

Restcomm JAIN SLEE MS Control Demo Example User Guide

Eduardo Martins <emmartins (at) gmail.com>

Bartosz Baranowski <baranowb (at) gmail.com>

Alexandre Mendonça <brainslog (at) gmail.com>

Restcomm JAIN SLEE MS Control Demo Example User Guide:

by Eduardo Martins, Bartosz Baranowski, and Alexandre Mendonça

Copyright © 2010 Telestax Inc.

Abstract

Preface	iv
1. Document Conventions	iv
1.1. Typographic Conventions	iv
1.2. Pull-quote Conventions	vi
1.3. Notes and Warnings	vi
2. Provide feedback to the authors!	vii
1. Introduction to Restcomm JAIN SLEE MS Control Demo Example	1
2. Setup	2
2.1. Pre-Install Requirements and Prerequisites	2
2.1.1. Hardware Requirements	2
2.1.2. Software Prerequisites	2
2.2. Restcomm JAIN SLEE MS Control Demo Example Source Code	2
2.2.1. Release Source Code Building	2
2.2.2. Development Master Source Building	3
2.3. Installing Restcomm JAIN SLEE MS Control Demo Example	4
2.4. Uninstalling Restcomm JAIN SLEE MS Control Demo Example	4
3. Design Overview	5
4. Source Code Overview	6
4.1. Service Descriptor	6
4.2. Root SBB XML Descriptor	6
4.3. The Root SBB Abstract Class	9
4.3.1. The setSbbContext(SbbContext) method	9
4.3.2. The SIP INVITE event handler	10
4.3.3. The AnswerGenerated handler	12
4.3.4. The JoinEvent handler	13
5. Running the Example	15
6. Traces and Alarms	16
6.1. Tracers	16
6.2. Alarms	16
A. Revision History	17
Index	18

Preface

1. Document Conventions

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

In PDF and paper editions, this manual uses typefaces drawn from the Liberation Fonts [<https://fedorahosted.org/liberation-fonts/>] 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/jain-slee/issues/list>], against the product Restcomm JAIN SLEE MS Control Demo Example, or contact the authors.

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

Chapter 1. Introduction to Restcomm JAIN SLEE MS Control Demo Example

This example demonstrates how MS Control RA can be used as Media Gateway Call Controller to control the Media Gateway (Media Server)

Prior knowledge of JSR309 (MSC) is necessary to understand this example. To learn about JSR309, look at JSR Homepage [<http://jcp.org/en/jsr/detail?id=309>]. Knowledge of MS Control RA is also desired.

Chapter 2. 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 Restcomm JAIN SLEE Hardware Requirements, refer to Restcomm JAIN SLEE documentation for more information.

2.1.2. Software Prerequisites

The Example requires Restcomm JAIN SLEE properly set, with following list of dependencies deployed/started.

MS Control RA

Its required that MS Control RA is deployed. The MS Control RA is responsible to fire the MS Control Events corresponding to Media Server activity

SIP11 RA

Its required that SIP11 RA is deployed. The SIP RA is responsible to fire the SIP Events like INVITE, BYE etc received from SIP User Agents

Restcomm Media Server

Demo requires Media Server running with MGCP Controller deployed. Refer to MS Control RA documentation for explanation

2.2. Restcomm JAIN SLEE MS Control Demo Example Source Code

This section provides instructions on how to obtain and build the MS Control Demo Example from source code.

2.2.1. Release Source Code Building

1. Downloading the source code



Important

Git is used to manage Restcomm JAIN SLEE source code. Instructions for downloading, installing and using Git can be found at <http://git-scm.com/>

Use Git to checkout a specific release source, the Git repository URL is <https://code.google.com/p/jain-slee.media/>, then switch to the specific release version, lets consider 2.8.12.

```
[usr]$ git clone https://code.google.com/p/jain-slee.media/ restcomm-jain-slee-media
[usr]$ cd restcomm-jain-slee-media
[usr]$ git checkout tags/2.8.12
```

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 examples/mscontrol-demo
[usr]$ mvn install
```

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

2.2.2. Development Master Source Building

Similar process as for Section 2.2.1, "Release Source Code Building", the only change is the Git reference should be the master. The `git checkout tags/2.8.12` command should not be performed. If already performed, the following should be used in order to switch back to the master:

```
[usr]$ git checkout master
```

2.3. Installing Restcomm JAIN SLEE MS Control Demo 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 Restcomm JAIN SLEE server profile deploy directory, to deploy to another server profile use the argument `-Dnode=` .

2.4. Uninstalling Restcomm JAIN SLEE MS Control Demo 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 Restcomm JAIN SLEE server profile deploy directory, to undeploy from another server profile use the argument `-Dnode=` .

Chapter 3. Design Overview

This example consists of single Service with single Sbb.

callSbb listens for incoming SIP call. To be specific, it awaits SIP INVITE. Once it receives INVITE, it creates session with Media Server, through MS Control RA .

Note that since RA is a preview, it does not support all operations.

Chapter 4. Source Code Overview

The example application is defined by a service descriptor, which refers the included root SBB.



Important

To obtain the example's complete source code please refer to Section 2.2, "Restcomm JAIN SLEE MS Control Demo 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>
    <service-name>MsControlDemo</service-name>
    <service-vendor>org.mobicens</service-vendor>
    <service-version>1.0</service-version>
    <root-sbb>
      <sbb-name>CallSbb</sbb-name>
      <sbb-vendor>org.mobicens</sbb-vendor>
      <sbb-version>1.0</sbb-version>
    </root-sbb>
    <default-priority>0</default-priority>
  </service>
</service-xml>
```

4.2. 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, along with the cmp fields and child relation.:

```
<sbb-classes>
  <sbb-abstract-class>
```

```
<sbb-abstract-class-name>org.mobicens.slee.example.msc.CallSbb</sbb-abstract-  
class-name>  
  <cmp-field>  
    <cmp-field-name>serverTransactionACI</cmp-field-name>  
  </cmp-field>  
  <cmp-field>  
    <cmp-field-name>dialogACI</cmp-field-name>  
  </cmp-field>  
  <cmp-field>  
    <cmp-field-name>mediaSessionACI</cmp-field-name>  
  </cmp-field>  
  <cmp-field>  
    <cmp-field-name>networkConnectionACI</cmp-field-name>  
  </cmp-field>  
  <cmp-field>  
    <cmp-field-name>mediaGroupACI</cmp-field-name>  
  </cmp-field>  
</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>Invite</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="ApplicationContext" />  
</event>  
<event event-direction="Receive" initial-event="False">  
  <event-name>AnswerGenerated</event-name>  
  <event-type-ref>  
    <event-type-name>javax.media.mscontrol.networkconnection.  
      SdpPortManagerEvent.ANSWER_GENERATED</event-type-name>  
    <event-type-vendor>org.mobicens</event-type-vendor>  
    <event-type-version>1.0</event-type-version>  
  </event-type-ref>  
</event>  
<event event-direction="Receive" initial-event="False">  
  <event-name>StreamFailure</event-name>  
  <event-type-ref>  
    <event-type-name>javax.media.mscontrol.networkconnection.  
      SdpPortManagerEvent.NETWORK_STREAM_FAILURE</event-type-name>  
    <event-type-vendor>org.mobicens</event-type-vendor>  
    <event-type-version>1.0</event-type-version>  
  </event-type-ref>  
</event>  
  
<event event-direction="Receive" initial-event="False">  
  <event-name>Joined</event-name>  
  <event-type-ref>
```

```
<event-type-name>javax.media.mscontrol.join.
    JoinEvent.JOINED</event-type-name>
<event-type-vendor>org.mobicens</event-type-vendor>
<event-type-version>1.0</event-type-version>
</event-type-ref>
</event>

<event event-direction="Receive" initial-event="False">
  <event-name>AnnouncementCompleted</event-name>
  <event-type-ref>
    <event-type-name>javax.media.mscontrol.mediagroup.
      PlayerEvent.PLAY_COMPLETED</event-type-name>
    <event-type-vendor>org.mobicens</event-type-vendor>
    <event-type-version>1.0</event-type-version>
  </event-type-ref>
</event>

<event event-direction="Receive" initial-event="False">
  <event-name>Disconnect</event-name>
  <event-type-ref>
    <event-type-name>javax.sip.Dialog.BYE</event-type-name>
    <event-type-vendor>net.java.slee</event-type-vendor>
    <event-type-version>1.2</event-type-version>
  </event-type-ref>
</event>
```

Finally, the Resource Adaptors 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>
      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>
```

```
<resource-adaptor-type-binding>
  <resource-adaptor-type-ref>
    <resource-adaptor-type-name>
      MSC-1.0-RA
    </resource-adaptor-type-name>
    <resource-adaptor-type-vendor>
      org.mobicens
    </resource-adaptor-type-vendor>
    <resource-adaptor-type-version>
      1.0
    </resource-adaptor-type-version>
  </resource-adaptor-type-ref>
  <activity-context-interface-factory-name>
    slee/resources/media/1.0/acifactory
  </activity-context-interface-factory-name>
  <resource-adaptor-entity-binding>
    <resource-adaptor-object-name>
      slee/resources/media/1.0/provider
    </resource-adaptor-object-name>
    <resource-adaptor-entity-link>
      MSCRA
    </resource-adaptor-entity-link>
  </resource-adaptor-entity-binding>
</resource-adaptor-type-binding>
```

4.3. The Root SBB Abstract Class

The class `org.mobicens.slee.example.msc.CallSbb` includes all the service logic for the example.

4.3.1. 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:

```
private SbbContext sbbContext;
private Tracer tracer;
private SLEEProvider sipRaSbbInterface;
private SipActivityContextInterfaceFactory sipRaAciFactory;
private MsControlFactory msRaSbbInterface;
private MsActivityContextInterfaceFactory mscRaAciFactory;

public void setSbbContext(SbbContext sbbContext) {
    this.sbbContext = sbbContext;
    this.tracer = sbbContext.getTracer("MS-Control-DEMO");
    try {
        Context ctx = (Context) new InitialContext()
```

```

        .lookup("java:comp/env");
sipRaSbbInterface = (SleeSipProvider) ctx
        .lookup("slee/resources/jainsip/1.2/provider");
sipRaAciFactory = (SipActivityContextInterfaceFactory) ctx
        .lookup("slee/resources/jainsip/1.2/acifactory");
msRaSbbInterface = (MsControlFactory) ctx
        .lookup("slee/resources/media/1.0/provider");
mscRaAciFactory = (MsActivityContextInterfaceFactory) ctx
        .lookup("slee/resources/media/1.0/acifactory");
} catch (Exception ne) {
    tracer.severe("Could not set SBB context:", ne);
}
}

```

4.3.2. The SIP INVITE event handler

The SIP INVITE is the starting point of each this example, its responsibility is:

- Do the initial SIP session setup, which means create the SIP Dialog and send provisional response.
- Do the initial Media session setup, which means create the MediaSession object (the session is container for all MS Control objects) and create the NetworkConnection, which is then provided with the SIP client SDP.

The event handler code:

```

/**
 * Handles the event notifying new SIP session invitation.
 *
 * @param event
 * @param aci
 */
public void onInvite(RequestEvent event, ActivityContextInterface aci) {
    tracer.info("Received new SIP session invitation.");
    try {
        initialSipSessionSetup(event, aci);
    } catch (Exception e) {
        tracer.severe("Failed to do initial sip session setup.", e);
        abortSipSessionSetup();
        return;
    }
    try {
        initialMediaSessionSetup(event.getServerTransaction());
    } catch (Exception e) {
        tracer.severe("Failed to process sip invite", e);
        abortSipSessionSetup();
        abortMediaSessionSetup();
    }
}

```

```

    * Setup of the media session: creates the media session, creates a network
    * connection on it and process the client sdp received on SIP
    */
private void initialMediaSessionSetup(ServerTransaction serverTransaction)
    throws MsControlException {
    // create media session
    MediaSession session = msRaSbbInterface.createMediaSession();
    ActivityContextInterface mediaSessionACI = mscRaAciFactory
        .getActivityContextInterface(session);
    SbbLocalObject sbbLocalObject = sbbContext.getSbbLocalObject();
    mediaSessionACI.attach(sbbLocalObject);
    // store the media session aci in a cmp shortcut
    setMediaSessionACI(mediaSessionACI);
    tracer.info("Created media session: " + session);
    // create network connection
    NetworkConnection connection = session
        .createNetworkConnection(NetworkConnection.BASIC);
    ActivityContextInterface connectionACI = mscRaAciFactory
        .getActivityContextInterface(connection);
    connectionACI.attach(sbbLocalObject);
    // store the network connection aci in a cmp shortcut
    setNetworkConnectionACI(connectionACI);
    tracer.info("Created network connection: " + connection);
    // process the received sdp
    SdpPortManager sdpManager = connection.getSdpPortManager();
    tracer.info("Created SDP Manager, sending client sdp...");
    sdpManager.processSdpOffer((byte[]) serverTransaction.getRequest()
        .getContent());
}

/*
 * Aborts the media session: releases the network connection and the media
 * session.
 */
private void abortMediaSessionSetup() {
    releaseMediaSession();
}

/*
 *
 */
private void releaseMediaSession() {
    // get sbb entity local object
    SbbLocalObject sbbLocalObject = sbbContext.getSbbLocalObject();
    // release media group (the ivr) if exists
    ActivityContextInterface mediaGroupACI = getMediaGroupACI();
    if (mediaGroupACI != null) {
        mediaGroupACI.detach(sbbLocalObject);
        try {
            MediaGroup mediaGroup = (MediaGroup) mediaGroupACI
                .getActivity();
            if (mediaGroup != null) {
                mediaGroup.release();
            }
        } catch (Exception e) {
            tracer.severe("failed to abort media network connection.", e);
        }
    }
    // release network connection if exists
}
```

```

    ActivityContextInterface networkConnectionACI = getNetworkConnectionACI();
    if (networkConnectionACI != null) {
        networkConnectionACI.detach(sbbLocalObject);
        try {
            NetworkConnection networkConnection = (NetworkConnection) networkConnectionACI
                .getActivity();
            if (networkConnection != null) {
                networkConnection.release();
            }
        } catch (Exception e) {
            tracer.severe("failed to abort media network connection.", e);
        }
    }
    // release media session if exists
    ActivityContextInterface mediaSessionACI = getMediaSessionACI();
    if (mediaSessionACI != null) {
        mediaSessionACI.detach(sbbLocalObject);
        try {
            MediaSession mediaSession = (MediaSession) mediaSessionACI
                .getActivity();
            if (mediaSession != null) {
                mediaSession.release();
            }
        } catch (Exception e) {
            tracer.severe("failed to abort media session.", e);
        }
    }
}

```

4.3.3. The AnswerGenerated handler

The MS Control AnswerGenerated is sent when Media Server successfully processes SDP offer. Event handler code responsibilities are:

- Finish the SIP session setup, that is, send the received SDP offer back to the sip client.
- Finish the Media session setup, that is, create the MediaGroup IVR and join it with the NetworkConnection.

The event handler code:

```

/**
 * Event with the media server generated sdp, send it back to the sip
 * client.
 *
 * @param event
 * @param aci
 */
public void onAnswerGenerated(SdpPortManagerEvent event,
    ActivityContextInterface aci) {
    tracer.info("Received SDP answer.");
}

```

```

    try {
        finishSipSessionSetup(event.getMediaServerSdp());
    } catch (Exception e) {
        tracer.severe("Unable to send OK response with generated SDP", e);
        abortSipSessionSetup();
        abortMediaSessionSetup();
        return;
    }
    try {
        finishMediaSessionSetup(aci);
    } catch (Exception e) {
        tracer.severe("Unable to initiate join.", e);
        terminateSipSession();
        abortMediaSessionSetup();
    }
}

/*
 * End of the media session setup: creates ivr and make it join the session
 * with the sip client.
 */
private void finishMediaSessionSetup(
    ActivityContextInterface networkConnectionAci)
    throws MsControlException {
    NetworkConnection connection = (NetworkConnection) networkConnectionAci
        .getActivity();
    MediaSession session = connection.getMediaSession();
    MediaGroup mediaGroup = session
        .createMediaGroup(MediaGroup.PLAYER_RECORDER_SIGNALDETECTOR);
    connection.joinInitiate(Direction.DUPLEX, mediaGroup, "context");
    ActivityContextInterface mediaGroupACI = mscRaAciFactory
        .getActivityContextInterface(mediaGroup);
    mediaGroupACI.attach(sbbContext.getSbbLocalObject());
    setMediaGroupACI(mediaGroupACI);
}

```

4.3.4. The JoinEvent handler

The MS Control JoinEvent is sent when Media Server controller joins resources . Event handler code responsibilities are:

- Request media playback from group player.

The event handler code:

```

/**
 * Event notifying the connection between the sip client and the media
 * server.
 *
 * @param event
 * @param aci
 */

```

```
public void onJoined(JoinEvent event, ActivityContextInterface aci) {
    tracer.info("SIP client and media server connected, requesting play of announcement...");
    try {
        ActivityContextInterface mediaGroupACI = getMediaGroupACI();
        MediaGroup mediaGroup = (MediaGroup) mediaGroupACI.getActivity();
        mediaGroup.getPlayer().play(new URI(WELCOME), null, null);
    } catch (Exception e) {
        tracer.severe(
            "Unexpected error playing announcement, terminating sip and media sessions.",
            e);
        terminateSipSession();
        terminateMediaSession();
    }
}

/*
 * Terminates the media session: releases the network connection and the
 * media session.
 */
private void terminateMediaSession() {
    releaseMediaSession();
}
```

Chapter 5. Running the Example

To run example simply call SLEE container. Example takes all addresses as its target. You should hear playback from speakers once call is established.

Chapter 6. Traces and Alarms

6.1. Tracers

The example Application uses multiple JAIN SLEE 1.1 Tracer facility instances. Below is full list:

Table 6.1. MS Control Demo Tracer and Log Categories

Sbb	Tracer name	LOG4J category
CallSbb	MS-Control-DEMO	javax.slee.SbbNotification[service= ServiceID[name= MsControlDemo ,vendor=org.mobicens, version=1.0], sbb=SbbID[name=CallSbb, vendor=mobicents, version=1.0]]. SubscriptionProfileSbb



Important

Spaces were introduced in LOG4J category column values, to correctly render the table. Please remove them when using copy/paste.

6.2. Alarms

The example Application does not set JAIN SLEE Alarms.

Appendix A. Revision History

Revision History

Revision 1.0

Tue Dec 30 2009

EduardoMartins

Creation of the Restcomm JAIN SLEE MS Control Demo Example User Guide.

Index

F

feedback, vii