated by the `AlarmMBean`.

Alarm notifications are intended for consumption by management clients external to the SLEE. The management client is responsible for registering to receive alarm notifications generated by the Alarm Facility through the external management interface of the `Alarm Facility`. The management client may optionally provide notification filters so that only the alarm notifications that the management client would like to receive are transmitted to the management client.

For further information on how to use `JAIN SLEE Alarm Facility` and receive JMX notifications refer to the JAIN SLEE 1.1 Specification.

5.3. Trace Facility

Notification sources such as SBBs, Resource Adaptors, Profiles, and SLEE internal components can use the `Trace Facility` to generate trace messages intended for consumption by external management clients. Management clients register to receive trace messages generated by the `Trace Facility` through the external management interface (MBean). Filters can be applied, in a similar way as in case of Alarms.

Within the SLEE, notification sources use a `tracer` to emit trace messages. A tracer is a named entity. Tracer names are case-sensitive and follow the Java hierarchical naming conventions. A tracer is considered to be an ancestor of another tracer if its name followed by a dot is a prefix of the descendant tracer's name. A tracer is considered to be a parent of a tracer if there are no ancestors between itself and the descendant tracer. For example, the tracer named `com` is the parent tracer of the tracer named `com.foo` and an ancestor of the tracer named `com.foo.bar`.

All tracers are implicitly associated with a notification source, which identifies the object in the SLEE that is emitting the trace message and is included in trace notifications generated by the `Trace MBean` on behalf of the tracer. For instance, an SBB notification source is composed by the SBB id and the Service id.

Multiple notification sources may have tracers with same name in SLEE. Comparing with common logging frameworks, this would mean that the notification source would be part of the log category or name.

For further information on how to use `JAIN SLEE Trace Facility` and receive JMX notifications refer to the JAIN SLEE 1.1 Specification.

5.3.1. JAIN SLEE Tracers and Log4j

Restcomm JAIN SLEE Tracers additionally log messages to `Apache Log4j`, being the log4j category, for notification source `X`, defined as `javax.slee.` concatenated with the `X.toString()`.

For instance, the full log4j logger `name` for tracer named `GoogleTalkBotSbb`, of sbb notification source with `SbbID[name=GoogleTalkBotSbb,vendor=restcomm,version=1.0]` and `ServiceID[name=GoogleTalkBotService,vendor=restcomm,version=1.0]`, would be `javax.slee.SbbNotification[service=ServiceID[name=GoogleTalkBotService, vendor=restcomm,version=0.1], sbb=SbbID[name=GoogleTalkBotSbb,vendor=restcomm, version=0.1]].GoogleTalkBotSbb` (without the spaces or breaks), which means a log4j category defining its level as `DEBUG` could be:

```
<category
    name="javax.slee.SbbNotification[service=ServiceID[name=GoogleTalkBotService,
    vendor=restcomm,version=0.1],sbb=SbbID[name=GoogleTalkBotSbb,
    vendor=restcomm,version=0.1]]">
    <priority value="DEBUG" />
</category>
```

The relation of JAIN SLEE `tracers` and log4j `loggers` goes beyond log4j showing tracer's messages, changing the tracer's log4j logger `effective level` changes the tracer level in SLEE, and vice-versa. Since JAIN SLEE tracer levels differ from log4j logger levels a mapping is needed:

Table 6. Mapping JAIN SLEE Tracer Levels with Apache Log4j Logger Levels

Tracer Level	Logger Level
OFF	OFF
SEVERE	ERROR
WARNING	WARN
INFO	INFO
CONFIG	INFO
FINE	DEBUG
FINER	DEBUG
FINEST	TRACE

Chapter 6. Restcomm JAIN SLEE Clustering

JAIN SLEE supports clustering, whether it is simple high availability () or complete fault tolerance () support. This is achieved through the replication of the container state. The Restcomm JAIN SLEE implementation also exposes a clustering extension for Resource Adaptors components, which live outside the container.

6.1. High Availability and Fault Tolerance

The used JAIN SLEE clustering mode is defined by the selected server profile:

JAIN SLEE reuses the JBoss Application Server clustering framework, and if all nodes of a cluster are in the same network then the underlying JBoss Application Server clustering will automatically handle the discovery of new cluster nodes and join these to the cluster. For more complicated setups, refer to the JBoss Application Server clustering documentation.

6.1.1. High Availability Mode

High availability mode provides no clustering functionality per say. The mode is useful when deploying for example single node, non-replicated or hot-cold configurations. In this mode all clustering needs to be explicitly done by the developer where applicable. In this sense, mode is not a clustered mode.

The `default` server profile is used to start the server in mode.

6.1.2. Fault Tolerant Mode

The fault tolerant mode is a fully clustered mode with state replication. An FT cluster can be viewed as one virtual container that extend over all the JAIN SLEE nodes that are active in the cluster. All activity context and Sbb entity data is replicated across the cluster nodes and is hence fully redundant. Events are not failed over, due to performance constraints, which means that an event fired and not yet routed will be lost if its cluster node fails.

The `all` server profile is used to start the server in mode.

6.2. Component Redundancy in Fault Tolerant Clusters

The fault tolerant clustering mode provides clustering for most of the JAIN SLEE components. JAIN SLEE components can be divided into internal and external components. Internal components are logically contained by the JAIN SLEE container, and external components are at least partly outside the container.

For a concrete example of how the container behaves in mode, see Fault Tolerant Clustering - A Concrete Example.

6.2.1. Internal Component Redundancy

Internal SLEE components are components that are completely inside the JAIN SLEE container. This group of components include entities, internal activities, events and timers. With the exception of events, all internal components will be fully redundant in a JAIN SLEE configuration.

SBB entities are fully replicated. entities are always serialized and saved by the container, regardless of the clustering profile. In an environment the container will replicate this serialized state to other nodes in the cluster so that it can be retrieved if the node fails or if the entity is processed in another node. All entities will hence be accessible by any node in the cluster at any given time.

Timers are fully replicated. Timers created in a given node will be executed in that same node. If the node leaves the cluster, all active timers from that node are recreated and run in another node.

Activity context interfaces (), as well as activity handles are fully replicated. The s for all activities are replicated within a fault tolerant cluster. However, the activity object is not replicated by default and needs to be handled by the resource adaptor that owns the activity in question if replication is required. The activity objects for all internal activities, e.g. null activities, profile table activities and service activities, are fully replicated.

Events are not replicated because of performance constraints. Hence, all events fired in a node is routed only in that node. However, if an event is fired in one node, and an entity created in another node has attached to that , the entity will be retrieved in the node that fired the event and the event will be delivered to it. Hence, even though the event is fired in a single node, the effects will be cluster-wide. Because the events are not replicated, any event currently being routed in a node that fails, will be lost.

6.2.2. External Component Redundancy

External JAIN SLEE components are components that are on the border between the SLEE container and the outside environment. This group of components include resource adaptors and external activities, neither of which are replicated by default.

The Resource adaptors may use the Fault Tolerant Resource Adaptor API extension of the JAIN SLEE 1.1 specification in order to be cluster-aware. Refer to the Fault Tolerant Resource Adaptor API and Resource Adaptor Activity Replication sections for more information on how to achieve resource adaptor and activity object redundancy.

6.3. Managing Components in Restcomm JAIN SLEE Cluster

JAIN SLEE clustering does not require special components management. Components can be deployed and undeployed in all cluster modes, including fault tolerant setups, and the cluster will handle the operation correctly. However, there are certain behaviours in fault tolerance setups to be aware of:

`Service Activation`

JAIN SLEE Service started events are only fired on the first cluster node started.

`Service Deactivation`

Only the last node will force the removal of the service's entities.

`Resource Adaptor Entity Deactivation`

Only the last node will force the removal of all its activities.

Chapter 7. Fault Tolerant Resource Adaptor API

JAIN SLEE Resource Adaptors exist on the boundary between the container and the underlying protocol. The specification contract requires the object to implement the `javax.slee.resource.ResourceAdaptor`interface. This interface defines callbacks, which SLEE uses to interact with the , including one to provide the `javax.slee.resource.ResourceAdaptorContext`object. The Resource Adaptor Context provides object facilities to interact with SLEE.

The JAIN SLEE 1.1 RA API is a major milestone, standardizing RA and JSLEE contract. However, it misses an API for clustering, which is critical for a RA deployed in a clustered JAIN SLEE environment. The JAIN SLEE 1.1 contract does not define any fault tolerant data source nor cluster state callbacks.

The `Fault Tolerant RA API`extends the JAIN SLEE 1.1 RA , providing missing features related to clustering. An effort has been made keep the API similar to the standard RA contract, so that anyone who has developed a JAIN SLEE 1.1 RA is able to easily use the proprietary API extension.

7.1. The Fault Tolerant Resource Adaptor Object

The core of the Fault Tolerant RA API is the `org.mobicents.slee.resource.cluster.FaultTolerantResourceAdaptor` `interface. It is intended to be used instead of the` `javax.slee.resource.ResourceAdaptor`interface from the JAIN SLEE 1.1 Specification.

The FaultTolerant interface provides three new callback methods used by the container:

`setFaultTolerantResourceAdaptorContext(FaultTolerantResourceAdaptorContext context)`

This method provides the RA with the `org.mobicents.slee.resource.cluster.FaultTolerantResourceAdaptorContext` ` object, which gives access to facilities related with the cluster. This method is invoked by SLEE after invoking` `raConfigure(ConfigProperties properties)`from JAIN SLEE 1.1 specs.

`unsetFaultTolerantResourceAdaptorContext()`

This method indicates that the RA should remove any references it has to the FaultTolerantResourceAdaptorContext, as it is not valid anymore. The method is invoked by SLEE before invoking `unsetResourceAdaptorContext()`from JAIN SLEE 1.1 specs.

`failOver(K key)`

Callback from SLEE when the local RA was selected to recover the state for a replicated data key, which was owned by a cluster member that failed. The RA may then restore any runtime resources associated with such data.

`dataRemoved(K key)`

Optional callback from SLEE when the replicated data key was removed from the cluster, this may be helpful when the local RA maintains local state.

7.2. The Fault Tolerant Resource Adaptor Context

The clustered RA context follows the contract of JAIN SLEE 1.1 specification interface `javax.slee.resource.ResourceAdaptorContext`. It gives access to facilities that the RA may use when run in a clustered environment.

The cluster contract is defined in: `org.mobicents.slee.resource.cluster.FaultTolerantResourceAdaptorContext`. It provides critical information, such as if SLEE is running in local mode (not clustered), if it is the head/master member of the cluster, and what the members of the cluster are.

7.2.1. The Fault Tolerant Resource Adaptor Replicated Data Sources

The Fault Tolerant Resource Adaptor Context provides two data sources to replicate data in cluster:

`ReplicatedData`
: A container for serializable data, which is replicated in the SLEE cluster, but don't require any failover.

`ReplicatedDataWithFailover`
: A `ReplicatedData`which requires fail over callbacks, this means, that for all data stored here, when a cluster member goes down, the SLEE in another cluster member will invoke the `failOver(Key k)`callback in the Fault Tolerant RA object.

When retrieved from the context through a boolean parameter, both types of `ReplicatedData` can activate the callback on the `FaultTolerantResourceAdaptor`which indicates that a specific data was removed from the cluster remotely.

7.2.2. The Fault Tolerant Resource Adaptor Timer

The standard Resource Adaptor Context provides a `java.util.Timer`, which can be used by the Resource Adaptor to schedule the execution of tasks, the Fault Tolerant Resource Adaptor Context provides `org.mobicents.slee.resource.cluster.FaultTolerantTimer`, an alternative scheduler which is able to fail over tasks scheduled.

The Fault Tolerant Timer has an interface that resembles the JDK's `ScheduledExecutorService`, with two fundamental changes to allow a proper interaction in a cluster environment:

`Task Interface`
: Instead of relying on pure `Runnable` tasks, tasks must follow a specific interface `FaultTolerantTimerTask`, to ensure that the timer is able to replicate the task's data, and failover the task in any cluster node.

`Task Cancellation`
: Cancellation of task is done through the Timer interface, not through `ScheduledFuture` objects, this allows the operation to be easily done in any cluster node.

The Fault Tolerant Timer interface:

`cancel(Serializable taskID)`
: Requests the cancellation of the FT Timer Task with the specified ID.

`configure(FaultTolerantTimerTaskFactory taskFactory, int threads)`
: Configures the fault tolerant timer, specifying the timer task factory and the number of threads the timer uses to execute tasks.

`isConfigured()`
: Indicates if the timer is configured.

`schedule(FaultTolerantTimerTask task, long delay, TimeUnit unit)`
: Creates and executes a one-shot action that becomes enabled after the given delay.

`scheduleAtFixedRate(FaultTolerantTimerTask task, long initialDelay, long period, TimeUnit unit)`:: Creates and executes a periodic action that becomes enabled first after the given initial delay, and subsequently with the given period; that is executions will commence after `initialDelay` then `initialDelay+period`, then `initialDelay + 2 * period`, and so on. If any execution of the task encounters an exception, subsequent executions are suppressed. Otherwise, the task will only terminate via cancellation or termination of the executor. If any execution of this task takes longer than its period, then subsequent executions may start late, but will not concurrently execute.

`scheduleWithFixedDelay(FaultTolerantTimerTask task, long initialDelay, long delay, TimeUnit unit)`:: Creates and executes a periodic action that becomes enabled first after the given initial delay, and subsequently with the given delay between the termination of one execution and the commencement of the next. If any execution of the task encounters an exception, subsequent executions are suppressed. Otherwise, the task will only terminate via cancellation or termination of the executor.

There is a single Fault Tolerant Timer per RA Entity, and when first retrieved, and before any task can be scheduled, the Fault Tolerant Timer must be configured, through its configure(...) method.

The Fault Tolerant Resource Adaptor Timer Task

As mentioned in previous section, tasks submitted to the Fault Tolerant Timer must follow a specific interface, `FaulTolerantTimerTask`, it is nothing more than a `Runnable`, which provides the replicable `FaultTolerantTimerTaskData`. The task data must be serializable and provide a Serializable task ID, which identifies the task, and may be used to cancel its execution.

The Fault Tolerant Resource Adaptor Timer Example Usage

A simple example for the usage of the Faul Tolerant Timer Task:

```
// data, task and factory implementation

package org.mobicents.slee.resource.sip11;

import java.io.Serializable;

import org.mobicents.slee.resource.cluster.FaultTolerantTimerTaskData;
```

```
public class FaultTolerantTimerTaskDataImpl implements
        FaultTolerantTimerTaskData {

    private final String taskID;

    public FaultTolerantTimerTaskDataImpl(String taskID) {
        this.taskID = taskID;
    }

    @Override
    public Serializable getTaskID() {
        return taskID;
    }
}

package org.mobicents.slee.resource.sip11;

import org.mobicents.slee.resource.cluster.FaultTolerantTimerTask;
import org.mobicents.slee.resource.cluster.FaultTolerantTimerTaskData;
import org.mobicents.slee.resource.cluster.FaultTolerantTimerTaskFactory;

public class FaultTolerantTimerTaskFactoryImpl implements
        FaultTolerantTimerTaskFactory {

    private final SipResourceAdaptor ra;

    public FaultTolerantTimerTaskFactoryImpl(SipResourceAdaptor ra) {
        this.ra = ra;
    }

    @Override
    public FaultTolerantTimerTask getTask(FaultTolerantTimerTaskData data) {
        return new FaultTolerantTimerTaskImpl(ra, data);
    }
}

package org.mobicents.slee.resource.sip11;

import org.mobicents.slee.resource.cluster.FaultTolerantTimerTask;
import org.mobicents.slee.resource.cluster.FaultTolerantTimerTaskData;

public class FaultTolerantTimerTaskImpl implements FaultTolerantTimerTask {

    private final SipResourceAdaptor ra;
    private final FaultTolerantTimerTaskData data;

    public FaultTolerantTimerTaskImpl(SipResourceAdaptor ra,
            FaultTolerantTimerTaskData data) {
        this.ra = ra;
        this.data = data;
```

```
    }

    @Override
    public void run() {
        ra.getTracer("FaultTolerantTimerTaskImpl").info("Timer executed.");
    }

    @Override
    public FaultTolerantTimerTaskData getTaskData() {
        return data;
    }
}

// ra code retrieving the timer, configuring it and submiting a task

public void setFaultTolerantResourceAdaptorContext(
        FaultTolerantResourceAdaptorContext<SipActivityHandle, String> context) {
    this.ftRaContext = context;
    FaultTolerantTimer timer = context.getFaultTolerantTimer();
    timer.config(new FaultTolerantTimerTaskFactoryImpl(this), 4);
    FaultTolerantTimerTaskDataImpl data = new FaultTolerantTimerTaskDataImpl("xyz");
    FaultTolerantTimerTaskImpl task = new FaultTolerantTimerTaskImpl(this,
            data);
    timer.schedule(task, 30, TimeUnit.SECONDS);
}
```

Chapter 8. Resource Adaptor Activity Replication

The Resource Adaptor API includes an optional component named `javax.slee.resource.Marshaler`, which is responsible, besides other functions, for converting Activity Handles to byte arrays and vice-versa. Also relevant, the Resource Adaptor, when starting activities, may provide a flag indicating that the container may marshall the activity (using the Marshaler). In case of a container cluster with data replication, if an activity is to be replicated then the Marshaler must be provided and the activity flags must activate the flag MAY_MARSHALL, otherwise the activity is not replicated and if a node fails all its activities are removed from the container cluster.

The activity replication doesn't mean that the activity object is replicated by any means, only the related Activity Handle. The Resource Adaptor must use the Fault Tolerant RA API or its own means to replicate any additional data to support that presence of the activity in all nodes of the cluster. Usage of the Fault Tolerant RA API is recommended since it reuses the clustering setup of the container.

Chapter 9. Firing Events from Java EE Applications

9.1. SLEE Connection Factory

The JAIN SLEE specification includes an API for interaction with JAIN SLEE container, with the interface `javax.slee.connection.SleeConnectionFactory` upfront, which allows external applications, such as an EJB, to create connections to a specific JAIN SLEE instance, and use that to fire events.

JAIN SLEE provides two different implementations of the API, one for access in the same JVM, another for remote access. Both implementations expose the SLEE Connection Factory in the local JNDI tree, thus the same code is used by the application independently of the implementation used.

9.1.1. Local SLEE Connection Factory

JAIN SLEE container instances always expose the local SLEE Connection Factory in its JVM, which means that an external application running in the same JVM doesn't need any additional tools or setup to use it.

9.1.2. Remote SLEE Connection Factory

JAIN SLEE includes a tool, named Remote SLEE Connection Tool, which is a JCA connector that can be deployed in any Java EE application server. Once deployed this connector installs the factory into JNDI, which communicates with the remote SLEE container through RMI.

Restcomm Remote SLEE Connection Tool is bundled in all Restcomm JAIN SLEE releases, in the tools/remote-slee-connection directory, including its own documentation.

9.1.3. Using SLEE Connection Factory

Below is example code which retrieves the SLEE Connection Factory and uses it to fire an event into the JAIN SLEE container. The code is the same whether the SLEE container is in the same JVM or not.

```
    // retrieves JNDI context
    InitialContext context = new InitialContext();

    // retrieves the connection factory from JNDI
    SleeConnectionFactory factory = (SleeConnectionFactory) context.lookup
("java:/MobicentsConnectionFactory");

    // creates a connection to the SLEE container
    SleeConnection connection = factory.getConnection();

    // creates the activity handle which will be used to fire the event
    ExternalActivityHandle handle = connection.createActivityHandle();

    // retrieves the event type ID
    EventTypeID eventTypeID = connection.getEventTypeID("CustomEvent", "...", "1.0");

    // creates the event object
    CustomEvent eventObject = new CustomEvent();

    // fires the event in the remote SLEE container
    connection.fireEvent(eventObject, eventTypeID, handle, null);

    // closes the connection
    connection.close();
```

Chapter 10. JAIN SLEE 1.1 Extensions

Restcomm exposes proprietary extensions to the 1.1 specification, to allow the development of easier or more powerful application code.

The extensions were discussed among multiple vendors, and should become part of the standard in next revision, but there is no guarantee that portability won't be lost when using those.

The extensions source code is available in the Restcomm SLEE Community Git repository, specifically at api/extensions subdirectory. Its javadocs are bundled in the SLEE binary release, in docs/container/javadoc subdirectory. The setup for the source project in Eclipse IDE is similar to the container core, as seen in Setting JAIN SLEE Source Code Projects in Eclipse IDE.

Java archives (JARs) with compiled classes, javadocs and sources are available in the Sonatype Maven Repository at https://oss.sonatype.org/content/groups/public/org/mobicents/servers/jainslee/api/jain-slee-11-ext/

10.1. SbbContext Extension

This extension to JAIN SLEE 1.1 introduces *org.mobicents.slee.SbbContextExt* interface, which extends *javax.slee.SbbContext* with methods to retrieve SLEE factories and facilities, avoiding the usage of JNDI context.

```
package org.mobicents.slee;

import javax.slee.ActivityContextInterface;
import javax.slee.Sbb;
import javax.slee.SbbContext;
import javax.slee.facilities.ActivityContextNamingFacility;
import javax.slee.facilities.AlarmFacility;
import javax.slee.facilities.TimerFacility;
import javax.slee.nullactivity.NullActivityContextInterfaceFactory;
import javax.slee.nullactivity.NullActivityFactory;
import javax.slee.profile.ProfileFacility;
import javax.slee.profile.ProfileTableActivityContextInterfaceFactory;
import javax.slee.resource.ResourceAdaptorTypeID;
import javax.slee.serviceactivity.ServiceActivityContextInterfaceFactory;
import javax.slee.serviceactivity.ServiceActivityFactory;

public interface SbbContextExt extends SbbContext {

    public Object getActivityContextInterfaceFactory(
            ResourceAdaptorTypeID raTypeID) throws NullPointerException,
            IllegalArgumentException;

    public ActivityContextNamingFacility getActivityContextNamingFacility();

    public AlarmFacility getAlarmFacility();
```

```
    public NullActivityContextInterfaceFactory
getNullActivityContextInterfaceFactory();

    public NullActivityFactory getNullActivityFactory();

    public ProfileFacility getProfileFacility();

    public ProfileTableActivityContextInterfaceFactory
getProfileTableActivityContextInterfaceFactory();

    public Object getResourceAdaptorInterface(ResourceAdaptorTypeID raTypeID,
            String raEntityLink) throws NullPointerException,
            IllegalArgumentException;

    public SbbLocalObjectExt getSbbLocalObject()
            throws TransactionRequiredLocalException, IllegalStateException,
            SLEEException;

    public ServiceActivityContextInterfaceFactory
getServiceActivityContextInterfaceFactory();

    public ServiceActivityFactory getServiceActivityFactory();

    public TimerFacility getTimerFacility();
}
```

The `getActivityContextInterfaceFactory(ResourceAdaptorTypeID)` *method*

Retrieves the ActivityContextInterface factory for the Resource Adaptor Type with the specified ID.

The `getActivityContextNamingFacility()` *method*

Retrieves the Activity Context Naming Facility.

The `getAlarmFacility()` *method*

Retrieves the Alarm Facility.

The `getNullActivityContextInterfaceFactory()` *method*

Retrieves the Null Activity Context Interface Factory.

The `getNullActivityFactory()` *method*

Retrieves the Null Activity Factor.

The `getProfileFacility()` *method*

Retrieves the Profile Facility.

The `getProfileTableActivityContextInterfaceFactory()` *method*

Retrieves the Profile Table Activity Context Interface Factory.

The `getResourceAdaptorInterface(ResourceAdaptorTypeID,String)` *method*

Retrieves the interface to interact with a specific Resource Adaptor entity, identified by both the entity link name and the Resource Adaptor Type ID.

The `getSbbLocalObject()` method

Exposes the SBB local object with the extension interface to avoid unneeded casts.

The `getServiceActivityContextInterfaceFactory()` method

Retrieves the Service Activity Context Interface Factory.

The `getServiceActivityFactory()` method

Retrieves the Service Activity Factory.

The `getTimerFacility()` method

Retrieves the Timer Facility.

10.2. ChildRelation Extension

This extension to JAIN SLEE 1.1 introduces the `org.mobicents.slee.ChildRelationExt` interface, which extends `javax.slee.ChildRelation` with methods to create and retrieve SBB entities by name.

```
package org.mobicents.slee;

public interface ChildRelationExt extends ChildRelation {

    public SbbLocalObjectExt create(String name) throws CreateException,
            TransactionRequiredLocalException, SLEEException;

    public SbbLocalObjectExt get(String name)
            throws TransactionRequiredLocalException, SLEEException;

}
```

The `create(String)` method

Creates a new SBB entity of the SBB type associated with the relation, with the specified name. The new SBB entity is automatically added to the relationship collection. The returned object may be cast to the required local interface type using the normal Java typecast mechanism. This method is a mandatory transactional method.

The `get(String)` method

Retrieves the SBB entity associated with the child relation with the specified name. This method is a mandatory transactional method.

10.3. SbbLocalObject Extension

This extension to JAIN SLEE 1.1 introduces the `org.mobicents.slee.SbbLocalObjectExt` interface, which extends `javax.slee.SbbLocalObject` with methods to retrieve the parent SBB Entity, if any, and to also retrieve information such as the child name, and the parent child relation name.

```
package org.mobicents.slee;

public interface SbbLocalObjectExt extends SbbLocalObject {

    public String getChildRelation() throws TransactionRequiredLocalException,
            SLEEException;

    public String getName() throws NoSuchObjectLocalException,
            TransactionRequiredLocalException, SLEEException;

    public SbbLocalObjectExt getParent() throws NoSuchObjectLocalException,
            TransactionRequiredLocalException, SLEEException;
}
```

The `getChildRelation()` method

Retrieves the name of the child relation used to create this object. This method is a mandatory transactional method.

The `getName()` method

Retrieves the name of the object. This method is a mandatory transactional method.

The `getParent()` method

Retrieves the parent SBB object. This method is a mandatory transactional method.

10.4. ProfileContext Extension

This extension to JAIN SLEE 1.1 introduces *org.mobicents.slee.ProfileContextExt* interface, which extends *javax.slee.ProfileContext* with methods to retrieve SLEE alarm facility, avoiding the usage of JNDI context.

```
package org.mobicents.slee;

import javax.slee.facilities.AlarmFacility;
import javax.slee.profile.Profile;
import javax.slee.profile.ProfileContext;

public interface ProfileContextExt extends ProfileContext {

    public AlarmFacility getAlarmFacility();

}
```

The `getAlarmFacility()` method

Retrieves the Alarm Facility.

10.5. ActivityContextInterface Extension

This simple extension to JAIN SLEE 1.1 introduces *org.mobicents.slee.ActivityContextInterfaceExt* interface, which extends *javax.slee.ActivityContextInterface* with methods to retrieve the timers and names bound to the ACI.

```
package org.mobicents.slee;

import javax.slee.ActivityContextInterface;
import javax.slee.facilities.TimerID;

public interface ActivityContextInterfaceExt extends ActivityContextInterface {

    public TimerID[] getTimers();

    public String[] getNamesBound();

}
```

The `getTimers()` *method*

Retrieves the IDs of timers currently set which are related to the ACI.

The `getNamesBound()` *method*

Retrieves the names currently bound to the ACI.

The `suspend()` *method*

This feature may be used before attaching to an ActivityContextInterface, to ensure that any event fired concurrently will be received. It suspends routing of events in the activity context immediately, till the active transaction ends.

10.6. Library References Extension

JAIN SLEE 1.1 standard introduced the Library component, a wrapper for a set of jars and/or classes to be referenced and used by other components types, such as SBBs.

The usage of the standard Library component is very limited, each Library can only refer other Library components. Due to this limitation a developer may not be able to include classes in a Library that depend, just as example, on Resource Adaptor Type interfaces, unless of course those interfaces are in a Library too.

This extension allows libraries to reference other component types, which the developer should use when the classes in the Library need to use classes from that component, by simply extending the JAIN SLEE 1.1 Library Jar XML descriptor.

10.6.1. Extended Library Jar XML Descriptor DTD

The DTD document changes for the extended library jar XML descriptor:

```
<!--
The library element defines a library.  It contains an optional description
about the library, the name, vendor, and version of the library being defined,
zero or more references to any other components that this library
depends on, and information about zero or more jar files that contain
prepackaged classes and resources to be included with the library.

The classes and resources for a library are the sum total of the classes and
resources contained in:
- the library component jar itself (if any)
- the library jars specified by the jar elements (if any)

All these classes are loaded by the same classloader.

Used in: library-jar
-->
<!ELEMENT library (description?, library-name, library-vendor, library-version,
 event-type-ref*, library-ref*, profile-spec-ref*, resource-adaptor-type-ref*,
 sbb-ref*, jar*)>

<!--
The event-type-ref element identifies an event type that the library classes depend.
It contains the name, vendor,and version of the event type.

Used in: library
-->
<!ELEMENT event-type-ref (event-type-name, event-type-vendor,
          event-type-version)>

<!--
The event-type-name element contains the name of an event type referred by
the library.

Used in: event-type-ref

Example:
    <event-type-name>
        javax.csapi.cc.jcc.JccCallEvent.CALL_CREATED
    </event-type-name>
-->
<!ELEMENT event-type-name (#PCDATA)>

<!--
The event-type-vendor element contains the vendor of an event type referred by
the library

Used in: event-type-ref

Example:
    <event-type-vendor>javax.csapi.cc.jcc</event-type-vendor>
-->
```

```
<!ELEMENT event-type-vendor (#PCDATA)>

<!--
The event-type-version element contains the version of an event type referred by
the library

Used in: event-type-ref

Example:
    <event-type-version>1.1</event-type-version>
-->
<!ELEMENT event-type-version (#PCDATA)>

<!--
The profile-spec-ref element identifies an profile specification that the library
classes depend. It contains an optional description about the reference, and the
name, vendor, and version of the referenced profile specification.

Used in: library
-->
<!ELEMENT profile-spec-ref (description?, profile-spec-name,
          profile-spec-vendor, profile-spec-version)>

<!--
The profile-spec-name element contains the name of a profile specification
component.

Used in: profile-spec-ref

Example:
    <profile-spec-name>AddressProfileSpec</profile-spec-name>
-->
<!ELEMENT profile-spec-name (#PCDATA)>

<!--
The profile-spec-vendor element contains the vendor of a profile specification
component.

Used in: profile-spec-ref

Example:
    <profile-spec-name>javax.slee</profile-spec-name>
-->
<!ELEMENT profile-spec-vendor (#PCDATA)>

<!--
The profile-spec-version element contains the version of a profile
specification component.

Used in: profile-spec-ref
```

```
Example:
    <profile-spec-version>1.0</profile-spec-version>
-->
<!ELEMENT profile-spec-version (#PCDATA)>

<!--
The resource-adaptor-type-ref element identifies an resource adaptor type that the
library classes depend. It contains the name, vendor,and version of the RA type.

Used in: library
-->
<!ELEMENT resource-adaptor-type-ref (resource-adaptor-type-name,
          resource-adaptor-type-vendor, resource-adaptor-type-version)>

<!--
The resource-adaptor-type-name element contains the name of a resource
adaptor type component referred by the library.

Used in: resource-adaptor-type-ref

Example:
    <resource-adaptor-type-name>JCC</resource-adaptor-type-name>
-->
<!ELEMENT resource-adaptor-type-name (#PCDATA)>

<!--
The resource-adaptor-type-vendor element contains the vendor of a resource
adaptor type component referred by the library.

Used in: resource-adaptor-type-ref

Example:
    <resource-adaptor-type-vendor>
        javax.csapi.cc.jcc
    </resource-adaptor-type-vendor>
-->
<!ELEMENT resource-adaptor-type-vendor (#PCDATA)>

<!--
The resource-adaptor-type-version element contains the version of a resource
adaptor type component referred by the library.

Used in: resource-adaptor-type-ref

Example:
    <resource-adaptor-type-version>1.1</resource-adaptor-type-version>
-->
<!ELEMENT resource-adaptor-type-version (#PCDATA)>

<!--
The sbb-ref element identifies an SBB that the library classes depend.
```

```
It contains the name, vendor,and version of the SBB.

Used in: library
-->
<!ELEMENT sbb-ref (sbb-name, sbb-vendor,
          sbb-version)>

<!--
The sbb-name element contains the name of a SBB component referred
 by the library.


Used in: sbb-ref

Example:
    <sbb-name>MySbb</sbb-name>
-->
<!ELEMENT sbb-name (#PCDATA)>

<!--
The sbb-vendor element contains the vendor of a SBB component referred
 by the library.

Used in: sbb-ref

Example:
    <sbb-vendor>My Company, Inc.</sbb-vendor>
-->
<!ELEMENT sbb-vendor (#PCDATA)>

<!--
The sbb-version element contains the version of a SBB component referred
 by the library.

Used in: sbb-ref

Example:
    <sbb-version>1.0</sbb-version>
-->
<!ELEMENT sbb-version (#PCDATA)>

<!--
The ID mechanism is to allow tools that produce additional deployment
information (ie. information beyond that contained by the standard SLEE
deployment descriptors) to store the non-standard information in a separate
file, and easily refer from those tools-specific files to the information in
the standard deployment descriptor.  The SLEE architecture does not allow the
tools to add the non-standard information into the SLEE-defined deployment
descriptors.
-->
<!ATTLIST library-jar id ID #IMPLIED>
```

```
<!ATTLIST description id ID #IMPLIED>
<!ATTLIST library id ID #IMPLIED>
<!ATTLIST library-name id ID #IMPLIED>
<!ATTLIST library-vendor id ID #IMPLIED>
<!ATTLIST library-version id ID #IMPLIED>
<!ATTLIST event-type-ref id ID #IMPLIED>
<!ATTLIST event-type-name id ID #IMPLIED>
<!ATTLIST event-type-vendor id ID #IMPLIED>
<!ATTLIST event-type-version id ID #IMPLIED>
<!ATTLIST library-ref id ID #IMPLIED>
<!ATTLIST profile-spec-ref id ID #IMPLIED>
<!ATTLIST profile-spec-name id ID #IMPLIED>
<!ATTLIST profile-spec-vendor id ID #IMPLIED>
<!ATTLIST profile-spec-version id ID #IMPLIED>
<!ATTLIST resource-adaptor-type-ref id ID #IMPLIED>
<!ATTLIST resource-adaptor-type-name id ID #IMPLIED>
<!ATTLIST resource-adaptor-type-vendor id ID #IMPLIED>
<!ATTLIST resource-adaptor-type-version id ID #IMPLIED>
<!ATTLIST sbb-ref id ID #IMPLIED>
<!ATTLIST sbb-name id ID #IMPLIED>
<!ATTLIST sbb-vendor id ID #IMPLIED>
<!ATTLIST sbb-version id ID #IMPLIED>
<!ATTLIST jar id ID #IMPLIED>
<!ATTLIST jar-name id ID #IMPLIED>
<!ATTLIST security-permissions id ID #IMPLIED>
<!ATTLIST security-permission-spec id ID #IMPLIED>
```

This full DTD is available at https://raw.githubusercontent.com/RestComm/jain-slee/master/api/descriptors/library/src/main/resources/slee-library-jar_1_1-ext.dtd

10.6.2. Extended Library Jar XML Descriptor Example

The following XML descriptor examples the definition of references to JAIN SLEE 1.1 Component types other than Library

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE library-jar PUBLIC
        "-//Sun Microsystems, Inc.//DTD JAIN SLEE Ext Library 1.1//EN"
        "https://raw.githubusercontent.com/RestComm/jain-
slee/master/api/descriptors/library/src/main/resources/slee-library-jar_1_1-ext.dtd">

<library-jar>
    <library>
        <library-name>extended-library-example</library-name>
        <library-vendor>com.redhat</library-vendor>
        <library-version>1.0</library-version>

        <event-type-ref>
            <event-type-name>ExampleX</event-type-name>
            <event-type-vendor>com.redhat</event-type-vendor>
            <event-type-version>1.0</event-type-version>
        </event-type-ref>

        <library-ref>
            <library-name>LibraryX</library-name>
            <library-vendor>com.redhat</library-vendor>
            <library-version>1.0</library-version>
        </library-ref>

        <profile-spec-ref>
            <profile-spec-name>ProfileX</profile-spec-name>
            <profile-spec-vendor>com.redhat</profile-spec-vendor>
            <profile-spec-version>1.0</profile-spec-version>
        </profile-spec-ref>

        <resource-adaptor-type-ref>
            <resource-adaptor-type-name>ResourceAdaptorTypeX</resource-adaptor-type-
name>
            <resource-adaptor-type-vendor>com.redhat</resource-adaptor-type-vendor>
            <resource-adaptor-type-version>1.0</resource-adaptor-type-version>
        </resource-adaptor-type-ref>

        <sbb-ref>
            <sbb-name>SbbX</sbb-name>
            <sbb-vendor>com.redhat</sbb-vendor>
            <sbb-version>1.0</sbb-version>
        </sbb-ref>

    </library>
</library-jar>
```

Note how the DOCTYPE element is set to the extended DTD instead of the standard one.

10.7. Preferred Packages Extension

This extension to JAIN SLEE 1.1 introduces the ability to override server provided classes from SBB/Lib components.

The SBB developer may define the list of packages to be overriden, by using an special environment entry in SBB descriptor. If SBB declares SLEE library dependencies, the list of preferred packages will be applied to the library as well.

```
<env-entry>
    <env-entry-name>org.restcomm.slee.preferred-package-list</env-entry-name>
    <env-entry-type>java.lang.String</env-entry-type>
    <env-entry-
value>org.hibernate,dom4j,javax.transaction,org.jboss.logging,javax.persistence,net.sf
.ehcache</env-entry-value>
</env-entry>
```

The "org.restcomm.slee.preferred-package-list" entry contains a comma separated list of base packages that will be used during class loading. Any class under those packages, will have the classloading order inverted, so local class definition to SBB will be taken before server classes.

Chapter 11. Advanced Topics

11.1. Class Loading

Each JAIN SLEE Component has its own classloader (`ComponentClassLoader`) in the package named `org.mobicents.slee.container.component.deployment.classloading`. This classloader sees the component classes contained in the component jar (`URLClassLoaderDomain`) by declaring it as the parent classloader, and adding the classes seen by each component it refers. It does not see classes from a component that it does not depend on.

JAIN SLEE defines a class loading domain with the s required in the JAIN SLEE 1.1 container (for example, JAIN SLEE, and). This domain (`JBoss Microcontainer ClassLoadingDomain`) imports all classes shared in the JBoss Application Server , and acts like the parent domain for all `URLClassLoaderDomain`s, which means that a class imported by a SLEE classloading domain will always be used first.

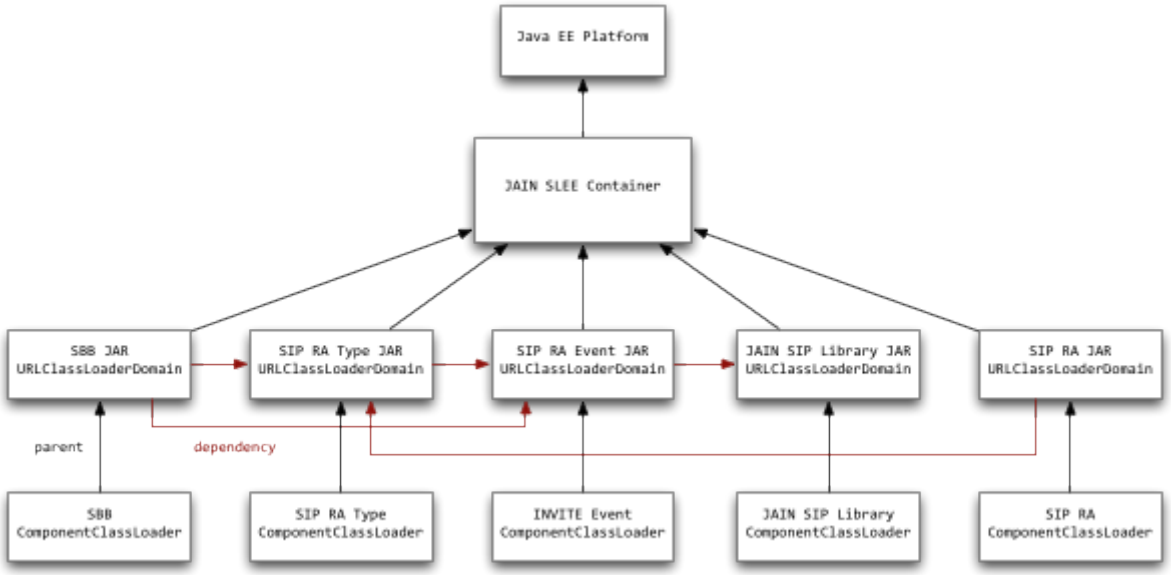

Figure 3. Classloading example in Restcomm JAIN SLEE

- The `SIP INVITE Event` component refers to the `JAIN SIP Library` in its XML descriptor, and its classloader domain depends on the classloader domain of the JAIN SIP Library.
- The `SIP RA Type` component refers to all Events in the SIP RA Event jar in its XML descriptor, and its classloader domain depends on the classloader domain of the SIP Event JAR and inherits its dependencies, including the JAIN SIP library classloading domain.
- The `SIP RA` component refers to the `SIP RA Type` component in its XML descriptor, and its classloader domain depends on the classloader domain of the SIP RA Type Component jar, and inherits its dependencies. This includes the SIP Event jar and JAIN SIP library classloading domains.
- The `SBB` component refers to the `SIP RA Type` component and `SIP INVITE Event` in its XML descriptor. Its classloader domain depends on the class loader domain of the SIP RA Type Component jar, and inherits its dependencies; the SIP Event jar and the JAIN SIP library classloading domains. It also depends on the classloader domain of the SIP Event jar.

JBoss Application Server does not see the classes of deployed JAIN SLEE components. This means that if it exports its classes for components that are complemented with Java EE components, the common classes must be deployed on JBoss Application Server , either directly or included in the Java EE component.

11.2. Congestion Control

JAIN SLEE can monitor the memory available in the JVM. In case it drops to a certain level (percentage), new events and/or activity startups are rejected, and at the same time a JAIN SLEE Alarm (which can send JMX notifications) is raised. This feature is called Congestion Control, and the container will turn it off automatically once another available memory level is reached.

If Congestion Control rejects an operation, a `javax.slee.SleeException` is thrown. This means that if the feature is to be used, the Resource Adaptors and Applications need to handle such use case, and behave properly.

The type of JAIN SLEE Alarm raised is `org.mobicents.slee.management.alarm.congestion`.

Congestion Control is turned off by default.

11.2.1. Congestion Control Configuration

The Congestion Control feature is configured through an XML file or through a JMX MBean. Changes applied through JMX are not persisted, and once the container is restarted the configuration will revert to the one in the XML file.

Congestion Control Persistent Configuration

Configuration is done through a XML descriptor for each Server Profiles. The XML file is named *jboss-beans.xml* and is located at *$JBOSS_HOME/server/profile_name/deploy/restcomm-slee/META-INF*, where `profile_name` is the server profile name.

The configuration is exposed a JBoss Microcontainer Bean:

```
<bean name="Mobicents.JAINSLEE.CongestionControlConfiguration"
    class=
"org.mobicents.slee.container.management.jmx.CongestionControlConfiguration">
    <annotation>@org.jboss.aop.microcontainer.aspects.jmx.JMX(name=
        "org.mobicents.slee:name=CongestionControlConfiguration",exposedInterface=

org.mobicents.slee.container.management.jmx.CongestionControlConfigurationMBean.class,
        registerDirectly=true)</annotation>
    <property name="periodBetweenChecks">0</property>
    <property name="minFreeMemoryToTurnOn">10</property>
    <property name="minFreeMemoryToTurnOff">20</property>
    <property name="refuseStartActivity">true</property>
    <property name="refuseFireEvent">false</property>
</bean>
```

Table 7. JAIN SLEE Congestion Control Bean Configuration

Property Name	Property Type	Description
periodBetweenChecks	int	The available memory level is checked periodically, this property defines the period, in seconds, between these checks, and if set to 0 turns off the Congestion Control feature.
minFreeMemoryToTurnOn	int	This property defines the minimum free memory percentage, which if reached turns ON the Congestion Control feature.
minFreeMemoryToTurnOff	int	This property defines the minimum free memory percentage, which if reached turns OFF the Congestion Control feature. This value should be considerably higher than minFreeMemoryToTurnOn, otherwise the feature may be turning on and off all the time.
refuseStartActivity	boolean	If true and the Congestion Control feature is ON, the container rejects activity startups, no matter it's a request from a Resource Adaptor or SBB.

Property Name	Property Type	Description
refuseFireEvent	boolean	If true and the Congestion Control feature is ON, the container rejects the firing of events, no matter it's a request from a Resource Adaptor or SBB.

Congestion Control JMX Configuration

Through JMX, the Congestion Control feature configuration can be changed with the container running. These configuration changes are not persisted.

The JMX MBean which can be used to change the Congestion Control configuration is named `org.mobicents.slee:name=CongestionControlConfiguration`, and provides getters and setters to change each property defined in the persistent configuration. The JMX Console can be used to use this MBean, see JMX Console.

11.3. JAIN SLEE 1.1 Profiles JPA Mapping

As mentioned in the containers configuration section, Restcomm JAIN SLEE uses JPA to store all JAIN SLEE 1.1 Profiles, and in mentioned section it was explained how to define which JPA / Hibernate data source. In this section more details are provided about how JAIN SLEE 1.1 Profiles are mapped to a JPA datasource schema.

11.3.1. Profile Specification JPA Tables And columns

For each Profile Specification, at least one Table is created, and is named `SLEE_PE_` concatenated with the Profile CMP interface simple name (`obtained as java.lang.Class.getSimpleName()`), then `_`, and finally the absolute value of the `hashCode()` method of the javax.slee.ComponentID of the Profile Specification.

This table has a primary key composed by the profile name and profile table name, and a column for each attribute of the Profile Specification CMP, except for those of array type. Those columns are named `C`, concatenated with the cmp attribute name.

For each Profile CMP attribute of `array` type, a join table is created, and is named `SLEE_PEAAV_` concatenated with the Profile CMP interface simple name (`obtained as java.lang.Class.getSimpleName()`), then `_`, then the absolute value of `hashCode()` method of the javax.slee.ComponentID of the Profile Specification, and finally the CMP attribute name. This table has a generated primary key column named `ID`, the foreign key, and two columns to store the CMP attribute value:

`SERIALIZABLE`
: Used to store the value if its type does not allow it to be converted to a String.

`STRING`
: Used when the CMP attribute type can be converted to a `java.lang.String`, for instance an `Integer`.

11.3.2. Profile Specification JPA Datasource

Unless configured manually, Restcomm JAIN SLEE uses the default datasource of &JEE.PLATFORM;. Please refer to its documentation to learn about it.

11.4. Testing the JAIN SLEE 1.1 TCK

To run the JAIN SLEE 1.1 TCK:

1. Checkout and build the container source code as explained in Installing Restcomm JAIN SLEE.
2. Install the TCK Resource Adaptor and Plugin:

```
cd tck/jain-slee-1.1
mvn install
```

3. Setup JAVA_OPTS environment variable to include -XX:MaxPermSize=128M -Djboss.defined.home=/Volumes/2/jboss-5.1.0.GA -Djava.security.manager=default -Djava.security.policy=file:///Volumes/2/workspace/restcomm-jainslee-2/tck/jain-slee-1.1/tck-security.policy, replacing the absolute paths with $JBOSS_HOME and the path of the Git checkout.
4. Unzip and run the JAIN SLEE 1.1 TCK distribution:

```
unzip testsuite.zip
cd testsuite
ant
```

No test should fail.

11.5. Setting JAIN SLEE Source Code Projects in Eclipse IDE

The JAIN SLEE Core, each RA, and each example, are worked out with separated Eclipse IDE Projects.

There are two alternatives to set up a specific project:

Procedure: Via Command Line

1. In the checked out directory of the project, and with Eclipse IDE closed, open a terminal.
2. Run the following:

```
]mvn restcomm:eclipse
mvn -Declipse.workspace=YOUR_RELATIVE_PATH_TO_ECLIPSE_WORKSPACE eclipse:add-maven-repo
```

3. Install M2Eclipse if you want to do maven builds within Eclipse.

Procedure: With Eclipse IDE

1. Install the M2Eclipse plugin and use "Import..." and subselect the "Maven Projects" feature.

Ensure the "Resolve Workspace projects" and "Separate projects for modules" in the "Advanced" options on the bottom of the window are turned off. If the project is large, such as the JAIN SLEE Core, M2Eclipse may be a considerable slower option, due to dynamic Maven2 Dependency Management.

Appendix A: Java Development Kit (): Installing, Configuring and Running

The [app]` Platform` is written in Java; therefore, before running any server, you must have a working Java Runtime Environment () or Java Development Kit () installed on your system. In addition, the JRE or JDK you are using to run [app] must be version 5 or higher [1: At this point in time, it is possible to run most servers, such as the JAIN SLEE, using a Java 6 JRE or JDK. Be aware, however, that presently the XML Document Management Server does not run on Java 6. We suggest checking the web site, forums or discussion pages if you need to inquire about the status of running the XML Document Management Server with Java 6.].

Should I Install the JRE or JDK?

Although you can run servers using the Java Runtime Environment, we assume that most users are developers interested in developing Java-based, [app]-driven solutions. Therefore, in this guide we take the tact of showing how to install the full Java Development Kit.

Should I Install the 32-Bit or the 64-Bit JDK, and Does It Matter?

Briefly stated: if you are running on a 64-Bit Linux or Windows platform, you should consider installing and running the 64-bit JDK over the 32-bit one. Here are some heuristics for determining whether you would rather run the 64-bit Java Virtual Machine (JVM) over its 32-bit cousin for your application:

- Wider datapath: the pipe between RAM and CPU is doubled, which improves the performance of memory-bound applications when using a 64-bit JVM.
- 64-bit memory addressing gives virtually unlimited (1 exabyte) heap allocation. However large heaps affect garbage collection.
- Applications that run with more than 1.5 GB of RAM (including free space for garbage collection optimization) should utilize the 64-bit JVM.
- Applications that run on a 32-bit JVM and do not require more than minimal heap sizes will gain nothing from a 64-bit JVM. Barring memory issues, 64-bit hardware with the same relative clock speed and architecture is not likely to run Java applications faster than their 32-bit cousin.

Note that the following instructions detail how to download and install the 32-bit JDK, although the steps are nearly identical for installing the 64-bit version.

Downloading

You can download the Sun JDK 5.0 (Java 2 Development Kit) from Sun's website: http://java.sun.com/javase/downloads/index_jdk5.jsp. Click on the Download link next to "JDK 5.0 Update <x>`" (where [replaceable]<x>` is the latest minor version release number). On the next page, select your language and platform (both architecture—whether 32- or 64-bit—and operating system), read and agree to the Java Development Kit 5.0 License Agreement, and proceed to the download page.

The Sun website will present two download alternatives to you: one is an RPM inside a self-extracting file (for example, *jdk-1_5_0_16-linux-i586-rpm.bin*), and the other is merely a self-extracting file (e.g. *jdk-1_5_0_16-linux-i586.bin*). If you are installing the JDK on Red Hat Enterprise

Linux, Fedora, or another RPM-based Linux system, we suggest that you download the self-extracting file containing the RPM package, which will set up and use the SysV service scripts in addition to installing the JDK. We also suggest installing the self-extracting RPM file if you will be running [app]``` in a production environment.

Installing

The following procedures detail how to install the Java Development Kit on both Linux and Windows.

Procedure: Installing the JDK on Linux

1. Regardless of which file you downloaded, you can install it on Linux by simply making sure the file is executable and then running it:

```
~]$ chmod +x "jdk-1_5_0_<minor_version>-linux-<architecture>-rpm.bin"
~]$ ./"jdk-1_5_0_<minor_version>-linux-<architecture>-rpm.bin"
```

You Installed Using the Non-RPM Installer, but Want the SysV Service Scripts

If you download the non-RPM self-extracting file (and installed it), and you are running on an RPM-based system, you can still set up the SysV service scripts by downloading and installing one of the `-compat` packages from the JPackage project. Remember to download the `-compat` package which corresponds correctly to the minor release number of the JDK you installed. The compat packages are available from link:ftp://jpackage.hmdc.harvard.edu/JPackage/1.7/generic/RPMS.non-free/.

You do not need to install a `-compat` package in addition to the JDK if you installed the self-extracting RPM file! The `-compat` package merely performs the same SysV service script set up that the RPM version of the JDK installer does.

Procedure: Installing the JDK on Windows

1. Using Explorer, simply double-click the downloaded self-extracting installer and follow the instructions to install the JDK.

Configuring

Configuring your system for the JDK consists in two tasks: setting the `JAVA_HOME` environment variable, and ensuring that the system is using the proper JDK (or JRE) using the `alternatives` command. Setting `JAVA_HOME` usually overrides the values for `java`, `javac` and `java_sdk_1.5.0` in `alternatives`, but we will set them all just to be safe and consistent.

Setting the `JAVA_HOME` Environment Variable on Generic Linux

After installing the JDK, you must ensure that the `JAVA_HOME` environment variable exists and points to the location of your JDK installation.

Setting `java`, `javac` and `java_sdk_1.5.0` Using the `alternatives` command

As the root user, call `/usr/sbin/alternatives` with the `--config java` option to select between

JDKs and JREs installed on your system:

Setting the `JAVA_HOME` Environment Variable on Windows

For information on how to set environment variables in Windows, refer to http://support.microsoft.com/kb/931715.

Testing

Finally, to make sure that you are using the correct JDK or Java version (5 or higher), and that the java executable is in your `PATH`, run the `java -version ` command in the terminal from your home directory:

```
~]$ java -version
java version "1.5.0_16"
Java(TM) 2 Runtime Environment, Standard Edition (build 1.5.0_16-b03)
Java HotSpot(TM) Client VM (build 1.5.0_16-b03, mixed mode, sharing)
```

Uninstalling

There is usually no reason (other than space concerns) to remove a particular JDK from your system, given that you can switch between JDKs and JREs easily using `alternatives`, and/or by setting `JAVA_HOME`.

Uninstalling the JDK on Linux

On RPM-based systems, you can uninstall the JDK using the `yum remove <jdk_rpm_name> ` command.

Uninstalling the JDK on Windows

On Windows systems, check the JDK entry in the `Start` menu for an uninstall command, or use `Add/Remove Programs`.

Appendix B: Setting the JBOSS_HOME Environment Variable

The [app]` Platform` () is built on top of the [app]. You do not need to set the JBOSS_HOME environment variable to run any of the [app]` Platform` servers *unless* JBOSS_HOME is *already* set.

The best way to know for sure whether JBOSS_HOME was set previously or not is to perform a simple check which may save you time and frustration.

Checking to See If JBOSS_HOME is Set on Unix

At the command line, echo $JBOSS_HOME to see if it is currently defined in your environment:

```
~]$ echo $JBOSS_HOME
```

The [app]` Platform` and most &THIS.PLATFORM; servers are built on top of the ([app]). When the [app]` Platform` or &THIS.PLATFORM; servers are built *from source*, then JBOSS_HOME *must* be set, because the &THIS.PLATFORM; files are installed into (or "over top of" if you prefer) a clean installation, and the build process assumes that the location pointed to by the JBOSS_HOME environment variable at the time of building is the [app] installation into which you want it to install the &THIS.PLATFORM; files.

This guide does not detail building the [app]` Platform` or any &THIS.PLATFORM; servers from source. It is nevertheless useful to understand the role played by JBoss AS and JBOSS_HOME in the &THIS.PLATFORM; ecosystem.

The immediately-following section considers whether you need to set JBOSS_HOME at all and, if so, when. The subsequent sections detail how to set JBOSS_HOME on Unix and Windows

Even if you fall into the category below of *not needing* to set JBOSS_HOME, you may want to for various reasons anyway. Also, even if you are instructed that you do *not need* to set JBOSS_HOME, it is good practice nonetheless to check and make sure that JBOSS_HOME actually *isn't* set or defined on your system for some reason. This can save you both time and frustration.

You *DO NOT NEED* to set JBOSS_HOME if...

- ...you have installed the [app]` Platform` binary distribution.
- ...you have installed a &THIS.PLATFORM;server binary distribution *which bundles [app]``*.

You *MUST* set JBOSS_HOME if...

- ...you are installing the [app]` Platform` or any of the &THIS.PLATFORM; servers *from source*.
- ...you are installing the [app]` Platform` binary distribution, or one of the &THIS.PLATFORM; server binary distributions, which *do not* bundle [app]``.

Naturally, if you installed the [app]` Platform` or one of the &THIS.PLATFORM; server binary

releases which *do not* bundle `, yet requires it to run, then you should install before setting [var]`JBOSS_HOME or proceeding with anything else.

Setting the JBOSS_HOME Environment Variable on Unix

The JBOSS_HOME environment variable must point to the directory which contains all of the files for the [app]` Platform` or individual &THIS.PLATFORM; server that you installed. As another hint, this topmost directory contains a *bin* subdirectory.

Setting JBOSS_HOME in your personal *~/.bashrc* startup script carries the advantage of retaining effect over reboots. Each time you log in, the environment variable is sure to be set for you, as a user. On Unix, it is possible to set JBOSS_HOME as a system-wide environment variable, by defining it in */etc/bashrc*, but this method is neither recommended nor detailed in these instructions.

Procedure: To Set JBOSS_HOME on Unix...

1. Open the *~/.bashrc* startup script, which is a hidden file in your home directory, in a text editor, and insert the following line on its own line while substituting for the actual install location on your system:

```
export JBOSS_HOME="/home/<username>/<path>/<to>/<install_directory>"
```

2. Save and close the *.bashrc* startup script.
3. You should source the *.bashrc* script to force your change to take effect, so that JBOSS_HOME becomes set for the current session [2: Note that any other terminals which were opened prior to your having altered .bashrc will need to source ~/.bashrc as well should they require access to JBOSS_HOME.].

```
~]$ source ~/.bashrc
```

4. Finally, ensure that JBOSS_HOME is set in the current session, and actually points to the correct location:

The command line usage below is based upon a binary installation of the [app]` Platform`. In this sample output, JBOSS_HOME has been set correctly to the topmost_directory of the installation. Note that if you are installing one of the standalone [app] servers (with JBoss AS bundled!), then JBOSS_HOME would point to the topmost_directory of your server installation.

```
~]$ echo $JBOSS_HOME
/home/silas/<path>/<to>/<install_directory>
```

Setting the JBOSS_HOME Environment Variable on Windows

The JBOSS_HOME environment variable must point to the directory which contains all of the files for the &THIS.PLATFORM;Platform or individual &THIS.PLATFORM;server that you installed. As another hint, this topmost directory contains a *bin* subdirectory.

For information on how to set environment variables in recent versions of Windows, refer to http://support.microsoft.com/kb/931715.

Appendix C: Fault Tolerant Clustering - A Concrete Example

In order to highlight the behavior of the Restcomm JAIN SLEE server when run in fault tolerant clustering mode, a concrete situation is described below.

Example Overview

The example below outlines a situation where two Restcomm JAIN SLEE nodes (`Node-1` and `Node-2`) are configured to run in fault tolerant mode. Both nodes have the same deployable unit containing a custom . A null activity is used by the s to communicate.

The service is a simple application that relays information received in INFO messages to another incoming dialog. This might for example be a rudimentary chat application. The flow outlines the clustered execution of the base case of this application, e.g. receive one message on one dialog and send it out on another dialog.

The example starts with the creation of two entities. One of the entities is the receiver, and the other one is the sender. The receiver will create an and register this with the naming facility. The sender retrieves the and sends a message to the receiver using an event. The example ends when the receiver sends the acquired message on its dialog as an INFO message.

Creating Sbb entities

The base case starts with a INVITE being received in `Node-1`. The event will be routed inside `Node-1` and trigger the creation of a new entity (`Sbb-1`). The entity in turn create and attach to a new null activity (`ACI-1`). Concurrently another INVITE is received in `Node-2`, which causes the JAIN SLEE container to create another entity (`Sbb-2`). Both s and s are fully clustered, so even though they are created in different physical nodes, they are logically inside the same container. In the image below, the cloud represents this logical relationship.

Note that balancer is not a requirement. It is present in this example to show basic message flow with balancer in front of deployed containers.

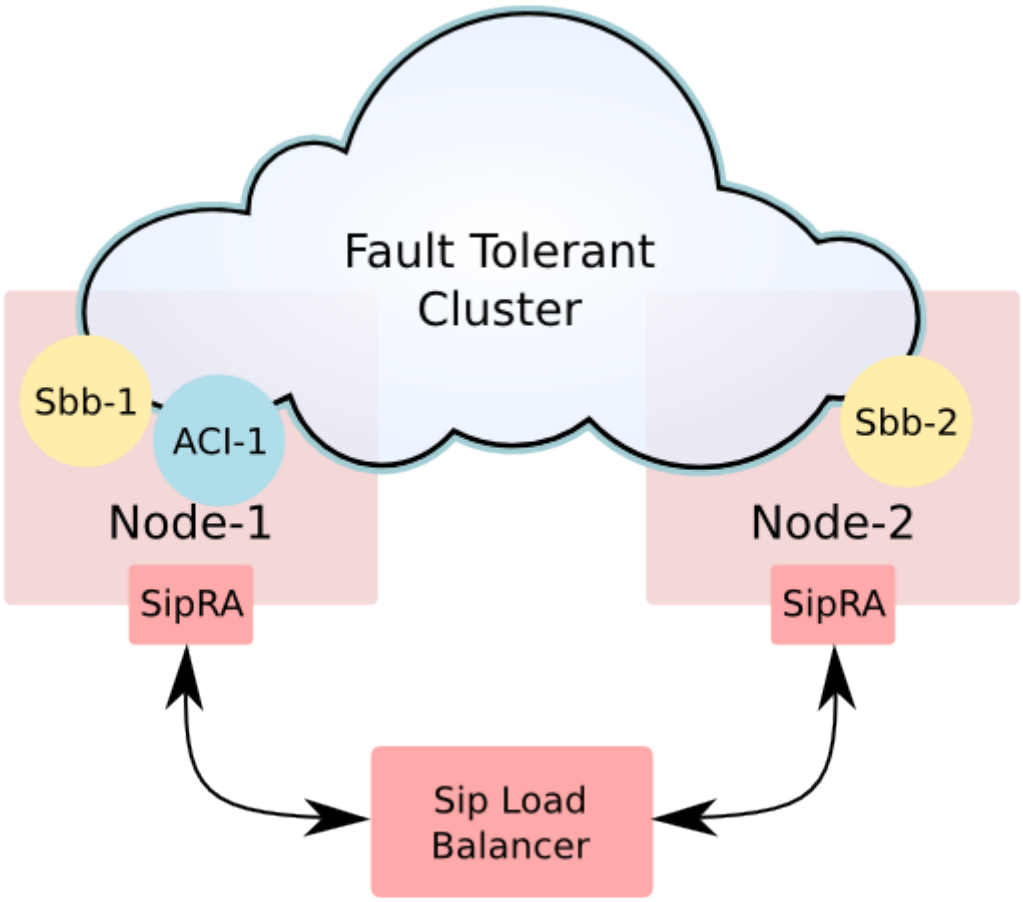

Figure 4. Fault tolerant cluster consisting of two nodes that both have one created on SBB entity for chat application. The SIP load balancer is multiplexing the SIP traffic.

Relaying the Message

After the s have been set up, a INFO is received by `Node-2` and relayed to `Sbb-2`. `Sbb-2` then looks up `ACI-1` from the ACI naming facility. `Node-2` retrieves the clustered state of `ACI-1` and de-serializes it for `Sbb-2`. Note that this means that `ACI-1` is currently being handled in `Node-2`.

When `ACI-1` has been retrieved from the cluster, `Sbb-2` fires a MessageEvent on `ACI-1`. Since events are not clustered, the event will be routed only on `Node-2`. The event will, however, be delivered to all attached s. This is achieved by `Node-2` retrieving the `Sbb-1` entity from the cluster and delivering the event to it.

The MessageEvent is parsed by `Sbb-1` and an outgoing INFO message is constructed with the appropriate payload. `Sbb-1` then forwards the INFO message to the SipRA. The incoming dialog that spawned `Sbb-1` is in `Node-1`, hence the SipRA will retrieve the activity object from the cluster and send the INFO message. The load balancer will then handle de-multiplexing. Note that retrieving the activity object from the clustered state only works because the SipRA is explicitly handling the replication of the SIP activity objects. Had another than the SipRA been used, a similar kind of clustering would have been needed to be implemented using the `FaultTolerantResourceAdaptor` interfaces.

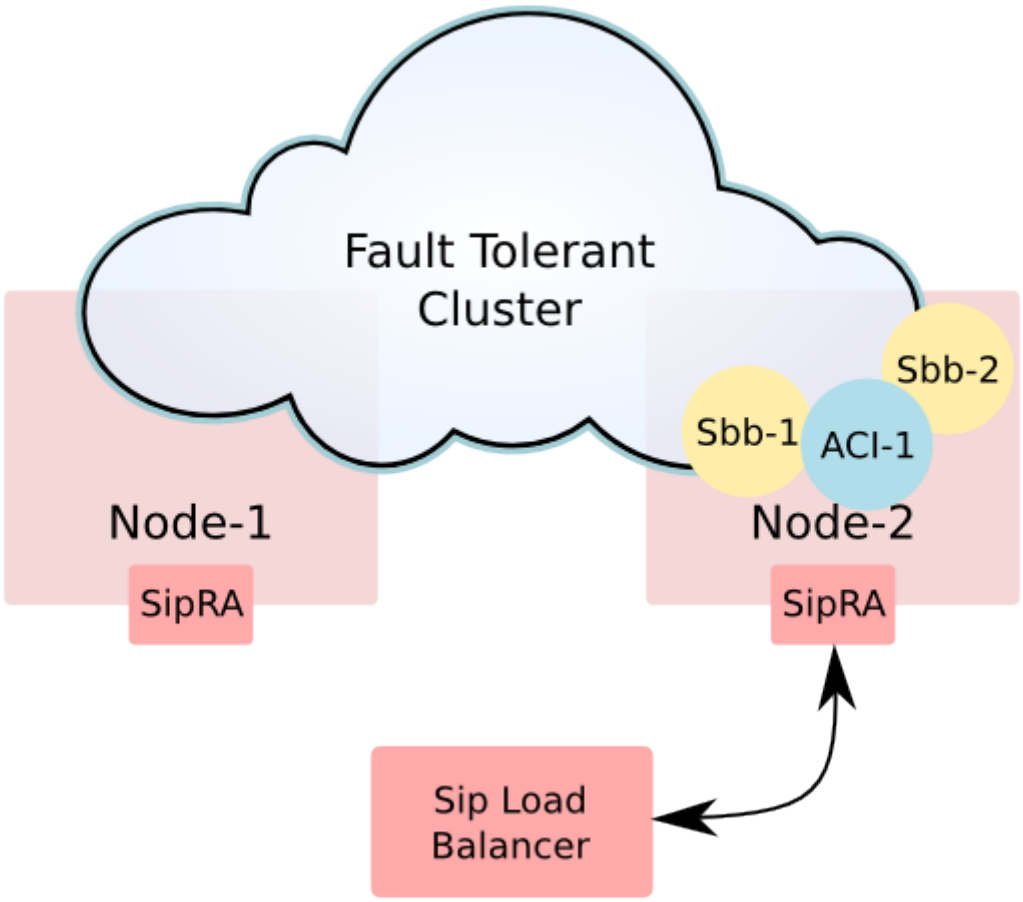

Figure 5. The situation when relaying the INFO message. Both SBB entities are running in the same node.

Appendix D: Revision History