

Restcomm JAIN SLEE MGCP

Demo Example User Guide

Amit Bhayani <amit.bhayani (at) gmail.com>

Oleg Kulikoff <amit.bhayani (at) gmail.com>

Restcomm JAIN SLEE MGCP Demo Example User Guide :

by Amit Bhayani and Oleg Kulikoff

Copyright © 2009 Telestax Inc.

Abstract

MGCP Demo is Media Gateway Controller Application demonstrating the usage of MGCP RA

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 MGCP 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 MGCP 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 MGCP Demo Example	4
2.4. Uninstalling Restcomm JAIN SLEE MGCP Demo Example	4
3. Design Overview	5
3.1. Design	5
4. Source Code Overview	7
5. Running the Example	16
6. Traces and Alarms	17
6.1. Tracers	17
6.2. Alarms	17
A. Revision History	18
Index	19

Preface

1. Document Conventions

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

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

1.1. Typographic Conventions

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

`Mono-spaced Bold`

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

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

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

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

Press **Enter** to execute the command.

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

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

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

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

Proportional Bold

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

Choose System > Preferences > Mouse from the main menu bar to launch Mouse Preferences. In the Buttons tab, click the Left-handed mouse check box and click Close to switch the primary mouse button from the left to the right (making the mouse suitable for use in the left hand).

To insert a special character into a gedit file, choose Applications > Accessories > Character Map from the main menu bar. Next, choose Search > Find... from the Character Map menu bar, type the name of the character in the Search field and click Next. The character you sought will be highlighted in the Character Table. Double-click this highlighted character to place it in the Text to copy field and then click the Copy button. Now switch back to your document and choose Edit > Paste from the gedit menu bar.

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

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

Mono-spaced Bold Italic Of *Proportional Bold Italic*

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

To connect to a remote machine using ssh, type **ssh *username@domain.name*** at a shell prompt. If the remote machine is `example.com` and your username on that machine is john, type **ssh *john@example.com***.

The **mount -o remount *file-system*** command remounts the named file system. For example, to remount the `/home` file system, the command is **mount -o remount */home***.

To see the version of a currently installed package, use the **rpm -q *package*** command. It will return a result as follows: ***package-version-release***.

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

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

When the Apache HTTP Server accepts requests, it dispatches child processes or threads to handle them. This group of child processes or threads is known as a *server-pool*. Under Apache HTTP Server 2.0, the responsibility for creating and maintaining these server-pools has been abstracted to a group of modules called *Multi-Processing Modules (MPMs)*. Unlike other modules, only one module from the MPM group can be loaded by the Apache HTTP Server.

1.2. Pull-quote Conventions

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

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

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

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

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

import javax.naming.InitialContext;

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

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

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

1.3. Notes and Warnings

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



Note

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



Important

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



Warning

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

2. Provide feedback to the authors!

If you find a typographical error in this manual, or if you have thought of a way to make this manual better, we would love to hear from you! Please submit a report in the the Issue Tracker [<https://github.com/RestComm/jain-slee.media/issues>], against the product Restcomm JAIN SLEE MGCP Demo Example, or contact the authors.

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

MGCP assumes a call control architecture where the call control "intelligence" is outside the gateways and handled by external call control elements known as Call Agents. The MGCP assumes that these call control elements, or Call Agents, will synchronize with each other to send coherent commands and responses to the gateways (media servers) under their control. If this assumption is violated, inconsistent behavior should be expected. MGCP does not define a mechanism for synchronizing Call Agents. MGCP is, in essence, a master/slave protocol, where the gateways are expected to execute commands sent by the Call Agents.

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

Prior knowledge of MGCP is necessary to understand this example. To learn about MGCP, look at RFC 3435 [<http://www.faqs.org/rfcs/rfc3435.html>]. Knowledge of MGCP 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.

MGCP RA

Its required that MGCP RA is deployed. The MGCP RA is responsible to fire the MGCP Events corresponding to MGCP Request/Response received from Media Gateway

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

The demo sends MGCP Signals to Media Gateway (Media Server) to play announcements, text-to-speech, initiate conference etc and also requests DTMF events to be notified back to Application. The media part is taken care by Restcomm Media Server; its required that Restcomm Media Server is started before the User dials respective digits to test demo.

2.2. Restcomm JAIN SLEE MGCP Demo Example Source Code

This section provides instructions on how to obtain and build the MGCP 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://github.com/RestComm/jain-slee.media/>, then switch to the specific release version, lets consider 7.1.13.

```
[usr]$ git clone https://github.com/RestComm/jain-slee.media/ restcomm-jain-slee-media
[usr]$ cd restcomm-jain-slee-media
[usr]$ git checkout tags/7.1.13
```

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/mgcp-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/7.1.13` 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 MGCP 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 MGCP 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

The example is designed to demonstrate the usage of JAIN MGCP API including the MGCP RA. At the same time it also demonstrates various features/capabilities of Restcomm Media Server. MGCP messages are transmitted over UDP. Commands are sent to IP addresses defined in the domain for the endpoint. The responses are sent back to the source address (i.e., IP address and UDP port number) of the commands. The domain name specified for Endpoint can include port number as colon separated value. For example 122.64.4.108:2427

When no port is specified for the endpoint, the commands by default is sent:

by the Call Agents, to the default MGCP port for gateways, 2427.

by the Gateways, to the default MGCP port for Call Agents, 2727.

The MGCP RA bounds the MGCP Stack to IP address and Port specified for `jain.mgcp.IP_ADDRESS`, `jain.mgcp.PORT` properties in `resource-adaptor-jar.xml` for MGCP RA. For further details look at MGCP RA Documentation.

3.1. Design

The Example is composed of 3 Services.

First Service is composed of CallSbb as parent/root listening for SIP INVITE as initial event. The IVRSbb and RecorderSbb are its children.

Second Service is composed of only ConferenceSbb acting as root. Its listening for SIP INVITE too and has lower priority than above Service.

Third Service is composed of only ConfLegSbb acting as root. Its listening for Custom Event fired by ConferenceSbb when the first participant joins the Conference.

CallSbb listens for incoming SIP call and depending on To field of INVITE it creates respective child; Child does further processing of SIP Message.

2010 : If user dialed 2010, IVRSbb child is created. As name suggest IVRSbb is responsible for playing announcement to user and listening for DTMF inputs from user. Depending on which DTMF is dialed by user, corresponding announcement is played again.

2011 : If user dialed 2011, RecorderSbb child is created. As name suggest RecorderSbb is responsible for recording user's voice and playing it back after 30 seconds.

2012 : If user dialed 2012, CallSbb ignores message and relies task of creating ConferenceSbb to SLEE.

ConferenceSbb is responsible for maintaining Conference. User gets connected to Conference endpoint by dialing 2012 from SIP Phone. If this is the first User joining Conference, ConferenceSbb fires Custom Event that is consumed by ConfLegSbb. Multiple users can dial 2012 and get connected to same Conference endpoint and participate in conference call.

ConfLegSbb is acting as one of the automated leg of Conference. As soon as conference is initiated, this leg of conference will start a back ground music so even if there is only one participant in conference he/she can listen to music that verifies Conference has started.

As explained earlier, the main purpose of this demo is to make developers understand how to use the MGCP API and MGCP RA. Once there is clear understanding of MGCP API and RA, the business logic can be written as required. Let us go to Section 4, where MGCP usage from respective SBB is explained.

Chapter 4. Source Code Overview

The bellow source code is from IVRSbb, it hides SIP related logic and highlights MGCP specific code. In case if you want to look at complete source, please get the source code as explained in Section 2.2.

```
package org.restcomm.mgcp.demo;

.....

public abstract class IVRSbb implements Sbb {

    public final static String ENDPOINT_NAME = "/restcomm/media/IVR/$";

    public final static String JBOSS_BIND_ADDRESS = System.getProperty("jboss.bind.address",
"127.0.0.1");

    public final static String WELCOME = "http://" + JBOSS_BIND_ADDRESS + ":8080/mgcpdemo/audio/
RQNT-ULAW.wav";

    .....

    private SbbContext sbbContext;

    .....

    // MGCP
    private JainMgcpProvider mgcpProvider;
    private MgcpActivityContextInterfaceFactory mgcpAcif;

    public static final int MGCP_PEER_PORT = 2427;
    public static final int MGCP_PORT = 2727;

    private Tracer logger;

    /** Creates a new instance of CallSbb */
    public IVRSbb() {
    }

    public void onCallCreated(RequestEvent evt, ActivityContextInterface aci) {
        .....
        //SIP Related handling
        .....

        // respond(evt, Response.RINGING);

        CallIdentifier callID = mgcpProvider.getUniqueCallIdentifier();
        this.setCallIdentifier(callID.toString());
        EndpointIdentifier endpointID = new EndpointIdentifier(ENDPOINT_NAME, JBOSS_BIND_ADDRESS +
":" + MGCP_PEER_PORT);

        CreateConnection createConnection = new CreateConnection(this, callID, endpointID,
ConnectionMode.SendRecv);
```

```
try {
    String sdp = new String(evt.getRequest().getRawContent());
    createConnection.setRemoteConnectionDescriptor(new ConnectionDescriptor(sdp));
} catch (ConflictingParameterException e) {
    // should never happen
}

int txID = mgcpProvider.getUniqueTransactionHandler();
createConnection.setTransactionHandle(txID);

MgcpConnectionActivity connectionActivity = null;
try {
    connectionActivity = mgcpProvider.getConnectionActivity(txID, endpointID);
    ActivityContextInterface epnAci = mgcpAcif.getActivityContextInterface(connectionActivity);
    epnAci.attach(sbbContext.getSbbLocalObject());
} catch (FactoryException ex) {
    ex.printStackTrace();
} catch (NullPointerException ex) {
    ex.printStackTrace();
} catch (UnrecognizedActivityException ex) {
    ex.printStackTrace();
}

mgcpProvider.sendMgcpEvents(new JainMgcpEvent[] { createConnection });
}

...
....
}
```

IVRSbb is listening for INVITE SIP RA Event. The Slee Container calls corresponding onCallCreated event handler method.

New instance of `CallIdentifier` is created as this is new call. Through out the call, `CallIdentifier` will remain the same.

New instance of `EndpointIdentifier` is created. The Endpoint local name is `"/restcomm/media/IVR/$"`. A term represented by a dollar sign ("`$`") is to be interpreted as, use any available free endpoint. Endpoint domain name is `JBOSS_BIND_ADDRESS + ":" + MGCP_PEER_PORT`. The MGCP Command will be delivered to MGCP Stack listening at `JBOSS_BIND_ADDRESS` IP Address and `MGCP_PEER_PORT` port. Make sure that the Restcomm Media Server is bound to same IP address as `JBOSS_BIND_ADDRESS` and port in `mgcp-conf.xml` (MGCP Configuration file in Restcomm Media Server) is same as `MGCP_PEER_PORT`.

New instance of `CreateConnection` object is created. This `CreateConnection` Request when received by Media Gateway, will cause Media Gateway to select any one free IVR Endpoint, create a connection on it and send back `CreateConnectionResponse`. If there is any error while creating connection the returned `CreateConnectionResponse` will carry the corresponding cause in `ReturnCode`

Setting the SDP of User Agent. This is not mandatory. If this is set the Connection created on Endpoint on media gateway will start sending the media (depending on `ConnectionMode`) to

User Agent at IP:Port specified in SDP. If this is not set, the Connection can latter be modified by sending `ModifyConnection` (not shown in this example)

Set a new transaction for this MGCP Request.

Create a new `MgcpConnectionActivity` for this MGCP Request, attach the `SbbLocalObject` to this activity to receive `CreateConnectionResponse` event from MGCP RA.

Finally send this `CreateConnection` MGCP Command to Media Gateway.

Once the MGCP command reaches Media Gateway, it replies back and MGCP RA will fire corresponding event.

```
public void onCreateConnectionResponse(CreateConnectionResponse event, ActivityContextInterface
aci) {
    logger.info("Receive CRCX response: " + event.getTransactionHandle());

    .....
    ....

    ReturnCode status = event.getReturnCode();

    switch (status.getValue()) {
    case ReturnCode.TRANSACTION_EXECUTED_NORMALLY:

        this.setEndpointName(event.getSpecificEndpointIdentifier().getLocalEndpointName());

        ConnectionIdentifier connectionIdentifier = event.getConnectionIdentifier();

        this.setConnectionIdentifier(connectionIdentifier.toString());
        String sdp = event.getLocalConnectionDescriptor().toString();

                                //Send OK to UA with SDP from media gateway
                                .....

        .....

        //Play Announcement
        sendRQNT(WELCOME, false);
        .....
        .....

        break;
    default:
        //CRCX failed at Media Gateway. Take necessary action
        .....
    }
}
```

MGCP RA will fire the `CreateConnectionResponse` event once CRCX Response received from Media Gateway.

`ReturnCode` specifies if Media Gateway has successfully created the connection on Endpoint or not.

Return codes are

100 and 199 indicate a provisional response

200 and 299 indicate a successful completion

400 and 499 indicate a transient error

500 and 599 indicate a permanent error

If connection created successfully, `CreateConnectionResponse` will carry the specific `EndpointIdentifier` indicating the concrete Endpoint selected by media gateway. Next all MGCP Request will be fired on same endpoint till `DeleteConnection` is requested which represents end of call.

If connection created successfully, `CreateConnectionResponse` will carry the specific `ConnectionIdentifier` identifying the connection created by media gateway on above Endpoint. The request to apply Signal or detect Event will be for this `ConnectionIdentifier`.

If connection created successfully, `CreateConnectionResponse` will carry the `ConnectionDescriptor` indicating the SDP of above created connection. This SDP can then be sent to UA as OK Response to INVITE received and RTP flow begins between UA and Media Gateway

Since the RTP connection is established between UA and Media Gateway, `NotificationRequest` can be send to Media Gateway to play an announcement.

If creation of connection failed for some reason indicated by `ReturnCode`, necessary action can be taken.

Let us see how `NotificationRequest` is used to request Media Gateway to play an Announcement

```
private void sendRQNT(String mediaPath, boolean createActivity) {
    EndpointIdentifier endpointID = new EndpointIdentifier(this.getEndpointName(),
        JBOSS_BIND_ADDRESS + ":"
        + MGCP_PEER_PORT);

    NotificationRequest notificationRequest = new NotificationRequest(this, endpointID,
        mgcpProvider
        .getUniqueRequestIdIdentifier());

    ConnectionIdentifier connectionIdentifier = new
        ConnectionIdentifier(this.getConnectionIdentifier());

    EventName[] signalRequests = { new EventName(PackageName.Announcement,
        MgcpEvent.ann.withParm(mediaPath), connectionIdentifier) };
    notificationRequest.setSignalRequests(signalRequests);

    RequestedAction[] actions = new RequestedAction[] { RequestedAction.NotifyImmediately };

    RequestedEvent[] requestedEvents = {
        new RequestedEvent(new EventName(PackageName.Announcement, MgcpEvent.oc,
            connectionIdentifier), actions),
    }
```

```
        new RequestedEvent(new EventName(PackageName.Announcement, Mgcpevent.of,
connectionIdentifier), actions),
        new RequestedEvent(new EventName(PackageName.Dtmf, Mgcpevent.dtmf0, connectionIdentifier),
actions),
        new RequestedEvent(new EventName(PackageName.Dtmf, Mgcpevent.dtmf1, connectionIdentifier),
actions),
        new RequestedEvent(new EventName(PackageName.Dtmf, Mgcpevent.dtmf2, connectionIdentifier),
actions),
        new RequestedEvent(new EventName(PackageName.Dtmf, Mgcpevent.dtmf3, connectionIdentifier),
actions),
        new RequestedEvent(new EventName(PackageName.Dtmf, Mgcpevent.dtmf4, connectionIdentifier),
actions),
        new RequestedEvent(new EventName(PackageName.Dtmf, Mgcpevent.dtmf5, connectionIdentifier),
actions),
        new RequestedEvent(new EventName(PackageName.Dtmf, Mgcpevent.dtmf6, connectionIdentifier),
actions),
        new RequestedEvent(new EventName(PackageName.Dtmf, Mgcpevent.dtmf7, connectionIdentifier),
actions),
        new RequestedEvent(new EventName(PackageName.Dtmf, Mgcpevent.dtmf8, connectionIdentifier),
actions),

        new RequestedEvent(new EventName(PackageName.Dtmf, Mgcpevent.dtmf9, connectionIdentifier),
actions),
        new RequestedEvent(new EventName(PackageName.Dtmf, Mgcpevent.dtmfA, connectionIdentifier),
actions),
        new RequestedEvent(new EventName(PackageName.Dtmf, Mgcpevent.dtmfB, connectionIdentifier),
actions),
        new RequestedEvent(new EventName(PackageName.Dtmf, Mgcpevent.dtmfC, connectionIdentifier),
actions),
        new RequestedEvent(new EventName(PackageName.Dtmf, Mgcpevent.dtmfD, connectionIdentifier),
actions),
        new RequestedEvent(new EventName(PackageName.Dtmf, Mgcpevent.dtmfStar, connectionIdentifier),
actions),
        new RequestedEvent(new EventName(PackageName.Dtmf, Mgcpevent.dtmfHash, connectionIdentifier),
actions) };

notificationRequest.setRequestedEvents(requestedEvents);
notificationRequest.setTransactionHandle(mgcpeventProvider.getUniqueTransactionHandler());

NotifiedEntity notifiedEntity = new NotifiedEntity(JBOSS_BIND_ADDRESS, JBOSS_BIND_ADDRESS,
MGCP_PORT);
notificationRequest.setNotifiedEntity(notifiedEntity);

if (createActivity) {
    MgcpeventEndpointActivity endpointActivity = null;
    try {
        endpointActivity = mgcpeventProvider.getEndpointActivity(endpointID);
        ActivityContextInterface epnAci = mgcpeventAcif.getActivityContextInterface(endpointActivity);
        epnAci.attach(sbbContext.getSbbLocalObject());
    } catch (FactoryException ex) {
        ex.printStackTrace();
    } catch (NullPointerException ex) {
        ex.printStackTrace();
    } catch (UnrecognizedActivityException ex) {
        ex.printStackTrace();
    }
} // if (createActivity)

mgcpeventProvider.sendMgcpevents(new JainMgcpevent[] { notificationRequest });
```

```
logger.info(" NotificationRequest sent");
}
```

The `NotificationRequest` should be fired on same Endpoint where connection is created
Create a new `NotificationRequest` Object passing `EndpointIdentifier` created above and use new `RequestIdentifier`

The `NotificationRequest` should be fired on same connection on Endpoint.

The `NotificationRequest` carries Signal to play an announcement on connection represented by `connectionIdentifier`

The `NotificationRequest` carries request to detect DTMF Events and also detect events if Announcement completed successfully or failed.

Since none of the Signals/Events are fired/detected on Endpoint, Endpoint Activity is not created. The above Events when detected would be fired on Connection Activity.

Finally send the request to Media Gateway.

Once Media Gateway receives the `NotificationRequest`, it will process the Signals / Events and send back `NotificationRequestResponse` which carries `ReturnCode` indicating if Signals can be applied or not and Events can be detected or not.

```
public void onNotificationRequestResponse(NotificationRequestResponse event,
ActivityContextInterface aci) {
    logger.info("onNotificationRequestResponse");

    ReturnCode status = event.getReturnCode();

    switch (status.getValue()) {
    case ReturnCode.TRANSACTION_EXECUTED_NORMALLY:
        logger.info("The Announcement should have been started");
        break;
    default:
        ReturnCode rc = event.getReturnCode();
        logger.severe("RQNT failed. Value = " + rc.getValue() + " Comment = " + rc.getComment());

        //Send DLCX to MMS. Send BYE to UA
        break;
    }
}
```

The Media Gateway will fire the `Notify` command to Application when ever it detects any of the above Events requested by `NotificationRequest`

```
public void onNotifyRequest(Notify event, ActivityContextInterface aci) {
    logger.info("onNotifyRequest");
}
```

```
NotifyResponse response = new NotifyResponse(event.getSource(),
    ReturnCode.Transaction_Executed_Normally);
response.setTransactionHandle(event.getTransactionHandle());

mgcpProvider.sendMgcpEvents(new JainMgcpEvent[] { response });

EventName[] observedEvents = event.getObservedEvents();

for (EventName observedEvent : observedEvents) {
    switch (observedEvent.getEventIdentifier().intValue()) {
        case MgcpEvent.REPORT_ON_COMPLETION:
            logger.info("Announcemnet Completed NTFY received");
            break;
        case MgcpEvent.REPORT_FAILURE:
            logger.info("Announcemnet Failed received");
            // TODO : Send DLCX and Send BYE to UA
            break;
        case MgcpEvent.DTMF_0:
            logger.info("You have pressed 0");
            sendRQNT(DTMF_0, false);
            break;
        case MgcpEvent.DTMF_1:
            logger.info("You have pressed 1");
            sendRQNT(DTMF_1, false);
            break;
        case MgcpEvent.DTMF_2:
            logger.info("You have pressed 2");
            sendRQNT(DTMF_2, false);
            break;
        case MgcpEvent.DTMF_3:
            logger.info("You have pressed 3");
            sendRQNT(DTMF_3, false);
            break;
        case MgcpEvent.DTMF_4:
            logger.info("You have pressed 4");
            sendRQNT(DTMF_4, false);
            break;
        case MgcpEvent.DTMF_5:
            logger.info("You have pressed 5");
            sendRQNT(DTMF_5, false);
            break;
        case MgcpEvent.DTMF_6:
            logger.info("You have pressed 6");
            sendRQNT(DTMF_6, false);
            break;
        case MgcpEvent.DTMF_7:
            logger.info("You have pressed 7");
            sendRQNT(DTMF_7, false);
            break;
        case MgcpEvent.DTMF_8:
            logger.info("You have pressed 8");
            sendRQNT(DTMF_8, false);
            break;
        case MgcpEvent.DTMF_9:
            logger.info("You have pressed 9");
            sendRQNT(DTMF_9, false);
            break;
        case MgcpEvent.DTMF_A:
```

```
        logger.info("You have pressed A");
        sendRQNT(A, false);
        break;
    case MgcpEvent.DTMF_B:
        logger.info("You have pressed B");
        sendRQNT(B, false);
        break;
    case MgcpEvent.DTMF_C:
        logger.info("You have pressed C");
        sendRQNT(C, false);
        break;
    case MgcpEvent.DTMF_D:
        logger.info("You have pressed D");
        sendRQNT(D, false);

        break;
    case MgcpEvent.DTMF_STAR:
        logger.info("You have pressed *");
        sendRQNT(STAR, false);

        break;
    case MgcpEvent.DTMF_HASH:
        logger.info("You have pressed C");
        sendRQNT(POUND, false);

        break;
    }
}
```

Send the `NotifyResponse` immediately to avoid Media Gateway sending the `Notify` again on expiration of response Timer.

The `Notify` command carries list of Events depending on which all occurred at Media Gateway. Iterate through this list and act accordingly. In our example we are simply asking Media Gateway to play corresponding audio file for DTMF pressed by user.

Finally when user hangs-up, we need to delete the connection on Endpoint and free the resources

```
public void onCallTerminated(RequestEvent evt, ActivityContextInterface aci) {
    EndpointIdentifier endpointID = new EndpointIdentifier(this.getEndpointName(),
        JBOSS_BIND_ADDRESS + ":"
        + MGCP_PEER_PORT);
    DeleteConnection deleteConnection = new DeleteConnection(this, endpointID);

    deleteConnection.setTransactionHandle(mgcpProvider.getUniqueTransactionHandler());
    mgcpProvider.sendMgcpEvents(new JainMgcpEvent[] { deleteConnection });

    ServerTransaction tx = evt.getServerTransaction();
    Request request = evt.getRequest();

    try {
        Response response = messageFactory.createResponse(Response.OK, request);
        tx.sendResponse(response);
    }
```

```
} catch (Exception e) {  
    logger.severe("Error while sending DLCX ", e);  
}  
}
```

The SIP11 RA fires BYE event, s1cc container calls onCallTerminated method on SBB. New `DeleteConnection` Object is created passing the same Endpoint on which original connection was created. Once Media Gateway receives `DeleteConnection` command, it closes the connection and frees Endpoint from all resources allocated.

Chapter 5. Running the Example

The easiest way to try the example application is to start the JAIN SLEE container, deploy the RA's as mentioned in 2.1.2. Software Prerequisites. Start the Restcomm Media Server.

Fire your favorite SIP Phone. Point the proxy of your SIP Phone to Restcomm JAIN SLEE Server and dial

2010

To test IVR functionality

2011

To test Recording functionality

2012

To test Conference functionality

Chapter 6. Traces and Alarms

6.1. Tracers

Each SBB has its own JAIN SLEE 1.1 Tracer

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

AmitBhayani

Creation of the Restcomm JAIN SLEE MGCP Demo Example User Guide.

Index

F

feedback, vii