JBoss Communications JAIN SLEE HTTP Client Resource Adaptor User Guide

by Amit Bhayani and 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 HTTP Client Resource Adaptor 1
2. Resource Adaptor Type 3
2.1. Activities 3
2.2. Events 4
2.3. Activity Context Interface Factory 4
2.4. Resource Adaptor Interface 5
2.5. Restrictions 6
2.6. Sbb Code Examples 6
2.6.1. Synchronous Operations 7
2.6.2. Asynchronous Operations 7
3. Resource Adaptor Implementation 9
3.1. Configuration 9
3.2. Default Resource Adaptor Entities 9
3.3. Traces and Alarms 10
3.3.1. Tracers 10
3.3.2. Alarms 10
4. Setup 11
4.1. Pre-Install Requirements and Prerequisites 11
4.1.1. Hardware Requirements 11
4.1.2. Software Prerequisites 11
4.2. JBoss Communications JAIN SLEE HTTP Client Resource Adaptor Source Code 11
4.2.1. Release Source Code Building 11
4.2.2. Development Trunk Source Building 12
4.3. Installing JBoss Communications JAIN SLEE HTTP Client Resource Adaptor 12
4.4. Uninstalling JBoss Communications JAIN SLEE HTTP Client Resource Adaptor 12
5. Clustering 13
A. Revision History 15
Index 17

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 HTTP Client Resource Adaptor**, or contact the authors.

When submitting a bug report, be sure to mention the manual's identifier: JAIN_SLEE_HttpClient_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 HTTP Client Resource Adaptor

The Hyper-Text Transfer Protocol (HTTP) is perhaps the most significant protocol used on the Internet today. Web services, network-enabled appliances and the growth of network computing continue to expand the role of the HTTP protocol beyond user-driven web browsers, while increasing the number of applications that require HTTP support; for example Application developed using SLEE. The Jakarta Commons HttpClient component provides an efficient, up-to-date, and feature-rich package implementing the client side of the most recent HTTP standards and recommendations. Http Client RA provides the client side HTTP standard within the SLEE environment, using the popular Apache Commons Http Client library. An SBB can use the HTTP Client RA to make a request and get the Response Synchronously or Asynchronously.

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 `HttpClientResourceAdaptorType`, its vendor is `org.mobicents` and its version is `1.0`.

2.1. Activities

The single activity object for HTTP Client Resource Adaptor is the `net.java.client.slee.resource.http.HttpClientActivity` interface. Through the activity an SBB can send multiple HTTP 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 the response is received, or when the SBB invokes its `endActivity()` method.

The `HttpClientActivity` interface is defined as follows:

```
package net.java.client.slee.resource.http;

import org.apache.commons.httpclient.HttpMethod;

public interface HttpClientActivity {

    public String getSessionId();

    public void endActivity();

    public boolean getEndOnReceivingResponse();

    public void executeMethod(HttpMethod httpMethod);
}
```

The `getSessionId()` method:
Retrieves the activity unique Id.

The `getEndOnReceivingResponse()` method:
Returns true if this Activity is set to end as soon as the Response is received

The `executeMethod(HttpMethod)` method:

Executes an `HttpMethod` , i.e., sends an HTTP request, created through the RA SBB Interface.

The `endActivity()` method:

Ends the activity and its related Activity Context.

2.2. Events

There is a single event fired by HTTP Client Resource Adaptor, which represents a response to a request, received in a specific `HttpClientActivity` instance.

Table 2.1. Events fired on the HttpClientActivity

Name	Vendor	Version	Event Class	Description
net.java. client.slee. resource.http. event. ResponseEvent	net.java. client.slee	1.0	net.java. client.slee. resource.http. event. ResponseEvent	A response event to an asynchronous HTTP request.

Important

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

The response event class provides the result of sending an HTTP request, which can be response or exception, due to error.

2.3. Activity Context Interface Factory

The Resource Adaptor's Activity Context Interface Factory is of type `net.java.client.slee.resource.http.HttpClientActivityContextInterfaceFactory`, it allows the SBB to retrieve the `ActivityContextInterface` related with an existing Resource Adaptor activity object. The interface is defined as follows:

```
package net.java.client.slee.resource.http;

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

```
public interface HttpClientActivityContextInterfaceFactory {

    public ActivityContextInterface getActivityContextInterface(
            HttpClientActivity acivity) throws NullPointerException,
            UnrecognizedActivityException, FactoryException;

}
```

2.4. Resource Adaptor Interface

The HTTP Client Resource Adaptor interface, of type `net.java.client.slee.resource.http.HttpClientResourceAdaptorSbbInterface`, which an SBB uses to create new `HttpClientActivity` instances or send synchronous requests, its interface is defined as follows:

```
package net.java.client.slee.resource.http;

import java.io.IOException;

import javax.slee.resource.StartActivityException;

import net.java.client.slee.resource.http.event.Response;

import org.apache.commons.httpclient.HttpException;
import org.apache.commons.httpclient.HttpMethod;
import org.apache.commons.httpclient.HttpState;
import org.apache.commons.httpclient.params.HttpClientParams;

public interface HttpClientResourceAdaptorSbbInterface {

    public HttpClientActivity createHttpClientActivity() throws StartActivityException;

    public HttpClientActivity createHttpClientActivity(
            boolean endOnReceivingResponse) throws StartActivityException;

    public HttpMethod createHttpMethod(HttpMethodName methodName, String uri);

    public Response executeMethod(HttpMethod method) throws HttpException, IOException;

    public HttpClientParams getParams();
```

```
    public HttpState getState();

    public void setParams(HttpClientParams params);

    public void setState(HttpState state);
}
```

The `createHttpClientActivity()` method:
: Creates a new `HttpClientActivity` instance which has to be ended by the SBB, that is, it won't end once the HTTP Request response is received.

The `createHttpClientActivity(boolean)` method:
: Creates a new `HttpClientActivity` instance. If the parameter is true then the activity will end once the HTTP Request response is received, if false then the SBB needs to end the activity through its endActivity() method.

The `createHttpMethod(HttpMethodName, String)` method:
: Creates an HttpMethod instance, which request will be sent to the specified URI.

The `executeMethod(HttpMethod)` method:
: Sends the HttpMethod's request synchronously, blocking till a response is received.

The `getParams()` method:
: Retrieves the HTTP protocol parameters associated with the HttpClient.

The `getState()` method:
: Retrieves the HTTP state associated with the HttpClient.

The `setParams(HttpClientParams)` method:
: Sets the HTTP protocol parameters for the HttpClient.

The `setState(HttpState)` method:
: Sets the HTTP state for the HttpClient.

2.5. Restrictions

The HTTP Client Resource Adaptor Type does not defines any restriction on the usage of its interfaces.

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 HTTP requests:

```
HttpMethod httpMethod = raSbbInterface.createHttpMethod(
        HttpMethodName.GET, syndFeed.getLink());
try {
   Response response = raSbbInterface.executeMethod(httpMethod);
} catch (Throwable e) {
   tracer.severe("Error while sending request",e);
}
```

2.6.2. Asynchronous Operations

The following code examples the creation and usage of the HttpClientActivity to send an async HTTP GET requests the optimal way to use the RA, since it doesn't block the SLEE container event routing threads:

```
HttpMethod httpMethod = raSbbInterface.createHttpMethod(
        HttpMethodName.GET, syndFeed.getLink());
try {
   HttpClientActivity clientActivity = raSbbInterface
        .createHttpClientActivity(true);
   ActivityContextInterface clientAci = httpClientAci
        .getActivityContextInterface(clientActivity);
   clientAci.attach(sbbContext.getSbbLocalObject());
   clientActivity.executeMethod(httpMethod);
} catch (Throwable e) {
   tracer.severe("Error while creating HttpClientActivity",e);
}
```

Resource Adaptor Implementation

This chapter documents the HTTP 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 `HttpClientResourceAdaptor`, its vendor is `org.mobicents` and its version is `1.0`.

3.1. Configuration

The Resource Adaptor implementation does not have any configuration properties.

3.2. Default Resource Adaptor Entities

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

The `HttpClientResourceAdaptor` entity is also bound to Resource Adaptor Link Name `HttpClientResourceAdaptor`, 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>
         HttpClientResourceAdaptorType
      </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/http-client/acifactory
   </activity-context-interface-factory-name>
   <resource-adaptor-entity-binding>
      <resource-adaptor-object-name>
         slee/resources/http-client/sbbinterface
      </resource-adaptor-object-name>
      <resource-adaptor-entity-link>
         HttpClientResourceAdaptor
      </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 `HttpClientResourceAdaptor`. The related Log4j Logger category, which can be used to change the Tracer level from Log4j configuration, is `javax.slee.RAEntityNotification[entity=HttpClientResourceAdaptor]`

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 HTTP 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 2.3.0.FINAL.

```
[usr]$ svn co ?/2.3.0.FINAL slee-ra-http-client-2.3.0.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-http-client-2.3.0.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 HTTP 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 HTTP 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 HTTP Client Resource Adaptor is not cluster aware, which means there is no failover process for a cluster node's requests being handled, 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 HTTP Client RA User Guide.

Index

F

feedback, viii