JBoss Communications JAIN SLEE XCAP Client Resource Adaptor User Guide

by Eduardo Martins

Preface v
1. Document Conventions v
1.1. Typographic Conventions v
1.2. Pull-quote Conventions vii
1.3. Notes and Warnings vii
2. Provide feedback to the authors! viii
1. Introduction to JBoss Communications JAIN SLEE XCAP Client Resource Adaptor 1
2. Resource Adaptor Type 3
2.1. Activities 3
2.2. Events 6
2.3. Activity Context Interface Factory 6
2.4. Resource Adaptor Interface 7
2.5. Restrictions 10
2.6. Sbb Code Examples 10
2.6.1. Synchronous Operations 11
2.6.2. Asynchronous Operations 12
3. Resource Adaptor Implementation 15
3.1. Configuration 15
3.2. Default Resource Adaptor Entities 15
3.3. Traces and Alarms 16
3.3.1. Tracers 16
3.3.2. Alarms 16
4. Setup 17
4.1. Pre-Install Requirements and Prerequisites 17
4.1.1. Hardware Requirements 17
4.1.2. Software Prerequisites 17
4.2. JBoss Communications JAIN SLEE XCAP Client Resource Adaptor Source Code 17
4.2.1. Release Source Code Building 17
4.2.2. Development Trunk Source Building 18
4.3. Installing JBoss Communications JAIN SLEE XCAP Client Resource Adaptor 18
4.4. Uninstalling JBoss Communications JAIN SLEE XCAP Client Resource Adaptor 18
5. Clustering 19
A. Revision History 21
Index 23

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://bugzilla.redhat.com/bugzilla/], against the product **JBoss Communications JAIN SLEE XCAP Client Resource Adaptor**, or contact the authors.

When submitting a bug report, be sure to mention the manual's identifier: JAIN_SLEE_XCAPClient_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 JBoss Communications JAIN SLEE XCAP Client Resource Adaptor

The XCAP Client Resource Adaptor adapts a XCAP Client API into JAIN SLEE domain. The RA provides means to send XCAP requests to a XCAP Server, such as the JBoss Communications XDM Server. There can be two different types of usage with this component, sending requests and receiving the Response synchronously, blocking the requester thread until the Response is retrieved, and sending requests and receiving the Response asynchronously, which is preferred since it matches the JAIN SLEE application model.

Resource Adaptor Type

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

The name of the RA Type is `XCAPClientResourceAdaptorType`, its vendor is `org.mobicents` and its version is `2.0`.

The RA Type uses its own XCAP Client API, with an implementation built on top of `Apache HTTP Client 4.x`, for further information about the Apache API refer to its website at *http://hc.apache.org/httpcomponents-client/index.html*.

2.1. Activities

The single activity object for XCAP Client Resource Adaptor is the `org.mobicents.slee.resource.xcapclient.AsyncActivity` interface. Through the activity an SBB can send multiple XCAP requests, and receive the related responses asynchronously. Due to the nature of SLEE activities, this RA activity acts like a queue of requests, allowing the processing of their responses - the events- in a serialized way

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

The `AsyncActivity` interface is defined as follows:

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

import java.net.URI;
import org.mobicents.xcap.client.auth.Credentials;
import org.mobicents.xcap.client.header.Header;

public interface AsyncActivity {

    public void get(URI uri, Header[] additionalRequestHeaders,
            Credentials credentials);

    public void put(URI uri, String mimetype, String content,
            Header[] additionalRequestHeaders, Credentials credentials);

    public void put(URI uri, String mimetype, byte[] content,
            Header[] additionalRequestHeaders, Credentials credentials);

    public void putIfMatch(URI uri, String eTag, String mimetype,
```

```
            String content, Header[] additionalRequestHeaders,
            Credentials credentials);

    public void putIfMatch(URI uri, String eTag, String mimetype,
            byte[] content, Header[] additionalRequestHeaders,
            Credentials credentials);

    public void putIfNoneMatch(URI uri, String eTag, String mimetype,
            String content, Header[] additionalRequestHeaders,
            Credentials credentials);

    public void putIfNoneMatch(URI uri, String eTag, String mimetype,
            byte[] content, Header[] additionalRequestHeaders,
            Credentials credentials);

    public void delete(URI uri, Header[] additionalRequestHeaders,
            Credentials credentials);

    public void deleteIfMatch(URI uri, String eTag,
            Header[] additionalRequestHeaders, Credentials credentials);

    public void deleteIfNoneMatch(URI uri, String eTag,
            Header[] additionalRequestHeaders, Credentials credentials);

    public void endActivity();

}
```

The `get(URI, Header[], Credentials)` method:
: Retrieves the XCAP resource specified by the URI parameter. Additional HTTP headers, to be added in the XCAP request, and authentication credentials can be specified too.

The `put(URI, String, String, Header[], Credentials)` method:
: Puts the provided XML content, in `String` format, in the XCAP resource specified by the URI parameter. The request mimetype needs to be provided, according to the content type to be put. Additional HTTP headers, to be added in the XCAP request, and authentication credentials can be specified too.

The `put(URI, String, byte[], Header[], Credentials)` method:
: Puts the provided XML content, in `byte[]` format, in the XCAP resource specified by the URI parameter. The request mimetype needs to be provided, according to the content type to be put. Additional HTTP headers, to be added in the XCAP request, and authentication credentials can be specified too.

The `putIfMatch(URI, String, String, String, Header[], Credentials)` method:
Conditional put of the provided XML content, in `String` format, in the XCAP resource specified by the URI parameter. The request only succeeds if the XCAP resource entity tag matches the provided `eTag` parameter. The request mimetype needs to be provided, according to the content type to be put. Additional HTTP headers, to be added in the XCAP request, and authentication credentials can be specified too.

The `putIfMatch(URI, String, String, byte[], Header[], Credentials)` method:
Conditional put of the provided XML content, in `byte[]` format, in the XCAP resource specified by the URI parameter. The request only succeeds if the XCAP resource entity tag matches the provided `eTag` parameter. The request mimetype needs to be provided, according to the content type to be put. Additional HTTP headers, to be added in the XCAP request, and authentication credentials can be specified too.

The `putIfNoneMatch(URI, String, String, String, Header[], Credentials)` method:
Conditional put of the provided XML content, in `String` format, in the XCAP resource specified by the URI parameter. The request only succeeds if the XCAP resource entity tag doesn't match the provided `eTag` parameter. The request mimetype needs to be provided, according to the content type to be put. Additional HTTP headers, to be added in the XCAP request, and authentication credentials can be specified too.

The `putIfNoneMatch(URI, String, String, byte[], Header[], Credentials)` method:
Conditional put of the provided XML content, in `byte[]` format, in the XCAP resource specified by the URI parameter. The request only succeeds if the XCAP resource entity tag doesn't match the provided `eTag` parameter. The request mimetype needs to be provided, according to the content type to be put. Additional HTTP headers, to be added in the XCAP request, and authentication credentials can be specified too.

The `delete(URI, Header[], Credentials)` method:
Deletes the XCAP resource specified by the URI parameter. Additional HTTP headers, to be added in the XCAP request, and authentication credentials can be specified too.

The `deleteIfMatch(URI, String, Header[], Credentials)` method:
Conditional delete of the XCAP resource specified by the URI parameter. The request only succeeds if the XCAP resource entity tag matches the provided `eTag` parameter. Additional HTTP headers, to be added in the XCAP request, and authentication credentials can be specified too.

The `deleteIfNoneMatch(URI, String, Header[], Credentials)` method:
Conditional delete of the XCAP resource specified by the URI parameter. The request only succeeds if the XCAP resource entity tag doesn't match the provided `eTag` parameter. Additional HTTP headers, to be added in the XCAP request, and authentication credentials can be specified too.

The `endActivity()` method:
Ends the activity and its related Activity Context.

2.2. Events

There are three events fired by XCAP Client Resource Adaptor, which represents a response to each type of request, received in a specific `AsyncActivity` instance.

Table 2.1. Events fired on the AsyncActivity

Name	Vendor	Version	Event Class	Description
GetResponseEvent	org.mobicents	2.0	org.mobicents.slee.resource.xcapclient.ResponseEvent	A response to a XCAP GET request.
PutResponseEvent	org.mobicents	2.0	org.mobicents.slee.resource.xcapclient.ResponseEvent	A response to a XCAP PUT request.
DeleteResponseEvent	org.mobicents	2.0	org.mobicents.slee.resource.xcapclient.ResponseEvent	A response to a XCAP DELETE request.

Important

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

2.3. Activity Context Interface Factory

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

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

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

public interface XCAPClientActivityContextInterfaceFactory {
```

```
    public ActivityContextInterface getActivityContextInterface(
            AsyncActivity activity) throws NullPointerException,
            UnrecognizedActivityException, FactoryException;

}
```

2.4. Resource Adaptor Interface

The XCAP Client Resource Adaptor interface, of type `org.mobicents.slee.resource.xcapclient.XCAPClientResourceAdaptorSbbInterface`, which an SBB uses to create new `AsyncActivity` instances or send synchronous requests, its interface is defined as follows:

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

import javax.slee.resource.ActivityAlreadyExistsException;
import javax.slee.resource.StartActivityException;

import org.mobicents.xcap.client.XcapClient;

public interface XCAPClientResourceAdaptorSbbInterface extends XcapClient {

    public AsyncActivity createActivity()
            throws ActivityAlreadyExistsException, StartActivityException;

}
```

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

The XCAP Client Resource Adaptor interface extends type `org.mobicents.xcap.client.XcapClient`, its interface is defined as follows:

```
package org.mobicents.xcap.client;
```

```
import java.io.IOException;
import java.net.URI;

import org.mobicents.xcap.client.auth.Credentials;
import org.mobicents.xcap.client.header.Header;

public interface XcapClient {

    public void setAuthenticationCredentials(Credentials credentials);

    public void unsetAuthenticationCredentials();

    public void shutdown();

    public XcapResponse get(URI uri, Header[] additionalRequestHeaders,
            Credentials credentials) throws IOException;

    public XcapResponse put(URI uri, String mimetype, String content,
            Header[] additionalRequestHeaders, Credentials credentials)
            throws IOException;

    public XcapResponse put(URI uri, String mimetype, byte[] content,
            Header[] additionalRequestHeaders, Credentials credentials)
            throws IOException;

    public XcapResponse putIfMatch(URI uri, String eTag, String mimetype,
            String content, Header[] additionalRequestHeaders,
            Credentials credentials) throws IOException;

    public XcapResponse putIfMatch(URI uri, String eTag, String mimetype,
            byte[] content, Header[] additionalRequestHeaders,
            Credentials credentials) throws IOException;

    public XcapResponse putIfNoneMatch(URI uri, String eTag, String mimetype,
            String content, Header[] additionalRequestHeaders,
            Credentials credentials) throws IOException;

    public XcapResponse putIfNoneMatch(URI uri, String eTag, String mimetype,
            byte[] content, Header[] additionalRequestHeaders,
            Credentials credentials) throws IOException;

    public XcapResponse delete(URI uri, Header[] additionalRequestHeaders,
            Credentials credentials) throws IOException;
```

```
    public XcapResponse deleteIfMatch(URI uri, String eTag,
            Header[] additionalRequestHeaders, Credentials credentials)
            throws IOException;

    public XcapResponse deleteIfNoneMatch(URI uri, String eTag,
            Header[] additionalRequestHeaders, Credentials credentials)
            throws IOException;

}
```

The `setAuthenticationCredentials(Credentials)` method:
Sets default authentication credentials to be used on XCAP requests, when those do not provide specific authentication credentials.

The `unsetAuthenticationCredentials()` method:
Unsets default authentication credentials.

The `shutdown()` method:
Unsupported operation.

The `get(URI, Header[], Credentials)` method:
Retrieves the XCAP resource specified by the URI parameter. Additional HTTP headers, to be added in the XCAP request, and authentication credentials can be specified too.

The `get(URI, Header[], Credentials)` method:
Retrieves the XCAP resource specified by the URI parameter. Additional HTTP headers, to be added in the XCAP request, and authentication credentials can be specified too.

The `put(URI, String, String, Header[], Credentials)` method:
Puts the provided XML content, in `String` format, in the XCAP resource specified by the URI parameter. The request mimetype needs to be provided, according to the content type to be put. Additional HTTP headers, to be added in the XCAP request, and authentication credentials can be specified too.

The `put(URI, String, byte[], Header[], Credentials)` method:
Puts the provided XML content, in `byte[]` format, in the XCAP resource specified by the URI parameter. The request mimetype needs to be provided, according to the content type to be put. Additional HTTP headers, to be added in the XCAP request, and authentication credentials can be specified too.

The `putIfMatch(URI, String, String, String, Header[], Credentials)` method:
Conditional put of the provided XML content, in `String` format, in the XCAP resource specified by the URI parameter. The request only succeeds if the XCAP resource entity tag matches the provided `eTag` parameter. The request mimetype needs to be provided, according to the content type to be put. Additional HTTP headers, to be added in the XCAP request, and authentication credentials can be specified too.

The `putIfMatch(URI, String, String, byte[], Header[], Credentials)` method:
Conditional put of the provided XML content, in `byte[]` format, in the XCAP resource specified by the URI parameter. The request only succeeds if the XCAP resource entity tag matches the provided `eTag` parameter. The request mimetype needs to be provided, according to the content type to be put. Additional HTTP headers, to be added in the XCAP request, and authentication credentials can be specified too.

The `putIfNoneMatch(URI, String, String, String, Header[], Credentials)` method:
Conditional put of the provided XML content, in `String` format, in the XCAP resource specified by the URI parameter. The request only succeeds if the XCAP resource entity tag doesn't match the provided `eTag` parameter. The request mimetype needs to be provided, according to the content type to be put. Additional HTTP headers, to be added in the XCAP request, and authentication credentials can be specified too.

The `putIfNoneMatch(URI, String, String, byte[], Header[], Credentials)` method:
Conditional put of the provided XML content, in `byte[]` format, in the XCAP resource specified by the URI parameter. The request only succeeds if the XCAP resource entity tag doesn't match the provided `eTag` parameter. The request mimetype needs to be provided, according to the content type to be put. Additional HTTP headers, to be added in the XCAP request, and authentication credentials can be specified too.

The `delete(URI, Header[], Credentials)` method:
Deletes the XCAP resource specified by the URI parameter. Additional HTTP headers, to be added in the XCAP request, and authentication credentials can be specified too.

The `deleteIfMatch(URI, String, Header[], Credentials)` method:
Conditional delete of the XCAP resource specified by the URI parameter. The request only succeeds if the XCAP resource entity tag matches the provided `eTag` parameter. Additional HTTP headers, to be added in the XCAP request, and authentication credentials can be specified too.

The `deleteIfNoneMatch(URI, String, Header[], Credentials)` method:
Conditional delete of the XCAP resource specified by the URI parameter. The request only succeeds if the XCAP resource entity tag doesn't match the provided `eTag` parameter. Additional HTTP headers, to be added in the XCAP request, and authentication credentials can be specified too.

2.5. Restrictions

The `shutdown()` method exposed by the XCAP Client Resource Adaptor SBB Interface underlying XcapClient, throws a `UnsupportedOperationException` if invoked.

2.6. Sbb Code Examples

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

2.6.1. Synchronous Operations

The following code examples the usage of the RA's SBB Interface to send synchronous XCAP requests:

```
        // create auth credentials
        Credentials credentials = ra.getCredentialsFactory().getHttpDigestCredentials(username,password);

        // create doc uri
StringdocumentSelector=DocumentSelectorBuilder.getUserDocumentSelectorBuilder("resource-lists", userName, documentName).toPercentEncodedString();
        UriBuilder uriBuilder = new UriBuilder()
            .setSchemeAndAuthority("http://127.0.0.1:8080")
            .setXcapRoot("/mobicents")
            .setDocumentSelector(documentSelector);
        URI documentURI = uriBuilder.toURI();

        // the doc to put
        String initialDocument =
            "<?xml version=\"1.0\" encoding=\"UTF-8\"?>" +
            "<resource-lists xmlns=\"urn:ietf:params:xml:ns:resource-lists\">" +
                "<list name=\"friends\"/>" +
            "</resource-lists>";

        // put the document and get sync response
                    XcapResponse response = ra.put(documentURI,"application/resource-lists+xml",initialDocument,null,credentials);

        // check put response
        if (response != null) {
            if(response.getCode() == 200 || response.getCode() == 201) {
                log.info("document created in xcap server...");
            } else {
                log.severe("bad response from xcap server: "+response.toString());
            }
        } else {
            log.severe("unable to create document in xcap server...");
        }

        // let's create an uri selecting an element
        // create uri
        String elementSelector = new ElementSelectorBuilder()
```

```
            .appendStepByName("resource-lists")
            .appendStepByAttr("list","name","friends")
            .appendStepByAttr("entry","uri","sip:alice@example.com")
            .toPercentEncodedString();
        URI elementURI = uriBuilder.setElementSelector(elementSelector).toURI();

        // put an element and get sync response
                            String element = "<entry uri=\"sip:alice@example.com\" xmlns=
\"urn:ietf:params:xml:ns:resource-lists\"/>";
        response = ra.put(elementURI,ElementResource.MIMETYPE,element,null,credentials);

        // check put response
        if (response != null) {
            if(response.getCode() == 201) {
                log.info("element created in xcap server...");
            } else {
                log.severe("bad response from xcap server: "+response.toString());
            }
        } else {
            log.severe("unable to create element in xcap server...");
        }

        // get the document and check content is ok
        response = ra.get(documentURI,null,credentials);

        // check get response
        if (response != null) {
            if(response.getCode() == 200) {
                log.info("document successfully retreived in xcap server.");
                // delete the document
                ra.delete(documentURI,null,credentials);
            } else {
                log.severe("bad response from xcap server: "+response.toString());
            }
        } else {
            log.severe("unable to retreive document in xcap server...");
        }
```

2.6.2. Asynchronous Operations

The following code examples the usage of the AsyncActivity to send async XCAP requests, the optimal way to use the RA, since it doesn't block the SLEE container event routing threads:

```
        // now we will use JAXB marshalling and unmarshalling too

        // let's create a list containing  someone
        ObjectFactory of = new ObjectFactory();
        ListType listType = of.createListType();
        listType.setName("enemies");
        EntryType entry = of.createEntryType();
        entry.setUri("sip:winniethepooh@disney.com");
        listType.getListOrExternalOrEntry().add(entry);

        // create the uri selecting the new element
        String elementSelector = new ElementSelectorBuilder()
            .appendStepByName("resource-lists")
            .appendStepByAttr("list","name","enemies")
            .toPercentEncodedString();
StringdocumentSelector=DocumentSelectorBuilder.getUserDocumentSelectorBuilder("resource-
lists", userName, documentName).toPercentEncodedString();
        UriBuilder uriBuilder = new UriBuilder()
        .setSchemeAndAuthority("http://127.0.0.1:8080")
        .setXcapRoot("/mobicents")
        .setDocumentSelector(documentSelector)
        .setElementSelector(elementSelector);
        URI uri = uriBuilder.toURI();

        // marshall content to byte array
        ByteArrayOutputStream baos = new ByteArrayOutputStream();
        jAXBContext.createMarshaller().marshal(listType, baos);

        // lets put the element using the sync interface
        XcapResponse response = ra.put(uri,ElementResource.MIMETYPE,
            baos.toByteArray(),null,credentials);
        // check put response
        if (response != null) {
            if(response.getCode() == 201) {
                log.info("list element created in xcap server...");
            } else {
                log.severe("bad response from xcap server: "+response.toString());
            }
        } else {
            log.severe("unable to create list element in xcap server...");
        }
```

```
    // now lets get it using the async interface

    // get a async request activity from the xcap client ra
    AsyncActivity activity = ra.createActivity();

    // attach this sbb entity to the activity's related aci
    ActivityContextInterface aci = acif.getActivityContextInterface(activity);
    aci.attach(sbbContext.getSbbLocalObject());

    // send request
    activity.get(uri,null,credentials);
```

And the next code snippet examples the handling of the ResponseEvent, and the ending of the activity instance:

```
public void onGetResponseEvent(ResponseEvent event, ActivityContextInterface aci) {

    // check put response
    XcapResponse response = event.getResponse();
    if (response != null) {
        if(response.getCode() == 200) {
            log.info("list element retreived from xcap server...");
        } else {
            log.severe("bad response from xcap server: "+response.toString());
        }
    } else {
        log.severe("unable to create list element in xcap server...");
    }

    // end the activity
    AsyncActivity activity = (AsyncActivity)aci.getActivity();
    if (activity != null) {
        activity.endActivity();
    }

}
```

Resource Adaptor Implementation

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

The name of the RA is `XCAPClientResourceAdaptor`, its vendor is `org.mobicents` and its version is `2.0`.

3.1. Configuration

The Resource Adaptor implementation does not have any configuration.

3.2. Default Resource Adaptor Entities

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

The `XCAPClientRA` entity is also bound to Resource Adaptor Link Name `XCAPClientRA`, 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>
         XCAPClientResourceAdaptorType
      </resource-adaptor-type-name>
      <resource-adaptor-type-vendor>
         org.mobicents
      </resource-adaptor-type-vendor>
      <resource-adaptor-type-version>
         2.0
      </resource-adaptor-type-version>
   </resource-adaptor-type-ref>
   <activity-context-interface-factory-name>
      slee/resources/xcapclient/2.0/acif
   </activity-context-interface-factory-name>
   <resource-adaptor-entity-binding>
      <resource-adaptor-object-name>
         slee/resources/xcapclient/2.0/sbbrainterface
      </resource-adaptor-object-name>
      <resource-adaptor-entity-link>
         XCAPClientRA
      </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 `XCAPClientResourceAdaptor`. The related Log4j Logger category, which can be used to change the Tracer level from Log4j configuration, is `javax.slee.RAEntityNotification[entity=XCAPClientRA]`

3.3.2. Alarms

No alarms are set by this Resource Adaptor.

Setup

4.1. Pre-Install Requirements and Prerequisites

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

4.1.1. Hardware Requirements

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

4.1.2. Software Prerequisites

The RA requires JBoss Communications JAIN SLEE properly set.

4.2. JBoss Communications JAIN SLEE XCAP Client 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 ?, then add the specific release version, lets consider 2.2.2.FINAL.

```
[usr]$ svn co ?/2.2.2.FINAL slee-ra-xcap-client-2.2.2.FINAL
```

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-xcap-client-2.2.2.FINAL
[usr]$ mvn install
```

Once the process finishes you should have the `deployable-unit` jar file in the `target` directory, if JBoss Communications JAIN SLEE is installed and environment variable JBOSS_HOME is pointing to its underlying JBoss Enterprise Application Platform 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 NOT AVAILABLE.

4.3. Installing JBoss Communications JAIN SLEE XCAP Client 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` JBoss Communications JAIN SLEE server profile deploy directory, to deploy to another server profile use the argument `-Dnode=`.

4.4. Uninstalling JBoss Communications JAIN SLEE XCAP Client 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` JBoss Communications JAIN SLEE server profile deploy directory, to undeploy from another server profile use the argument `-Dnode=`.

Clustering

The XCAP Client Resource Adaptor is not cluster aware, which means there is no failover process for a cluster node's requests being made once the node fails.

Appendix A. Revision History

Revision History
Revision 1.0 Tue Dec 30 2009 EduardoMartins
Creation of the JBoss Communications JAIN SLEE XCAP Client RA User Guide.

Index

F

feedback, viii