Mobicents JAIN SLEE SIP UAS Example User Guide

by Eduardo Martins and Bartosz Baranowski

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://code.google.com/p/mobicents/issues/list], against the product **Mobicents JAIN SLEE SIP UAS Example**, or contact the authors.

When submitting a bug report, be sure to mention the manual's identifier: JAIN_SLEE_SipUAS_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 Mobicents JAIN SLEE SIP UAS Example

This example application plays the User Agent Server (UAS) role in a SIP session setup, using the JAIN SIP Resource Adaptor specified in the JAIN SLEE 1.1 specification.

The application receives and sets up new sessions. After setup is complete, a specific timer expiration will signal the application to terminate the session.

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

2.1.2. Software Prerequisites

The Example requires Mobicents JAIN SLEE properly set, with SIP11 Resource Adaptor deployed.

2.2. Mobicents JAIN SLEE SIP UAS Example Source Code

This section provides instructions on how to obtain and build the SIP UAS 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 http:// mobicents.googlecode.com/svn/tags/servers/jain-slee/2.x.y/examples/sip-uas, then add the specific release version, lets consider 2.3.0.FINAL.

    ```
    [usr]$ svn co http://mobicents.googlecode.com/svn/tags/servers/jain-slee/2.x.y/examples/
    sip-uas/2.3.0.FINAL slee-example-sip-uas-2.3.0.FINAL
    ```

2. **Building the source code**

> **Important**
>
> Maven 2.0.9 (or higher) is used to build the release. Instructions for using Maven2, including install, can be found at *http://maven.apache.org*

Use Maven to build the deployable unit binary.

```
[usr]$ cd slee-example-sip-uas-2.3.0.FINAL
[usr]$ mvn install
```

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

> **Important**
>
> This procedure does not install the 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 http://mobicents.googlecode.com/svn/trunk/servers/jain-slee/examples/sip-uas.

2.3. Installing Mobicents JAIN SLEE SIP UAS 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` Mobicents 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 Mobicents JAIN SLEE SIP UAS Example

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

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

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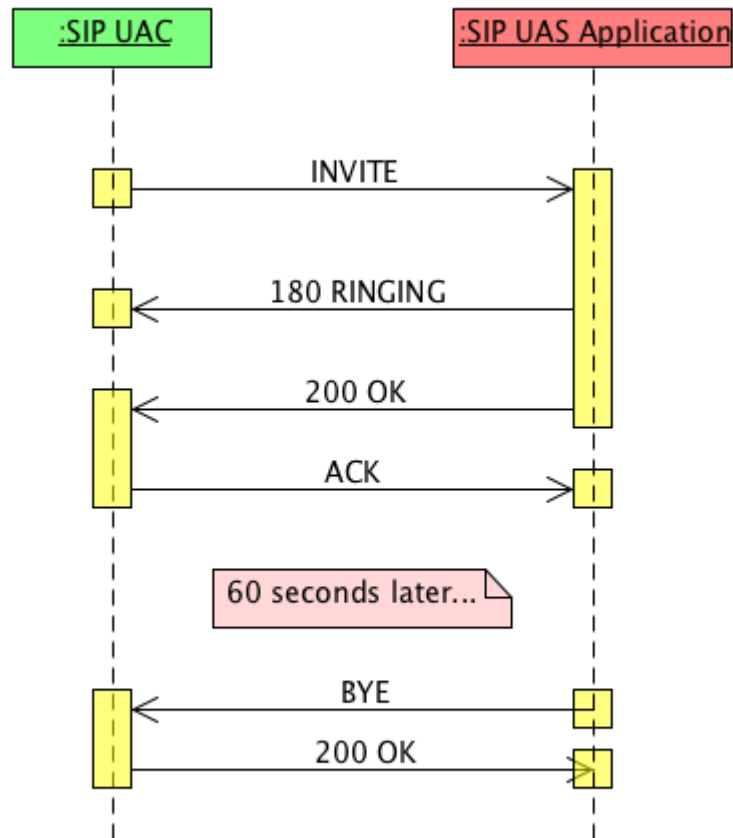
Important

This procedure also uninstalls the Example's dependencies.

Design Overview

The SIP UAS Example is JAIN SLEE 1.1 Application which participates in SIP sessions, as an UAS. The SIP messages exchange is depicted in the image below.

SIP UAS Example Functionality

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, "Mobicents JAIN SLEE SIP UAS 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
<?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>
        <service-name>SIP UAS</service-name>
        <service-vendor>org.mobicents</service-vendor>
        <service-version>1.0</service-version>
        <root-sbb>
            <sbb-name>SIP UAS</sbb-name>
            <sbb-vendor>org.mobicents</sbb-vendor>
            <sbb-version>1.0</sbb-version>
        </root-sbb>
        <default-priority>0</default-priority>
    </service>
</service-xml>
```

4.2. The Root SBB

The SIP UAS 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.slee.example.sip.SipUASExampleSbb` 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, which are reused by all service logic entities, a.k.a. SbbEntities, and are stored in the JNDI environment.

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

```java
public void setSbbContext(SbbContext context) {
    sbbContext = (SbbContextExt) context;
    sipActivityContextInterfaceFactory = (SipActivityContextInterfaceFactory) sbbContext
        .getActivityContextInterfaceFactory(sipRATypeID);
    sleeSipProvider = (SleeSipProvider) sbbContext
        .getResourceAdaptorInterface(sipRATypeID, sipRALink);
    addressFactory = sleeSipProvider.getAddressFactory();
    headerFactory = sleeSipProvider.getHeaderFactory();
    messageFactory = sleeSipProvider.getMessageFactory();
    timerFacility = sbbContext.getTimerFacility();
}
```

4.2.1.2. The SIP INVITE Event Handler Method

The SIP INVITE is the starting point of each instance of the service logic, its responsibility is:

- Create the SIP Dialog activity, which represents the session, and attach the SbbEntity, to receive further SIP messages.

- Set the session timer.

- Send RINGING response back to the UAC.

- Send OK response back to the UAC, to continue the session setup.

The event handler code:

```
      // Initial request
public void onInviteEvent(RequestEvent event, ActivityContextInterface aci) {
   final SbbLocalObject sbbLocalObject = this.sbbContext
         .getSbbLocalObject();
   aci.detach(sbbLocalObject);
   final ServerTransaction serverTransaction = requestEvent
         .getServerTransaction();
   try {
      // create dialog and attach this entity to it's aci
      final DialogActivity dialog = (DialogActivity) sleeSipProvider
            .getNewDialog(serverTransaction);
      final ActivityContextInterfaceExt dialogAci =
            (ActivityContextInterfaceExt) sipActivityContextInterfaceFactory
               .getActivityContextInterface(dialog);
      dialogAci.attach(sbbLocalObject);
      // set timer of 60 secs on the dialog aci
      timerFacility.setTimer(dialogAci, null,
            System.currentTimeMillis() + 60000L, getTimerOptions());
      // send 180
      Response response = messageFactory.createResponse(Response.RINGING,
            requestEvent.getRequest());
      response.addHeader(getContactHeader());
      serverTransaction.sendResponse(response);
      // send 200 ok
      response = messageFactory.createResponse(Response.OK, requestEvent
            .getRequest());
      response.addHeader(getContactHeader());
      serverTransaction.sendResponse(response);
   } catch (Exception ex) {
      ex.printStackTrace();
   }
}
```

4.2.1.3. The Timer Event Handler Method

The Timer Event is received when the session timer expires, the application then sends a BYE request to the other party, requesting the session termination.

The event handler method's code:

```
   public void onTimerEvent(TimerEvent event, ActivityContextInterface aci) {
```

```
   aci.detach(sbbContext.getSbbLocalObject());
   final DialogActivity dialog = (DialogActivity) aci.getActivity();
   try {
      dialog.sendRequest(dialog.createRequest(Request.BYE));
   } catch (Exception ex) {
      ex.printStackTrace();
   }
}
```

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:

```xml
<sbb-classes>
   <sbb-abstract-class reentrant="True">
      <sbb-abstract-class-name>org.mobicents.slee.example.sip.SipUASExampleSbb</sbb-abstract-class-name>
   </sbb-abstract-class>
</sbb-classes>
```

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

```xml
<event event-direction="Receive" initial-event="True">
   <event-name>ServiceStartedEvent</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>
```

```xml
<event event-direction="Receive" initial-event="True">
   <event-name>InviteEvent</event-name>
   <event-type-ref>
      <event-type-name>javax.sip.message.Request.INVITE</event-type-name>
      <event-type-vendor>net.java.slee</event-type-vendor>
      <event-type-version>1.2</event-type-version>
   </event-type-ref>
   <initial-event-select variable="ActivityContext"/>
</event>

<event event-direction="Receive" initial-event="False">
   <event-name>TimerEvent</event-name>
   <event-type-ref>
      <event-type-name>javax.slee.facilities.TimerEvent</event-type-name>
      <event-type-vendor>javax.slee</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 sbb logic, remaining events all happen in activities that the service instance is already attached, abstracting the application from calculating which session it handles.

Finally, the SIP11 Resource Adaptor must be specified also, otherwise SLEE won't put its SBB Interface in the SBB's JNDI Context:

```xml
<resource-adaptor-type-binding>
   <resource-adaptor-type-ref>
      <resource-adaptor-type-name>
         JAIN SIP
      </resource-adaptor-type-name>
      <resource-adaptor-type-vendor>
         javax.sip
      </resource-adaptor-type-vendor>
      <resource-adaptor-type-version>
         1.2
      </resource-adaptor-type-version>
   </resource-adaptor-type-ref>
   <activity-context-interface-factory-name>
```

```
            slee/resources/jainsip/1.2/acifactory
        </activity-context-interface-factory-name>
        <resource-adaptor-entity-binding>
          <resource-adaptor-object-name>
              slee/resources/jainsip/1.2/provider
          </resource-adaptor-object-name>
          <resource-adaptor-entity-link>
              SipRA
          </resource-adaptor-entity-link>
        </resource-adaptor-entity-binding>
    </resource-adaptor-type-binding>
```

Running the Example

To run the example application, first start the JAIN SLEE container, wait for the message `Service activated, now execute SIPP script.` to be displayed in the console, then execute the SIPP script, `run.sh` or `run.bat` depending on which Operating System being used, in `sipp` directory.

The script is UAC party, responsible to create the sessions. All the traffic should be printed in the application server console. The usage of SIPP scripts requires SIPP to be in `$PATH` environment variable.

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 `SipUASExampleSbb`. The related log4j category is `javax.slee.SbbNotification[service=ServiceID[name= SIP UAS,vendor=org.mobicents,version=1.0],` `sbb=SbbID[name=SIP UAS,vendor=org.mobicents,version=1.0]].`

6.2. Alarms

The example Application does not sets JAIN SLEE Alarms.

Appendix A. Revision History

Revision History

Revision 1.0	Thu Jan 06 2011	EduardoMartins

Creation of the Mobicents JAIN SLEE SIP UAS Example User Guide.

Index

F
feedback, viii