JBoss Communications JAIN SLEE Google Talk Bot Example User Guide

by Eduardo Martins and Ivelin Ivanov

1. Document Conventions

This manual uses several conventions to highlight certain words and phrases and draw attention to specific pieces of information.

In PDF and paper editions, this manual uses typefaces drawn from the *Liberation Fonts* [https://fedorahosted.org/liberation-fonts/] set. The Liberation Fonts set is also used in HTML editions if the set is installed on your system. If not, alternative but equivalent typefaces are displayed. Note: Red Hat Enterprise Linux 5 and later includes the Liberation Fonts set by default.

1.1. Typographic Conventions

Four typographic conventions are used to call attention to specific words and phrases. These conventions, and the circumstances they apply to, are as follows.

`Mono-spaced Bold`

Used to highlight system input, including shell commands, file names and paths. Also used to highlight key caps and key-combinations. For example:

> To see the contents of the file `my_next_bestselling_novel` in your current working directory, enter the `cat my_next_bestselling_novel` command at the shell prompt and press **Enter** to execute the command.

The above includes a file name, a shell command and a key cap, all presented in Mono-spaced Bold and all distinguishable thanks to context.

Key-combinations can be distinguished from key caps by the hyphen connecting each part of a key-combination. For example:

> Press **Enter** to execute the command.
>
> Press **Ctrl**+**Alt**+**F1** to switch to the first virtual terminal. Press **Ctrl**+**Alt**+**F7** to return to your X-Windows session.

The first sentence highlights the particular key cap to press. The second highlights two sets of three key caps, each set pressed simultaneously.

If source code is discussed, class names, methods, functions, variable names and returned values mentioned within a paragraph will be presented as above, in `Mono-spaced Bold`. For example:

> File-related classes include `filesystem` for file systems, `file` for files, and `dir` for directories. Each class has its own associated set of permissions.

Proportional Bold

This denotes words or phrases encountered on a system, including application names; dialogue box text; labelled buttons; check-box and radio button labels; menu titles and sub-menu titles. For example:

> Choose **System > Preferences > Mouse** from the main menu bar to launch **Mouse Preferences**. In the **Buttons** tab, click the **Left-handed mouse** check box and click **Close** to switch the primary mouse button from the left to the right (making the mouse suitable for use in the left hand).
>
> To insert a special character into a **gedit** file, choose **Applications > Accessories > Character Map** from the main menu bar. Next, choose **Search > Find** from the **Character Map** menu bar, type the name of the character in the **Search** field and click **Next**. The character you sought will be highlighted in the **Character Table**. Double-click this highlighted character to place it in the **Text to copy** field and then click the **Copy** button. Now switch back to your document and choose **Edit > Paste** from the **gedit** menu bar.

The above text includes application names; system-wide menu names and items; application-specific menu names; and buttons and text found within a GUI interface, all presented in Proportional Bold and all distinguishable by context.

Note the **>** shorthand used to indicate traversal through a menu and its sub-menus. This is to avoid the difficult-to-follow 'Select **Mouse** from the **Preferences** sub-menu in the **System** menu of the main menu bar' approach.

`Mono-spaced Bold Italic` or ***Proportional Bold Italic***

Whether Mono-spaced Bold or Proportional Bold, the addition of Italics indicates replaceable or variable text. Italics denotes text you do not input literally or displayed text that changes depending on circumstance. For example:

> To connect to a remote machine using ssh, type `ssh` *`username@domain.name`* at a shell prompt. If the remote machine is `example.com` and your username on that machine is john, type `ssh john@example.com`.
>
> The `mount -o remount` *`file-system`* command remounts the named file system. For example, to remount the `/home` file system, the command is `mount -o remount /home`.
>
> To see the version of a currently installed package, use the `rpm -q` *`package`* command. It will return a result as follows: *`package-version-release`*.

Note the words in bold italics above username, domain.name, file-system, package, version and release. Each word is a placeholder, either for text you enter when issuing a command or for text displayed by the system.

Aside from standard usage for presenting the title of a work, italics denotes the first use of a new and important term. For example:

> When the Apache HTTP Server accepts requests, it dispatches child processes or threads to handle them. This group of child processes or threads is known as

a *server-pool*. Under Apache HTTP Server 2.0, the responsibility for creating and maintaining these server-pools has been abstracted to a group of modules called *Multi-Processing Modules* (*MPMs*). Unlike other modules, only one module from the MPM group can be loaded by the Apache HTTP Server.

1.2. Pull-quote Conventions

Two, commonly multi-line, data types are set off visually from the surrounding text.

Output sent to a terminal is set in `Mono-spaced Roman` and presented thus:

```
books        Desktop   documentation  drafts  mss    photos   stuff  svn
books_tests  Desktop1  downloads      images  notes  scripts  svgs
```

Source-code listings are also set in `Mono-spaced Roman` but are presented and highlighted as follows:

```
package org.jboss.book.jca.ex1;

import javax.naming.InitialContext;

public class ExClient
{
  public static void main(String args[])
      throws Exception
  {
    InitialContext iniCtx = new InitialContext();
    Object         ref    = iniCtx.lookup("EchoBean");
    EchoHome       home   = (EchoHome) ref;
    Echo           echo   = home.create();

    System.out.println("Created Echo");

    System.out.println("Echo.echo('Hello') = " + echo.echo("Hello"));
  }

}
```

1.3. Notes and Warnings

Finally, we use three visual styles to draw attention to information that might otherwise be overlooked.

Note

A note is a tip or shortcut or alternative approach to the task at hand. Ignoring a note should have no negative consequences, but you might miss out on a trick that makes your life easier.

Important

Important boxes detail things that are easily missed: configuration changes that only apply to the current session, or services that need restarting before an update will apply. Ignoring Important boxes won't cause data loss but may cause irritation and frustration.

Warning

A Warning should not be ignored. Ignoring warnings will most likely cause data loss.

2. Provide feedback to the authors!

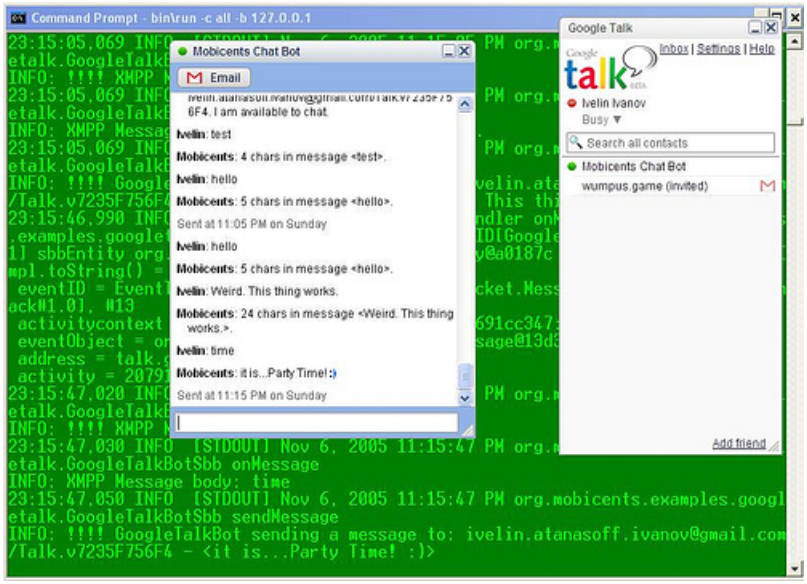
If you find a typographical error in this manual, or if you have thought of a way to make this manual better, we would love to hear from you! Please submit a report in the the *Issue Tracker* [http://bugzilla.redhat.com/bugzilla/], against the product **JBoss Communications JAIN SLEE Google Talk Bot Example**, or contact the authors.

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

If you have a suggestion for improving the documentation, try to be as specific as possible when describing it. If you have found an error, please include the section number and some of the surrounding text so we can find it easily.

Introduction to JBoss Communications JAIN SLEE Google Talk Bot Example

The Google Talk Bot is a simple JAIN SLEE example using the XMPP Resource Adaptor in client mode. It demonstrates connection with the Google Talk service. The bot appears as a regular Google Talk user.



Google Talk Bot Example Screenshot.

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

2.1.2. Software Prerequisites

The Example requires JBoss Communications JAIN SLEE properly set, with XMPP Resource Adaptor deployed.

2.2. JBoss Communications JAIN SLEE Google Talk Bot Example Source Code

This section provides instructions on how to obtain and build the Google Talk Bot Example from source code.

2.2.1. Release Source Code Building

1. **Downloading the source code**

Important

Subversion is used to manage its source code. Instructions for using Subversion, including install, can be found at *http://svnbook.red-bean.com*

Use SVN to checkout a specific release source, the base URL is ?, then add the specific release version, lets consider 2.4.0.CR1.

```
[usr]$ svn co ?/2.4.0.CR1 slee-example-google-talk-bot-2.4.0.CR1
```

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-example-google-talk-bot-2.4.0.CR1
[usr]$ mvn install
```

Once the process finishes you should have the `deployable-unit` jar file in the `target` directory, if JBoss Communications JAIN SLEE is installed and environment variable JBOSS_HOME is pointing to its underlying JBoss Enterprise Application Platform directory, then the deployable unit jar will also be deployed in the container.

Important

This procedure does not install the Example's dependencies

2.2.2. Development Trunk Source Building

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

2.3. Installing JBoss Communications JAIN SLEE Google Talk Bot Example

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

```
[usr]$ ant
```

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

Important

This procedure also installs the Example's dependencies.

2.4. Uninstalling JBoss Communications JAIN SLEE Google Talk Bot Example

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

```
[usr]$ ant undeploy
```

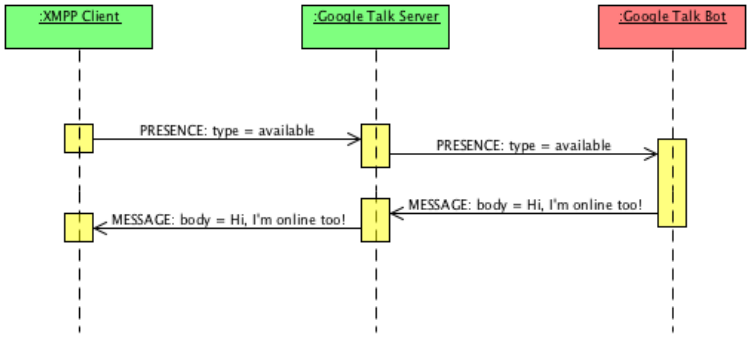
The script will delete the Example's deployable unit jar from the `default` JBoss Communications JAIN SLEE server profile deploy directory, to undeploy from another server profile use the argument `-Dnode=`.

Important

This procedure also uninstalls the Example's dependencies.

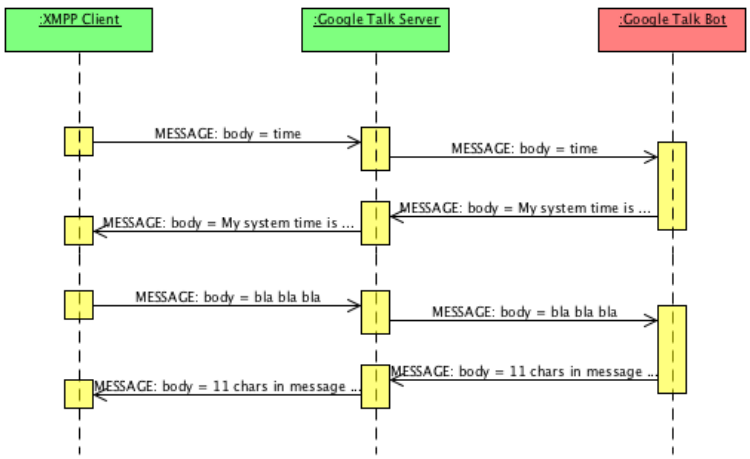
Design Overview

The Google Talk Bot Example is a simple JAIN SLEE 1.1 Application, which creates an XMPP client connection to Google Talk Service when the application is activated by the SLEE Container. The application then handles XMPP `PRESENCE` and `MESSAGE` stanzas.



Google Talk Bot Example Handling of `PRESENCE` XMPP stanzas.

The image above depicts the handling of XMPP `PRESENCE` stanzas, which are used in XMPP to signal presence state changing. The application sends a message containing body `Hi, I'm online too!` to the users which sent a stanza with `available` type, () XMPP users which Google Talk Bot Example is a simple JAIN SLEE 1.1 Application, which creates an XMPP client connection to Google Talk Service when the application is activated by the SLEE Container. The application then handles XMPP `presence` and `message` stanzas.



Google Talk Bot Example Handling of `MESSAGE` XMPP stanzas.

The image above depicts the handling of XMPP `MESSAGE` stanzas, which are used in XMPP to exchange instant messages among users. The application sends a message with body containing

the system time, if the received message body starts with `time`, otherwise it will send back the number of chars in the body of the message received.

The connection to Google Talk Service is closed when the Google Talk Bot Example JAIN SLEE Service is deactivated.

Source Code Overview

The example application is defined by a service descriptor, which refers the Root SBB. The Root SBB does not defines child relations, which means the application is a single SBB.

Important

To obtain the example's complete source code please refer to *Section 2.2, "JBoss Communications JAIN SLEE Google Talk Bot Example Source Code"*.

4.1. Service Descriptor

The service descriptor is plain simple, it just defines the service ID, the ID of the root SBB and its default priority. The complete XML is:

```
<?xml version="1.0" encoding="utf-8"?>
<!DOCTYPE service-xml PUBLIC
   "-//Sun Microsystems, Inc.//DTD JAIN SLEE Service 1.1//EN"
   "http://java.sun.com/dtd/slee-service-xml_1_1.dtd">
<service-xml>
   <service>
      <description />
      <service-name>GoogleTalkBotService</service-name>
      <service-vendor>mobicents</service-vendor>
      <service-version>0.1</service-version>
      <root-sbb>
         <description />
         <sbb-name>GoogleTalkBotSbb</sbb-name>
         <sbb-vendor>mobicents</sbb-vendor>
         <sbb-version>0.1</sbb-version>
      </root-sbb>
      <default-priority>0</default-priority>
   </service>
</service-xml>
```

4.2. The Root SBB

The Google Talk Bot Example's Root SBB is composed by the abstract class and the XML descriptor.

4.2.1. The Root SBB Abstract Class

The class `org.mobicents.examples.googletalk.GoogleTalkBotSbb` includes all the service logic for the example.

4.2.1.1. The setSbbContext(SbbContext) method

The `javax.slee.SbbObject`'s `setSbbContext(SbbContext)` is used by SBBs to store the SBB's context into a class field. The SBB should take the opportunity to also store objects, such as SLEE facilities, or env entries property values, which are common for all service logic entities, a.k.a. SbbEntities, and thus may be stored in the SbbObject instance.

The class fields and `setSbbContext(SbbContext)` method's and related code:

```
/*
 * (non-Javadoc)
 *
 * @see javax.slee.Sbb#setSbbContext(javax.slee.SbbContext)
 */
public void setSbbContext(SbbContext context) {
    this.sbbContext = context;
    this.tracer = sbbContext.getTracer(getClass().getSimpleName());
    try {
        Context myEnv = (Context) new InitialContext()
                .lookup("java:comp/env");

        xmppSbbInterface = (XmppResourceAdaptorSbbInterface) myEnv
                .lookup("slee/resources/xmpp/2.0/xmppinterface");
        xmppActivityContextInterfaceFactory = (XmppActivityContextInterfaceFactory) myEnv
                .lookup("slee/resources/xmpp/2.0/factoryprovider");
        // env-entries
        username = (String) myEnv.lookup("username");
        password = (String) myEnv.lookup("password");

        tracer.info("setSbbContext() Retrieved uid[" + username + "],"
                + " passwd[" + password + "]");
    } catch (NamingException ne) {
        tracer.severe("Could not set SBB context:" + ne.getMessage(), ne);
    }
}
```

4.2.1.2. The ServiceStartedEvent Handler Method

The ServiceStartedEvent is fired by the SLEE Container when the service activation is complete, in the example that is the time to connect to Google Talk Server.

The event handler method's code:

```
/**
 * Init the xmpp connection to GOOGLE TALK when the service is activated by
 * SLEE
 *
 * @param event
 * @param aci
 */
public void onStartServiceEvent(
        javax.slee.serviceactivity.ServiceStartedEvent event,
        ActivityContextInterface aci) {
    try {
        // connect to google talk xmpp server
        XmppConnection connection = xmppSbbInterface.connectClient(
                connectionID, serviceHost, servicePort, serviceName,
                username, password, resource, Arrays
                        .asList(packetsToListen));
        xmppActivityContextInterfaceFactory.getActivityContextInterface(
                connection).attach(sbbContext.getSbbLocalObject());
    } catch (XMPPException e) {
        tracer.severe("Connection to server failed!",e);
    }
}
```

4.2.1.3. The ActivityEndEvent Handler Method

The ActivityEndEvent is fired by the SLEE Container whenever an activity ends. Regarding the example application, it needs to close the XMPP connection when the service is deactivated, and for an SBB the only way to known this happen is by checking for ActivityEndEvents on the ServiceActivity.

The event handler method's code:

```
/**
 * Handler to disconnect from Google when the service is being deactivated.
```

```
     *
     * @param event
     * @param aci
     */
    public void onActivityEndEvent(ActivityEndEvent event,
            ActivityContextInterface aci) {
        if (aci.getActivity() instanceof ServiceActivity) {
            try {
                xmppSbbInterface.disconnect(connectionID);
            } catch (Exception e) {
                tracer.severe(e.getMessage(), e);
            }
        }
    }
```

4.2.1.4. The Presence Event Handler Method

The Presence event indicates that an XMPP `PRESENCE` stanza was received in the managed XMPP connection. The application simply checks if its `TYPE` is `available`, and if that is the case the `Hi, I'm online too!` message is sent back to the sender.

The event handler method's code:

```
    /**
     * Here we handle the Presence messages.
     *
     * @param packet
     * @param aci
     */
    public void onPresence(org.jivesoftware.smack.packet.Presence packet,
            ActivityContextInterface aci) {
        tracer.info("XMPP Presence event type! Status: '" + packet.getType()
                + "'. " + "Sent by '" + packet.getFrom() + "'.");
        // reply hello msg if receives notification of available presence state
        if (packet.getType() == Presence.Type.AVAILABLE) {
            Message msg = new Message(packet.getFrom(), Message.Type.CHAT);
            msg.setBody("Hi. I'am online too.");
            xmppSbbInterface.sendPacket(connectionID, msg);
        }
    }
```

4.2.1.5. The Message Event Handler Method

The Message event indicates that an XMPP `MESSAGE` stanza was received in the managed XMPP connection. The application simply checks if its `body` is `time`, and if that is the case sends back the system time, otherwise sends the count of chars received..

The event handler method's code:

```
/**
 * This is the point where we already have a chat session with the user, so,
 * when they send us messages, we count the chars and reply or tell time :)
 *
 * @param message
 * @param aci
 */
public void onMessage(org.jivesoftware.smack.packet.Message message,
        ActivityContextInterface aci) {
    // only process messages which are not an error and not sent by another
    // bot instance
    if (!message.getType().equals(Message.Type.ERROR)
            && !StringUtils.parseBareAddress(message.getFrom()).equals(
                    username + "@" + serviceName)) {
        tracer.info("XMPP Message event type! Message Body: '"
                + message.getBody() + "'. " + "Sent by '"
                + message.getFrom() + "'.");
        String body = null;
        if (message.getBody() != null) {
            if (message.getBody().equalsIgnoreCase("time")) {
                body = "My system time is " + new Date().toString();
            } else {
                body = message.getBody().length() + " chars in message <"
                        + message.getBody() + ">.";
            }
        }
        Message msg = new Message(message.getFrom(), message.getType());
        msg.setBody(body);
        xmppSbbInterface.sendPacket(connectionID, msg);
    }
}
```

4.2.2. Root SBB XML Descriptor

The Root SBB XML Descriptor has to be provided and match the abstract class code.

First relevant part is the declaration of the `sbb-classes` element, where the sbb class abstract name must be specified:

```
<sbb-classes>
   <sbb-abstract-class>
      <sbb-abstract-class-name>
         org.mobicents.examples.googletalk.GoogleTalkBotSbb
      </sbb-abstract-class-name>
   </sbb-abstract-class>
</sbb-classes>
```

Then the events handled by the SBB must be specified too:

```
<event event-direction="Receive" initial-event="True">
   <event-name>StartServiceEvent</event-name>
   <event-type-ref>
      <event-type-name>
         javax.slee.serviceactivity.ServiceStartedEvent
      </event-type-name>
      <event-type-vendor>javax.slee</event-type-vendor>
      <event-type-version>1.1</event-type-version>
   </event-type-ref>
   <initial-event-select variable="ActivityContext" />
</event>
<event event-direction="Receive" initial-event="False">
   <event-name>ActivityEndEvent</event-name>
   <event-type-ref>
      <event-type-name>
         javax.slee.ActivityEndEvent
      </event-type-name>
      <event-type-vendor>javax.slee</event-type-vendor>
      <event-type-version>1.0</event-type-version>
   </event-type-ref>
</event>
<event event-direction="Receive" initial-event="False">
   <event-name>Message</event-name>
```

```
      <event-type-ref>
        <event-type-name>
          org.jivesoftware.smack.packet.Message
        </event-type-name>
        <event-type-vendor>
          org.jivesoftware.smack
        </event-type-vendor>
        <event-type-version>1.0</event-type-version>
      </event-type-ref>
    </event>
    <event event-direction="Receive" initial-event="False">
      <event-name>Presence</event-name>
      <event-type-ref>
        <event-type-name>
          org.jivesoftware.smack.packet.Presence
        </event-type-name>
        <event-type-vendor>
          org.jivesoftware.smack
        </event-type-vendor>
        <event-type-version>1.0</event-type-version>
      </event-type-ref>
    </event>
```

Note that there is a single event defined as initial, which triggers the xmpp connection creation, remaining events all happen in the XMPP connection activity, that the service instance is already attached.

Next are the `env-entries`, which are used as configuration properties, and contains the Google Talk account credentials:

```
    <env-entry>
      <env-entry-name>username</env-entry-name>
      <env-entry-type>java.lang.String</env-entry-type>
      <env-entry-value>mobicents.gtalk.bot</env-entry-value>
    </env-entry>

    <env-entry>
      <env-entry-name>password</env-entry-name>
      <env-entry-type>java.lang.String</env-entry-type>
      <env-entry-value>m0b1c3nts</env-entry-value>
    </env-entry>
```

Finally, the XMPP Resource Adaptor must be specified also, otherwise SLEE won't put its SBB Interface in the SBB's JNDI Context:

```
<resource-adaptor-type-binding>
   <resource-adaptor-type-ref>
      <resource-adaptor-type-name>
         XMPPResourceAdaptorType
      </resource-adaptor-type-name>
      <resource-adaptor-type-vendor>
         org.mobicents
      </resource-adaptor-type-vendor>
      <resource-adaptor-type-version>
         2.0
      </resource-adaptor-type-version>
   </resource-adaptor-type-ref>
   <activity-context-interface-factory-name>
      slee/resources/xmpp/2.0/factoryprovider
   </activity-context-interface-factory-name>
   <resource-adaptor-entity-binding>
      <resource-adaptor-object-name>
         slee/resources/xmpp/2.0/xmppinterface
      </resource-adaptor-object-name>
      <resource-adaptor-entity-link>
         XMPPRA
      </resource-adaptor-entity-link>
   </resource-adaptor-entity-binding>
</resource-adaptor-type-binding>
```

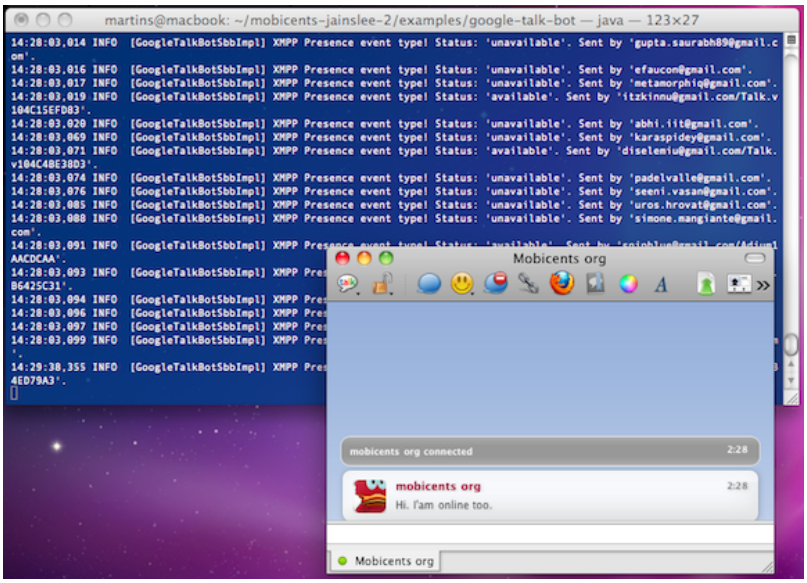
Running the Example

To try the example application a Google Talk account and client is needed, refer to the *Google Talk* [http://www.google.com/talk/] website for instructions.

Important

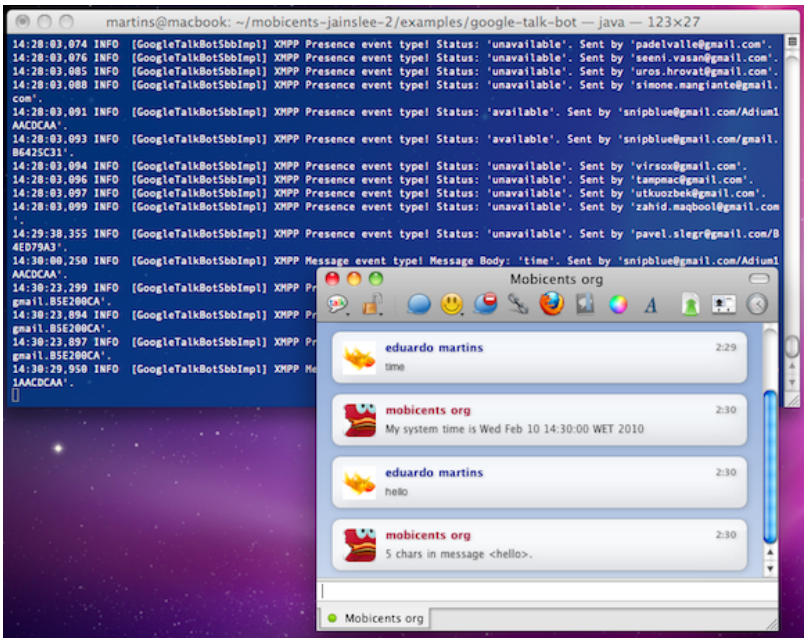
To exchange messages, the Google Talk account needs to add the Google Talk Bot account in its buddy list. The Google Talk Bot account is `mobicents.gtalk.bot@gmail.com`.

Once Google Talk is fully setup and running, start JBoss Communications JAIN SLEE and once the service is activated the `Hi, I'm online too!` message should popup, as depicted in the image below.



Google Talk Bot saying Hi!

At this point messages can be exchanged, sending `time` will get a reply with the system time, and sending `hello` will get a `5 chars in message ...` reply, as depicted in the image below.



Google Talk Bot message exchanging.

Traces and Alarms

6.1. Tracers

Important

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

The example Application uses a single JAIN SLEE 1.1 Tracer, named `GoogleTalkBotSbb`. The related log4j category is `javax.slee.SbbNotification[service=ServiceID[name=GoogleTalkBotService,vendor=mobicents,version=0.1],sbb=SbbID[name=GoogleTalkBotSbb,vendor=mobicents,version=0.1]]`.

6.2. Alarms

The example Application does not sets JAIN SLEE Alarms.

Appendix A. Revision History

Revision History

Revision 1.0 Tue Dec 30 2009 EduardoMartins

Creation of the JBoss Communications JAIN SLEE Google Talk Bot Example User Guide.

Index

F