Mobicents SIP Presence Service User Guide

by Douglas Silas, Eduardo Martins, and Jared Morgan

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.

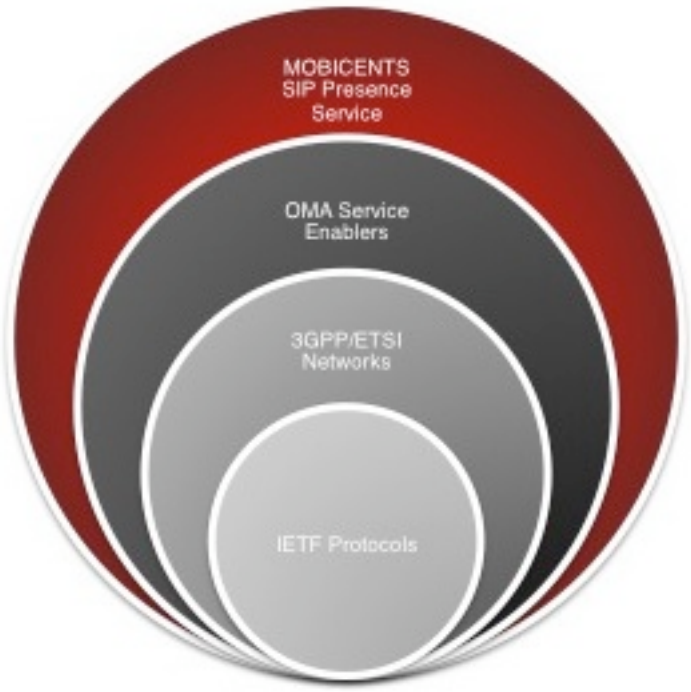
2. Provide feedback to the authors!

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 Bugzilla: *http://bugzilla.redhat.com/bugzilla/* against the product **${product.name}**, or contact the authors.

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

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 the Mobicents SIP Presence Service

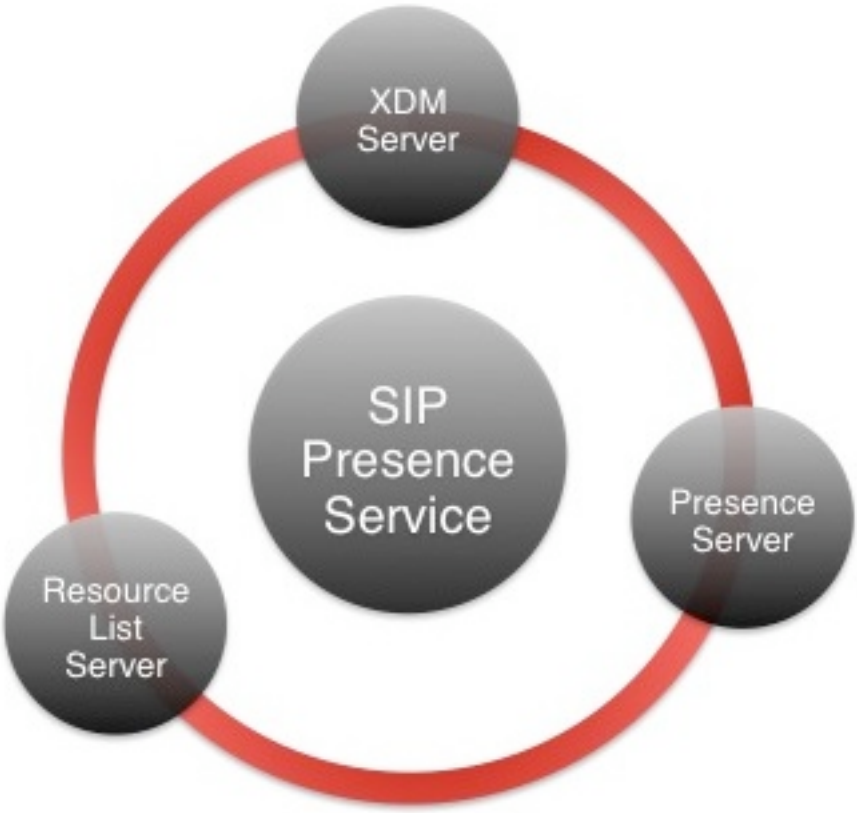


Mobicents SIP Presence Service relation with standard groups

The **Mobicents SIP Presence Service** provides presence functionalities to SIP-based networks using standards developed by the Internet Engineering Task Force (IETF), the Open Mobile Alliance (OMA), the 3rd Generation Partnership Project (3GPP) and the European Telecommunications Standards Institute (ETSI).

1.1. Architecture of the Mobicents SIP Presence Service

The **SIP Presence Service** is comprised of three separate but interrelated servers.



Mobicents SIP Presence Service servers

The Three Servers Comprising the SIP Presence Service

The SIP Presence Server

The **Mobicents SIP Presence Server** (PS) is an entity that accepts, stores and distributes SIP Presence Information. The **Presence Server** performs the following functions:

- It manages publications from one or multiple presence source(s) of a certain *presentity*. This includes refreshing presence information, replacing existing presence information with newly-published information, or removing presence information.
- It manages subscriptions from watchers to presence information and generates notifications about presence information state changes, retrieving the presence authorization rules from the **XDM Server**.
- It manages subscriptions from watcher information subscribers to watcher information and generates notifications about watcher information state changes.

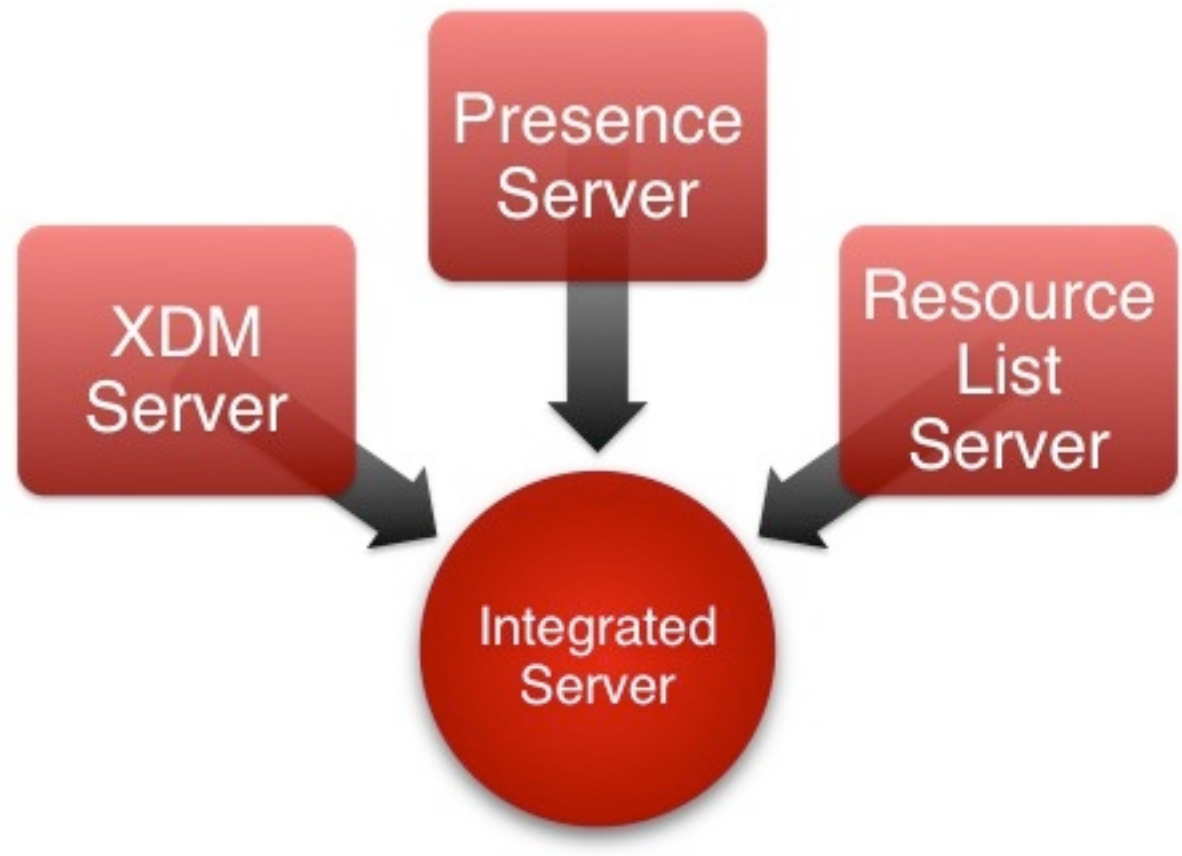
The XML Document Management Server

The **XML Document Management Server** (XDMS) is a functional element of next-generation IP communications networks is responsible for handling the management of user XML

documents stored on the network side, such as presence authorization rules, static presence information, contact and group lists (also known as “resource lists”), policy data, and many others.

The Resource List Server

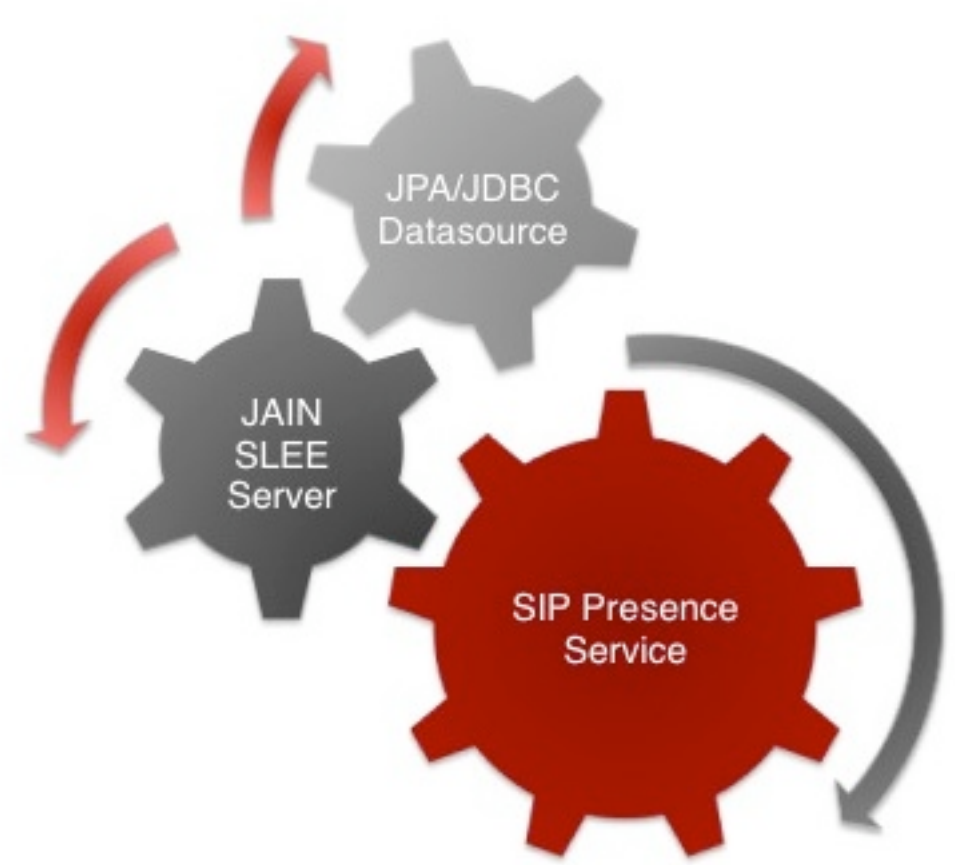
The **Resource List Server** (RLS) handles subscriptions to presence lists. It creates and manages back-end subscriptions to all resources in the presence list. The list content is retrieved from the **XDM Server**.



Mobicents SIP Presence Service Integrated server

A major advantage of the **Mobicents SIP Presence Service** is that, depending on your needs, each server can be deployed separately, or all servers can be integrated on the same host.

The **Mobicents SIP Presence Service** is built on top of **Mobicents JAIN SLEE**, a high performance and scalable Application Server and uses many additional Java Enterprise (JEE) technologies, such as Java Persistence API (JPA) to manage data.



Mobicents SIP Presence Service Integrated implementation

In addition, there are JAIN SLEE internal client interfaces available for interaction with each server, which distinguishes the **Mobicents SIP Presence Service** from other presence services.

Resources and Further Information about the Mobicents SIP Presence Service. For further information on the **Mobicents SIP Presence Service**, here is a list of additional resources:

Sources

Source Code Location [http://mobicents.googlecode.com/svn/trunk/servers/sip-presence/]

Community

Mobicents Community [http://groups.google.com/group/mobicents-public]

Installing the Mobicents SIP Presence Service

2.1. Mobicents SIP Presence Service: Installing, Configuring and Running

There are multiple binary distributions of the **Mobicents SIP Presence Service**.

Description of the different Mobicents SIP Presence Service Distributions

The **Integrated SIP Presence Service** binary distribution *with Mobicents JAIN SLEE*
: These installation instructions detail the installation, running and configuring of the *Integrated* binary **Mobicents SIP Presence Service** distribution. This distribution includes the **XDM** and **SIP Presence Servers**, the servers are pre-installed in a version of the **Mobicents JAIN SLEE**, and the Mobicents JAIN SLEE SIP11 and HTTP Servlet Resource Adaptors. Examples of JAIN SLEE applications interacting with the **Mobicents Integrated SIP Presence Service** are also included and come pre-installed.

The stand-alone **Mobicents XDM Server** binary distribution *with Mobicents JAIN SLEE*
: Users who wish to deploy the **Mobicents XML Document Server** on a different host or who do not require the **Mobicents Presence Server** should install the stand-alone **Mobicents XDM Server** binary distribution. The following installation, running and configuring instructions provide parallel instructions specific to the **Mobicents XDM Server**.

The **Mobicents SIP Presence Service** binary distribution *without* Mobicents JAIN SLEE
: Users who have already installed and set up a separate **Mobicents JAIN SLEE** installation may want to install one or more servers of the **Mobicents SIP Presence Service**.

2.1.1. Pre-Install Requirements and Prerequisites

You should ensure that a few requirements have been met before continuing with the install.

Hardware Requirements

Anything Java Itself Will Run On
: The **Mobicents SIP Presence Service** is an 100% Java application. **Mobicents SIP Presence Service** will run on the same hardware that the **Mobicents JAIN SLEE** runs on.

Software Prerequisites

JDK 5
: A working installation of the Java Development Kit (JDK) version 5 or higher is required in order to run the **Mobicents SIP Presence Service**.

Apache Ant 1.6 or later
: A working installation of the Apache Ant 1.6 or later is required in order to install the **Mobicents SIP Presence Service** release without **Mobicents JAIN SLEE**.

Mobicents JAIN SLEE 2.x
: The **Mobicents SIP Presence Service** is a set of JAIN SLEE and JEE components built on top of **Mobicents JAIN SLEE** container.

JBOSS_HOME Environment Variable
: The environment variable JBOSS_HOME, if set, must be pointing to the **JBoss AS** within **Mobicents JAIN SLEE**.

2.1.2. Downloading

You can download the latest version of the **Mobicents SIP Presence Service** distribution you need from the Mobicents `Downloads` page at *https://sourceforge.net/projects/mobicents/files/Mobicents%20SIP%20Presence%20Service/*. The latest releases are nearer the top.

If you are unsure which distribution zip file to download, refer to *Description of the different Mobicents SIP Presence Service Distributions*, and then to the following list of release binaries.

Mobicents SIP Presence Service Binary Distribution Zip Files

mobicents-sip-presence-integrated-1.0.0.BETA6.zip
: Download this zip file to obtain the **Mobicents Integrated SIP Presence Service** binary distribution, which includes the **Mobicents SIP Presence Server**, the **Mobicents XDM Server**, and the **JBoss Application Server** with **Mobicents JAIN SLEE**, well as all required JAIN SLEE Resource Adaptors.

mobicents-sip-presence-xdms-1.0.0.BETA6.zip
: Download this zip file to obtain the **Mobicents XDM Server** binary distribution, which bundles the **JBoss Application Server** with **Mobicents JAIN SLEE**.

2.1.3. Configuring (and Setting JBOSS_HOME)

2.1.3.1. Setting the JBOSS_HOME Environment Variable

The **Mobicents Platform** (**Mobicents**) is built on top of the **JBoss Application Server** (**JBoss AS**). You do not need to set the `JBOSS_HOME` environment variable to run any of the **Mobicents Platform** servers *unless* `JBOSS_HOME` is *already* set.

The best way to know for sure whether `JBOSS_HOME` was set previously or not is to perform a simple check which may save you time and frustration.

Checking to See If JBOSS_HOME is Set on Unix. At the command line, `echo` **`$JBOSS_HOME`** to see if it is currently defined in your environment:

```
~]$ echo $JBOSS_HOME
```

The **Mobicents Platform** and most Mobicents servers are built on top of the **JBoss Application Server** (**JBoss AS**). When the **Mobicents Platform** or Mobicents servers are built *from source*, then `JBOSS_HOME` *must* be set, because the Mobicents files are installed into (or "over top of" if you prefer) a clean **JBoss AS** installation, and the build process assumes that the location pointed to by the `JBOSS_HOME` environment variable at the time of building is the **JBoss AS** installation into which you want it to install the Mobicents files.

This guide does not detail building the **Mobicents Platform** or any Mobicents servers from source. It is nevertheless useful to understand the role played by **JBoss AS** and `JBOSS_HOME` in the Mobicents ecosystem.

The immediately-following section considers whether you need to set `JBOSS_HOME` at all and, if so, when. The subsequent sections detail how to set `JBOSS_HOME` on Unix and Windows

Important

Even if you fall into the category below of *not needing* to set `JBOSS_HOME`, you may want to for various reasons anyway. Also, even if you are instructed that you do *not need* to set `JBOSS_HOME`, it is good practice nonetheless to check and make sure that `JBOSS_HOME` actually *isn't* set or defined on your system for some reason. This can save you both time and frustration.

You *DO NOT NEED* to set `JBOSS_HOME` if...

- ...you have installed the **Mobicents Platform** binary distribution.
- ...you have installed a Mobicents server binary distribution *which bundles **JBoss AS**.*

You *MUST* set `JBOSS_HOME` if...

- ...you are installing the **Mobicents Platform** or any of the Mobicents servers *from source*.
- ...you are installing the **Mobicents Platform** binary distribution, or one of the Mobicents server binary distributions, which *do not* bundle **JBoss AS**.

Naturally, if you installed the **Mobicents Platform** or one of the Mobicents server binary releases which *do not* bundle **JBoss AS**, yet requires it to run, then you should *install **JBoss AS*** [http://www.jboss.org/file-access/default/members/jbossas/freezone/docs/Installation_Guide/4/html/index.html] before setting `JBOSS_HOME` or proceeding with anything else.

Setting the JBOSS_HOME Environment Variable on Unix. The `JBOSS_HOME` environment variable must point to the directory which contains all of the files for the **Mobicents Platform** or individual Mobicents server that you installed. As another hint, this topmost directory contains a `bin` subdirectory.

Setting JBOSS_HOME in your personal ~/.bashrc startup script carries the advantage of retaining effect over reboots. Each time you log in, the environment variable is sure to be set for you, as a user. On Unix, it is possible to set JBOSS_HOME as a system-wide environment variable, by defining it in /etc/bashrc, but this method is neither recommended nor detailed in these instructions.

Procedure 2.1. To Set JBOSS_HOME on Unix...

1. Open the ~/.bashrc startup script, which is a hidden file in your home directory, in a text editor, and insert the following line on its own line while substituting for the actual install location on your system:

```
export JBOSS_HOME="/home/<username>/<path>/<to>/<install_directory>"
```

2. Save and close the .bashrc startup script.

3. You should source the .bashrc script to force your change to take effect, so that JBOSS_HOME becomes set for the current session[1].

```
~]$ source ~/.bashrc
```

4. Finally, ensure that JBOSS_HOME is set in the current session, and actually points to the correct location:

Note

The command line usage below is based upon a binary installation of the **Mobicents Platform**. In this sample output, JBOSS_HOME has been set correctly to the *topmost_directory* of the **Mobicents** installation. Note that if you are installing one of the standalone **Mobicents** servers (with **JBoss AS** bundled!), then JBOSS_HOME would point to the *topmost_directory* of your server installation.

```
~]$ echo $JBOSS_HOME
/home/silas/
```

Setting the JBOSS_HOME Environment Variable on Windows. The JBOSS_HOME environment variable must point to the directory which contains all of the files for the Mobicents Platform or individual Mobicents server that you installed. As another hint, this topmost directory contains a bin subdirectory.

For information on how to set environment variables in recent versions of Windows, refer to *http://support.microsoft.com/kb/931715*.

[1] Note that any other terminals which were opened prior to your having altered .bashrc will need to source ~/.bashrc as well should they require access to JBOSS_HOME.

2.1.4. Installing

Once the requirements and prerequisites have been met, and you have downloaded the correct zip file for the binary distribution you need, you are ready to install, please follow the instructions below.

Procedure 2.2. Installing a Mobicents SIP Presence Service Binary Distribution bundled *with* Mobicents JAIN SLEE

1. Unzip the release file
2. Ensure that the environment variable JBOSS_HOME is either not set, or pointing to the `<install_directory>` directory extracted from the release file.

Procedure 2.3. Installing a Mobicents SIP Presence Service Binary Distribution *without* Mobicents JAIN SLEE

1. Unzip the release file
2. Ensure that the environment variable JBOSS_HOME is set and pointing to the JBoss AS with the **Mobicents JAIN SLEE**, where **Mobicents SIP Presence Service** components should be installed.
3. Invoke the correct Apache Ant target in build.xml script to install:

Install the Integrated SIP Presence Service:

```
~]$ ant integrated-deploy
```

Install the stand-alone XDM Server:

```
~]$ ant xdms-deploy
```

2.1.5. Running

Once installed, you can run server(s) by executing the run.sh (Unix) or run.bat (Microsoft Windows) startup scripts in the `<install_directory>/bin` directory (on Unix or Windows).

Note

By default the server(s) start and bind to 127.0.0.1 IP, to use a different hostname or IP use the -b HOST parameter when executing the startup script (e.g. run.sh -b 172.31.1.1).

2.1.6. Stopping

You can shut down the server(s) you can run server(s) by executing the shutdown.sh (Unix) or shutdown.bat (Microsoft Windows) scripts in the `<install_directory>/bin` directory (on Unix or Windows). Note that if you properly stop the server, you will see the following three lines as the last output in the Unix terminal or Command Prompt:

```
[Server] Shutdown complete
Shutdown complete
Halting VM
```

2.1.7. Uninstalling

Procedure 2.4. Uninstalling a Mobicents SIP Presence Service Binary Distribution bundled *with* Mobicents JAIN SLEE

- To uninstall the **SIP Presence Service** or **XDM Server**, simply delete the directory you decompressed the binary distribution archive into.

Procedure 2.5. Uninstalling a Mobicents SIP Presence Service Binary Distribution *without* Mobicents JAIN SLEE

- Invoke the correct Apache Ant target in build.xml script to uninstall:

Uninstall the Integrated SIP Presence Service:

```
~]$ ant integrated-undeploy
```

Uninstall the stand-alone XDM Server:

```
~]$ ant xdms-undeploy
```

2.1.8. Building from Source Project

Note

The source building process requires access to the Internet. It also requires an **SVN Client** (to sources checkout) and **Apache Maven2 2.0.9+** (for the building process) installed.

The source project can be downloaded using SVN, the checkout URL is `http://mobicents.googlecode.com/svn/tags/servers/sip-presence/mobicents-sip-presence-service-1.0.0.BETA6`

Mobicents SIP Presence Service configurations can be done through the root `pom.xml`.

To build the binaries from source, enter the release directory inside the directory used to checkout the source project and:

```
~]$ ant
```

2.1.9. Binary Releases Daily Snapshots

Everyday a binary release snapshot is built using current sources in development trunk, those are accessible from `http://hudson.jboss.org/hudson/view/Mobicents/job/MobicentsSipPresenceRelease/`

Mobicents XML Document Management Server

The **Mobicents XML Document Management Server** (**XDM Server**) is part of the **Mobicents SIP Presence Service**; it is the first free and open source implementation of an **XML Document Management Server** as defined in the *Open Mobile Alliance (OMA) XML Document Management v1.1 specification*. This functional element of next-generation IP communication networks is responsible for handling the management of user XML documents stored on the network side, such as presence authorization rules, contact and group lists (also known as resource lists), static presence information, and much more.

Important

The SIP interface partially implements the XCAP Diff Event IETF draft, version 3. Subscriptions to a single document or usage by an entire application are supported. However, these differing usages do not extend to the single-XML element or attribute value level. Regarding the notifications, the diff-processing subscription parameter, if present, is ignored, and patching of content is not available at the moment, which means that only the document etags, new and/or old, will be provided.

The **XDM Server** comprises the following functional elements:

Functional Elements of the XDM Server

Data Source
: The **XDM Server** data source is where all user XML documents are stored. Information related to the server itself is also stored in this element along with the user's provisioned data

  The data source also handles subscriptions to updates on specific documents, or complete XCAP application usages.

Aggregation Proxy
: The aggregation proxy is responsible for handling an XDM client's XCAP requests

Authentication Proxy
: The authentication proxy is responsible for authentication of the user related with each XCAP request handled.

Request Processor
: This element includes the XCAP Server logic to process an XCAP request and return a proper response, including authorization for the authenticated user.

XDM Event Subscription Control

This element, using the SIP protocol, is responsible for handling subscriptions to documents managed by the XDM. Its functions include the authentication and authorization of a subscription, attachment to update events on specific documents or application usages, and the sending of notifications when documents change.

3.1. Configuring the XDM Server

3.1.1. Configuring the XDM Server XCAP root

The Mobicents XDM Server comes pre-configured for an XCAP root of http://<hostname>:8080/mobicents, hostname being the host/IP used to start the server (127.0.0.1 by default). It is possible to change the host, the port and the last path segment:

- Rename `$JBOSS_HOME/server/<server_profile>/deploy/mobicents.war` to the desired last path segment in the XCAP root (e.g. rename to xcap-root.war for an XCAP root of http://<hostname>:8080/xcap-root). The <server_profile> is the server configuration/profile used in the underlying **JBoss AS**, by default it is *default*

- Edit `$JBOSS_HOME/server/<server_profile>/deploy/http-servlet-ra-DU-*.jar/META-INF/deploy-config.xml`. Uncomment and set custom servlet name again to the desired last path segment in the XCAP root.

- Edit the properties in file `$JBOSS_HOME/server/<server_profile>/deploy/mobicents-xdms/3-beans/configuration/xdms/META-INF/jboss-beans.xml`. Note that the xcapRoot has a leading /. Also note that for the **Integrated Server** the path segment mobicents-xdms is mobicents-sip-presence.

  This configuration part can also be done through JMX, using the MBean named *org.mobicents.slee:sippresence=XDMServerConfiguration*. The configuratian changes through JMX are not persistent.

3.1.2. Other configurations in the XDM Server XCAP Interface

There are other configurable features related with the XCAP Interface:

- Edit the properties in file `$JBOSS_HOME/server/<server_profile>/deploy/mobicents-xdms/3-beans/configuration/xdms/META-INF/jboss-beans.xml`. Note that for the **Integrated Server** the path segment mobicents-xdms is mobicents-sip-presence.

  This configuration part can also be done through JMX, using the MBean named *org.mobicents.slee:sippresence=XDMServerConfiguration*. The configuratian changes through JMX are not persistent.

3.1.3. Configuring the XDM Server XCAP Diff SIP Subscription Interface

The Mobicents XDM Server SIP Interface can be configured regarding several features, such as subscription timers:

- Edit the properties in file `$JBOSS_HOME/server/<server_profile>/deploy/mobicents-xdms/3-beans/configuration/sip-event/subscription/META-INF/jboss-beans.xml`. Note that for the **Integrated Server** the path segment mobicents-xdms is mobicents-sip-presence.

  This configuration part can also be done through JMX, using the MBean named *org.mobicents.slee:sippresence=SipEventSubscriptionControl*. The configuratian changes through JMX are not persistent.

3.1.4. XDM Server User Profile Provisioning

XCAP interface is public, used by users to manage their information such as buddy list, presence authorization rules, etc. thus it needs to enforce user authentication. To do the user authentication, the server relies on the User Profile Enabler managed data, such as the users passwords, and this information must be provisioned, this can be done in two ways, both requiring the server to be running:

User Provisioning through an JMX Client
: Users can be added/removed through the MBean named *org.mobicents.slee:userprofile=UserProfileControl*

User Provisioning through the **JBoss AS** default datasource.
: Users can be added/removed through adding/removing rows of the table named *MOBICENTS_SLEE_ENABLER_USERPROFILES*.

3.1.4.1. User Asserted IDs

The XDM Server allows the usage of asserted user IDs, though the usage of `X-3GPP-Asserted-Identity` or `X-XCAP-Asserted-Identity` header in the XCAP request. If the XDM Server is directly exposed to public this feature should be disabled, through the configuration of the XCAP Interface.

3.1.4.2. Local XCAP Requests

By default local (same host) XCAP requests will go around user authentication, this can also be disabled through configuration of the XCAP interface.

3.1.5. XCAP Application Usages

What is an XCAP Application Usage?

"Each XCAP resource on a server is associated with an application. In order for an application to use those resources, application specific conventions must be specified. Those conventions include the XML schema that defines the structure and constraints of the data, well-known URIs to bootstrap access to the data, and so on. All of those application specific conventions are defined by the application usage." RFC 4825

Each XCAP Application Usage defines:

Application Unique ID
: The AUID used in XCAP URIs to point to a specific App Usage, e.g. `resource-lists` in http://xdms.mobicents.org:8080/xcap-root/resource-lists/users/sip:user@mobicents.org/index

Default Document Namespace
: Defines the namespace of elements/attributes without prefix in XCAP URIs, usually it matches the default namespace of the XML Schema for documents of the App Usage,e.g. in http://xdms.mobicents.org/xcap-root/pres-rules/users/sip:eduardo@mobicents.org/index/~~/watcherinfo/watcher-list/watcher[@id="8ajksjda7s"], selection is made on `watcher` elements with the pres-rules default document namespace.

MIME Type
: MIME Type used when exchanging XML content.

XML Schema and Data Constraints
: The XML Schema to validate documents; Data constraints, which are impossible to validate with XML Schema, e.g. one element value must be a ISO country name (2 char) that belongs to Europe.

Data Semantics
: Semantic definition on documents content, used by applications filling data, not validated by servers.

Naming Conventions
: What is the document name for each user? Are there global documents under a specific name? XCAP Clients usually forget to follow these!

Resource Interdependencies
: One request may update other documents as well, e.g. global/index document in rls-services, a composition of all users/*/index `service` elements.

Authorization Policies
: What each user can read or write?

3.1.5.1. XCAP Application Usages Deployed

The **Mobicents XDM Server** includes the following XCAP application usages:

- *IETF Presence Rules (RFC 5025)* [http://tools.ietf.org/html/rfc5025]
- *OMA Presence Rules (OMA Presence Simple v1.1)* [http://www.openmobilealliance.org/Technical/release_program/Presence_simple_v1_1.aspx]

- *IETF Resource Lists* [http://tools.ietf.org/html/rfc4826]
- *OMA Group Usage List (OMA XDM v1.1)* [http://www.openmobilealliance.org/Technical/release_program/xdm_v1_1.aspx]
- *IETF RLS Services (RFC 4826)* [http://tools.ietf.org/html/rfc4826]
- *OMA User Profile (OMA XDM v2.0)* [http://www.openmobilealliance.org/Technical/release_program/xdm_v2_0.aspx]
- *OMA Locked User Profile (OMA XDM v2.0)* [http://www.openmobilealliance.org/Technical/release_program/xdm_v2_0.aspx]
- *IETF XCAP-CAPS (RFC 4825)* [http://tools.ietf.org/html/rfc4825]
- *OMA XCAP Directory (OMA XDM v1.1)* [http://www.openmobilealliance.org/Technical/release_program/xdm_v1_1.aspx]

3.1.5.2. Developing XCAP Application Usages

The **Mobicents XDM Server** XCAP Application Usages are implemented with a few simple Java classes and some meta data, it is very easy to develop additional ones.

3.1.5.2.1. The AppUsage Class

Each Application Usage is represented by a Java class extending the abstract `org.mobicents.xdm.server.appusage.AppUsage` class:

```
package org.mobicents.xcap.server.slee.appusage.presrules;

// ...

public class PresRulesAppUsage extends AppUsage {

    public static final String ID = "pres-rules";
   public static final String DEFAULT_DOC_NAMESPACE = "urn:ietf:params:xml:ns:pres-rules";
    public static final String MIMETYPE = "application/auth-policy+xml";
    private static final String AUTH_ONLY_DOCUMENT_NAME = #index#;

    public PresRulesAppUsage(Validator schemaValidator) {
        super(ID,DEFAULT_DOC_NAMESPACE,MIMETYPE,schemaValidator,AUTH_ONLY_DOCUMENT_NAME);
    }
}
```

Methods for data constraints and resource interdependencies can be overriden:

```
public void checkConstraintsOnPut();
public void checkConstraintsOnDelete();
public void processResourceInterdependenciesOnPutDocument();
public void processResourceInterdependenciesOnDeleteElement();
//...
```

Important

RLSServicesAppUsage and ResourceListsAppUsage are good examples on how to implement those methods.

Multiple constructors exposed to provide your App Usage XML Schemas Validators and/or Authorization Policies:

```
public AppUsage(String auid, String defaultDocumentNamespace, String mimetype,
   Validator schemaValidator, String authorizedUserDocumentName);

public AppUsage(String auid, String defaultDocumentNamespace, String mimetype,
   Validator schemaValidator, AuthorizationPolicy authorizationPolicy);

public AppUsage(String auid, String defaultDocumentNamespace, String mimetype,
   Validator schemaValidator, Validator uniquenessSchemaValidator,
   String authorizedUserDocumentName);

public AppUsage(String auid, String defaultDocumentNamespace, String mimetype,
   Validator schemaValidator, Validator uniquenessSchemaValidator,
   AuthorizationPolicy authorizationPolicy);
```

Important

Default Authorization Policy if custom is not provided, an user can read/write his/her own documents, with the specified document name.

3.1.5.2.2. The AppUsageFactory Class

An implementation of an object factory is required, which should extend class named `org.mobicents.xdm.server.appusage.AppUsageFactory`:

```
package org.mobicents.xcap.server.slee.appusage.presrules;

// ...

public class PresRulesAppUsageFactory implements AppUsageFactory {

   private Schema schema = null;

   public PresRulesAppUsageFactory(Schema schema) {
      this.schema = schema;
   }

   public AppUsage getAppUsageInstance() {
      return new PresRulesAppUsage(schema.newValidator());
   }

   public String getAppUsageId() {
      return PresRulesAppUsage.ID;
   }

   public AppUsageDataSourceInterceptor getDataSourceInterceptor() {
      return null;
   }
}
```

The factory is used to maintain a cache/pool of your app usage objects, since XML Schema Validator are expensive objects to create.

The factory can also provide a DataSource Interceptor, which will be used to generate a document on request (for instance the oma-xcap-directory generates the user document for each request).

3.1.5.2.3. The AppUsageDeployer Class And XML Descriptor

A deployer to load/unload the App Usage into the XDM Server, it should extend class named `org.mobicents.xdm.server.appusage.AppUsageDeployer`:

```
package org.mobicents.xcap.server.slee.appusage.presrules;

// ...

public class PresRulesAppUsageDeployer extends AppUsageDeployer {

   @Override
   public AppUsageFactory createAppUsageFactory(Schema schema) {
      return new PresRulesAppUsageFactory(schema);
   }

   @Override
   public String getSchemaRootNamespace() {
      return PresRulesAppUsage.DEFAULT_DOC_NAMESPACE;
   }

}
```

Multiple XML schema files may be combined, starting point is the namespace returned by `getSchemaRootNamepsace()`, which not always is the same as the app usage's default doc namespace.

The deployer is actually a JBoss Microcontainer Bean, and a `jboss-beans.xml` file is needed in the META-INF directory of the app usage jar:

```
<?xml version="1.0" encoding="UTF-8"?>

<deployment xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   xmlns="urn:jboss:bean-deployer:2.0">

   <!-- Registers the APP USAGE DEPLOYER AS JBOSS MICROCONTAINER BEAN -->
```

```
e.Deployer" class="org.mobicents.xcap.server.slee.appusage.presrules.PresRulesAppUsageDeployer">

    <!-- this app usage depends on app usage management only -->
    <depends>Mobicents.XDMS.AppUsageManagement</depends>
  </bean>

</deployment>
```

A unique bean “name” is need, and of course the “class” name of the Deployer, nothing else needs to be changed.

3.1.5.2.4. Packaging and Deploying the XCAP Application Usage

The Application Usage classes and metadata should be packed in a jar, with a structure similar to:

```
example-appusage.jar
|-META-INF
|---jboss-beans.xml (jboss mc bean descriptor)
|-org
|---mobicents
|------xdm
|---------server
|------------appusage
|--------------example
|------------------ExampleAppUsage (app usage class)
|------------------ExampleAppUsageFactory (factory class)
|------------------ExampleAppUsageDeployer (deployer class)
|------------------ExampleAuthorizationPolicy (optional auth policy class)
```

To deploy simply drop the jar in `$JBOSS_HOME/server/default/deploy/mobicents-xdms(or mobicents-sip-presence)/3.beans/appusages`. To undeploy simply delete the jar.

What about the XSD file(s)? Simply copy to the `xsd` directory inside the `appusages` directory, a few limitations on multiple XSD files combination:

- References between namespaces on different files must be done through `import` element.
- A namespace can only be defined within a single xsd file.

3.1.5.2.5. Submiting XCAP Application Usages to Mobicents

Contribution of additional App Usages are welcome, but a few rules apply:

Implements a standard app usage, defined by IETF, OMA or other standard organization.

A document is provided defining the app usage. This document may be the one defined by the standard organization.

The app usage jar and XSDs are ready to deploy, i.e., if the app usage was already tested and works.

Includes JUnit basic tests to validate put/get and delete of a document, similar to the ones included in RLS Services app usage source code, see the ones in the SVN, at `trunk/servers/sip-presence/xdm/server/appusages/rls-services/tests`. If the app usage defines data constraints or resource interdependencies then these should be validated by tests too.

Mobicents SIP Presence Server

The **Mobicents SIP Presence Server** is a free and open source implementation of a SIP Presence Server, as defined by the Internet Engineering Task Force (IETF), the Open Mobile Alliance (OMA), the 3rd Generation Partnership Project (3GPP) and the European Telecommunications Standards Institute (ETSI).

The **SIP Presence Server** is an entity that accepts, stores and distributes SIP presence information.

4.1. Functional Architecture of the SIP Presence Server

The **SIP Presence Server** is comprised of the following functional elements:

Presence Publication Control
: This functional element manages the publication of presence events, which includes not only the handling of new publications, but also the refreshing, modification or removal of, already-published information.

  Because the presence resource, which is also called a *presentity*, can have multiple publications simultaneously, such as some state published by a user agent or device, and some location data published by a Presence Network Agent (on behalf of the presentity), this element is also responsible for composing all of the different publications for the same resource.

  In some presence networks, it may be of interest to allow resources to have a static presence state which is stored in the XDM Server. In cases like these, Presence Publication Control may need to interface with the **XDM Server** to retrieve and subscribe to (learn about changes to) that information, and use it when composing the final presence information document.

Presence Subscription Control
: This functional element handles subscriptions to presence events or to the list of subscribers (watchers), for any specific resource. It is, of course, responsible for emitting notifications related to those subscriptions.

  Presence authorization rules, which define if a subscription is allowed or rejected and, if allowed, define which transformations to the original presence events are needed, are stored on the **XDM Server** by the user. Thus, Presence Subscription Control needs to retrieve and subscribe to that information.

Presence Rules Cache
: This element is responsible for interfacing with the **XDM Server** that manages the user's XML presence rules documents. It is responsible for providing the presence rules to the Presence Subscription Control, which are used to authorize the subscriptions it handles.

The implementation architecture of the SIP Presence Server also contains client-side components:

Presence Client SBB

The `PresenceClientSBB` is the interface to a JAIN SLEE SBB intended to be used as a client for the **Mobicents SIP Presence Server** (and other servers compliant with same standards), in JAIN SLEE child relations.

Two implementations of this interface are provided: the `InternalPresenceClientSBB` that is used with applications running in the **Mobicents SIP Presence Server** JAIN SLEE container, and the `ExternalPresenceClientSBB`, used with applications running in a different JAIN SLEE container than the **Mobicents SIP Presence Server**.

4.2. Configuring The SIP Presence Server

Several features of the SIP Presence Server are configurable, through XML files or JMX.:

4.2.1. Configuring the Abstract SIP Event Publication Interface

Edit the properties in file `$JBOSS_HOME/server/<server_profile>/deploy/mobicents-sip-presence/3-beans/configuration/sip-event/publication/META-INF/jboss-beans.xml`.

This configuration part can also be done through JMX, using the MBean named *org.mobicents.slee:sippresence=SipEventPublicationControl*. The configuratian changes through JMX are not persistent.

4.2.2. Configuring the Abstract SIP Event Subscription Interface

Edit the properties in file `$JBOSS_HOME/server/<server_profile>/deploy/mobicents-sip-presence/3-beans/configuration/sip-event/subscription/META-INF/jboss-beans.xml`.

This configuration part can also be done through JMX, using the MBean named *org.mobicents.slee:sippresence=SipEventSubscriptionControl*. The configuratian changes through JMX are not persistent.

4.2.3. Configuring the Concrete SIP Event Interfaces

Edit the properties in file `$JBOSS_HOME/server/<server_profile>/deploy/mobicents-sip-presence/3-beans/configuration/sip-presence/META-INF/jboss-beans.xml`.

This configuration part can also be done through JMX, using the MBean named *org.mobicents.slee:sippresence=SipPresenceServerManagement*. The configuratian changes through JMX are not persistent.

Mobicents Resource List Server

The **Mobicents Resource List Server**, or simply RLS, is the functional element which handles subscriptions to resources lists. A resource list is defined as a list of any kind of SIP presence entities, be it single presentities or other resource lists.

The RLS is specified by IETF RFC 5367. It is a XDM Client, which watches all RLS Services documents (each define a list of presence entities) stored in the related XDM Server, and processes SIP presence subscriptions to each RLS Service state, that is, the state for all presence entities deferred by from the service. When handling a subscription to a RLS Service, the RLS creates and manages (possibly virtual) subscriptions to each presence entity on the Presence Server, and notifies the subscriber for entity state change.

RLS Services are typically used to store the list of entities which the subscriber watch, and the list of entities which are allowed to subscribe its state.

Mobicents Resource List Server extends the Presence Server, it introduces an additional functional element, the RLS Services Cache. This element is responsible for managing the flat list of entities pointed by each RLS Service, and for that it subscribes changes in referred docs (RLS Services and related Resource Lists). Each time an RLS Service changes the cache notifies the related subscriptions, to ensure the subscriber is always subscribing the correct list of presence entities.

Important

The Mobicents RLS is currently limited to RLS Services stored in the integrated XDM Server, and such services should not refer other XDM Servers, otherwise the RLS will set the state for the related service as Bad Gateway.

5.1. Disabling the Resource List Server

It is possible to disable the RLS function from the integrated server, that is achieved by configuring the Presence Server's Subscription Interface. See *Section 4.2.2, "Configuring the Abstract SIP Event Subscription Interface"* for additional information.

Client JAIN SLEE Applications

The **Mobicents SIP Presence Service** is built on top of **Mobicents JAIN SLEE**, which means JAIN SLEE applications can be deployed and run in same JVM as the servers. Better yet, there are XDM and SIP Presence client enablers which can be integrated in such JAIN SLEE applications, allowing an easy interaction with the platform servers.

6.1. XDM Client JAIN SLEE Enabler

The **Mobicents SIP Presence** exposes a JAIN SLEE enabler for applications which want to interact as clients of the XDM Server. The enabler is an extension of the XDM Client Enabler which exists in Mobicents JAIN SLEE, the only difference is that upon requests targeting the local and integrated XDM Server, the enabler does not uses XCAP or SIP network protocols, thus providing better performance and less overhead to network communications.

Please refer to the bundled JAIN SLEE documentation for complete details about how to integrate the enabler, the only difference to note in this document, is the configuration of the client JAIN SLEE application SBB's XML Descriptor. The extended XDM Client Enabler SBB has the following ID:

```
<sbb-name>InternalXDMClientControlSbb</sbb-name>
<sbb-vendor>org.mobicents</sbb-vendor>
<sbb-version>1.0</sbb-version>
```

In concrete this means that when integrating the enabler, the Parent's (the client application) Sbb XML Descriptor will refer the ID above instead of:

```
<sbb-name>XDMClientChildSbb</sbb-name>
<sbb-vendor>org.mobicents</sbb-vendor>
<sbb-version>1.0</sbb-version>
```

6.2. The Mobicents SIP Event Publication Client Enabler

The **Mobicents SIP Event Publication** exposes a JAIN SLEE enabler for applications which want to interact as clients of a SIP Event Publication Server. The enabler does not uses SIP network protocols, thus providing better performance and less overhead to network communications.

The Enabler consists in an SBB which can be used in child relations, with a simple synchronous interface.

6.2.1. Integrating the Mobicents SIP Event Publication 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.

6.2.1.1. The Parent's SbbLocalObject Interface

The Mobicents SIP Event Publication Client Enabler Sbb does not provides asynchronous callbacks to the Parent's Sbb at the moment, that is, all operations invoked in the child sbb will return a response. Thus the Parent does not needs to implement a specific interface.

6.2.1.2. The Parent's Sbb Abstract Class

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 `javax.slee.ChildRelation` object, which is used to create or access specific Child Sbbs. This method should be:

```
public abstract ChildRelation getSIPEventPublicationClientChildRelation();
```

6.2.1.3. 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>PublicationControlSbb</sbb-name>
   <sbb-vendor>org.mobicents</sbb-vendor>
   <sbb-version>1.0</sbb-version>
   <sbb-alias>sipEventPublicationClientChildSbb</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>sipEventPublicationClientChildSbb</sbb-alias-ref>
    <get-child-relation-method-name>getSIPEventPublicationClientChildRelation</get-child-relation-method-name>
   <default-priority>0</default-priority>
</get-child-relation-method>
```

6.2.2. Using the Mobicents SIP Event Publication Client Enabler

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

6.2.2.1. The Child's SbbLocalObject Interface

The Mobicents SIP Event Publication Client Enabler Sbb, the Child Sbb, implements the `org.mobicents.slee.sipevent.server.publication.PublicationClientControlSbbLocalObject`, which extends the `javax.slee.SbbLocalObject` and `org.mobicents.slee.sipevent.server.publication.PublicationClientControl` interfaces, the latter declares the methods which can be used to interact with the PS and/or RLS:

```
package org.mobicents.slee.sipevent.server.publication;
```

```
public interface PublicationClientControl {

   public Result newPublication(String entity, String eventPackage,
         String document, String contentType, String contentSubType,
         int expires);

   public Result refreshPublication(String entity, String eventPackage,
         String eTag, int expires);

   public Result modifyPublication(String entity, String eventPackage,
         String eTag, String document, String contentType,
         String contentSubType, int expires);

   public int removePublication(String entity, String eventPackage, String eTag);

}
```

The `newPublication(String, String, String, String, String, int)` method:
Requests a new publication, for the specified Entity and SIP Event Package.

The `refreshPublication(String, String, String, int)` method:
Requests a publication refresh, for the specified Entity, SIP Event Package and ETag.

The `modifyPublication(String, String, String, String, String, String, int)` method:
Requests a publication modification, for the specified Entity, SIP Event Package and ETag.

The `removePublicationOk(String, String, String)` method:
Requests a publication removal, for the specified Entity, SIP Event Package and ETag.

6.2.2.2. 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 PublicationClientControl getPresenceClientChildSbb() {
      final ChildRelation childRelation = getSIPEventPublicationClientChildRelation();
      if (childRelation.isEmpty()) {
         try {
            // creates new instance
```

```
            return (PublicationClientControl) childRelation.create();
        } catch (Exception e) {
            tracer.severe("Failed to create child sbb", e);
            return null;
        }
    }
    else {
        // reuse the existent one
        return (PublicationClientControl) childRelation.iterator().next();
    }
}
```

6.3. The Mobicents SIP Event Subscription Client Enabler

The **Mobicents SIP Event Publication** exposes a JAIN SLEE enabler for applications which want to interact as clients of a SIP Event Subscription Server, such as a PS or RLS. The enabler does not uses SIP network protocols, thus providing better performance and less overhead to network communications.

The Enabler consists in an SBB which can be used in child relations, with a simple asynchronous interface.

6.3.1. Integrating the Mobicents SIP Event 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.

6.3.1.1. The Parent's SbbLocalObject Interface

The Mobicents SIP Event Subscription 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, 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.mobicents.slee.sipevent.server.subscription.SubscriptionClientControlParentSbbLocalObject`, which extends the `javax.slee.SbbLocalObject` and `org.mobicents.slee.sipevent.server.subscription.SubscriptionClientControlParent` interfaces, the latter declares the callbacks which must be implemented in the Parent's Sbb Abstract Class:

```
package org.mobicents.slee.sipevent.server.subscription;

import org.mobicents.slee.sipevent.server.subscription.data.Subscription;

public interface SubscriptionClientControlParent {

    public void subscribeOk(String subscriber, String notifier,
            String eventPackage, String subscriptionId, int expires,
            int responseCode);

    public void resubscribeOk(String subscriber, String notifier,
            String eventPackage, String subscriptionId, int expires);

    public void unsubscribeOk(String subscriber, String notifier,
            String eventPackage, String subscriptionId);

    public void subscribeError(String subscriber, String notifier,
            String eventPackage, String subscriptionId, int error);

    public void resubscribeError(String subscriber, String notifier,
            String eventPackage, String subscriptionId, int error);

    public void unsubscribeError(String subscriber, String notifier,
            String eventPackage, String subscriptionId, int error);

    public void notifyEvent(String subscriber, String notifier,
            String eventPackage, String subscriptionId,
            Subscription.Event terminationReason, Subscription.Status status,
            String content, String contentType, String contentSubtype);

}
```

The `subscribeOk(String, String, String, String, int, int)` method:
Callback from the Enabler indicating that the new subscription request succeed.

The `resubscribeOk(String, String, String, String, int)` method:
Callback from the Enabler indicating that the refresh subscription request succeed.

The `unsubscribeOk(String, String, String, String)` method:
Callback from the Enabler indicating that the remove subscription request succeed.

The `subscribeError(String, String, String, String, int)` method:
Callback from the Enabler indicating that the new subscription request failed.

The `resubscribeError(String, String, String, String, int)` method:
Callback from the Enabler indicating that the refresh subscription request failed.

The `unsubscribeError(String, String, String, String, int)` method:
Callback from the Enabler indicating that the remove subscription request failed.

The `notifyEvent(String, String, String, String, Subscription.Event, Subscription.Status, String, String, String)` method:
Callback from the Enabler notifying an event related with notifier state change.

6.3.1.2. The Parent's Sbb Abstract Class

The Parent Sbb Abstract Class must implement the callbacks on it's SbbLocalObject, that is, must implement the `org.mobicents.slee.sipevent.server.subscription.SubscriptionClientControlParent` 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 `javax.slee.ChildRelation` object, which is used to create or access specific Child Sbbs. This method should be:

```
public abstract ChildRelation getSIPEventSubscriptionClientChildRelation();
```

6.3.1.3. 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>SubscriptionControlSbb</sbb-name>
   <sbb-vendor>org.mobicents</sbb-vendor>
   <sbb-version>1.0</sbb-version>
   <sbb-alias>sipEventSubscriptionClientChildSbb</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>sipEventSubscriptionClientChildSbb</sbb-alias-ref>
  <get-child-relation-method-name>getSIPEventSubscriptionClientChildRelation</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>...</sbb-local-interface-name>
</sbb-local-interface>
```

6.3.2. Using the Mobicents SIP Event Subscription Client Enabler

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

6.3.2.1. The Child's SbbLocalObject Interface

The Mobicents SIP Event Subscription Client Enabler Sbb, the Child Sbb, implements the `org.mobicents.slee.sipevent.server.subscription.SubscriptionClientControlSbbLocalObject`, which extends the `javax.slee.SbbLocalObject` and `org.mobicents.slee.sipevent.server.subscription.SubscriptionClientControlSbbLocalObject` interfaces, the latter declares the methods which can be used to interact with the SIP Event Subscription Server:

```
package org.mobicents.slee.sipevent.server.subscription;

public interface SubscriptionClientControl {

   public void setParentSbb(
         SubscriptionClientControlParentSbbLocalObject sbbLocalObject);

   public void subscribe(String subscriber, String subscriberdisplayName,
         String notifier, String eventPackage, String subscriptionId,
         int expires, String content, String contentType,
         String contentSubtype);

   public void resubscribe(String subscriber, String notifier,
         String eventPackage, String subscriptionId, int expires);

   public void unsubscribe(String subscriber, String notifier,
         String eventPackage, String subscriptionId);

}
```

The `setParentSbb(SubscriptionClientControlParentSbbLocalObject)` 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 `subscribe(String, String, String, String, String, int, String, String, String)` method:
Requests a new subscription.

The `resubscribe(String, String, String, String, int)` method:
Requests a subscription refresh.

The `unsubscribe(String, String, String, String)` method:
Requests a subscription removal.

6.3.2.2. 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 SubscriptionClientControl getSIPEventSubscriptionClientChildSbb() {
    final ChildRelation childRelation = getSIPEventSubscriptionClientChildRelation();
    if (childRelation.isEmpty()) {
        try {
            // creates new instance
            SubscriptionClientControl sbb = (SubscriptionClientControl) childRelation.create();
            // passes the parent sbb local object to the child
            sbb.setParentSbb((SubscriptionClientControlParentSbbLocalObject) sbbContext.getSbbLocalObject());
            return sbb;
        } catch (Exception e) {
            tracer.severe("Failed to create child sbb", e);
            return null;
        }
    }
    else {
        // reuse the existent one
        return (SubscriptionClientControl) childRelation.iterator().next();
    }
}
```

6.4. The Mobicents Presence Client Enabler

The **Mobicents SIP Presence** exposes a JAIN SLEE enabler for applications which want to interact as clients of the Presence (PS) or Resource List Server (RLS). The enabler does not uses SIP network protocols, thus providing better performance and less overhead to network communications.

The Enabler consists in an SBB which can be used in child relations, with a simple asynchronous interface.

Important

The Presence Client Enabler reuses the SIP Event Subscription and Publication Client Enablers, and for the best performance it is best to use these instead. This is due to less internal state needed.

6.4.1. Integrating the Mobicents Presence 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.

6.4.1.1. The Parent's SbbLocalObject Interface

The Mobicents Presence 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, 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.mobicents.slee.sippresence.client.PresenceClientControlParentSbbLocalObject`, which extends the `javax.slee.SbbLocalObject` and `org.mobicents.slee.sippresence.client.PresenceClientControlParent` interfaces, the latter declares the callbacks which must be implemented in the Parent's Sbb Abstract Class:

```
package org.mobicents.slee.sippresence.client;

import org.mobicents.slee.sipevent.server.subscription.data.Subscription;

public interface PresenceClientControlParent {

    public void newPublicationOk(Object requestId, String eTag, int expires);

    public void refreshPublicationOk(Object requestId, String eTag, int expires);
```

```
    public void modifyPublicationOk(Object requestId, String eTag, int expires);

    public void removePublicationOk(Object requestId);

    public void newPublicationError(Object requestId, int error);

    public void refreshPublicationError(Object requestId, int error);

    public void modifyPublicationError(Object requestId, int error);

    public void removePublicationError(Object requestId, int error);

    public void newSubscriptionOk(String subscriber, String notifier,
            String eventPackage, String subscriptionId, int expires,
            int responseCode);

    public void refreshSubscriptionOk(String subscriber, String notifier,
            String eventPackage, String subscriptionId, int expires);

    public void removeSubscriptionOk(String subscriber, String notifier,
            String eventPackage, String subscriptionId);

    public void newSubscriptionError(String subscriber, String notifier,
            String eventPackage, String subscriptionId, int error);

    public void refreshSubscriptionError(String subscriber, String notifier,
            String eventPackage, String subscriptionId, int error);


    public void removeSubscriptionError(String subscriber, String notifier,
            String eventPackage, String subscriptionId, int error);

    public void notifyEvent(String subscriber, String notifier,
            String eventPackage, String subscriptionId,
            Subscription.Event terminationReason, Subscription.Status status,
            String content, String contentType, String contentSubtype);

}
```

The `newPublicationOk(Object, String, int)` method:

Callback from the Enabler indicating that the new publication request succeed.

The `refreshPublicationOk(Object, String, int)` method:
Callback from the Enabler indicating that the refresh publication request succeed.

The `modifyPublicationOk(Object, String, int)` method:
Callback from the Enabler indicating that the modify publication request succeed.

The `removePublicationOk(Object)` method:
Callback from the Enabler indicating that the remove publication request succeed.

The `newPublicationError(Object, int)` method:
Callback from the Enabler indicating that the new publication request failed.

The `refreshPublicationError(Object, int)` method:
Callback from the Enabler indicating that the refresh publication request failed.

The `modifyPublicationError(Object, int)` method:
Callback from the Enabler indicating that the modify publication request failed.

The `removePublicationError(Object, int)` method:
Callback from the Enabler indicating that the remove publication request failed.

The `newSubscriptionOk(String, String, String, String, int, int)` method:
Callback from the Enabler indicating that the new subscription request succeed.

The `refreshSubscriptionOk(String, String, String, String, int)` method:
Callback from the Enabler indicating that the refresh subscription request succeed.

The `removeSubscriptionOk(String, String, String, String)` method:
Callback from the Enabler indicating that the remove subscription request succeed.

The `newSubscriptionError(String, String, String, String, int)` method:
Callback from the Enabler indicating that the new subscription request failed.

The `refreshSubscriptionError(String, String, String, String, int)` method:
Callback from the Enabler indicating that the refresh subscription request failed.

The `removeSubscriptionError(String, String, String, String, int)` method:
Callback from the Enabler indicating that the remove subscription request failed.

The `notifyEvent(String, String, String, String, Subscription.Event, Subscription.Status, String, String, String)` method:
Callback from the Enabler notifying an event related with notifier state change.

6.4.1.2. The Parent's Sbb Abstract Class

The Parent Sbb Abstract Class must implement the callbacks on it's SbbLocalObject, that is, must implement the `org.mobicents.slee.sippresence.client.PresenceClientControlParent` 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 `javax.slee.ChildRelation` object, which is used to create or access specific Child Sbbs. This method should be:

```
public abstract ChildRelation getPresenceClientChildRelation();
```

6.4.1.3. 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>InternalPresenceClientControlSbb</sbb-name>
   <sbb-vendor>org.mobicents</sbb-vendor>
   <sbb-version>1.0</sbb-version>
   <sbb-alias>presenceClientChildSbb</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>presenceClientChildSbb</sbb-alias-ref>
      <get-child-relation-method-name>getPresenceClientChildRelation</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>...</sbb-local-interface-name>
</sbb-local-interface>
```

6.4.2. Using the Mobicents Presence Client Enabler

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

6.4.2.1. The Child's SbbLocalObject Interface

The Mobicents Presence Client Enabler Sbb, the Child Sbb, implements the `org.mobicents.slee.sippresence.client.PresenceClientControlSbbLocalObject`, which extends the `javax.slee.SbbLocalObject` and `org.mobicents.slee.sippresence.client.PresenceClientControl` interfaces, the latter declares the methods which can be used to interact with the PS and/or RLS:

```
package org.mobicents.slee.sippresence.client;

public interface PresenceClientControl {

   public void setParentSbb(PresenceClientControlParentSbbLocalObject parentSbb);

   public void newPublication(Object requestId, String entity,
         String document, String contentType, String contentSubType,
         int expires);

   public void refreshPublication(Object requestId, String entity,
         String eTag, int expires);

   public void modifyPublication(Object requestId, String entity, String eTag,
         String document, String contentType, String contentSubType,
         int expires);

   public void removePublication(Object requestId, String entity, String eTag);
```

```
    public void newSubscription(String subscriber,
            String subscriberdisplayName, String notifier, String eventPackage,
            String subscriptionId, int expires);

    public void refreshSubscription(String subscriber, String notifier,
            String eventPackage, String subscriptionId, int expires);

    public void removeSubscription(String subscriber, String notifier,
            String eventPackage, String subscriptionId);

}
```

The `setParentSbb(PresenceClientControlParentSbbLocalObject)` 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 `newPublication(Object, String, String, String, String, int)` method:
Requests a new publication, for the specified Entity. The object argument is an ID that identifies the publication, and which will be provided in the response callback.

The `refreshPublication(Object, String, String, int)` method:
Requests a publication refresh, for the specified Entity and ETag. The object argument is an ID that identifies the publication, and which will be provided in the response callback.

The `modifyPublication(Object, String, String, String, String, String, int)` method:
Requests a publication modification, for the specified Entity and ETag. The object argument is an ID that identifies the publication, and which will be provided in the response callback.

The `removePublicationOk(Object, String, String)` method:
Requests a publication removal, for the specified Entity and ETag. The object argument is an ID that identifies the publication, and which will be provided in the response callback.

The `newSubscription(String, String, String, String, String, int)` method:
Requests a new subscription.

The `refreshSubscription(String, String, String, String, int)` method:
Requests a subscription refresh.

The `removeSubscription(String, String, String, String)` method:
Requests a subscription removal.

6.4.2.2. 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 PresenceClientControlSbbLocalObject getPresenceClientChildSbb() {
    final ChildRelation childRelation = getPresenceClientChildRelation();
    if (childRelation.isEmpty()) {
        try {
            // creates new instance
            PresenceClientControlSbbLocalObject sbb = (PresenceClientControlSbbLocalObject) childRelation.create
            // passes the parent sbb local object to the child
            sbb.setParentSbb((PresenceClientControlSbbLocalObject) sbbContext.getSbbLocalObject());
            return sbb;
        } catch (Exception e) {
            tracer.severe("Failed to create child sbb", e);
            return null;
        }
    }
    else {
        // reuse the existent one
        return (PresenceClientControlSbbLocalObject) 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.

6.5. Client Application Examples

TODO

Logging, Traces and Alarms

7.1. Log4j Logging Service

In Mobicents SIP Presence `Apache log4j` is used for logging. If you are not familiar with the log4j package and would like to use it in your applications, you can read more about it at the *Jakarta web site* [http://jakarta.apache.org/log4j/].

Logging is controlled from a central `conf/jboss-log4j.xml` file, in each server configuration profile. This file defines a set of appenders specifying the log files, what categories of messages should go there, the message format and the level of filtering. By default, in produces output to both the console and a log file (`log/server.log`).

There are 6 basic log levels used: TRACE, DEBUG, INFO, WARN, ERROR and FATAL.

Logging is organized in categories and appenders. Appenders control destination of log entries. Different appenders differ in configuration, however each supports threshold. Threshold filters log entries based on their level. Threshold set to WARN will allow log entry to pass into appender if its level is WARN, ERROR or FATAL, other entries will be discarded. For more details on appender configuration please refer to its documentation or java doc.

The logging threshold on the console is INFO, by default. In contrast, there is no threshold set for the server.log file, so all generated logging messages are logged there.

Categories control level for loggers and its children, for details please refer to log4j manual.

By default Mobicents SIP Presence inherits level of INFO from root logger. To make platform add more detailed logs, file `conf/jboss-log4j.xml` has to be altered. Explicit category definition for Mobicents SIP Presence looks like:

```
<category name="org.mobicents.slee">
   <priority value="INFO"/>
</category>
```

This limits the level of logging to INFO for all Mobicents SIP Presence classes. It is possible to declare more categories with different level, to provide logs with greater detail.

For instance, to provide detailed information on Mobicents SIP Presence transaction engine in separate log file(`txmanager.log`), file `conf/jboss-log4j.xml` should contain entries as follows:

```
<appender name="TXMANAGER" class="org.jboss.logging.appender.RollingFileAppender">
```

```
    <errorHandler class="org.jboss.logging.util.OnlyOnceErrorHandler"/>
    <param name="File" value="${jboss.server.home.dir}/log/txmanager.log"/>
    <param name="Append" value="false"/>
    <param name="MaxFileSize" value="500KB"/>
    <param name="MaxBackupIndex" value="1"/>

    <layout class="org.apache.log4j.PatternLayout">
       <param name="ConversionPattern" value="%d %-5p [%c] %m%n"/>
    </layout>
</appender>

<category name="org.mobicents.slee.runtime.transaction">
    <priority value="DEBUG" />
    <appender-ref ref="TXMANAGER"/>
</category>
```

This creates a new file appender and specifies that it should be used by the logger (or category) for the package org.mobicents.slee.runtime.transaction.

The file appender is set up to produce a new log file every day rather than producing a new one every time you restart the server or writing to a single file indefinitely. The current log file is txmanager.log. Older files have the date they were written added to their filenames.

7.1.1. Simplified Global Log4j Configuration

Besides manual logging configuration, described previously, Mobicents SIP Presence also exposes management operations that greatly simplify such configuration, allowing the administrator to select through predefined and complete logging configuration presets. Such operations are available in MBean named `org.mobicents.slee%3Aservice%3DMobicentsManagement`, and the available presets are:

Level

- DEFAULT: Regular logging, at INFO level, displaying most user-related messages;
- DEBUG: More verbose logging, mostly using DEBUG/TRACE level, displaying message of interest for developers;
- PRODUCTION: Low verbosity and async logging, mostly in WARN level, for systems in production so that logging does impact performance.

The available management operations are:

JMX Operation

- getLoggingConfiguration: retrieves what is the current logging configuration;

- switchLoggingConfiguration: allows switching to a different configuration preset;
- setLoggingConfiguration: used to upload a complete logging configuration.

Custom presets can be easily deployed in the application server too. Simply name the configuration file as `jboss-log4j.xml.PRESET_NAME`, where `PRESET_NAME` should be unique preset name, and copy it to directory `$JBOSS_HOME/server/profile_name/deploy/mobicents-slee/log4j-templates`, where **profile_name** is the server profile name.

Note

These procedures changes the whole platform logging configuration, so it will affect also logging for other running applications besides the SIP Presence elemnts.

7.2. Alarms

Currently Mobicents SIP Presence does not uses JAIN SLEE Alarms.

7.3. Trace Facility

Notification sources such as SBBs, Resource Adaptors, Profiles, and SLEE internal components use the `JAIN SLEE Trace Facility` to generate trace messages intended for consumption by external management clients. Management clients register to receive trace messages generated by the `Trace Facility` through the external management interface (MBean). Filters can be applied, in a similar way as in case of Alarms.

Within the SLEE, notification sources use a `tracer` to emit trace messages. A tracer is a named entity. Tracer names are case-sensitive and follow the Java hierarchical naming conventions. A tracer is considered to be an ancestor of another tracer if its name followed by a dot is a prefix of the descendant tracer's name. A tracer is considered to be a parent of a tracer if there are no ancestors between itself and the descendant tracer. For example, the tracer named `com` is the parent tracer of the tracer named `com.foo` and an ancestor of the tracer named `com.foo.bar`.

All tracers are implicitly associated with a notification source, which identifies the object in the SLEE that is emitting the trace message and is included in trace notifications generated by the `Trace MBean` on behalf of the tracer. For instance, an SBB notification source is composed by the SBB id and the Service id.

Important

Multiple notification sources may have tracers with same name in SLEE. Comparing with common logging frameworks, this would mean that the notification source would be part of the log category or name.

For further information on how to use `JAIN SLEE Trace Facility` and receive JMX notifications refer to the JAIN SLEE 1.1 Specification.

7.3.1. JAIN SLEE Tracers and Log4j

Mobicents SIP Presence Tracers additionally log messages to **Apache Log4j**, being the log4j category, for notification source `X`, defined as `javax.slee.` concatenated with the `X.toString()`.

For instance, the full log4j logger `name` for tracer named `GoogleTalkBotSbb`, of sbb notification source with `SbbID[name=GoogleTalkBotSbb,vendor=mobicents,version=1.0]` and `ServiceID[name=GoogleTalkBotService,vendor=mobicents,version=1.0]`, would be `javax.slee.SbbNotification[service=ServiceID[name=GoogleTalkBotService, vendor=mobicents,version=0.1], sbb=SbbID[name=GoogleTalkBotSbb,vendor=mobicents, version=0.1]].GoogleTalkBotSbb` (without the spaces or breaks), which means a log4j category defining its level as `DEBUG` could be:

```
<category
   name="javax.slee.SbbNotification[service=ServiceID[name=GoogleTalkBotService,
   vendor=mobicents,version=0.1],sbb=SbbID[name=GoogleTalkBotSbb,
   vendor=mobicents,version=0.1]]">
   <priority value="DEBUG" />
</category>
```

The relation of JAIN SLEE `tracers` and log4j `loggers` goes beyond log4j showing tracer's messages, changing the tracer's log4j logger `effective level` changes the tracer level in SLEE, and vice-versa. Since JAIN SLEE tracer levels differ from log4j logger levels a mapping is needed:

Table 7.1. Mapping JAIN SLEE Tracer Levels with Apache Log4j Logger Levels

Tracer Level	Logger Level
OFF	OFF
SEVERE	ERROR
WARNING	WARN
INFO	INFO
CONFIG	INFO
FINE	DEBUG
FINER	DEBUG
FINEST	TRACE

Appendix A. Revision History

Revision History

Revision 3.1 Fri Dec 11 2009 EduardoMartins

Migration to Mobicents JAIN SLEE 2.x introduces major refactoring, specially on configurations.

Revision 3.0 Fri Jul 10 2009 EduardoMartins

Major update to include XDM Server authentication and authorization, creation of XCAP Application Usages, SIP Client configuration examples and Resource List Server.

Revision 2.0 Fri Mar 06 2009 DouglasSilas

First release of the "parameterized" and much-improved Mobicents documentation.

Revision 1.0 Tue Jan 20 2009 DouglasSilas

Creation of the Mobicents SIP Presence User Guide separate from the Mobicents Platform User Guide.

Index