

Mobicents JAIN SLEE SIP Subscription Client Enabler User Guide

by 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 Mobicents JAIN SLEE SIP Subscription 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. Mobicents JAIN SLEE SIP Subscription Client Enabler Source Code	3
2.2.1. Release Source Code Building	3
2.2.2. Development Trunk Source Building	4
2.3. Installing Mobicents JAIN SLEE SIP Subscription Client Enabler	4
2.4. Uninstalling Mobicents JAIN SLEE SIP Subscription Client Enabler	5
3. Integrating the Mobicents JAIN SLEE SIP Subscription Client Enabler	7
3.1. The Parent's SbbLocalObject Interface	7
3.2. The Parent's Sbb Abstract Class	9
3.3. The Parent Sbb XML Descriptor	9
4. Using the Mobicents JAIN SLEE SIP Subscription Client Enabler	11
4.1. The Child's SbbLocalObject Interface	11
4.2. Creating And Retrieving The Child Sbb	13
4.3. Enabler configuration	14
5. Traces and Alarms	15
5.1. Tracers	15
5.2. Alarms	15
A. Revision History	17
Index	19

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://code.google.com/p/mobicents/issues/list) [http://code.google.com/p/mobicents/issues/list], against the product **Mobicents JAIN SLEE SIP Subscription Client Enabler**, or contact the authors.

When submitting a bug report, be sure to mention the manual's identifier: JAIN_SLEE_ENABLER_SIPSubscriptionClient_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 Mobicents JAIN SLEE SIP Subscription Client Enabler

The JAIN SLEE SIP Subscription Client Enabler allows JAIN SLEE Applications to interact with SIP Event Servers, such as Presence Servers, hiding the network protocol complexity. The Enabler consists of SBB, which can be used in child relations, with a simple synchronous interface.

Enabler abstracts communication described in [RFC3265](#).

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 Mobicents JAIN SLEE Hardware Requirements, refer to Mobicents JAIN SLEE documentation for more information.

2.1.2. Software Prerequisites

The Enabler requires Mobicents JAIN SLEE properly set, with the SIP Resource Adaptor deployed.

2.2. Mobicents JAIN SLEE SIP Subscription Client Enabler Source Code

This section provides instructions on how to obtain and build the SIP Subscription 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 <http://mobicents.googlecode.com/svn/tags/servers/jain-slee/2.x.y/enablers/sip-subscription-client>, then add the specific release version, lets consider 1.0.0.BETA2.

```
[usr]$ svn co http://mobicents.googlecode.com/svn/tags/servers/jain-slee/2.x.y/enablers/sip-subscription-client/1.0.0.BETA2 slee-enabler-sip-subscription-client-1.0.0.BETA2
```

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-sip-subscription-client-1.0.0.BETA2
[usr]$ mvn install
```

Once the process finishes you should have the `deployable-unit` jar file in the `target` directory, if Mobicents 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.



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 <http://mobicents.googlecode.com/svn/trunk/servers/jain-slee/enablers/sip-subscription-client>.

2.3. Installing Mobicents JAIN SLEE SIP Subscription 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` Mobicents 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 Mobicents JAIN SLEE SIP Subscription 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` Mobicents 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 Mobicents JAIN SLEE SIP Subscription Client Enabler

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

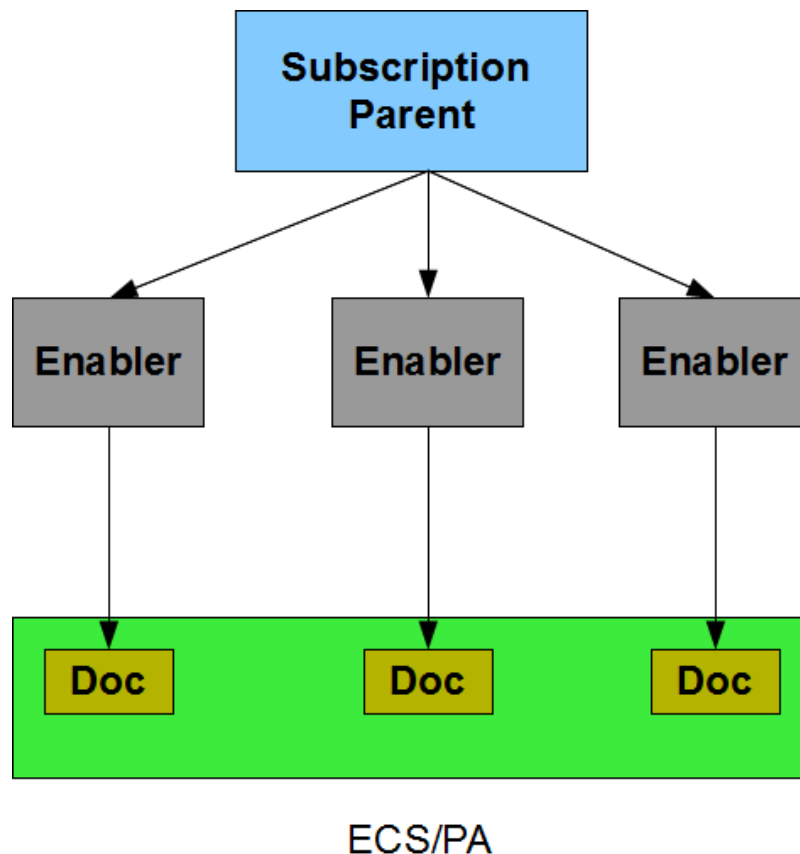
In short terms, a Service's Sbb will define the Enabler's Sbb as a child, and to achieve that it will need to setup the XML Descriptor, Abstract Class and SbbLocalObject interface.



Important

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

Relation between Parent Sbb, Enabler instance and subscriptions to Presence Server, look as follows:



JAIN SLEE SIP Subscription Client Enabler design

3.1. The Parent's SbbLocalObject Interface

The Mobicents JAIN SLEE SIP Subscription Client Enabler Sbb provides synchronous 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, and provides it's SbbLocalObject to the Enabler's Sbb, through a specific method exposed by the latter interface. The Enabler stores the Parent's SbbLocalObject and uses it 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.sip.SubscriptionClientParentSbbLocalObject`, which extends the `javax.slee.SbbLocalObject` and `org.mobicens.slee.enabler.sip.SubscriptionClientParent` interfaces, the latter declares the callbacks which must be implemented in the Parent's Sbb Abstract Class:

```
package org.mobicens.slee.enabler.sip;

public interface SubscriptionClientParent {

    public void subscribeSucceed(int responseCode,
        SubscriptionClientChildLocalObject enabler);

    public void unsubscribeSucceed(int responseCode,
        SubscriptionClientChildLocalObject enabler);

    public void onNotify(Notify notify,
        SubscriptionClientChildLocalObject enabler);

    public void subscribeFailed(int responseCode, S
        ubscriptionClientChildLocalObject sbbLocalObject);

    public void resubscribeFailed(int responseCode,
        SubscriptionClientChildLocalObject sbbLocalObject);

    public void unsubscribeFailed(int responseCode,
        SubscriptionClientChildLocalObject sbbLocalObject);

}
```

The `subscribeSucceed(int responseCode, SubscriptionClientChildLocalObject enabler);` method:

Callback from the Enabler providing details about initial subscription. If it indicates error, enabler instance should be discarded.

The `unsubscribeSucceed(int responseCode, SubscriptionClientChildLocalObject enabler);` method:

Callback from the Enabler providing details about remove subscription. If it indicates error, enabler instance should be discarded.

The `onNotify(Notify notify, SubscriptionClientChildLocalObject enabler);` method:

Callback from the Enabler providing details about notification. If notification indicates termination of subscription, enabler can be safely removed.

The `subscribeFailed(SubscriptionClientChildLocalObject sbbLocalObject);` method:

Callback from the Enabler indicating communication failure, enabler instance must be discarded.

The `resubscribeFailed(SubscriptionClientChildLocalObject sbbLocalObject);` method:

Callback from the Enabler indicating communication failure, enabler instance must be discarded.

The `unsubscribeFailed(SubscriptionClientChildLocalObject sbbLocalObject);` method:

Callback from the Enabler indicating communication failure, enabler instance must be discarded.

3.2. The Parent's Sbb Abstract Class

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

The Enabler is a Child Sbb. Parent requires JAIN SLEE 1.1 Child Relation to access Enabler. SLEE specification mandates parent to declare an abstract method to retrieve the `javax.slee.ChildRelation` object, which is used to create or access specific Child Sbbs. This method may be declared as follows:

```
public abstract ChildRelation getSipSubscriptionClientChildRelation();
```

3.3. The Parent Sbb XML Descriptor

The Parent Sbb must define following information in its descriptor

child reference

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>SipSubscriptionClientChildSbb</sbb-name>
  <sbb-vendor>org.mobicens</sbb-vendor>
  <sbb-version>1.0</sbb-version>
  <sbb-alias>sipSubClientChildSbb</sbb-alias>
</sbb-ref>
```

child relation method

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>sipSubClientChildSbb</sbb-alias-ref>
  <get-child-relation-method-name>getSipSubscriptionClientChildRelation</get-
child-relation-method-name>
  <default-priority>0</default-priority>
</get-child-relation-method>
```

Sbb Local Object

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

```
<sbb-local-interface>
  <sbb-local-interface-name>...</sbb-local-interface-name>
</sbb-local-interface>
```

Using the Mobicents JAIN SLEE SIP Subscription 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.

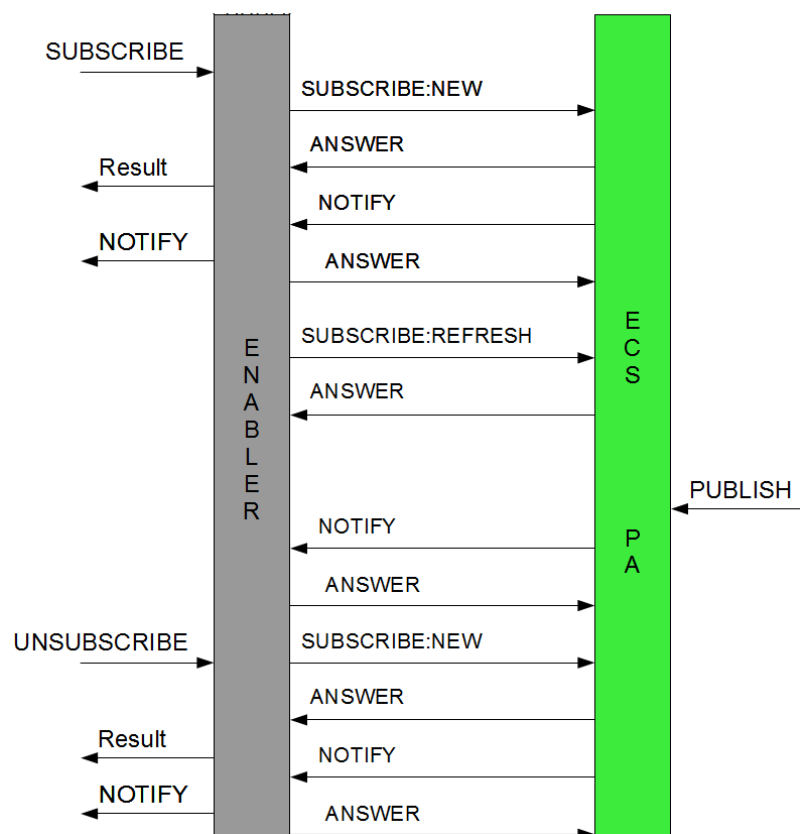
Enabler performs following tasks:

manage subscription

send subscription events on behalf of Parent Sbb

automatic refresh

based on values exchanged between Enabler and Server, Enabler keeps track of subscription life time(expiration) and issues refresh requests to Server



JAIN SLEE SIP Subscription Client Enabler control flow

4.1. The Child's SbbLocalObject Interface

The Mobicents JAIN SLEE SIP Subscription Client Enabler Sbb, the Child Sbb, implements the `org.mobicents.slee.enabler.sip.SubscriptionClientChildSbbLocalObject` , which

extends the `javax.slee.SbbLocalObject` and `org.mobicents.slee.enabler.sip.SubscriptionClientChild` interfaces, the latter declares the methods which can be used to interact with the SIP Event Server:

```
package org.mobicents.slee.enabler.sip;

public void setParentSbb(SubscriptionClientParentSbbLocalObject parent);

public String getSubscriber();

public String getEventPackage();

public String getNotifier();

public void subscribe(String subscriber, String subscriberdisplayName, String notifier,
    int expires, String eventPackage, Map<String, String> eventsParameters,
    String acceptedContentType, String acceptedContentSubtype) throws SubscriptionException;

public void subscribe(String subscriber, String subscriberdisplayName, String notifier,
    int expires, String eventPackage, Map<String, String> eventParameters,
    String acceptedContentType, String acceptedContentSubtype, String contentType,
    String contentSubType, String content) throws SubscriptionException;

public void unsubscribe() throws SubscriptionException;
```

The `setParentSbb(SubscriptionClientParentLocalObject parent);` method:

Passes the Parent's `SbbLocalObject`, which will be used by the Child `Sbb` to provide async results. If not invoked after the child creation the Enabler won't be able to callback the Parent `Sbb`.

The `getSubscriber();` method:

returns subscriber used for subscription in Server.

The `getNotifier();` method:

returns notifier used as target for subscription.

The `getEventPackage();` method:

returns event package used in subscription, ie. `presence` or `presence.wininfo`

The `public void subscribe(String subscriber, String subscriberdisplayName, String notifier, int expires, String eventPackage, Map<String, String> eventsParameters, String acceptedContentType, String acceptedContentSubtype)` throws `SubscriptionException`; **method:**

this method should be called to subscribe enabler to certain document in Server, optionally user may pass `eventsParameters`. Event Parameters are sent with each SUBSCRIBE. Accepted content type arguments indicate type of expected content generated by Presence Server. It throws exception in case of transport error or wrong arguments.

The `public void subscribe(String subscriber, String subscriberdisplayName, String notifier, int expires, String eventPackage, Map<String, String> eventParameters, String acceptedContentType, String acceptedContentSubtype, String contentType, String contentSubType, String content)` throws `SubscriptionException`; **method:**

this method is similar to previously defined method. In addition, this method allows to pass `filter` document in each SUBSCRIBE sent to Presence Server. This is useful, for instance in case of subscription to RLS service in XDM server. Content type arguments must match type of document passed to Presence Server.

The `unsubscribe()` throws `SubscriptionException`; **method:**

this method should be called to explicitly unsubscribe from publication. If enabler is refreshing, when this method is called, it will throw exception.

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, it is important to not forget to pass the Parent's `SbbLocalObject` to the Child after creation:

```
public SubscriptionClientChildLocalObject getPublicationClientChildSbb() {
    final ChildRelation childRelation = getSipPublicationClientChildRelation();
    if (childRelation.isEmpty()) {
        try {
            // creates new instance
            SubscriptionClientChildLocalObject sbb =
                (SubscriptionClientChildLocalObject) childRelation.create();
            // passes the parent sbb local object to the child
            sbb.setParentSbb((SubscriptionClientParentLocalObject)
                sbbContext.getSbbLocalObject());
            return sbb;
        } catch (Exception e) {
            tracer.severe("Failed to create child sbb", e);
            return null;
        }
    }
    else {
```

```
        return (SubscriptionClientChildLocalObject)
            childRelation.iterator().next();
    }
}
```

The SbbLocalObject of the Child could also be stored in a CMP Field for the simplest retrieval, but unless you are going to reuse each instance several times it's better to have less state, specially in clustered environments.

4.3. Enabler configuration

Enabler can be configured with SLEE environment entries. Currently following entries are supported:

Table 4.1. Environment entry table

Name	Type	Description
server.address	java.lang.String	Specifies address to which requests should be forwarded. It has form of ip:port pair.
expires.drift	java.lang.Integer	Specifies time drift, in seconds, between value of Expires/Min-Expires values and automatic refresh performed by enabler. For instance if Expires value passed(and accepted by Server) is 3600, time drift set to 10, Enabler will refresh publication after 3590

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 SIP Subscription Client Enabler uses a single JAIN SLEE 1.1 Tracer, named `SipSubscriptionClientChildSbb`. The related log4j category is `javax.slee.SbbNotification[service=ServiceID[name=ServiceX,vendor=VendorY,version=VersionZ],sbb=SbbID[name=SipSubscriptionClientChildSbb,vendor=org.mobicens,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 SIP Subscription Client Enabler does not set JAIN SLEE Alarms.

Appendix A. Revision History

Revision History

Revision 1.0

Tue Oct 26 2010

EduardoMartins

Creation of the Mobicents JAIN SLEE SIP Subscription Client Enabler User Guide.

Index

F

feedback, viii

