TRITUX JAIN SLEE AMQP
Resource Adaptor User Guide

by Akrem BenMarzouk and Helmi BenAbdallah

1. Document Conventions

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

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

1.1. Typographic Conventions

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

`Mono-spaced Bold`

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

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

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

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

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

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

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

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

Proportional Bold

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

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

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

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

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

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

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

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

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

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

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

1.2. Pull-quote Conventions

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

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

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

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

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

import javax.naming.InitialContext;

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

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

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

}
```

1.3. Notes and Warnings

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

Note

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

Important

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

Warning

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

2. Provide feedback to the authors!

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

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

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

Introduction to TRITUX JAIN SLEE AMQP Resource Adaptor

Advanced Message Queuing Protocol (AMQP) is an open standard for passing business messages between applications or organizations. It connects systems, feeds business processes with the information they need and reliably transmits onward the instructions that achieve their goals.

For further details please look at *https://www.amqp.org/resources/specifications*

For spec details please look at *http://docs.oasis-open.org/amqp/core/v1.0/amqp-core-complete-v1.0.pdf*

This resource adaptor provides an AMQP API for JAIN SLEE applications.

Resource Adaptor Type

AMQP Resource Adaptor Type is defined by TRITUX team as part of effort to standardize RA Types.

2.1. Activities

An AMQP activity object represents a set of related events in an AMQP resource. This RA Type defines only one activity object:

AMQPWrapper
: All the events related to AMQP are fired on this activity. Class name is `org.mobicents.slee.resource.amqp.AMQPWrapper`

  New `AMQPWrapper` activity objects are created via specific AMQP Service interface. Check *Section 2.4, "Resource Adaptor Interface"* section for available services.

  Depending on amqp implementation AMQPActivity is used for sending/receiving messages and managing the server.

2.2. Events

Events represent's AMQP's common services are fired on `AMQPTemplate`

Important

For proper render of this table prefixes, for entries on some columns are ommited. For prefix values, for each column, please see list below:

Name
: org.springframework.amqp.core.Message.

Event Class
: org.mobicents.slee.resource.amqp.AMQPEvent

Version for all defined events is 1.0

Vendor for all defined events is org.mobicents

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

Table 2.1. Dialog events

Name	Event Class	Comments
AMQPEvent	AMQPEvent	Event representing sending and receiving messages.

2.3. Activity Context Interface Factory

The interface of the AMQP resource adaptor type specific Activity Context Interface Factory is defined as follows:

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


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

import org.springframework.amqp.core.AmqpTemplate;

public interface AMQPActivityContextInterfaceFactory {

   public static final ResourceAdaptorTypeID RESOURCE_ADAPTOR_TYPE_ID = new
 ResourceAdaptorTypeID(
           "AMQPResourceAdaptorType", "org.mobicents", "1.0");

 public ActivityContextInterface getActivityContextInterface(AmqpTemplate amqpTemplate)
   throws NullPointerException, UnrecognizedActivityException,
   FactoryException;
}
```

2.4. Resource Adaptor Interface

The AMQP Resource Adaptor SBB Interface provides SBBs with access to the AMQP objects required for creating a new amqp Message Request/Response. It is defined as follows:

```
package org.springframework.amqp.core;
```

```
import org.springframework.amqp.AmqpException;
import org.springframework.amqp.support.converter.MessageConverter;


public interface AMQPActivity {

 void send(Message message);

 void send(String routingKey, Message message);

 void send(String exchange, String routingKey, Message message);



 void convertAndSend(Object message);

 void convertAndSend(String routingKey, Object message);

 void convertAndSend(String exchange, String routingKey, Object message);

 void convertAndSend(Object message, MessagePostProcessor messagePostProcessor);

   void convertAndSend(String routingKey, Object message, MessagePostProcessor
messagePostProcessor);

    void convertAndSend(String exchange, String routingKey, Object message,
MessagePostProcessor messagePostProcessor);



 Message receive();

 Message receive(String queueName);



 Object receiveAndConvert();

 Object receiveAndConvert(String queueName);



 <R, S> boolean receiveAndReply(ReceiveAndReplyCallback<R, S> callback);
```

```
 <R, S> boolean receiveAndReply(String queueName, ReceiveAndReplyCallback<R, S> callback);

 <R, S> boolean receiveAndReply(ReceiveAndReplyCallback<R, S> callback, String replyExchange, String replyRoutingKey);

 <R, S> boolean receiveAndReply(String queueName, ReceiveAndReplyCallback<R, S> callback, String replyExchange, String replyRoutingKey);

 <R, S> boolean receiveAndReply(ReceiveAndReplyCallback<R, S> callback, ReplyToAddressCallback<S> replyToAddressCallback);

 <R, S> boolean receiveAndReply(String queueName, ReceiveAndReplyCallback<R, S> callback, ReplyToAddressCallback<S> replyToAddressCallback);



Message sendAndReceive(Message message);

Message sendAndReceive(String routingKey, Message message);

Message sendAndReceive(String exchange, String routingKey, Message message);



Object convertSendAndReceive(Object message);

Object convertSendAndReceive(String routingKey, Object message);

Object convertSendAndReceive(String exchange, String routingKey, Object message);

    Object convertSendAndReceive(Object message, MessagePostProcessor messagePostProcessor);

 Object convertSendAndReceive(String routingKey, Object message, MessagePostProcessor messagePostProcessor);

  Object convertSendAndReceive(String exchange, String routingKey, Object message, MessagePostProcessor messagePostProcessor);

}
```

send methods for messages.

void send(Message message);
this method Send a message to a default exchange with a default routing key.

void send(String routingKey, Message message);
this method Send a message to a default exchange with a specific routing key.

void send(String exchange, String routingKey, Message message);
this method Send a message to a specific exchange with a specific routing key.

send methods with conversion.

void convertAndSend(Object message);
this method Convert a Java object to an Amqp Message and send it to a default exchange with a default routing key.

void convertAndSend(String routingKey, Object message);
this method Convert a Java object to an Amqp Message and send it to a default exchange with a specific routing key.

void convertAndSend(String exchange, String routingKey, Object message);
this method Convert a Java object to an Amqp Message and send it to a specific exchange with a specific routing key.

void convertAndSend(Object message, MessagePostProcessor messagePostProcessor);
this method Convert a Java object to an Amqp Message and send it to a default exchange with a default routing key.

void convertAndSend(String routingKey, Object message, MessagePostProcessor messagePostProcessor);
this method Convert a Java object to an Amqp Message and send it to a default exchange with a specific routing key.

void convertAndSend(String exchange, String routingKey, Object message, MessagePostProcessor messagePostProcessor);
this method Convert a Java object to an Amqp Message and send it to a specific exchange with a specific routing key.

receive methods for messages.

Message receive();
Receive a message if there is one from a default queue. Returns immediately, possibly with a null value.

Message receive(String queueName);
Receive a message if there is one from a specific queue. Returns immediately, possibly with a null value.

receive methods with conversion.

Object receiveAndConvert();

Receive a message if there is one from a default queue and convert it to a Java object. Returns immediately, possibly with a null value.

Object receiveAndConvert(String queueName);

Receive a message if there is one from a specific queue and convert it to a Java object. Returns immediately, possibly with a null value.

receive and send methods for provided callback.

<R, S> boolean receiveAndReply(ReceiveAndReplyCallback<R, S> callback);

Receive a message if there is one from a default queue, invoke provided ReceiveAndReplyCallback and send reply message, if the callback returns one, to the replyTo org.springframework.amqp.core.Address from org.springframework.amqp.core.MessageProperties or to default exchange and default routingKey.

<R, S> boolean receiveAndReply(String queueName, ReceiveAndReplyCallback<R, S> callback);

Receive a message if there is one from provided queue, invoke provided ReceiveAndReplyCallback and send reply message, if the callback returns one, to the replyTo org.springframework.amqp.core.Address from org.springframework.amqp.core.MessageProperties or to default exchange and default routingKey.

<R, S> boolean receiveAndReply(ReceiveAndReplyCallback<R, S> callback, String replyExchange, String replyRoutingKey);

Receive a message if there is one from default queue, invoke provided ReceiveAndReplyCallback and send reply message, if the callback returns one, to the provided exchange and routingKey.

<R, S> boolean receiveAndReply(String queueName, ReceiveAndReplyCallback<R, S> callback, String replyExchange, String replyRoutingKey);

Receive a message if there is one from provided queue, invoke provided ReceiveAndReplyCallback and send reply message, if the callback returns one, to the provided exchange and routingKey.

<R, S> boolean receiveAndReply(ReceiveAndReplyCallback<R, S> callback, ReplyToAddressCallback<S> replyToAddressCallback);

Receive a message if there is one from a default queue, invoke provided ReceiveAndReplyCallback and send reply message, if the callback returns one, to the replyTo org.springframework.amqp.core.Address from result of ReplyToAddressCallback.

<R, S> boolean receiveAndReply(String queueName, ReceiveAndReplyCallback<R, S> callback, ReplyToAddressCallback<S> replyToAddressCallback);

Receive a message if there is one from provided queue, invoke provided ReceiveAndReplyCallback and send reply message, if the callback returns one, to the replyTo org.springframework.amqp.core.Address from result of ReplyToAddressCallback.

send and receive methods for messages.

Message sendAndReceive(Message message);
: Basic RPC pattern. Send a message to a default exchange with a default routing key and attempt to receive a response. Implementations will normally set the reply-to header to an exclusive queue and wait up for some time limited by a timeout.

Message sendAndReceive(String routingKey, Message message);
: Basic RPC pattern. Send a message to a default exchange with a specific routing key and attempt to receive a response. Implementations will normally set the reply-to header to an exclusive queue and wait up for some time limited by a timeout.

Message sendAndReceive(String exchange, String routingKey, Message message);
: Basic RPC pattern. Send a message to a specific exchange with a specific routing key and attempt to receive a response. Implementations will normally set the reply-to header to an exclusive queue and wait up for some time limited by a timeout.

send and receive methods with conversion.

Object convertSendAndReceive(Object message);
: Basic RPC pattern with conversion. Send a Java object converted to a message to a default exchange with a default routing key and attempt to receive a response, converting that to a Java object. Implementations will normally set the reply-to header to an exclusive queue and wait up for some time limited by a timeout.

Object convertSendAndReceive(String routingKey, Object message);
: Basic RPC pattern with conversion. Send a Java object converted to a message to a default exchange with a specific routing key and attempt to receive a response, converting that to a Java object. Implementations will normally set the reply-to header to an exclusive queue and wait up for some time limited by a timeout.

Object convertSendAndReceive(String exchange, String routingKey, Object message);
: Basic RPC pattern with conversion. Send a Java object converted to a message to a specific exchange with a specific routing key and attempt to receive a response, converting that to a Java object. Implementations will normally set the reply-to header to an exclusive queue and wait up for some time limited by a timeout.

Object convertSendAndReceive(Object message, MessagePostProcessor messagePostProcessor);
: Basic RPC pattern with conversion. Send a Java object converted to a message to a default exchange with a default routing key and attempt to receive a response, converting that to a Java object. Implementations will normally set the reply-to header to an exclusive queue and wait up for some time limited by a timeout.

Object convertSendAndReceive(String routingKey, Object message, MessagePostProcessor messagePostProcessor);
: Basic RPC pattern with conversion. Send a Java object converted to a message to a default exchange with a specific routing key and attempt to receive a response, converting that to a

Java object. Implementations will normally set the reply-to header to an exclusive queue and wait up for some time limited by a timeout.

Object convertSendAndReceive(String exchange, String routingKey, Object message, MessagePostProcessor messagePostProcessor);

Basic RPC pattern with conversion. Send a Java object converted to a message to a specific exchange with a specific routing key and attempt to receive a response, converting that to a Java object. Implementations will normally set the reply-to header to an exclusive queue and wait up for some time limited by a timeout.

Note

we use spring amqp stack services to support all implemented services.

2.5. Restrictions

The resource adaptor implementation should prevent SBBs from adding themselves as AMQP listeners, or changing the AMQP network configuration. Any attempt to do so should be rejected by throwing a SecurityException.

2.6. Sbb Code Examples

The following code shows complete flow of application receiving the AMQP Message request. Application sends back an AMQP mesasge and finally application closes the AMQP connection

```
public abstract class AmqpExampleSbb implements Sbb {

 private Tracer tracer;


 public void onAMQPEvent(org.mobicents.slee.resources.amqp.AMQPEvent event,
   ActivityContextInterface aci/*, EventContext eventContext*/) {


   // Get the amqpMessage
   Massage received = event.getAmqpMessage();

   tracer.info(  "AMQP Event received ");

   Message response = new Message("this is a reply".getBytes(), null);

     // Get the activity object from the ACI.
     AMQPActivity amqp = ()AMQPActivity)aci.createcreateAMQPActivity();
```

```
    //reply
    amqp.basicPublish(response);

    aci.detach(getSbbLocalObject());
    tracer.info("Activity detached." );

}



// TODO: Perform further operations if required in these methods.
public void setSbbContext(SbbContext context) {

 this.sbbContext = (SbbContextExt) context;
 tracer = context.getTracer(this.getClass().getSimpleName());

}
public void unsetSbbContext() { this.sbbContext = null; }

// TODO: Implement the lifecycle methods if required
public void sbbCreate() throws javax.slee.CreateException {}
public void sbbPostCreate() throws javax.slee.CreateException {}
public void sbbActivate() {}
public void sbbPassivate() {}
public void sbbRemove() {}
public void sbbLoad() {}
public void sbbStore() {}
 public void sbbExceptionThrown(Exception exception, Object event, ActivityContextInterface
activity) {}
public void sbbRolledBack(RolledBackContext context) {}



/**
 * Convenience method to retrieve the SbbContext object stored in setSbbContext.
 *
 * TODO: If your SBB doesn't require the SbbContext object you may remove this
 * method, the sbbContext variable and the variable assignment in setSbbContext().
 *
 * @return this SBB's SbbContext object
 */

protected SbbContextExt getSbbContext() {
```

```
  return sbbContext;
 }

 private SbbContextExt sbbContext; // This SBB's SbbContext

}
```

Resource Adaptor Implementation

The RA implementation uses the Spring AMQP stack, this stack uses the rabbit-mq implementation. Other implementation can be added depending on spring-amqp project

3.1. Configuration

The Resource Adaptor supports configuration only at Resource Adaptor Entity creation time. It supports following properties:

Table 3.1. Resource Adaptor's Configuration Properties - amqp-default-ra.properties

Property Name	Description	Property Type	Default Value
amqpHost	Amqp server Host	java.lang.String	localhost
amqpPort	Amqp server Port	java.lang.Integer	5672
amqpQueueName	the name of the queue	java.lang.String	myQueue
amqpExchangeName	the name of the exchange	java.lang.String	myExchange

Important

JAIN SLEE 1.1 Specification requires values set for properties without a default value, which means the configuration for those properties are mandatory, otherwise the Resource Adaptor Entity creation will fail!

3.2. Default Resource Adaptor Entities

There is a single Resource Adaptor Entity created when deploying the Resource Adaptor, named `AMQPRA`. The `AMQPRA` entity uses the default Resource Adaptor configuration, specified in *Section 3.1, "Configuration"*.

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

```
<resource-adaptor-type-binding>
  <resource-adaptor-type-ref>
    <resource-adaptor-type-name>
      AMQPResourceAdaptorType
```

```
            </resource-adaptor-type-name>
            <resource-adaptor-type-vendor>
               org.mobicents
            </resource-adaptor-type-vendor>
            <resource-adaptor-type-version>
               1.0
            </resource-adaptor-type-version>
         </resource-adaptor-type-ref>
         <activity-context-interface-factory-name>
            slee/resources/amqp/1.0/acifactory
         </activity-context-interface-factory-name>
         <resource-adaptor-entity-binding>
            <resource-adaptor-object-name>
               slee/resources/amqp/1.0/provider
            </resource-adaptor-object-name>
            <resource-adaptor-entity-link>
               AMQPPRA
            </resource-adaptor-entity-link>
         </resource-adaptor-entity-binding>
      </resource-adaptor-type-binding>
```

3.3. Traces and Alarms

3.3.1. Tracers

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

3.3.2. Alarms

No alarms are set by this Resource Adaptor.

Setup

4.1. Pre-Install Requirements and Prerequisites

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

4.1.1. Hardware Requirements

The Resource Adaptor hardware's main concern is RAM memory and Java Heap size, the more the better.

Of course, memory is only needed to store the Resource Adaptor state, yet no particular CPU is a real requirement to use the RA.

4.1.2. Software Prerequisites

The RA requires TRITUX JAIN SLEE properly set.

4.2. TRITUX JAIN SLEE AMQP Resource Adaptor Source Code

4.2.1. Release Source Code Building

1. **Downloading the source code**

Important

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

Use SVN to checkout a specific release source, the base URL is http://mobicents.googlecode.com/svn/tags/servers/jain-slee/2.x.y/resources/AMQP, then add the specific release version, lets consider 1.0.

```
[usr]$ svn co http://mobicents.googlecode.com/svn/tags/servers/jain-slee/2.x.y/resources/AMQP/1.0 slee-ra-AMQP-1.0
```

2. **Building the source code**

Important

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

Use Maven to build the deployable unit binary.

```
[usr]$ cd slee-ra-AMQP-1.0
[usr]$ mvn install
```

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

4.2.2. Development Trunk Source Building

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

4.3. Installing TRITUX JAIN SLEE AMQP Resource Adaptor

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

```
[usr]$ ant
```

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

4.4. Uninstalling TRITUX JAIN SLEE AMQP Resource Adaptor

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

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

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

Appendix A. Revision History

Revision History		
Revision 1.0	Mon Mar 02 2015	AkremBenmarzouk, HelmiBenabdallah

Creation of the TRITUX JAIN SLEE AMQP RA User Guide.

Index

F