

JBoss Communications JAIN SLEE REST Client Enabler User Guide

by Eduardo Martins and Bartosz Baranowski

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 REST Client Enabler	1
2. Setup	3
2.1. Pre-Install Requirements and Prerequisites	3
2.1.1. Hardware Requirements	3
2.1.2. Software Prerequisites	3
2.2. JBoss Communications JAIN SLEE REST Client Enabler Source Code	3
2.2.1. Release Source Code Building	3
2.2.2. Development Trunk Source Building	4
2.3. Installing JBoss Communications JAIN SLEE REST Client Enabler	4
2.4. Uninstalling JBoss Communications JAIN SLEE REST Client Enabler	5
3. Integrating the JBoss Communications JAIN SLEE REST Client Enabler	7
3.1. The Parent's SbbLocalObject Interface	7
3.2. Handling HTTP Responses Provided by the Enabler	8
3.3. The Parent's Sbb Abstract Class	9
3.4. The Parent's Sbb XML Descriptor	9
4. Using the JBoss Communications JAIN SLEE REST Client Enabler	11
4.1. The Child's SbbLocalObject Interface	11
4.2. Creating And Retrieving The Child Sbb	11
4.3. Creating Requests for REST Resources	12
5. Traces and Alarms	13
5.1. Tracers	13
5.2. Alarms	13
A. Revision History	15
Index	17

Preface

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/) [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 Of 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/) [http://bugzilla.redhat.com/bugzilla/], against the product **JBoss Communications JAIN SLEE REST Client Enabler**, or contact the authors.

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

Client Enabler

The JAIN SLEE REST Client Enabler allows JAIN SLEE Applications to interact with REST Web Services, through an easy high level API. The Enabler consists in an SBB which can be used in child relations, to send REST requests, and receive responses asynchronously.

The JAIN SLEE REST Client Enabler uses the JAIN SLEE HTTP Client, based on Apache HTTP Client, to send the lower level HTTP requests. The enabler also uses the Signpost API to handle OAuth authentication and authorization procedures.

Setup

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

The Enabler doesn't change the JBoss Communications JAIN SLEE Hardware Requirements, refer to JBoss Communications JAIN SLEE documentation for more information.

2.1.2. Software Prerequisites

The Enabler requires JBoss Communications JAIN SLEE properly set, with the HTTP Client Resource Adaptor deployed.

2.2. JBoss Communications JAIN SLEE REST Client Enabler Source Code

This section provides instructions on how to obtain and build the REST Client Enabler from source code.

2.2.1. 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 1.0.0.BETA1.

```
[usr]$ svn co ?/1.0.0.BETA1 slee-enabler-rest-client-1.0.0.BETA1
```

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-enabler-rest-client-1.0.0.BETA1
[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.



Important

This procedure does not install the Enabler's dependencies

2.2.2. Development Trunk Source Building

Similar process as for [Section 2.2.1, "Release Source Code Building"](#), the only change is the SVN source code URL, which is NOT AVAILABLE.

2.3. Installing JBoss Communications JAIN SLEE REST Client Enabler

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

```
[usr]$ ant
```

The script will copy the Enabler's deployable unit jar to the `default` JBoss Communications JAIN SLEE server profile deploy directory, to deploy to another server profile use the argument `-Dnode=.`



Important

This procedure also installs the Enabler's dependencies.

2.4. Uninstalling JBoss Communications JAIN SLEE REST Client Enabler

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

```
[usr]$ ant undeploy-all
```

The script will delete the Enabler's deployable unit jar from the `default` JBoss Communications JAIN SLEE server profile deploy directory, to undeploy from another server profile use the argument `-Dnode=`.



Important

This procedure also uninstalls the Enabler's dependencies.

Integrating the JBoss Communications JAIN SLEE REST Client Enabler

This chapter explains how to setup a JAIN SLEE Service Sbb to use the Enabler.

In short terms, a Service's Sbb will define the Enabler's Sbb as a child, and to achieve that it will need to setup the XML Descriptor, Abstract Class and SbbLocalObject interface.



Important

The Service's Sbb will be referred as the Parent Sbb in the following sections.

3.1. The Parent's SbbLocalObject Interface

The JBoss Communications JAIN SLEE REST Client Enabler Sbb provides asynchronous callbacks to the Parent's Sbb, and that can only be achieved if the Parent's SbbLocalObject extends a specific Java interface, deployed also by the Enabler. The Enabler uses the Parent's SbbLocalObject when a callback to the Parent's Sbb is needed.

The SbbLocalObject which must be used or extended by the Parent's Sbb is named `org.mobicens.slee.enabler.rest.client.RESTClientEnablerParentSbbLocalObject`, which extends the `org.mobicens.slee.SbbLocalObjectExt` and `org.mobicens.slee.enabler.rest.client.RESTClientEnablerParent` interfaces, the latter declares the callbacks which must be implemented in the Parent's Sbb Abstract Class:

```
package org.mobicens.slee.enabler.rest.client;

public interface RESTClientEnablerParent {

    public void onResponse(RESTClientEnablerChildSbbLocalObject child,
        RESTClientEnablerResponse response);

}
```

The `onResponse(RESTClientEnablerChildSbbLocalObject, RESTClientEnablerResponse)` method:

Callback from the Enabler providing the result of executing a REST request, which besides containing the executed REST request, also contains either the received HTTP Response, or the exception which was thrown when executing the request. It also provides the child sbb which was used to execute the request.

3.2. Handling HTTP Responses Provided by the Enabler

Please refer to the [Apache HTTP Client 4.x Tutorial](http://hc.apache.org/httpcomponents-client-ga/tutorial/html/index.html) [http://hc.apache.org/httpcomponents-client-ga/tutorial/html/index.html] for detailed instructions on how to process the HTTP Response object, the code below is just a simple example of tracing the request information, the response status and content:

```
@Override
public void onResponse(RESTClientEnablerChildSbbLocalObject child,
    RESTClientEnablerResponse response) {

    String uri = response.getRequest().getUri();
    RESTClientEnablerRequest.Type type = response.getRequest().getType();
    HttpResponse httpResponse = response.getHttpResponse();
    if (httpResponse != null) {
        String content = null;
        if (httpResponse.getEntity() != null) {
            try {
                content = EntityUtils.toString(httpResponse.getEntity());
            } catch (Exception e) {
                tracer.severe("failed to convert response content to String", e);
            }
        }
        tracer.info("onResponse. Child " + child + ", request type "
            + type + ", uri " + uri + ", status "
            + httpResponse.getStatusLine().getStatusCode()
            + ", response content " + content + "");
    } else {
        tracer.info("onResponse. Child " + child + ", request type "
            + type + ", uri " + uri + ", exception "
            + response.getExecutionException() + "");
    }
}
```


3.3. The Parent's Sbb Abstract Class

The Parent Sbb Abstract Class must implement the callbacks on its SbbLocalObject, that is, must implement the `org.mobicens.slee.enabler.rest.client.RESTClientEnablerParent` interface discussed in last section.

The Enabler's Sbb is a Child Sbb, and JAIN SLEE 1.1 Child Relations requires an abstract method in the Sbb Abstract Class, to retrieve the `org.mobicens.slee.ChildRelationExt` object, which is used to create or access specific Child Sbbs. This method should be:

```
public abstract ChildRelationExt getRESTClientEnablerChildRelation();
```

3.4. The Parent's Sbb XML Descriptor

The Parent's Sbb must define a reference to the Enabler's Child Sbb, declare which is the method name to get the related ChildRelation object, and also ensure the SbbLocalObject interface is defined correctly.

A reference to the Enabler's Child Sbb is defined right after the Parent's Sbb Vendor ID element, using the following XML element:

```
<sbb-ref>
  <sbb-name>RESTClientEnabler</sbb-name>
  <sbb-vendor>org.mobicens</sbb-vendor>
  <sbb-version>1.0</sbb-version>
  <sbb-alias>restClientChildSbb</sbb-alias>
</sbb-ref>
```

The method name to get the Enabler's ChildRelation object must be defined after the CMP Fields (if any), this XML element links the sbb-alias previously defined with the abstract method declared in the Parent's Sbb Abstract Class:

```
<get-child-relation-method>
  <sbb-alias-ref>restClientChildSbb</sbb-alias-ref>
```

```
<get-child-relation-method-name>getRESTClientEnablerChildRelation</get-child-relation-  
method-name>  
  <default-priority>0</default-priority>  
</get-child-relation-method>
```

Finally, after the `sbb-abstract-class` element the Parent's `SbbLocalObject` interface name is defined:

```
<sbb-local-interface>  
  <sbb-local-interface-  
name>org.mobicens.slee.enabler.rest.client.RESTClientEnablerParentSbbLocalObject</sbb-  
local-interface-name>  
</sbb-local-interface>
```



Important

The Sbb local object interface may be an interface which extends `org.mobicens.slee.enabler.rest.client.RESTClientEnablerParentSbbLocalObject` for instance one that extends multiple Enabler Parent Sbb local object interfaces.

Using the JBoss Communications JAIN SLEE REST Client Enabler

In the last chapter we integrated the Enabler in the JAIN SLEE Service's Sbb, the Parent Sbb, in this chapter it is explained how to use the Enabler's Sbb, the Child Sbb.

4.1. The Child's SbbLocalObject Interface

The JBoss Communications JAIN SLEE REST Client Enabler Sbb, the Child Sbb, implements the `org.mobicens.slee.enabler.rest.client.RESTClientEnablerChildSbbLocalObject`, which extends the `org.mobicens.slee.SbbLocalObjectExt` and `org.mobicens.slee.enabler.rest.client.RESTClientEnablerChild` interfaces, the latter declares the methods which can be used to interact with the REST Web Service:

```
package org.mobicens.slee.enabler.rest.client;

public interface RESTClientEnablerChild {

    public void execute(RESTClientEnablerRequest request) throws Exception;

}
```

The `execute(RESTClientEnablerRequest)` method:
Executes the specified request asynchronously.

4.2. Creating And Retrieving The Child Sbb

The Child Relation in the Parent Sbb Abstract Class is used to create and retrieve the Child Sbb:

```
RESTClientEnablerChild sbb = null;
// creation
try {
    sbb = (RESTClientEnablerChild) getRESTClientEnablerChildRelation().create(childName);
}
catch (Exception e) {
    tracer.severe("Failed to create child sbb", e);
}
```

```
}  
// retrieval  
try {  
    sbb = (RESTClientEnablerChild) getRESTClientEnablerChildRelation().get(childName);  
}  
catch (Exception e) {  
    tracer.severe("Failed to retrieve child sbb", e);  
}
```

4.3. Creating Requests for REST Resources

Requests are instances of `org.mobicens.slee.enabler.rest.client.RESTClientEnablerRequest`, which include all the mandatory and optional data, needed by the Enabler to send the HTTP messages to the REST Web Services.

The only mandatory data is the request type (the HTTP message method name) and the REST Web Service URI, but the application using the enabler may also include a `SignPost OAuthConsumer` object, to sign the request according to OAuth protocol, additional headers and/or content to include in the HTTP message, and content. The following Java code examples the creation of a POST request to update status in the Twitter network:

```
String uri = "http://api.twitter.com/1/statuses/update.json?status=" +  
    + URLEncoder.encode(  
        "Hello Twitter!", "UTF-8");  
CommonsHttpOAuthConsumer consumer =  
    new CommonsHttpOAuthConsumer(twitterConsumerKey,  
        twitterConsumerSecret);  
consumer.setTokenWithSecret(twitterAccessToken,twitterAccessTokenKey);  
RESTClientEnablerRequest request = new RESTClientEnablerRequest(  
    RESTClientEnablerRequest.Type.POST, uri)  
    .setOAuthConsumer(consumer);
```

Traces and Alarms

5.1. Tracers



Important

Spaces were introduced in log4j category name to properly render page. Please remove them when using copy/paste.

The JAIN SLEE REST Client Enabler uses a single JAIN SLEE 1.1 Tracer, named `RESTClientEnabler`. The related log4j category is `javax.slee.SbbNotification[service=ServiceID[name=ServiceX,vendor=VendorY,version=VersionZ],sbb=SbbID[name=RESTClientEnabler,vendor=org.mobicients,version=1.0]]`. Where `ServiceX`, `VendorY` and `VersionZ` need to be replaced by the identifiers of the Service using the Enabler.

5.2. Alarms

The JAIN SLEE REST Client Enabler does not set JAIN SLEE Alarms.

Appendix A. Revision History

Revision History

Revision 1.0

Tue Aug 31 2011

EduardoMartins

Creation of the JBoss Communications JAIN SLEE REST Client Enabler User Guide.

Index

F

feedback, viii

