Mobicents

1.20

User Guide

Douglas Silas

ISBN:

Publication date:

Mobicents is a highly scalable event-driven application server with a robust component model. Mobicents is the first and only Open Source Platform certified for JSLEE 1.0 compliance. It complements J2EE to enable the convergence of voice, video and data in next-generation intelligent applications. Web and SIP can be combined together to achieve more sophisticated and natural user experience.

Mobicents: User Guide

by Red Hat Documentation Group and Douglas Silas

1. Document Conventions

Certain words in this manual are represented in different fonts, styles, and weights. This highlighting indicates that the word is part of a specific category. The categories include the following:

`Courier font`

Courier font represents `commands`, `file names and paths`, and `prompts` .

When shown as below, it indicates computer output:

```
Desktop        about.html       logs      paulwesterberg.png
Mail           backupfiles      mail      reports
```

`bold Courier font`

Bold Courier font represents text that you are to type, such as: **`service jonas start`**

If you have to run a command as root, the root prompt (`#`) precedes the command:

```
# gconftool-2
```

`italic Courier font`

Italic Courier font represents a variable, such as an installation directory: *`install_dir`*`/bin/`

bold font

Bold font represents **application programs** and **text found on a graphical interface**.

When shown like this: **OK** , it indicates a button on a graphical application interface.

Additionally, the manual uses different strategies to draw your attention to pieces of information. In order of how critical the information is to you, these items are marked as follows:

Note

A note is typically information that you need to understand the behavior of the system.

Tip

A tip is typically an alternative way of performing a task.

Important

Important information is necessary, but possibly unexpected, such as a configuration change that will not persist after a reboot.

Caution

A caution indicates an act that would violate your support agreement, such as recompiling the kernel.

Warning

A warning indicates potential data loss, as may happen when tuning hardware for maximum performance.

2. We Need Feedback!

You should over ride this by creating your own local Feedback.xml file.

Introduction

1. Introduction to the Mobicents Converged Application Server

The Mobicents Communication Platform is the best architecture for creating, deploying and managing services and applications integrating voice, video and data across a range of IP and communications networks. The Mobicents Communication Platform drives convergence by leveraging the following four core capabilities.

Core Capabilities of the Mobicents Converged Application Server

Mobicents JAIN-SLEE
: Mobicents JAIN-SLEE provides a highly-scalable event-driven application server with a robust component model and a fault-tolerant execution environment. Mobicents JAIN-SLEE is the first and only Open Source Platform certified for JSLEE 1.0 compliance. It complements J2EE to enable the convergence of voice, video and data in next-generation intelligent applications. Web and SIP can be combined together to achieve more sophisticated and natural user experience.

Mobicents Media Server
: The Mobicents Media Server is an open sourced server based on the Java Media Framework implementation, and aimed to:

Aims of the Mobicents Media Server

- Deliver a competitive, complete, best-of-breed media gateway functionality of the highest quality.
- Meet the demands of converged wireless, wireline, cable broadband access and fixed-mobile converged VoIP networks from a single media gateway platform.
- Increase flexibility with a media gateway that supports a wide variety of call control protocols, and which scales down to meet the demands of enterprises and small carrier providers.
- React quickly to dynamic market requirements.

Mobicents Presence (XDM) Server
: The first free and open source implementation of an XML Document Management (XDM) server, as defined in the Open Mobile Alliance (OMA)

specifications. This functional element of next-generation IP communication networks is responsible for handling the management of users' XML documents stored on the network side, such as presence authorization rules, contact and group lists (also known as resource lists), static presence information, and much more.

The JBoss Microcontainer
: The JBoss Microcontainer is the state-of-the-art hosting environment in which higher-level containers reside. It provides service registration, configuration and dependency management, classloading isolation control, package versioning, deployment, thread pooling and many other fundamental building blocks necessary for highly-scalable, fault-tolerant middleware servers.

2. Service Building Blocks and Next-Generation Intelligent Networks

In the scope of telecommunications' Next Generation Intelligent Networks (NGINs), Mobicents fits in as a high-performance core engine for Service Delivery Platforms (SDPs) and IP Multimedia Subsystems (IMSes).

Mobicents enables the composition of Service Building Blocks (SBBs) such as call control, billing, user-provisioning, administration, and presence-sensing features. The JAIN SLEE [1] specification allows popular protocol stacks such as SIP to be plugged in as resource adapters. The SLEE Service Building Blocks have many similarities to Enterprise Java Beans (EJBs), and naturally accommodate integration with enterprise applications, the Web, Customer Relationship Management (CRM) and Service-Oriented Architecture (SOA) end points.

Out-of-the-box monitoring and management of Mobicents components is achieved via SLEE standard-based Java Management Extensions (JMXes) and Simple Network Management Profile (SNMP) interfaces.

Beyond telecommunications, Mobicents is applicable to a wider variety of problems demanding high-volume, low-latency signaling. Examples include financial trading, online gaming, RFID sensor network integration, and distributed control.

3. Mobicents Use Cases

The Mobicents Converged Application Server has a wide variety of use cases, including all of the following domains:

- *VoIP Call Control Services*

[1] JAIN SLEE stands for Java API for Intelligent Network Service Logic and Execution Environment architecture. JSLEE (Java Service Logic Execution Environment) is a short synonym for JAIN SLEE.

- *Instant Messaging Services*
- *Online Multi-Player Games*
- *Financial Day Trading*
- *First Responder Communication Networks*
- *Sensor Network Integration and RFID*

VoIP Call Control Services. Voice-over-Internet Protocol (VoIP) services are some of the primary use cases for Mobicents. Examples of VoIP services include call routing, forwarding, termination, voice mailbox, conferencing and user provisioning.

Instant Messaging Services.

Online Multi-Player Games. SLEE is a great platform for Massively Multi-Player (MMP games. The following are some of the principle requirements for MMP servers, provided by the leader of *Java.net Games community* [https://games.dev.java.net/nonav/index.html]:

1. *Worst-case* latencies from end-to-end in the back-end system, including all database operations, of no more than 100 ms.
2. A programming model that allows the game programmer to program simple persistent objects, with no more work than any other standard Java object, and have them execute on data events coming into the system. These objects must be "real objects" in the simulation sense of the term.
3. A programming model that is optimistically parallel while *appearing* to the programmer as a single-threaded event-driven model. Each event has to be ACID-transactional (Atomicity, Consistency, Isolation, Durability), and atomic unto itself. The programmer cannot be required to be aware in any way of multiple threads, database access, or locking. It must be inherently and transparently race-condition-proof and deadlock-proof.
4. It must scale to massive numbers of simultaneous users (5-to-6 figures) online simultaneously accessing the same database of objects.
5. It needs to provide failover, fault-tolerance and efficient load-balancing. The last is critical. Game applications are cost-sensitive. Without the ability to load-balance the potentially heavy loads of multiple apps over banks of low-cost computers used to maximal efficiency, the economic model falls apart.

If you are familiar with SLEE, then you have already noticed that these requirements closely track the fundamental SLEE design principles.

Here are some pointers to books, blogs and papers on the subject of Massive Multi-Player games:

- *Internet-Based Games by R. Young* [http://liquidnarrative.csc.ncsu.edu/pubs/phic.pdf]
- *Massively Multiplayer Game Development* [http://www.amazon.com/exec/obidos/tg/detail/-/1584502436/002-3357427-0961607?v=glance] by Thor Alexander
- *MMP Blog* [http://hardcodedgames.com/mmpgamedev/]
- *A Mobile Gaming Platform for the IMS* [http://www.ibr.cs.tu-bs.de/users/wellnitz/papers/misc/ims-netgames2004.pdf]
- *SIP based Game Server* [http://forum.nokia.com.cn/doc/SIPpresentation(24workshopHK)-new.pdf]

Financial Day Trading. Financial trading is a well-understood problem domain demanding highly-available, fast-response technology to serve billions of trading calls per day. Each trading call is a concise message identifying the trader, ticker and price. Day trading applications fit nicely on top of event-driven containers.

First Responder Communication Networks. We live in an unsafe world. Project P25 is focused on public safety communications digital radio interoperability. It is spurred on by growing concern which has driven many countries' governmentsincluding the US Federal Governmentto reorganize in order to create focused positions to address homeland security. The National Institute of Standards and Technology (NIST) is building a test system for Radio-Frequency subsystem interoperability Standard, which is based on these three protocols: the Session Initiation Protocol (SIP), the Service Discovery Protocol (SDP), and the Real-time Transport Protocol (RTP). The reference implementation and test system will be built as SLEE services.

Sensor Network Integration and RFID. Gartner predicts that sensor technologies will be part of our everyday life by the year 2015. Sensors will be everywhere: as RFID tags on consumer products, devices monitoring tire pressure, location-tracking tags carried by workers in sensitive or hazardous environments, etc. In addition, all enterprise activities will be monitored using enterprise tools connected to the network; by 2010 these, as embedded Internet devices, will represent 95% of all Internet-connected systems.

Pervasive sensor technologies will transform the objective of IT from providing support for fundamentally manual processes to automating the execution of tasks in response to a continuously-changing environment monitored by sensors. This requires a new kind of architecture capable of delivering *extreme* transaction processing.

Installation

1. Installing the Mobicents Binary Distribution

The Mobicents Converged Application Server runs on top of the JBoss Application Server. The Mobicents 1.20 binary distribution comes bundled with the JBoss Application Server version 4.2.2 GA.

Hardware Requirements for the Binary Distribution. Both the Mobicents Converged Application Server and the JBoss Application Server are 100% Java. Mobicents will run on the same hardware that the JBoss Application Server runs on.

Pre-Install Requirements for the Binary Distribution. You should ensure a couple of things before downloading and installing the Mobicents binary release:

- You must have sufficient disk space in order to install the Mobicents binary release. Once unzipped, version 1.20 (`BETA3`) of the Mobicents binary release requires about 150 MB of free disk space. Keep in mind that disk space requirements may change from release to release.
- A working installation of Java 1.5 or higher is required in order to run Mobicents and the JBoss Application Server.

Installing the Mobicents Binary. Once the preconditions and requirements have been met, you are ready to install the Mobicents binary distribution.

1. Download the latest version of the Mobicents binary distribution. The latest version can be found by going to the *Mobicents project page on SourceForge.net* [https://sourceforge.net/projects/mobicents/], clicking on *Download Mobicents*, and then looking for *Mobicents Server* and *Latest Version*. The latest version as of the latest update of this document was `BETA3`.

2. Verify the integrity of the zip file. Along with the zip file there is a sha1 file[1] which contains a checksum that can be used to verify the integrity of the zip file, and can alert you if the file has been changed since it was uploaded, or if it was corrupted upon download.

   On a Linux system you can use the `sha1sum` command to verify the integrity of the binary distribution zip file.

   On a Windows system you will need to download a sha1sum-generating program such as the `sha1sum.exe` program. This will generate a checksum that you can compare with the sum in the sha1 file.

Note that if the two checksums are not identical, it means that the downloaded zip file was corrupted upon download, or has been changed since it was last uploaded to the server.

3. Finally, install the Mobicents Converged Application Server. Unzip the Mobicents binary distribution zip file to a location of your choice, thus completing the install.

2. Building the Mobicents 1.2.x Series

Besides simply installing the latest binary Mobicents release, you can also build the Mobicents Converged Application Server from source. There are several reasons for building Mobicents from source, including getting access to new features such as the latest resource adapters, bundled servers and examples, getting full support for debugging, and accessing extra documentation.

Note that building Mobicents from source requires a working installation of JBoss 4.2. This section on building Mobicents from source also includes the steps necessary for installing the JBoss 4.2.2 binary distribution.

Hardware Requirements for the Source Distribution. Both the Mobicents Converged Application Server and the JBoss Application Server are 100% Java. Mobicents will run on the same hardware that the JBoss Application Server runs on.

Pre-Install Requirements for the Source Distribution. There are a number of requirements that should be met before downloading and installing JBoss, and then acquiring the Mobicents sources and building the Mobicents Converged Application Server.

Source Distribution Installation Requirements

Sufficient Disk Space
: You must have sufficient disk space in order to install the Mobicents source as well as the JBoss binary distribution. The Mobicents source, once fetched from the development Subversion repository, will consume anywhere from 250-300 MB of disk space. In addition to this, the JBoss Application Server binary distribution (version 4.2.2 GA), once decompressed, requires approximately 115 MB, meaning that you should probably have *at least* 415 MB of free disk space available. Keep in mind also that disk space requirements will change with subsequent updates to the development repository.

Java 1.5 or Higher
: A working installation of Java 1.5 or higher is required in order to run Mobicents and the JBoss Application Server.

A Download Utility
: A way to download files. We will use the wget utility in the subsequent sections, but any common web browser will do.

The Subversion Version Control System or Client

The Subversion Version Control System (VCS) is required in order to check out the latest Mobicents source code. In addition to the command line client, there are a number of other clients that can be downloaded or installed via your package manager:

- **Basic Subversion (the shell client).** If you are reasonably proficient on the command line, this is the only client you will need. You can get a binary version for your operating system here: *http://subversion.tigris.org/project_packages.html*. Note that I will be using the command line (shell) Subversion client throughout this Guide.
- **The Meld Diff and Merge Tool.** On Linux and UNIX-based operating systems, Meld is the Swiss Army knife of version control tools. It can do 2- and 3-way diffs between files, diffs between directories, and also serves as a graphical VCS front-end for Subversion, as well as Darcs, Git, Mercurial, Bazaar(-NG), Monotone, Perforce, CVS, and probably others. Note that you will need the basic Subversion shell client installed before Meld can function as a client. Meld is available from *http://meld.sourceforge.net/install.html*.
- **TortoiseSVN.** A graphical Windows Subversion client which integrates with the Explorer file manager. Available from *http://tortoisesvn.tigris.org/*.
- **Other Subversion Clients.** A full list of Subversion clients is available from *http://subversion.tigris.org/links.html* (look under the `Clients and plugins` subheading).

Ant, version 1.7.0 or Higher

The Java-based Ant build tool “for tracking, resolving and managing project dependencies.”, which is available here: *http://ant.apache.org/*.

Both Ant and Maven Are Required!

Both the Ant and Maven build tools are required in order to build the Mobicents Converged Application Server and deploy resource adapters, etc.

Maven, version 2.0.6 or Higher

The Java-based Maven build and “software project management and comprehension tool”, available here: *http://maven.apache.org/*.

Downloading and Installing the Jboss Binary Distribution. Once the pre-install requirements have been met, you are ready to start. We will split the entire procedure, which isn't long but may take a while depending on the speed of your Internet connection, into two parts: in the first we will install JBoss; in the second, we will download and build Mobicents.

Procedure 2.1. Downloading and Installing JBoss

1. First, move to the directory where you want to install JBoss and download the JBoss 4.2.2 General Availability (GA) binary distribution[2]:

   ```
   cd <directory_to_install_jboss_into>
   wget
    http://surfnet.dl.sourceforge.net/sourceforge/jboss/jboss-
   4.2.2.GA.zip
   ```

2. Next, assuming you want to install JBoss in the same directory you have downloaded the zip file into, simply unzip it:

   ```
   unzip jboss-4.2.2.GA.zip
   ```

3. Finally, we must set the `JBOSS_HOME` environment variable. The quick-and-easy way to set it for the current session, but which will not last beyond the next reboot, is to `export` the variable on the command line, and then to `echo` the variable to make sure that it contains the right directory. For example:

   ```
   export JBOSS_HOME="${PWD}/jboss-4.2.2.GA"
   echo $JBOSS_HOME
   /home/my_name/jboss/jboss-4.2.2.GA
   ```

   If the `echo` command doesn't produce any output, make sure that you are in the directory into which you unzipped the JBoss binary distribution zip file, and that the name of that directory is correct in the `export` command (i.e., that the file name hasn't changed in some subtle way).

   Alternatively, you can set the `JBOSS_HOME` environment variable permanently in the startup file for your shell. If you are using the Bash shell, this would mean inserting the following line into your `~/.bashrc` startup file (remember to make sure that the path name, i.e. `<unzip_directory>`, is absolute!):

   ```
   export JBOSS_HOME="<unzip_directory>/jboss-4.2.2.GA"
   ```

   You Must Set JBOSS_HOME If You Are Building from Source

   The source distribution of Mobicents does not “bundle”, or come with, the JBoss Application Server, so you must remember to install it and set the environment variable. Conversely, the binary Mobicents distribution does come bundled with JBoss; therefore, if you are running Mobicents using the binary distribution, then you should *not* set `JBOSS_HOME`.

Downloading the Source and Building Mobicents. Now that that you have successfully installed the JBoss Application Server and set the `JBOSS_HOME` environment variable, we will next download the Mobicents source with Subversion and then build it.

1. First, move to a directory where you want to place the files for Mobicents, create a subdirectory named `mobicents` into which you will download everything, and then use Subversion to get the latest version of the sources:

```
cd <directory_to_install_mobicents_source_into>
mkdir mobicents
svn co http://mobicents.googlecode.com/svn/trunk mobicents
```

2. Building Mobicents from source is a simple affair consisting of issuing a `mvn install` command in the `<mobicents>` directory into which you checked out the sources with Subversion (this directory is named `trunk` on the Google Code Subversion repository):

```
cd <mobicents>
mvn install
```

If the build command is successful, you will see a `BUILD SUCCESSFUL` message in the terminal:

```
[INFO]
 --------------------------------------------------------------
----------
[INFO] BUILD SUCCESSFUL
[INFO]
 --------------------------------------------------------------
----------
[INFO] Total time: 22 seconds
[INFO] Finished at: Wed May 28 13:42:42 CEST 2008
[INFO] Final Memory: 17M/117M
[INFO]
 --------------------------------------------------------------
----------
```

On the other hand, if the build fails, you can either:

- Install and use the Mobicents binary distribution, at least for the time being.

- Wait a day or so and hope that the build failure is fixed in the Subversion sources. Update your copy of the sources by issuing a `svn update` command in the `<mobicents>` directory and try to build by invoking the `mvn install` command again.
- Create a new issue on the Google Code issue tracker, located at *http://code.google.com/p/mobicents/issues/list*.

TBD: Close with a sentence and direct the reader with a ulink to the Running Mobicents (which was installed from source) chapter. Issues that need to be addressed in this chapter include: installing GWT 1.3, which only comes in source for x86 (not version 1.4) and then building the management console.This chapter will have to be expanded with those instructions.

3. Installing EclipSLEE: The JAIN SLEE Eclipse Plugin

The Mobicents project's EclipSLEE Plugin for the Eclipse IDE enables and facilitates working with Mobicents and JAIN SLEE. In particular, EclipSLEE is designed to help in the creation of the following JAIN SLEE components:

- Profile Specifications
- Events
- Service Building Blocks (SBBs)
- Service XML Descriptors
- Deployable Units

With the EclipSLEE plugin, developers can construct complete services quickly and easily. The plug-in takes care of ensuring that the XML descriptors are correct, as well as creating skeleton Java classes to which developers can add service logic.

How to Install the EclipSLEE Eclipse Plugin. EclipSLEE can be installed simply by adding the URL of the Update site. In Eclipse, click on **Help** # **Software Updates** # **Find and Install...**. The `Install/Update` dialog box will pop up. Choose the second option, **Search for new features to install**, and click next to be presented with the `Install` dialog. On the right-hand side, click the **New Remote Site...** button to be presented with the `New Update Site` dialog. For the Name field, enter **`EclipSLEE`**, and in the `URL` field, enter the URL of the remote site: **`http://people.redhat.com/vralev/eclipslee/update/`**. Press **OK** to return to the `Install` dialog. Check the just-added `EclipSLEE` site and click the **Finish** button. Eclipse will then prompt you to read and accept the license agreement, and will also ask you whether you want to install all components. After following the instructions of Eclipse's plugin install wizard, the EclipSLEE plugin will be downloaded and installed (usually to the shared Eclipse plugins directory, unless you have directed Eclipse to install to another location). Restarting Eclipse is required in order to initialize the EclipSLEE plugin so that it can be used.

Running Mobicents

1. Running the Mobicents Application Server

Once installed, running Mobicents simply consists in changing to the `<topmost_mobicents_directory>/server/bin` directory and invoking `run.sh` (Linux) or `run.bat` (Windows) The precise instructions depend on which platform you are running Mobicents on.

Platform Mobicents is Running On

Linux

1. Change your working directory to the `<topmost_mobicents_directory>/server/bin` directory:

   ```
   cd <topmost_mobicents_directory>/server/bin
   ```

2. Ensure that the `run.sh` file is executable. If it isn't, modify its permissions so that it is:

   ```
   chmod +x run.sh
   ```

3. Finally, start the Mobicents Application Server:.

   ```
   ./run.sh
   ```

   Note

   The above command is a shortened version of this equivalent command:

   ```
   ./run.sh -c all -b 127.0.0.1
   ```

Windows

1. Change your working directory to the `<mobicents_folder>\server\bin` folder.

2. Start the Mobicents Application Server:

   ```
   run.bat
   ```

Note

The above command is a shortened version of this equivalent command:

```
run.bat -c all -b 127.0.0.1
```

Working with the Mobicents Management Console

Once Mobicents is running, you can access the management console by pointing your browser to *http://localhost:8080/management-console/* . If you are working with a remote machine, replace **localhost** with the name of the remote machine's host, and **8080** with the correct port.

The management console screen has the following components:

- The `Main Menu`, on the left.
- The `Current View`, on the right.
- The `Log Console`, on the bottom.

Figure 4.1. The Mobicents Management Console, initial view

New views my be selected by clicking on the `Main Menu` items. The `Log Console` displays important and relevant notifications when operations are executed, such as error and status messages.

The initial management console view, which can also be accessed by clicking on `SLEE` at the top of the `Main Menu`, displays a status message indicating whether Mobicents is currently running or not, and provides controls to start, stop, and shut down the Mobicents Converged Application Server.

`Deployable Units`. This view can be selected by clicking on `Deployable Units` from the Main Menu, and shows a list of all deployable units that can be deployed with the Mobicents Converged Application Server.

Figure 4.2. The `Deployable Units` view

A tab bar at the top of the view allows one to browse, search for, and install deployable units.

`Components`. The next view on the Main Menu is the `Components` view, which displays a list of component types, such as services, SBBs, resource adapters, etc., and their current count. Clicking on a component type will cause the management console to show a list of components for that type. From there, you can click on the different components to see their details.

Figure 4.3. The `Compenents` view

`Services`. The `Services` view naturally shows the list of available services, their state, whether `ACTIVE` or `INACTIVE`, and all currently possible actions for that resource adapter, such as `activate`, `deactivate` or `remove`.

Figure 4.4. The `Services` view

The tab bar at the top of the `Services` view also allows you to control the usage parameters of services.

`Resource Adapters`. Choosing `Resources` from the `Main Menu` will put the currently-deployed resource adapters in view. There, you can see their name, type, vendor and version. Clicking on one of the resource adapters will provide further information such as the name of the deployable unit, its state, whether `ACTIVE` or `INACTIVE`, and the currenty possible actions for specific services, such as activating, deactivating, or removing them.

Figure 4.5. The `Resources` view

`Activity Contexts`. The Activity Contexts view can be displayed by clicking on `Activities` from the `Main Menu`, and shows the current activity contexts. The `TTL` field shows how much time the activity can remain idle before being marked for garbage collection.

Figure 4.6. The `Activities` view

Resource Adapters

1. Deploying and Undeploying Resource Adapters

1.1. Before You Begin: Resource Adapters and Deployable Unit Files

Resource adapters are JAIN SLEE components that can be deployed (i.e. “loaded” or “installed”) into the Mobicents Converged Application Server, and undeployed (“uninstalled”) from it. At the file level, resource adapters consist of specially-named files called deployable unit files. Two of these deployable unit files must be loaded/installed before the Mobicents Converged Application Server will deploy the resource adapter. The next few sections will teach you how to locate where resource adapters and their files live, and how to deploy them and undeploy them through several different, though equivalent ways: via the command line; by copying the correct files to a special directory, or removing them from it; or with the Mobicents Management Console.

Which of these various ways of accomplishing the same task is easier depends on you. If you are comfortable with the command line, or like to script behavior, then deploying via the command line will probably be the preferred way for you. If you would prefer to simply copy or delete some files with the file manager, you can do it that way. And for those who prefer graphical user interfaces (GUIs), the Management Console provides an interface to the Mobicents Converged Application Server, and can easily be used to accomplish such tasks both locally and remotely. Note that if you are running Mobicents on the Windows platform, then you will want to deploy resource adapters by either copying files or using the Management Console. However, no matter which method you eventually choose, it will be necessary to read the following explanatory paragraphs, on where to find the directories corresponding to the resource adapters you want to install, and on how to recognize deployable unit files, and on the correct order in which to install the deployable unit files, before reading any or all of the method sections.

Where to Find Resource Adapters in the Mobicents Installation. The files belonging to resource adapters, and which you will need to access in order to deploy and undeploy them, are located in `<`*`topmost_mobicents_directory`*`>/resources`. It is trivial to figure out which directory in `resources` corresponds to the resource adapter that you wish to deploy or undeploy. For example, the `sipra` directory holds the files necessary for running the SIP resource adapter, and the `xmppra` directory holds the XMPP and Google Talk resource adapter.

How to Recognize Deployable Unit Files. Deployable unit files are JAR (Java ARchive) files. There are two deployable unit files in each resource adapter's directory. The following two patterns describe the file names:

Deployable Unit File Name Conventions

`<`*`resource_adapter_abbreviation`*`>-ra-DU.jar`
: ...where *ra* stands for “resource adapter”, and *DU* stands for "Deployable Unit (file). For example, if we want to deploy the SIP resource adapter, and we are located in `<`*`topmost_mobicents_directory`*`>/resources/sipra`, then this deployable unit file will be named `sip-ra-DU.jar`.

`<`*`resource_adapter_abbreviation`*`>-ratype-DU.jar`
: ...where *ratype* stands intuitively for “resource adapter type”, and *DU* again for “Deployable Unit (file)”. The file name for the “ratype” deployable unit file for the SIP resource adapter is therefore `sip-ratype-DU.jar`.

How to Determine the Correct Order for Installing Deployable Unit Files. Deployable unit files must be installed in a certain order, starting with the *ratype* deployable unit file, whose file name has the form: `<`*`resource_adapter_abbreviation`*`>-ratype-DU.jar`.

The second (and last) deployable unit file you must install is the *ra* file: `<`*`resource_adapter_abbreviation`*`>-ra-DU.jar`.

For example, if you wanted to deploy the SIP resource adapter, then you would want to first install the deployable unit file named `<`*`resource_adapter_abbreviation`*`>-ratype-DU.jar`, followed by the `<`*`resource_adapter_abbreviation`*`>-ra-DU.jar` file.

1.2. Deploying Resource Adapters

Note

The introductory section titled *Section 1.1, “Before You Begin: Resource Adapters and Deployable Unit Files”* should be read before proceeding to one of the following sections.

Section 1.2.1, “Deploying Resource Adapters Via the Command Line”

Section 1.2.2, “Deploying Resource Adapters By Copying Deployable Unit Files”

Section 1.2.3, “Deploying Resource Adapters With the Management Console”

1.2.1. Deploying Resource Adapters Via the Command Line

Resource adapters can be deployed by simply issuing an ant task command from the topmost Mobicents directory. Make sure that the Mobicents Converged Application Server has been started, and then change to the topmost directory:

```
cd <topmost_mobicents_directory>
```

Run the ant build-and-deploy script, remembering to replace `<resource_adapter>` with the correct directory name (refer to *Where to Find Resource Adapters in the Mobicents Installation* if you are unsure):

```
tools/ant/bin/ant -f resources/<resource_adapter>/build.xml
 ra-deploy
```

For example, the directory name in `<topmost_mobicents_directory>/resources` corresponding to the JAIN SIP resource adapter is (currently) named `sipra` ("SIP Resource Adapter"). To deploy the SIP resource adapter, therefore, you would issue the following command:

```
tools/ant/bin/ant -f resources/sipra/build.xml ra-deploy
```

If the resource adapter builds successfully, then a message similar to the following will be written to standard output:

```
ra-activate:

ra-deploy:

BUILD SUCCESSFUL
Total time: 5 seconds
```

To undeploy a resource adapter via the command line, refer to *Section 1.3.1, “Undeploying Resource Adapters Via the Command Line”*.

1.2.2. Deploying Resource Adapters By Copying Deployable Unit Files

It is essential to have read the previous sections on locating resource adapter directories, recognizing deployable unit files, and determining the correct order for installing the files, before continuing. See *Section 1.1, “Before You Begin: Resource Adapters and Deployable Unit Files”*.

You can deploy resource adapters by simply copying deployable unit files to a special directory with your file manager, or by doing the same via the command line. These instructions assume that you are using the file manager. You can also copy the two necessary files simultaneously, or one-after-the-other in the correct order, which is the way we will do it.

Warning

Make sure that you *copy* the deployable unit files, instead of *moving* them. If you accidentally move them, then make sure you copy them back to the correct resource adapter directory where they came from. If you accidentally move them and then delete them, you will probably have to reinstall Mobicents, so be careful to copy instead of move!

Open your file manager and move to the directory corresponding to the resource adapter you want to deploy. Find the `<`*`resource_adapter_abbreviation`*`>-ratype-DU.jar` file and copy it.

Copying the first deployable unit file, the ratype file, in order to deploy the SIP resource adatper.

Figure 5.1. Locating and Copying the First Deployable Unit File

Copying the first deployable unit file, the ratype file, in order to deploy the SIP resource adatper.

Where you will paste it to depends on whether you installed the Mobicents binary release, or built Mobicents from source. If you installed the binary version, then the correct directory to copy the files into is `<`*`topmost_mobicents_directory`*`>/server/server/default/deploy` (remembering that the bundled JBoss installation that comes with the binary Mobicents release is located in `<`*`topmost_mobicents_directory`*`>/server/server/default`; you are actually copying the file into JBoss's `deploy` directory). If, on the other hand, you installed Mobicents and JBoss separately and set the `JBOSS_HOME` environment variable, then the correct directory to copy the file to is `$JBOSS_HOME/deploy`.

After copying the *ratype* deployable unit file, Mobicents will send a message like the following to standard output, indicating that the file was successfully installed.

```
19:59:03,651 INFO  [RaTypeDeployer] Added RA Type with id
 ResourceAdaptorTypeID[JAIN SIP#javax.sip#1.2]
19:59:03,683 INFO  [STDOUT] 19:59:03,682
 INFO  [DeploymentMBeanImpl] Deployable
 unit with URL
 file:/home/silas/Desktop/temp/apps/mobicents/mobicents-all-
1_2_0_BETA2/server/server/default/deploy/sip-ratype-DU.jar deployed
 as DeployableUnitID[0]
```

After successfully copying the `<`*`resource_adapter_abbreviation`*`>-ratype-DU.jar` file, proceed to copy the `<`*`resource_adapter_abbreviation`*`>-ra-DU.jar` file to the same directory. Copying the *ra* file (after the *ratype* file, of course) will lead to successful installation of the resource adapter, and the Mobicents Converged Application Server will output information similar to the following to standard output:

```
19:59:59,940 INFO  [STDOUT] 19:59:59,940 INFO
 [ResourceManagementMBeanImpl] Activated RA Entity SipRA
20:00:00,191 INFO  [STDOUT] 20:00:00,190 INFO
 [ResourceManagementMBeanImpl] Created Link between RA Entity SipRA
 and Name SipRA
```

The easier way is simply to select and copy both deployable unit files and paste them simultaneously into `<`*`topmost_mobicents_directory`*`>/server/server/default/deploy` (or `$JBOSS_HOME/deploy` if you installed JBoss separately). When you do it that way, Mobicents can figure out in which order it should install the files.

To undeploy a resource adapter by removing or deleting deployable unit files, refer to *Section 1.3.2, “Undeploying Resource Adapters By Removing Files”*.

1.2.3. Deploying Resource Adapters With the Management Console

The Mobicents Management Console provides a graphical way to deploy and undeploy resource adapters. This section requires that you are familiar with the Management Console; for an introduction to the Console, refer first to *Chapter 4, Working with the Mobicents Management Console*.

Important

The following instructions also assume that you have already read introductory sections on locating resource adapter directories, recognizing deployable unit files, and determining their correct order for installation. It is essential to read those sections before attempting to deploy a resource adapter with the Management Console. See *Section 1.1, “Before You Begin: Resource Adapters and Deployable Unit Files”*

Once the Mobicents Converged Application Server is running, point your browser to *`http://localhost:8080/management-console/`* to open the Management Console. Click on **Deployable Units** in the Main Menu; this will open the **Deployable Units** main view, where you will see a list of currently-deployed units under the **Browse** tab, which will read `No deployable unit found` at first. Two other tabs provide the possibility of searching for and installing deployable units. Click on the **Install** tab, and you will see a **Package file:** prompt and a text-entry field in which the name of the deployable unit file you want to install will be listed. *But first we must locate the correct deployable unit file, so click the **Browse** button to the right of the text-entry field.*

Figure 5.2. The Install tab of the Deployable Units view in the Management Console

Before continuing, make sure that you know which deployable unit file to install first by reading the section titled *How to Determine the Correct Order for Installing Deployable Unit Files*.

Once armed with this knowledge, click the **Browse** button to find and install the first deployable unit file, whose file name will appear in the text entry field, and then click the **Install** button. Then install the second deployable unit file. If you install the files in the incorrect order, a message similar to the following will appear in the **Log Console** at the bottom of the Management Console main view:

```
[ERROR] javax.slee.management.DeploymentException: Could
 not deploy: No DeployableUniDeploymentDescriptor descriptor
 (META-INF/deployable-unit.xml) was found in deployable...
```

If you receive this error message, try installing the other deployable unit file first. Installing the right files in the correct order will lead to `[INFO] Deployable unit installed` messages in the **Log Console**.

If you see the name of the resource adapter you wanted to install listed in the **Resources** view of the Management Console, then that resource adapter has been successfully deployed.

Figure 5.3. Successful Deployment of the SIP Resource Adapter with the Mobicents Management Console

To undeploy a resource adapter with the Management Console, refer to *Section 1.3.3, “Undeploying Resource Adapters With the Management Console”*.

1.3. Undeploying Resource Adapters

Note

The introductory section titled *Section 1.1, “Before You Begin: Resource Adapters and Deployable Unit Files”* should be read before proceeding to one of the following sections.

Section 1.3.1, “Undeploying Resource Adapters Via the Command Line”

Section 1.3.2, “Undeploying Resource Adapters By Removing Files”

Section 1.3.3, “Undeploying Resource Adapters With the Management Console”

1.3.1. Undeploying Resource Adapters Via the Command Line

It is as easy to undeploy resource adapters via the command line as it is to deploy them that way: by simply issuing an ant task command from the topmost Mobicents directory, after the Mobicents Converged Application Server has been started, of course.

```
cd <topmost_mobicents_directory>
```

You will run the same ant script that you ran to build and deploy the resource adapter, but note that the last argument has changed from `ra-deploy` to `ra-undeploy`:

```
tools/ant/bin/ant -f resources/<resource_adapter>/build.xml
 ra-undeploy
```

For example, to undeploy the SIP resource adapter you would issue the following command:

```
tools/ant/bin/ant -f resources/sipra/build.xml ra-undeploy
```

Upon success, a message similar to the following will be written to standard output:

```
ra-uninstall:
     [echo] Uninstalling SipRA.
[slee-management] Apr 8, 2008 8:49:12 PM
 org.mobicents.ant.tasks.UninstallTask run
[slee-management] INFO: No response
[slee-management] Apr 8, 2008 8:49:12 PM
 org.mobicents.ant.tasks.UninstallTask run
[slee-management] INFO: No response

ra-undeploy:

BUILD SUCCESSFUL
Total time: 2 seconds
```

1.3.2. Undeploying Resource Adapters By Removing Files

If you deployed a resource adapter by copying the deployable unit files from the resource adapter's directory to the <*topmost_jboss_directory*>`/deploy` directory[1],

[1]The location of the <*topmost_jboss_directory*>`/deploy` directory depends on whether you installed the binary distribution of Mobicents, in which case it is

then you can undeploy that same resource adapter merely by removing the two deployable unit files from the `deploy` directory. In the file manager or on the command line, simply go to `<topmost_jboss_directory>/deploy` and either move the correct two deployable unit files out of it, or delete both of them from it.

Don't Delete if You Didn't Copy!

If you choose to delete the two deployable unit files, *make sure* that you indeed *copied* them into the `deploy` directory, and that they both still exist in the directory of the resource adapter you want to undeploy!

Successfully removing or deleting both files from the `deploy` directory will result in a long string of messages sent to standard output, and ending with a line similar to: `17:48:10,705 INFO [STDOUT] 17:48:10,705 INFO [DeploymentMBeanImpl] Uninstalled DU with id DeployableUnitID[6]`. It is also possible to simply remove or delete only the `<resource_adapter_abbreviation>-ra-DU.jar` file and have Mobicents undeploy the resource adapter, but it is recommended to always remove or delete both files.

Leaving Resource Adapters Deployed

One convenient way to have the Mobicents Converged Application Server deploy the correct resource adapters every time you start the server is to leave the deployable unit files in the `<topmost_jboss_directory>/deploy` directory. Upon startup, Mobicents will notice that the two files are still there, and will automatically deploy the resource adapter.

1.3.3. Undeploying Resource Adapters With the Management Console

If you are looking to undeploy a resource adapter with the Management Console, then it is assumed that you have first read *Section 1.1, "Before You Begin: Resource Adapters and Deployable Unit Files"* and *Section 1.2.3, "Deploying Resource Adapters With the Management Console"*.

You can undeploy resource adapters from the **Deployable Units** view in the Management Console by uninstalling the two deployable unit files. Click on **Deployable Units** from the Main Menu, and then make sure that the **Browse** tab

`<topmost_mobicents_directory>/server/server/default/deploy`, or downloaded and installed JBoss yourself, in which case it is `$JBOSS_HOME/deploy`.

is selected at the top of the view. You will see a list of deployable unit files for the currently-installed resource adapters.

The Browse tab of the Deployable Units view shows all deployable unit files for all resource adapters that are currently deployed

Figure 5.4. Uninstalling Deployable Unit Files with the Management Console

Find the deployable unit file that corresponds to the `<`*`resource_adapter_abbreviation`*`>-ra-DU.jar` pattern, and click on the **Uninstall** link in the **Actions** column. This will uninstall the file, and a message similar to `[INFO] Deployable unit sip-ra-DU uninstalled` will appear in the **Console Log** below. After uninstalling the *ra* file, then uninstall the `<`*`resource_adapter_abbreviation`*`>-ratype-DU.jar` file, which will print a success message like the previous one to the **Console Log**.

If you try to uninstall the *ratype* deployable unit file before the *ra* file, then you will receive a message similar to the following one, alerting you to the mistake: `[ERROR] javax.slee.management.ManagementException: Exception removing deployable Unit`.

2. SIP Resource Adapter

Introduction to the SIP Resource Adapter. The Mobicents SIP resource adapter is a Mobicents sub-project that aims to create a high-performance SLEE extenson for the Session Iniation Protocol (SIP).

The most recent version of the SIP resource adapter includes several enhancements which allow for more convenient development of SIP resources for a wide range of applications. The type name of this latest version is `JSIP v1.2`, the type vendor is `net.java.slee.sip`, and the type version is `1.2`.

The JAIN SIP Resource Adapter Version 1.1. In Java Specification Request 22 (JSR-22), this resource adapter type name is denoted as `JAIN SIP`, and the type vendor as `javax.sip`. This resource adapter type version is 1.1. Mobicents provides a fully-compliant implementation of this specification.

2.1. Installing the JAIN SIP Resource Adapter

To install the JAIN SIP Resource Adapter, you must first acquire the latest version. Navigate to the *Mobicents Sourceforge project page* [https://sourceforge.net/projects/mobicents] and click on the *Download Mobicents* link. On the next page, click on the *SLEE SIP RA* link so that it expands to reveal the necessary JAR file, which you can click to download[2] As of the time of this writing, the latest version was *1.2 RC1 - core rc2*.

2.2. Configuration of the JAIN SIP Resource Adapter

Currently, the JAIN SIP resource adapter can be configured in three different ways.

- *Via the provisioning mechanism*, by altering the `resource-adaptor.xml` configuration file.
- "Via properites file jared with classes (atleast to configure port and ip, it still is place to put gov.nist properties to configure other stack props)"

  Baranowb: is this deprecated? You wrote: "Properties also can be present in properties file present in ra/src/org/mobicents/slee/resources/sip/ diretory. It will be packed to jar with all classes, and wil be used to configure RA and stack(however its deprecated!!!)."
- *Via properties passed to the MBean which created the Resource Adapter Entity*

Configuring the SIP RA Via the Provisioning Mechanism.

You can set provisioned properties in the `resource-adaptor.xml` file, which is located in `resources/sipra/ra/xml/` [TBD: this directory does not exist in the binary installation: exactly which files should we be modifying?].

Provided here are explanations of the various property values which you can set. All of the following configuration entries, which are `<config-property>` elements in the XML file, should be unordered children of their parent `<resource-adapter>` element [3]

[2]As of the time this was written, the JAR file was named `jsipv1.2ra-package-10-02-2007.jar`.

[3]Here is an example of the complete `resource-adaptor.xml` configuration file.

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!DOCTYPE resource-adaptor-jar PUBLIC "-//Sun Microsystems,
 Inc.//DTD JAIN SLEE Resource Adaptor 1.0//EN"
 "http://java.sun.com/dtd/slee-resource-adaptor-jar_1_0.dtd">
<resource-adaptor-jar>
 <resource-adaptor
  id="jain-sip_1.1_RA">
  <description>JAIN SIP Resource Adaptor</description>
  <resource-adaptor-name>JainSipResourceAdaptor</resource-adaptor-name>
  <resource-adaptor-vendor>net.java.slee.sip</resource-adaptor-vendor>
  <resource-adaptor-version>1.2</resource-adaptor-version>

  <resource-adaptor-type-ref>
```

dialogIdleTimeTimeout
: This configuration property sets the timeout on dialogs, in milliseconds. After this amount of time, if the dialog hasn't received any requests or responses, and didn't send any, it is considered to be a “dead end” and is terminated.

```
    <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>

  <!--added oc-resource-adaptor.xml lines here altogether-->
   <resource-adaptor-classes>
    <resource-adaptor-class>
     <resource-adaptor-class-name>
                 org.mobicents.slee.resource.sip.SipResourceAdaptor
               </resource-adaptor-class-name>
    </resource-adaptor-class>
   </resource-adaptor-classes>

   <config-property>
    <description>uS after which dialog if it has not received/send any
  message is considered to be dead and is expunged</description>
    <config-property-name>dialogIdleTimeTimeout</config-property-name>
    <config-property-type>java.lang.Long</config-property-type>
    <config-property-value>360000</config-property-value>
   </config-property>

   <config-property>
    <description>uS after CANCEL event is going to be fired into SLEE. This
  atleast gives Sbbs chance to see INVITE and respond, rather than receiving
  INVITE in on Tx that is in terminated state.</description>
    <config-property-name>cancelWait</config-property-name>
    <config-property-type>java.lang.Long</config-property-type>
    <config-property-value>1000</config-property-value>
   </config-property>

   <config-property>
    <config-property-name>port</config-property-name>
    <config-property-type>java.lang.Integer</config-property-type>
    <config-property-value>5060</config-property-value>
   </config-property>

   <config-property>
    <description>IP to which SIP RAs stack should attach - if "null" it will
  cause RA to pick up Container bind address address</description>
    <config-property-name>ip</config-property-name>
    <config-property-type>java.lang.String</config-property-type>
    <config-property-value>null</config-property-value>
   </config-property>
```

```
<config-property>
 <description>uS after which dialog if it has not
 received/send any message is considered to be dead and is
 expunged</description>
 <config-property-name>dialogIdleTimeTimeout</config-property-
name>
 <config-property-type>java.lang.Long</config-property-type>
 <config-property-value>360000</config-property-value>
</config-property>
```

cancelWait

This configuration property sets the amount of time, in milleseconds, between receiving a `CANCEL` and triggering logic to handle it.

```
<config-property>
 <description>uS after CANCEL event is going to be fired into
 SLEE. This atleast gives Sbbs chance to see INVITE and respond,
 rather than receiving INVITE in on Tx that is in terminated
 state.</description>
 <config-property-name>cancelWait</config-property-name>
 <config-property-type>java.lang.Long</config-property-type>
 <config-property-value>1000</config-property-value>
</config-property>
```

port

This configuration property sets the port number on which the underlying SIP stack will listen.

```
<config-property>
 <config-property-name>port</config-property-name>
 <config-property-type>java.lang.Integer</config-property-type>
 <config-property-value>5060</config-property-value>
</config-property>
```

```
   <config-property>
    <description>List of transports, separated with ","</description>
    <config-property-name>transports</config-property-name>
    <config-property-type>java.lang.String</config-property-type>
    <config-property-value>UDP</config-property-value>
   </config-property>
  </resource-adaptor>
</resource-adaptor-jar>
```

ip

This configuration property sets the IP address of the underlying stack. If it is set to `null`, then the SIP resource adapter will use the binding address of the container as its IP address.

```
<config-property>
 <description>IP to which SIP RAs stack should attach - if
 "null" it will cause RA to pick up Container bind address
 address</description>
 <config-property-name>ip</config-property-name>
 <config-property-type>java.lang.String</config-property-type>
 <config-property-value>null</config-property-value>
</config-property>
```

transports

This configuration property sets the transports to be used by the stack. It is a string.

```
<config-property>
 <description>List of transports, separated with
 ","</description>
 <config-property-name>transports</config-property-name>
 <config-property-type>java.lang.String</config-property-type>
 <config-property-value>UDP</config-property-value>
</config-property>
```

Configuring the SIP RA Via Passing Properties to an MBean.

2.3. Example: End-to-End SIP

2.4. Example: Wake-Up Call

2.5. Example: Third-Party Call Control

2.6. Example: Back-to-Back User Agent

3. The XMPP and Google Talk Resource Adapter

3.1. Example: Google Talk Bot

4. Asterisk Resource Adapter

4.1. Installing the Asterisk Resource Adapter

The Asterisk resource adapter can be installed by referring to *Section 1, “Deploying and Undeploying Resource Adapters”*.

Remember to replace `<resource_adapter>` with the name of the directory in `<topmost_mobicents_directory>/resources` corresponding to the Asterisk resource adapter (currently this directory is named `asteriskra` , but this could change in the future).

5. J2EE and CA Resource Adapter

6. Diameter Resource Adapter

6.1. Installing the Diameter Resource Adapter

The Diameter resource adapter can be installed by referring to *Section 1, “Deploying and Undeploying Resource Adapters”*.

Remember to replace `<resource_adapter>` with the name of the directory in `<topmost_mobicents_directory>/resources` corresponding to the Diameter resource adapter (currently this directory is named `diameterra` , but this could change in the future).

7. Media Resource Adapter

7.1. Installing the Media Resource Adapter

The Media resource adapter can be installed by referring to *Section 1, “Deploying and Undeploying Resource Adapters”*.

Remember to replace `<resource_adapter>` with the name of the directory in `<topmost_mobicents_directory>/resources` corresponding to the Media resource adapter (currently this directory is named `media-ra` , but this could change in the future).

8. SMPP Resource Adapter

8.1. Installing the SMPP Resource Adapter

The SMPP resource adapter can be installed by referring to *Section 1, “Deploying and Undeploying Resource Adapters”*.

Remember to replace `<resource_adapter>` with the name of the directory in `<topmost_mobicents_directory>/resources` corresponding to the SMPP resource adapter (currently this directory is named `smppra`, but this could change in the future).

9. Media Gateway Resource Adapter

10. JCC INAP Resource Adapter

11. HTTP Servlet Resource Adapter

11.1. Installing the HTTP Servlet Resource Adapter

The HTTP Servlet resource adapter can be installed by referring to *Section 1, “Deploying and Undeploying Resource Adapters”*.

Remember to replace `<resource_adapter>` with the name of the directory in `<topmost_mobicents_directory>/resources` corresponding to the HTTP Servlet resource adapter (currently this directory is named `http-servlet-ra`, but this could change in the future).

11.2. Understanding the HTTP Servlet Resource Adapter

A servlet is used to extend the capabilities of servers which host accesses to applications via a request-response programming model. Although servlets can respond to any type of request, they are commonly used to extend the applications hosted by web servers. The aim of the HTTP Servlet resource adapter is to provide the ease of same request-response programming model in the Service Logic Execution Environment (SLEE). The HTTP resource adapter is not a replacement for a servlet but is supposed to work in close proximity with servlets to smooth the integration between web applications hosted in the web server and applications developed using Service Building Blocks (SBBs) hosted in the SLEE Server.

HttpSessionActivity is the only activity involved in the HTTP Servlet resource adapter

Figure 5.5. The Class Diagram for the HTTP Servlet Resource Adapter

`HttpSessionActivity` represents the `HttpSession` created from the incoming `javax.servlet.http.HttpServletRequest` received by `HttpServletRaServlet`, which in turn passes it on to HttpServletResourceEntryPoint.

HttpServletResourceEntryPoint is the link between the servlet and HttpServletResourceAdaptor. HttpServletResourceAdaptor fires the appropriate event like GET, POST, PUT, DELETE, etc., depending on the method of HttpServletRequest, and the service interested in these events will receive it as initial event. The service will use the HttpServletRequestEvent passed as an argument to an on*something* event method to get an HttpServletResponse object. A service can add content to the response, modify headers etc. Calling the flushBuffer() function on an HttpServletResponse will flush the buffer and the response will be committed.

The service can explicitly end the HttpSessionActivity by calling the invalidate() function on HttpSession. Calling invalidate() will invoke the sessionDestroyed() method of HttpServletRaSessionListener by the web container. Within the sessionDestroyed() method, the resource adapter's endHttpSessionActivity() is called to end the activity. If HttpSession times out, then it is the responsibility of the container to invoke the sessionDestroyed() function and the above steps in order to end activity.

HttpServletResourceAdaptor uses the request.getPathInfo() function to form the web address used to fire the Event:

```
//PathInfo can be an empty string and the creation of Address will
 throw exception
//for empty String hence hardcoding prefix /mobicents
String pathInfo = "/mobicents" + request.getPathInfo();

Address address = new Address(AddressPlan.URI, pathInfo);
sleeEndpoint.fireEvent(ah, event, eventID, address);
```

SBBs which are interested in events emmitted by HttpServletResourceAdaptor should use

```
<initial-event-select variable="Address" />
```

for calculating the convergence name, this way we can make sure that URL's submitted to HttpServletResourceAdaptor are mapped to convergence names

For example:

- The URL URL http://localhost:8080/mobicents/x/y maps to the convergence name /mobicents/x/y.
- URL http://localhost:8080/mobicents/sum/dum maps to the convergence name /mobicents/sum/dum.

A Service Building Block Example. An SBB can be inetersted in any of the HTTP Methods OPTIONS, GET, HEAD, POST, PUT, DELETE, TRACE, and for each HttpMethod, the resource adapter fires a corresponding event.

The following code is an example of Service Building Block code:

```
public void onPost(HttpServletRequestEvent event,
            ActivityContextInterface aci) {
        try {

            HttpServletResponse response = event.getResponse();
            PrintWriter w = response.getWriter();
            w.print("onPost OK! Served by SBB = " + getSbbId());
            w.flush();
            response.flushBuffer();
            log
                    .info("HttpServletRAExampleSbb: POST Request
 received and OK! response sent.");
        } catch (IOException e) {
            e.printStackTrace();
        }

    }

    public void onGet(HttpServletRequestEvent event,
            ActivityContextInterface aci) {
        .....
        .....

    }

    public void onHead(HttpServletRequestEvent event,
            ActivityContextInterface aci) {
        // The HEAD method is identical to GET except that the
 server MUST NOT
        // return a message-body in the response.
        log.info("HttpServletRAExampleSbb: HEAD Request
 received.");

    }

    public void onPut(HttpServletRequestEvent event,
            ActivityContextInterface aci) {
        ...........

    }

    public void onDelete(HttpServletRequestEvent event,
            ActivityContextInterface aci) {
           ............

    }
    public void onOptions(HttpServletRequestEvent event,
            ActivityContextInterface aci) {
               .............
    }

    public void onTrace(HttpServletRequestEvent event,
            ActivityContextInterface aci) {
               ..............
    }
```

```
sbb-jar.xml

<?xml version="1.0" encoding="utf-8"?>

<sbb-jar>
    <sbb>
        <description/>
        <sbb-name>HttpServletRAExampleSBB</sbb-name>
        <sbb-vendor>org.mobicents</sbb-vendor>
        <sbb-version>1.0</sbb-version>
        <sbb-classes>
            <sbb-abstract-class>

 <sbb-abstract-class-
name>org.mobicents.slee.service.httpservletra.example.HttpServletRAExampleSbb</
sbb-abstract-class-name>
                </sbb-abstract-class>
        </sbb-classes>

        <event initial-event="True" event-direction="Receive">
            <event-name>Post</event-name>
            <event-type-ref>

 <event-type-
name>net.java.slee.resource.http.events.incoming.POST</event-type-
name>

 <event-type-vendor>net.java.slee</event-type-vendor>
                <event-type-version>1.0</event-type-version>
            </event-type-ref>
            <initial-event-select variable="Address" />
        </event>


        <event initial-event="True" event-direction="Receive">
            <event-name>Get</event-name>
            ..................
        </event>

    .....
       ....
     <event-name>Head</event-name>
    ...
    <event-name>Put</event-name>
    .....
     ....

        <resource-adaptor-type-binding>
            <resource-adaptor-type-ref>

 <resource-adaptor-type-name>HttpServletResourceAdaptorType</
resource-adaptor-type-name>

 <resource-adaptor-type-vendor>org.mobicents</resource-adaptor-
type-vendor>
```

Example 5.1. A sample sbb-jar.xml configuration file

In order to deploy the HTTP Servlet resource adapter, execute 'ant build-and-deploy-ra' from mobicents/ra/http-servlet-ra folder. Make sure to set the MOBICENTS_HOME and JBOSS_HOME.

TBD: is this correct? Format and ensure the correctness of these last 2-3 paragraphs

To execute the test-case, follow the steps in
mobicents/ra/http-servlet-ra/tests/build.xml

There is a working example httpservletra-example
to demonstarte the usage of http-servlet-ra.
Follow the link to download example
https://mobicents-examples.dev.java.net/source/browse/mobicents-examples/httpservletra-example/ and read instructions README.txt to deploy the example

Note: Http Servlet RA is only for processing incoming HttpServletRequest and sending back the HttpServletResponse. If there is a need to create the Request from within the service make use of Http Client RA.

12. The HTTP Client Resource Adapter

12.1. Installing the HTTP Client Resource Adapter

The HTTP Client resource adapter can be installed by referring to *Section 1, “Deploying and Undeploying Resource Adapters”*.

Remember to replace `<resource_adapter>` with the name of the directory in `<topmost_mobicents_directory>/resources` corresponding to the HTTP Client resource adapter (currently this directory is named `http-client-ra`, but this could change in the future).

12.2. Understanding the HTTP Client Resource Adapter

The Hypertext Transfer Protocol (HTTP) is probably the most significant protocol used on the Internet today. Web services, network-enabled appliances and the growth of network computing continue to expand the role of the HTTP protocol beyond user-driven web browsers, while increasing the number of applications that require HTTP support. Applications developed using the Service Logic Execution Environment fall into this category. The Jakarta Commons `HttpClient` component provides an efficient, up-to-date, and feature-rich package implementing the client side of the most recent HTTP standards and recommendations. The HTTP Client resource adapter provides the client-side HTTP standard within the SLEE execution environment using the popular Apache Commons HTTP Client library.

A Service Building Block (SBB) can use the HTTP Client resource adapter to make a request and get the response either synchronously or asynchronously.

Figure 5.6. The Class Diagram for the HTTP Client Resource Adapter

HTTP Client Resource Adapter Methods

HttpClientResourceAdaptorSbbInterface

Provides the SBB with the interface to interact with the HTTP Client resource adapter. `HttpClientResourceAdaptorSbbInterface` is a wrapper over `org.apache.commons.httpclient.HttpClient` and exposes the most commonly-used methods of `HttpClient`.

`createHttpMethod(String method, String uri).` `HttpClientResourceAdaptorSbbInterface` is used by the SBB to generate the GET, POST, HEAD (etc.) requests (HttpMethod) by calling this method. The URI can be any external URI.

`public void executeMethod(HttpMethod method).` Calling this method will send the request synchronously and the calling application can then process the `HttpMethod` to get the response.

`public HttpClientActivity createHttpClientActivity().` Creates an instance of `HttpClientActivity` for a service that wants to send the request asynchronously. By default the value for `endOnReceivingResponse` is set to false, and the service has to explicitly end activity.

`public HttpClientActivity createHttpClientActivity(boolean endOn>>>ReceivingResponse).` Same method as above, with the only difference being that the service can pass `endOnReceivingResponse` as true, which means the activity ends as soon as the resource adapter executes the method and emits a `ResponseEvent`.

Methods Inherited from `HttpClient`

public HttpClientParams getParams(). The HTTP protocol parameters associated with this `HttpClient`.

public HttpClientParams getState(). The HTTP state associated with the `HttpClient`.

public void setParams(HttpClientParams params). The HTTP protocol parameters for this `HttpClient`.

public void setState(HttpState state). Assigns the HTTP state for the `HttpClient`.

HttpClientActivity. The HTTP Client resource adapter can be used in a synchronous or asynchronous way. `HttpClientActivity` is useful only when the HTTP Client resource adapter is used to send the Request asynchronously.

public String getSessionId(). Gets the unique Session ID.

public void endActivity();. To end the `HttpClientActivity`. If `endOnReceivingResponse` is set to true and if the SBB tries to forcefully end activity by calling `endActivity()`, it should throw an exception. `endActivity()` should only be allowed when `endOnReceivingResponse` is set to false.

public boolean getEndOnReceivingResponse(). Returns true or false depending on the value passed when the request is sent asynchronously.

public void executeMethod(HttpMethod httpMethod). The service that wants to send the request asynchronously first has to create an instance of `HttpMethod` by calling `createHttpMethod()` of `HttpClientResourceAdaptorSbbInterface` The service also creates an activity, attaches itself to this activity, and then calls `executeMethod`, passing the instance of `HttpMethod`.

ResponseEvent. `ResponseEvent` is fired by the HTTP Client resource adapter as soon as `HttpClient.executeMethod()` returns. The response can be a proper response or an exception depending on the environment and application logic.

public Response getResponse(). The interested SBB receives this event and can act on a response.

public Exception getException(). There may be exception due to a network failure or application logic. Interested SBBs can get the exact Exception using this method.

Response. This is the wrapper over the response part of `org.apache.commons.httpclient.HttpMethod`.

public byte[] getResponseBody(). Returns the response body of the HTTP method, if any, as an array of bytes.

public String getResponseBodyAsString(). The response body of the HTTP method, if any, as a String.

public int getStatusCode(). The status code associated with the latest response.

public Header[] getResponseHeaders(). The response headers from the most recent execution of this request.

Example

This is simple example of the code. To look at fully-working example, look at the httpclientra-example at *https://mobicents-examples.dev.java.net/source/browse/mobicents-examples/httpclientra-example/*.

The Synchronous Way to Send a Request

```
//get the resource adapter Sbb Interface
HttpClientResourceAdaptorSbbInterface raSbbInterface = ;

//create HttpMethod by passing the link
HttpMethod httpMethod = raSbbInterface.createHttpMethod("GET",
 "http://www.mobicents.org-a.googlepages.com/index.html"));

//send the request and get a response
Response response = raSbbInterface.executeMethod(httpMethod);

//get ResponseBody
String responseBody = response.getResponseBodyAsString();
```

The Asynchronous Way to Send a Request

```
//get the resource adapter sbb Interface and aci factory
HttpClientResourceAdaptorSbbInterface raSbbInterface = ;
HttpClientActivityContextInterfaceFactory httpClientAci = ;


//create HttpMethod by passing the link
HttpMethod httpMethod = raSbbInterface.createHttpMethod("GET",
 "http://www.mobicents.org-a.googlepages.com/index.html"));

//create HttpClientActivity
HttpClientActivity clientActivity =
 raSbbInterface.createHttpClientActivity(true);

// get the aci for the activity and attach an sbb local object
ActivityContextInterface clientAci =
 httpClientAci.getActivityContextInterface(clientActivity);
clientAci.attach(sbbContext.getSbbLocalObject());


//send a request by calling executeMethod
clientActivity.executeMethod(httpMethod);
```

The response arrives asynchronously

```
public void onResponseEvent(ResponseEvent event,
 ActivityContextInterface aci) {

//Get the Response from Event
Response response = event.getResponse();

//get ResponseBody
```

```
String responseBody = response.getResponseBodyAsString();
```

The HTTP Servlet resource adapter and HTTP Client resource adapter compliment each other in creating a complete SLEE Application capable of receiving requests as well as initiating them.

TBD: eventually remove this: If you have any suggestions/feedback or find a bug please discuss at http://forums.java.net/jive/thread.jspa?messageID=228634&#228634

13. Persistence Resource Adapter

13.1. Installing the Persistence Resource Adapter

The Persistence resource adapter can be installed by referring to *Section 1, "Deploying and Undeploying Resource Adapters"*.

Remember to replace `<resource_adapter>` with the name of the directory in `<topmost_mobicents_directory>/resources` corresponding to the Persistence resource adapter (currently this directory is named `persistencera` , but this could change in the future).

14. XCAP Client Resource Adapter

14.1. Installing the XCAP Client Resource Adapter

The XCAP Client resource adapter can be installed by referring to *Section 1, "Deploying and Undeploying Resource Adapters"*.

Remember to replace `<resource_adapter>` with the name of the directory in `<topmost_mobicents_directory>/resources` corresponding to the XCAP Client resource adapter (currently this directory is named `xcapclientra` , but this could change in the future).

15. Rules Resource Adapter

15.1. Installing the Rules Resource Adapter

The Rules resource adapter can be installed by referring to *Section 1, "Deploying and Undeploying Resource Adapters"*.

Remember to replace `<resource_adapter>` with the name of the directory in `<topmost_mobicents_directory>/resources` corresponding to the Rules resource adapter (currently this directory is named `rulesra` , but this could change in the future).

16. TTS Resource Adapter

16.1. Installing the TTS Resource Adapter

The TTS resource adapter can be installed by referring to *Section 1, “Deploying and Undeploying Resource Adapters”*.

Remember to replace `<resource_adapter>` with the name of the directory in `<topmost_mobicents_directory>/resources` corresponding to the TTS resource adapter (currently this directory is named `ttsra` , but this could change in the future).

17. SAP Resource Adapter

18. Third-Party Resource Adapters

Other Examples

1. Example: SLEE Graph Viewer

2. Example: Call-Blocking, Call-Forwarding and Voice Mail

3. Example: Multiple Services

4. Example: Parent-Child Interaction

5. Example: LDAP-Enabled JSLEE

Mobicents SIP Presence Service

1. Mobicents SIP Presence Service

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

1.1. Architecture of the Mobicents SIP Presence Service

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

Functional Diagram of the Mobicents SIP Presence Service

Figure 7.1. The Mobicents SIP Presence Service

The Three Servers Comprising the Mobicents Presence Service

Section 2, “Mobicents XML Document Management Server”
: The Mobicents XDM Server is a functional element of next-generation IP communications networks is responsible for handling the management of user XML documents stored on the network side, such as presence authorization rules, static presence information, contact and group lists (also known as “resource lists”), policy data, and many others.

Section 3, “Mobicents SIP Presence Server”
: The Mobicents SIP Presence Server is an entity that accepts, stores and distributes Presence Information. The SIP Presence Server performs the following functions:

  - It manages publications from one or multiple Presence Source(s) of a certain Presentity. This includes refreshing Presence Information, replacing existing Presence Information with newly-published information, or removing Presence Information.
  - It manages subscriptions from Watchers to Presence Information and generates notifications about the Presence Information state changes, retrieving the presence authorization rules from the XDM Server.
  - It manages subscriptions from Watcher Information Subscribers to Watcher information and generates notifications about Watcher information state changes.

Section 4, “Mobicents_Resource_List_Server”
: The Resource List Server (RLS) handles subscriptions to Presence Lists. It creates and manages back-end subscriptions to all resources in the Presence List. The list content is retrieved from the XDM Server.

To fill different needs you can deploy all servers separated, or all integrated in the same host, and on top of that, there are JAINSLEE internal client interfaces available for each, which can turn into a big advantage over other presence services.

For documentation on each server proceed to links under this page Resources section.

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

- Issue Tracker
- *Source Code Location* [http://mobicents.googlecode.com/svn/trunk/servers/sip-presence/]
- *Java.net Forums* [http://forums.java.net/jive/category.jspa?categoryID=36]

TBD: This version of the documentation is from *http://groups.google.com/group/mobicents-public/web/mobicents-sip-presence-service*, and the original author is Eduardo Martins, JBoss R & D.

2. Mobicents XML Document Management Server

2.1. Introduction to the XDM Server

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

2.2. Configuring, Installing and Testing the XDM Server

Requirements. The Mobicents XDM Server depends on the Mobicents JAIN SLEE server and the Mobicents SIP and HTTP Servlet resource adapters. Make sure that all of these requirements are deployed before deploying the XDM Server.

The XDM Server is Currently Incompatible with Java 1.6

Currently, the Mobicents XDM Server will not work with Java 1.6, although this will be fixed in the future. Therefore, if you are running the XDM Server, make sure that you are using Java 1.5.

Configuration. The Mobicents XML Document Management Server can be customized in the following aspects before being built.

Customizing the XDM Server

XCAP Root
: The Maven property `xdm.server.xcap.root`, in the `pom.xml` which is inside the Mobicents Presence Service root directory (in `/trunk/servers/sip-presence` to be exact) defines the relative path to root that is considered the XCAP Root for all XCAP URIs. This value must match the servlet name used in the Mobicents HTTP Servlet resource adapter. The default value is *`/mobicents`*.

Dynamic User Provisioning
: The Maven property `dynamicUserProvisioning` in the `pom.xml` inside the Mobicents Presence Service root directory (in `/trunk/servers/sip-presence` to be exact) defines whether the XDM Server should provision the user when a `PUT` request is being processed and the user is not installed for the request's application usage. The default value is true.

Note that you do not need to touch or configure anything in order to deploy the server.

Installing the Mobicents XML Document Management Server. How you install the XDM Server depends on whether you are using the Mobicents binary distribution, or running the source distribution.

- **Installing the XDM Server using the Mobicents Binary Distribution.** TBD: (Fix this with proper instructions): Using the binary release, you can install the XDM Server by doing an ant deploy on the `servers/sip-presence/xdms` directory.

- **Installing the XDM Server after Building Mobicents from Source.** TBD: (Fix this with proper instructions): From the source code repository, you can install the XDM Server by doing a mvn install on the `/trunk/servers/sip-presence/xdms` directory.

Uninstalling the Mobicents XML Document Management Server. How you uninstall the XDM Server depends on whether you are using the Mobicents binary distribution, or running the source distribution.

- **Uninstalling the XDM Server using the Mobicents Binary Distribution.** TBD: (Fix this with proper instructions): Using the binary release, you can uninstall the XDM Server by doing an ant undeploy on the `servers/sip-presence/xdms` directory.
- **Uninstalling the XDM Server after Building Mobicents from Source.** TBD: (Fix this with proper instructions): From the source code repository, you can uninstall the XDM Server by doing a mvn clean on the `/trunk/servers/sip-presence/xdms` directory.

Testing the Mobicents XDM Server. TBD (Make sure this is correct): is recommended that you test the XDM Server after you deploy it. For instructions on how to test the XDM Server, read and follow the instructions in `.../xdms/tests/README.txt`. No tests should fail.

2.3. Functional Architecture of the XDM Soerver

The Mobicents XDM Server includes the following XCAP application usages:

- *IETF Presence Rules (RFC 5025)* [http://tools.ietf.org/html/rfc5025]
- *OMA Presence Rules (OMA Presence Simple v1.1 Candidate Release)* [http://www.openmobilealliance.org/Technical/release_program/Presence_simple_v1_1.aspx]
- *IETF Resource Lists* [http://tools.ietf.org/html/rfc4826]
- *IETF RLS Services (RFC 4826)* [http://tools.ietf.org/html/rfc4826]
- *IETF XCAP-CAPS (RFC 4825)* [http://tools.ietf.org/html/rfc4825]

Functional Architecture of the XML Document Management Server

Figure 7.2. The Mobicents XML Document Management Server

The XDM Server comprises the following functional elements:

Functional Elements of the XDM Server

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

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

Aggregation Proxy
: The aggregation proxy is responsible for handling an XDM client's XCAP requests, which includes authentication and authorization of the requester.

Request Processor
: This element includes the XCAP Server logic to process an XCAP request and return a proper response.

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

Implementation Architecture of the Mobicents XML Document Management Server. The XDM Server is built on top of the Mobicents JAIN SLEE container. This figure depicts the architecture of the XDM Server implementation.

Implementation Architecture of the XML Document Management Server

Figure 7.3. Mobicents XML Document Management Server

The Functional Elements Which Compose the XML Document Management Server

Data Source Resource Adapter
: This Resource Adaptor implements the Data Source functional element.

The `RA Type` defines two activities objects, `DocumentActivity` and `AppUsageActivity`, both of which are used to fire events that signal that a document, element or attribute was updated.

The `RA Type` also defines a Service Building Block (SBB) RA interface to manage the users and documents stored in the XDM Server, and create activities, where events will be fired. The resource adapter will only fire events on activities that exist; that is, the RA won't create activities implicitly if a document is updated.

The `RA Type` also provides a base abstract implementation of the resource adapter, making it very simple to change the underlying resource used to store information, which is by default the internal JDBC datasource of the JBoss Application Server.

AppUsage Cache Resource Adaptor
: This resource adapter stores the XCAP application usages installed in the server.

Each `AppUsage` is an object that includes the logic to validate XCAP documents that result from XCAP requests and are expensive to create; this resource adapter thus provides caching of AppUsages, using a pool model.

The resource adapter doesn't possess events or activities.

AppUsage Service

XCAP Application Usages are installed through a JAIN SLEE service, making it possible to add and/or remove application usages while the server is running.

Aggregation Proxy Service

This JAIN SLEE service implements the aggregation proxy functional element. It handles events fired by the Mobicents HTTP Servlet resource adapter and then uses two child SBBs: the `User Profile Enabler SBB` to retrieve information regarding the user needed for authentication/authorization of the XCAP request, and the `Request Processor SBB`, which handles the XCAP request.

Request Processor SBB

The `Request Processor SBB` implements the request processor functional element, providing a synchronous SBB interface to process XCAP requests. It uses the `AppUsage Cache` resource adapter to borrow AppUsage objects, and the Data Source resource adapter to retrieve or set documents stored in the server's data source.

User Profile Enabler SBB (TBD: not available yet)

This SBB provides a synchronous SBB interface used in JAIN SLEE child relations in order to retrieve user information. Two different implementations of the interface are provided, one considers that the information is stored in the XDM Data Source, another interfaces to a Diameter Sh Server such as IMS HSS.

XCAP Diff Subscription Control Service (TBD: not available yet)

TBD

The implementation architecture figure also contains client-side components:

Client-Side Components of the XML Document Management Server

XCAP Client

The XCAP client is a simple API to interact with an XCAP Server that internally uses the Apache HTTP Client.

The API documentation and example code snippets can be found TBD

XCAP Client Resource Adaptor

The XCAP Client Resource Adaptor adapts the XCAP Client API into JAIN SLEE domain. It provides methods to interact with the XCAP server in both syncronous and asyncronous ways.

The RA Type description and code snippets using the RA can be found here.

XDM Client SBB

The XDMClientSBB is an interface of a JAIN SLEE SBB to be used as a client to the Mobicents XDM Server (and others compliant with same standards), in JAIN SLEE child relations.

Two implementations of this interface are provided:

- `InternalXDMClientSBB` is intended to be used on applications running in the Mobicents XDM Server JAIN SLEE container, and
- `ExternalXDMClientSBB`, which is intended to be used on applications running in a different JAIN SLEE container than the Mobicents XDM Server.

TBD: This version of the documentation is from http://groups.google.com/group/mobicents-public/web/mobicents-xdm-server and the original author is Eduardo Martins, JBoss R&D.

2.4. Resources and Further Information about the XDM Server

For further information on the Mobicents XDM Server, here is a list of additional resources:

- How to Manage the Mobicents XDM Server
- How to Create an XCAP App(lication)Usage
- Integrating XDM in your JAIN SLEE Apps: Code Snippets for XDM Client SBB Usage
- *XCAP Client API Documentation* [http://groups.google.com/group/mobicents-public/web/xcap-client-api]
- *XCAP Client RA Type Description and Example Code Snippets* [http://groups.google.com/group/mobicents-public/web/xcap-client-resource-adaptor]
- Mobicents SIP Presence Service
- Want to contribute?

3. Mobicents SIP Presence Server

3.1. Introduction to the SIP Presence Server

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

The SIP Presence Server is an entity that accepts, stores and distributes SIP Presence Information.

3.2. Configuring, Installing and Testing the SIP Presence Server

Requirements. The Mobicents SIP Presence Server depends on the Mobicents Converged Application Server and the Mobicents SIP resource adapter, which should be deloyed before starting the SIP Presence Server.

Configuration. The Mobicents SIP Presence Server can be customized in the following aspects before being built.

Customizing the SIP Presence Server

XCAP Aware User Agent

The SIP Presence Server calculates the presence-rules XCAP URIs of users in a non-standard way by default, but one which is compatible with CounterPath eyeBeam. You can change this behavior to conform with standards by changing the contents of the `clientUAIsEyebeam` field of `PresenceSubscriptionControlSbb`, which is in `ps-core/subscription-sbb/src/main/java/...`[1]

Note that you do not need to touch or configure anything in order to deploy the server.

Installing the Mobicents SIP Presence Server. TBD (Fix this with proper instructions): Currently, the Mobicents SIP Presence Server can only be installed as integrated with the Mobicents XML Document Management Server, i.e. in an all-in-one SIP Presence Server architecture[2].

Installing the SIP Presence Server Also Installs the XDM Server

You should also make sure that all the install requirements of the Mobicents XDM Server are met before attempting to install the Mobicents SIP Presence Server. See *Requirements*.

Installing the Mobicents XML Document Management Server. How you install the XDM Server depends on whether you are using the Mobicents binary distribution, or running the source distribution.

[1]This is a temporary solution due to a feature in the current JAIN SIP stack that removes invalid User Agent hedaers of SIP requests. In the future, this configuration is exected to be deprecated or removed.

[2]This limitation is due to the current lack of an external XDM Client SBB component.

- **Installing the XDM Server using the Mobicents Binary Distribution.** TBD: (Fix this with proper instructions): Using the binary release, you can install the XDM Server by doing an ant deploy on the `servers/sip-presence/integrated` directory.

- **Installing the XDM Server after Building Mobicents from Source.** TBD: (Fix this with proper instructions): From the source code repository, you can install the XDM Server by doing a mvn install on the `/trunk/servers/sip-presence/integrated` directory.

Uninstalling the Mobicents XML Document Management Server. How you uninstall the XDM Server depends on whether you are using the Mobicents binary distribution, or running the source distribution.

- **Uninstalling the XDM Server using the Mobicents Binary Distribution.** TBD: (Fix this with proper instructions): Using the binary release, you can uninstall the XDM Server by doing an ant undeploy on the `servers/sip-presence/integrated` directory.

- **Uninstalling the XDM Server after Building Mobicents from Source.** TBD: (Fix this with proper instructions): From the source code repository, you can uninstall the XDM Server by doing a mvn clean on the `/trunk/servers/sip-presence/integrated` directory.

Testing the Mobicents SIP Presence Server. A test framework for the Mobicents SIP Presence Server is forthcoming but not available yet.

3.3. Functional Architecture of the SIP Presence Server

Functional Architecture of the Mobicents SIP Presence Server.

Functional Diagram of the Mobicents SIP Presence Server

Figure 7.4. The Mobicents SIP Presence Server

The SIP Presence Server comprises the following functional elements:

The Functional Elements Which Compose the SIP Presence Server

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

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

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

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

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

XDM Client Control
: This last element is responsible for interfacing with the XDM Server that manages the user's XML documents, and is related to the main functions of the presence server. It's capable not only of retrieving a document (or part of one), but also of subscribing to either updates of a single, specific document, or to a full collection of documents of a specific type or application.

Implementation Architecture of the Mobicents SIP Presence Server.

The Mobicents SIP Presence Server

Figure 7.5. Implementation Architecture of the Mobicents SIP Presence Server

The implementation of the Mobicents SIP Presence Server comprises the following functional elements:

The Two Services Which Compose the SIP Presence Server

Presence Publication Control Service
: This JAIN SLEE service includes the root Service Building Block (SBB), `PresencePublicationControlSbb`, which is the implementation of the abstract

SIP event `PublicationControlSbb`. It handles publications on the "presence" event package.

The `PresencePublicationControlSbb` provides the following capabilities:

- It provides the logic to authorize a publication; however, it only authorizes `PUBLISH` requests when the request URI matches the PIDF document “entity” attribute.
- It provides JAXB unmarshellers to validate and parse the PIDF document for the abstract `PublicationControlSbb`.
- It demands that notifying subscribers occur through a child relation to the root SBB of the Presence Subscription Control Service.
- Finally, it also provides an `SbbLocalObject` interface that can be used, in JAIN SLEE child relations, to obtain the composed presence information for a specific resource.

Presence Subscription Control Service.

This JAIN SLEE service includes the root SBB `PresenceSubscriptionControlSbb`, which is the implementation of the abstract SIP Event `SubscriptionControlSbb`. It handles subscriptions on the "presence" event package.

The standout SBB logic item is the usage of presence-rules documents, obtained through the XDM Client SBB child relation, in order to authorize subscriptions and transform the content notified (TBD: feature not used yet). It also defines a child relation to the root SBB of `PresencePublicationService` to retrieve the composed PIDF document for the subscription's notifier.

The SBB also provides an `SbbLocalObject` interface that can be used, in JAIN SLEE child relations, to make the presence event known to the subscribers of a specific resource.

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

Presence Client SBB (TBD: not yet available)

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

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

TBD: This documentation is originally from http://groups.google.com/group/mobicents-public/web/mobicents-sip-presence-server by Eduardo Martins, JBoss R&D.

3.4. Resources and Further Information about the SIP Presence Server

For further information on the Mobicents SIP Presence Server, see the following list of additional resources:

- How to Manage the Mobicents SIP Presence Server
- Integrating the Mobicents SIP Presence Server into Your JAIN SLEE Applications: Code Snippets for Internal Presence Client SBB Usage.
- Integrating the Mobicents SIP Presence Server into Your JAIN SLEE Applications: Code Snippets for External Presence Client SBB Usage.
- *Mobicents Sip Event Publication and Subscription Control Components* [http://groups.google.com/group/mobicents-public/web/mobicents-sip-event-components]

4. Mobicents_Resource_List_Server

4.1. Introduction to the Mobicents Resource List Server

Mobicents Media Server

1. Mobicents Media Server

1.1. Overview of Media Servers

The Reasoning and Need for Media Servers. With the continued progress of globalization, more corporations than ever before have workgroups spread across countries and continents across the world. To support and increase the productivity of remote and telecommuting workgroups, communications companies are considering more cost effective network solutions that combine voice, wireless, data and video functionality. Businesses like these expect that the services they select and eventually implement will have call quality comparable to conventional telephone service, and they expect those services to boost productivity and reduce overall communications costs. Acquiring these desired network services requires connections from the Internet and wireless and wireline networks to Public Switched Telephone Networks (PSTNs) using a flexible, robust, scalable and cost-effective media gateway. The ability of such gateways to reduce overall communications costs for dispersed workgroups forms the foundation for media services and servers.

Media Gateways Bridge Multiple Technologies. Today, all communications can be routed through computers. Widespread access to broadband Internet and the ubiquity of Internet Protocol (IP) enable the convergence of voice, data and video. Media gateways provide the ability to switch voice media between a network and its access point. Using Digital Subscriber Line (DSL) and fast-Internet cable technology, a media gateway converts, compresses and packetizes voice data for transmission back-and-forth across the Internet backbone for wireline and wireless phones. Media gateways sit at the intersection of the PSTNs and wireless or IP-based networks.

The Justification for Media Gateways for VoIP. Multiple market demands are pushing companies to converge all of their media services using media gateways with VoIP capabilities. Companies expect such a convergent architecture to:

Company Expectations of a Convergent Architecture

Lower Initial Costs
: Capital investment is decreased because low-cost commodity hardware can be used for multiple functions.

Lower Development Costs
: Open system hardware and software standards with well-defined applications mean lower costs, and Application Programmable Interfaces (APIs) accelerate development.

Handle Multiple Media Types
: Companies want VoIP solutions today, but also need to choose extensible solutions that will handle video in the near future.

Lower the Costs of Deployment and Maintenance
: Standardized, modular systems reduce training costs and maintenance while also improving uptime.

Enable Rapid Time-to-Market
: Early market entry hits the window of opportunity and maximizes revenue.

What Is the Mobicents Media Server? The Mobicents Media Gateway is an open source Media Server based on the Java Media Framework and aimed to:

- Deliver competitive, complete, best-of-breed media gateway functionality of the highest quality.
- Meet the demands of converged wireless and wireline networks, DSL and cable broadband access, and fixed-mobile converged VoIP networks from a singleand singularly-capablemedia gateway platform.
- Increase flexibility with a media gateway that supports a wide variety of call control protocols and scales to meet the demands of enterprises and small-carrier providers.

1.2. Installing, Building and Uninstalling the MMS

Building the MMS. The server is available as a source code distribution. The source code is available from the server source repository located at http://code.google.com/p/mobicents/source/checkout. The source code availability allows you to debug the server, learn its inner workings and create customized versions for your personal use.

Pre-requisites. Before installing and running the media server, you should check that your have installed Mobicents SLEE server.

Accessing the MMS SVN Source Code Repositories. Access the source code repository for this project in one of following ways:

1. Browse source code online to view this project's directory structure and files: *https://mobicents.googlecode.com/svn/trunk/servers/media*
2. Check out source code with a Subversion client using the following command. Note: replace the last username with your own username:

```
svn checkout http://mobicents.googlecode.com/svn/trunk/
 mobicents-read-only
```

If you are new to Subversion, you may want to visit the *Subversion Project website* [http://subversion.tigris.org/] and read *Version Control with Subversion* [http://svnbook.red-bean.com/].

Subversion Source Code Repository Structure. The top-level directory structure under the /trunk/servers/media source tree is shown in the follwing table:

Path	Description
/trunk/servers/media/constants	Shared library for constants
/trunk/servers/media/jar	Media server SPI and implementation
/trunk/servers/media/sar	JBoss service deployment unit
/trunk/servers/media/mgcpcontrol	MGCP module
/trunk/servers/media/mscontrol	The API for media server control

Building the MMS from Source Code. The build process is driven by maven configuration. The main maven script is the pom.xml file located in the project home directory. The purpose of the main pom.xml file is to compile the various module directories and then to integrate their output to produce the binary release. To build and install media server perform the following actions:

1. **Installing media server endpoints**

   switch to the /servers/media and run mvn install

2. **Installing Media Resource Adaptor with local link**

   switch to the /servers/jain-slee/resources/media and run mvn install

3. **Installing MGCP control module**

   switch to the /servers/jain-slee/resources/mgcp and run mvn install

1.3. Writing and Running Tests Against the Mobicents Media Server

As platform became rich with functions and capabilities it became clear that it requires also a lot of man power to sustain its functionality. To help us achieve as highest availability and stability we developed tests.

MMS has two different set of tests:

In-Container Tests
: Tests which require setup of full container, same JVM as MMS.

Components Tests
: Regular junit tests for components that not require MMS to be in running state

Test in both cases are driven by maven plugins - as whole build depends on maven. However in case of incontainer tests its different - this module is not enabled. This default setup is due to requirement of incontainer test. In future we will make effort to automate whole process without any need to modify poms. Incontainer tests require MMS to be deployed in jboss. Despite availability of “cargo” plugin we were unsuccessfull on integrating both test types in one module due to “cactus” for m2, being still in alpha state.

Incontainer tests reside is separate module as they require 3rd party libraries and framework to act. It can be found in `trunk/servers/media/tests`

Writing In-Container Tests. This is fairly easy. We use cactus to run tests within contianer. Here are basic steps to acomplish:

1. Create the appropriate pacakge in `trunk/servers/media/tests/src/main/java`.
2. Create regular junit test case - but extends org.apache.cactus.ServletTestCase.
3. Create testXXX methods, setUp, tearDown, etc - just as in regular test case - but keep in mind, it runs within container, so all local container resources are available - like JNDI entries, envirment properties.
4. Make public static Test suite() method return instance of org.apache.cactus.ServletTestSuite object.

Writing Regular Tests. This is straightforward as those tests are regular junit tests, run with surefire plugin.

Running the In-Container Tests. As mentioned running those tests needs a little bit of setup. Here they are:

1. Deploy MMS with mvn install command invoked from `trunk/server/media`.
2. Change the directory to `tests` (its a mvn module).
3. Set proper container home and proper containerId (see cargo reference manual) in `pom.xml`:

```
<container>
 <containerId>jboss42x</containerId>
 <home>D:\java\servers\jboss-4.2.2.GA</home>
</container>
```

Example 8.1. Setting the Container Home and containerId

4. Simply do mvn install - this will create war file, deploy it, run container, run tests.

5. Do mvn clean to remove tests war from container.

In the future incotnainer run should be simpler, without need of any configuration. However please keep in mind that there are plans to separate MMS entirely from container.

Running Regular Tests. Those test are usually run while MMS is beeing build, however it is possible to run them on demand.

Simply move to `trunk/servers/media/jar` and type mvn test.

2. Mobicents Media Server Architecture

Media services have played an important role in the traditional Time Division Multiplexing (TDM)-based telephone network. As the network migrates to an Internet Protocol (IP)-based environment, media services are also moving to new environments.

One of the most exciting trends is the emergence and adoption of complementary modular standards that leverage the Internet to enable media services to be developed, deployed and updated more rapidly than before in a network architecture that supports the two concepts we will call provisioning-on-demand and scaling-on-demand.

2.1. Design Overview and Typical Deployment Scenario

General Design Overview. Mobicents Media Server is developed on top of existing Java technologies. The Java platform is the ideal platform for network computing. It offers a single, unifying programming model that can connect all elements of a business infrastructure. The modularization effort is supported by the use of the Java Management Extension (JMX) API, and the industry-standard Service Logic Execution Environment (SLEE) container. Using JMX enables easy management of both the server's media components and the control modules hosted by SLEE.

This high degree of modularity benefits the application developer in several ways. The already-tight code can be further trimmed down to support applications that must have small footprints. For example, if PSTN (Public Switched Telephone Network) interconnection is unnecessary in your application, then simply take the D-channel feature out of the server. If you later decide to deploy the same application within a Signaling System 7 (SS7) network, then simply enable the appropriate endpoint. Another example is the freedom you have to drop your favorite media control protocol directly into the SLEE container.

Figure 8.1. Media Server Overview

The components that are involved in media processing or endpoints are implemented with JMF, the Java Media Framework. JMF provides a platform-neutral framework for displaying time-sensitive media. The JMF API:

- Scales across different protocols and delivery mechanisms
- Scales across different types of media data
- Provides an event model for asynchnronous communication between JMF and applications or applets

The JMF architecture allows advanced developers nto create and integrate new types of controllers and data sources. The JMF API declares abstract interfaces which are used for handling new media sources and sinks such as audio and video files, PSTN cards, webcams, etc.

The Media Server architecture assumes that call control intelligence is outside of the Media Server and handled by an external entity. The Mebia Server assumes that call controllers will use control procedures such as MGCP, Mecago or MSML, among others. Each specific control module can be plugged in directly to the server as a standard SLEE deployable unit. The usage of a SLEE container for the implementation of the control protocol-specific communication logic provides simple deployment and also the easy deployment of such control modules. The developer will not have to mess with low-level transaction and state management details, multi-threading, connection-pooling and other similar complex, low-level APIs.

Note

The Mobicents Media Server uses SLEE for implementing its own communication capabilities. The SLEE container doesn't serve here as a call controller.

In addition to control protocol modules, the SLEE container is aimed at providing high-level features like Interactive Voice Response (IVR) and Drools or VoiceXML engines.

The modules deployed under SLEE control interact with the Media Server Service Provider Interface (SPI) through the Media Server Control Resource Adapter, or MSC-RA. The MSC-RA follows the recommendations of JSR-309 and implements asynchronous interconnection with the Media Server SPI stack. This local implementation is restricted and does not all the use of high-level abstractions like VoiceXML dialogs, etc..

Typical Deployment Scenario. Mobicents Media Server offers a complete solution for all aspects of being a media gateway and server. This includes the Digital Signal Processors required to covert and compress TDB voice circuits into IP packets, announcement access points, conferencing, high-level IVR engines, etc. The gateway is able to provide signaling conversation and can operate as a Session Border Controller at the boundaries of Local Access Networks. The Media Server is always controlled by an external JBCP application server which implements the call control logic.

Typical Deployment Scenario

2.2. Endpoints

Endpoints. It is convenient to consider a media gateway as a collection of endpoints. An endpoint is a logical representation of a physical entity such as an analog phone or a channel in a trunk. Endpoints are sources or sinks of data and can be physical or virtual. Physical endpoint creation requires hardware installation whil esoftware is sufficient for creating a virtual endpoint. An interface on a gateway that terminates a trunk connected to a PSTN switch is an example of a physical endpoint. An audio source in an audio content server is an example of a virtual endpoint.

The type of the endpoint determines its functionality. Our analysis, so far, has led us to isolate the following basic endpoint types:

- Digital Signal 0 (DS0)
- Analog line
- Announcement server access point
- Conference bridge acces point
- Packet relay
- Asynchronous Transfer Mode (ATM) "trunk side" interface

This list is not exhaustive: there may be other types of endpoints defined in the future, for example test endpoints that could be used to check network quality, or frame-relay endpoints that could be used to manage audio channels multiplexed over a frame-relay virtual circuit.

Description of Various Access Point Types

Announcement Server Access Point

An announcement server endpoint naturally provides access to an announcement server. Upon receiving requests from the call agent, the announcement server will "play" a specified announcement. A given announcement endpoint is not expected to support more than one connection at a time. Connections to an announcement server are typically one-way, or "half-duplex":the announcement server is not expected to listen to the audio signals from the connection.

Interactive Voice Response Access Point
: An Interactive Voice Repsonse (IVR) endpoint provides access to an IVR service. Upon requests from the call agent, the IVR server will "play" announcements and tones, and will "listen" to responses, such as (DTMF) input or voice messages, from the user. A given IVR endpoint is not expected to support more than one connection at a time.

Conference Bridge Access Point
: A conference bridge endpoint is used to provide access to a specific conference. Media gateways should be able to establish several connections between the endpoint and the packet networks, or between the endpoint and other endpoints in the same gateway. The signals originating from these connections shall be mixed according to the connection "mode" (as specified later in this document). The precise number of connections that an endpoint supports is characteristic of the gateway, and may in fact vary according to the alloction of resources within the gateway.

Packet Relay
: A packet relay endpoint is a specific form of conference bridge that typically only supports two connections. Packet relays can be found in firewalls between a protected and an open network, or in transcoding servers used to provide interoperation between incompatible gateways, such as gateways which don't support compatible compression algorithms, or gateways that operate over different transmission networks such as IP and ATM.

Signal Generators (SGs) and Signal Detectors (SDs). This endpoint contains a set of resources which provide the media-processing functionality. It manages the interconnection of media streams between the resources, and arbitrates the flow of media stream data between them. Media servicesor commands are invoked by a client application on the endpoint; that endpoint causes the resources to perform the desired services, and directs events sent by the resources to the appropriate client. The resources in the endpoint include a primary resource and zero or more secondary resources. The primary resource is typically connected to an external media stream, and provides the data from that stream to the secondary resources. The secondary resources may process that stream (e.g. by recording it and/or performing automatic speech recognition on it), or may themselves generate generate media stream data (e.g. by playing a voice file) which is then transmitted to the primary resource.

A resource is statically prepared if the preparation takes place at the time of creation. A resource is dynamically prepared if preparation of a particular resource (and its associated media streams) does not occur until it is required by a media operation. Static preparation can lead to less efficient usage of a server's resources, because resources may tend to be allocated for a longer time before use. However, once a resource has been prepared, it is guaranteed to be available for use. Dynamic preparation may utilize resources more efficiently because of Just-In-Time (JIT) allocation algorithms may be used.

An endpoint is divided logically into a Service Provider Interface (SPI) that is used to implement specific a endpoint, and a management interface which is used for implementing manageable resources of that endpoint. All endpoints are plugged into the Mobicents SLEE server by registering with the MBean server. The kernel in that sense is only an aggregator, and not a source of actual functionality. The functionality is provided by the SPI implementation of the Mbeans, and in fact, all major endpoints are manageable MBeans interconnected through the MBean server. The best way to add endpoints to a Media Server is to write a new JMX bean which provides implementation of the endpoint's SPI.

The SPI layer is an abstraction that endpoint providers must implement to enable their media-processing features. An implementation of SPI for an endpoint is referred to as an *Endpoint Provider*.

Figure 8.2. EndpointManagementMBean UML Diagram

2.3. Endpoint Identifiers

The endpoint is identified by its local name. The syntax of the local name depends on the type of endpoint being named. However, the local name for each of these types is naturally hierarchical, beginning with a term which identifies the physical gateway containing the given endpoint, and ending in a term which specifies the individual endpoint concerned. With this in mind, the JNDI naming rules are applied to the endpoint identifiers

2.4. Connections

Connections are created on the call agent on each endpoint that will be involved in the "call." In the classic example of a connection between two "DS0" endpoints (EP1 and EP2), the call agents controlling the end points will establish two connections (C1 and C2):

Media Server Connections

Each connection will be designated locally by a connection identifier, and will be characterized by connection attributes.

Resources and Connection Attributes. Many types of resources will be associated to a connection, such as specific signal processing functions or packetization functions. Generally, these resources fall in two categories:

Two Types of Resources

Externally-Visible Resources
: Externally visible resources, that affect the format of "the bits on the network" and must be communicated to the second endpoint involved in the connection.

Internal Resources
: Internal resources, that determine which signal is being sent over the connection and how the received signals are processed by the endpoint

The resources allocated to a connection, and more generally the handling of the connection, are chosen by the media server under instructions from the call agent. The call agent will provide these instructions by sending two set of parameters to the media server:

- The local directives instruct the gateway on the choice of resources that should be used for a connection.
- When available, the "session description" provided by the other end of the connection.

The local directives specify such parameters as the mode of the connection (e.g. send only, send-receive), preferred coding or packetization methods, usage of echo cancellation or silence suppression. (A detailed list can be found in the specification of the LocalConnectionOptions parameter of the CreateConnection command.) For each of these parameters, the call agent can either specify a value, a range of value, or no value at all. This allow various implementations to implement various level of control, from a very tight control where the call agent specifies minute details of the connection handling to a very loose control where the call agent only specifies broad guidelines, such as the maximum bandwidth, and let the gateway choose the detailed values.

Based on the value of the local directives, the gateway will determine the resources allocated to the connection. When this is possible, the gateway will choose values that are in line with the remote session description - but there is no absolute requirement that the parameters be exactly the same.

Once the resource have been allocated, the gateway will compose a "session description" that describes the way it intends to receive packets. Note that the session description may in some cases present a range of values. For example, if the gateway is ready to accept one of several compression algorithm, it can provide a list of these accepted algorithms.

Local Connections are a Special Case. Large gateways include a large number of endpoints which are often of different types. In some networks, we may often have to setup connections between endpoints that are located within the same gateway. Examples of such connections may be:

- Connecting a trunk line to a wiretap device,
- Connecting a call to an Interactive Voice-Response unit
- Connecting a call to a Conferencing unit
- Routing a call from on endpoint to another, something often described as a "hairpin" connection.

Local connections are much simpler to establish than network connections. In most cases, the connection will be established through some local interconnecting device, such as for example a TDM bus.

2.5. Events and Signals

The concept of events and signals is central to the Media Server. A Call Controller may ask to be notified about certain events occurring in an endpoint (e.g., off-hook events) by passing event's identifier as parameter to endpoint's subscribe(...) method.

A Call Controller may also request certain signals to be applied to an endpoint (e.g., dial-tone) by supplying the identifier of the event in as argument to endpoint's apply(...) method.

Events and signals are grouped in packages, within which they share the same name space which we will refer to as event identifier in the following. Event identifiers are integer constants. Some of the events may be parametrized with additional data such as DTMF mask.

Signals are divided into different types depending on their behavior:

Types of Signals

On/off (OO)
: Once applied, these signals last until they are turned off. This can only happen as the result of a reboot/restart or a new signal request where the signal is explicitly turned off. Signals of type OO are defined to be idempotent, thus multiple requests to turn a given OO signal on (or off) are perfectly valid. An On/Off signal could be a visual message-waiting indicator (VMWI). Once turned on, it MUST NOT be turned off until explicitly instructed to by the Call Agent, or as a result of an endpoint restart, i.e., these signals will not turn off as a result of the detection of a requested event.

Time-out (TO)
: Once applied, these signals last until they are either cancelled (by the occurrence of an event or by explicit releasing of signal generator), or a signal-specific period of time has elapsed. A TO signal that times out will generate an "operation complete" event. If an event occurs prior to the 180

seconds, the signal will, by default, be stopped (the "Keep signals active" action - will override this behavior). If the signal is not stopped, the signal will time out, stop and generate an "operation complete" event, about which the server controller may or may not have requested to be notified. A TO signal that fails after being started, but before having generated an "operation complete" event will generate an "operation failure" event which will include the name of the signal that failed. Deletion of a connection with an active TO signal will result in such a failure.

Brief (BR)
: The duration of these signals is normally so short that they stop on their own. If a signal stopping event occurs, or a new signal requests is applied, a currently active BR signal will not stop. However, any pending BR signals not yet applied will be cancelled (a BR signal becomes pending if a signal request includes a BR signal, and there is already an active BR signal). As an example, a brief tone could be a DTMF digit. If the DTMF digit "1" is currently being played, and a signal stopping event occurs, the "1" would play to completion. If a request to play DTMF digit "2" arrives before DTMF digit "1" finishes playing, DTMF digit "2" would become pending.

Signal(s) may be generated on endpoints or on connections. To each event is associated one or more actions, which can be:

- Notify the event immediately, together with the accumulated list of observed events
- Accumulate the event in an event buffer, but don't notify yet
- Keep Signal(s) active
- Ignore the event

3. MMS Configuration

All endpoints are plugged into the Mobicents SLEE server by registering with the MBean server. After the Media server has succesfully started, you can then locate the JMX console at *http://localhost:8080/jmx-console* in the default distribution. Note that if you have configured the servlet container (i.e. Tomcat) to service a different port, then you will need to supply a different port number in the URL.

3.1. Announcement Server Access Points

An Announcement Server endpoint provides access to an announcement service. Upon requests from the Call Agent, an Announcement Server will “play” a specified announcement. A given announcement endpoint is not expected to support more than one connection at a time. Connections to an Announcement Server are typically one-way, i.e. “half-duplex”: the Announcement Server is not expected to listen to audio signals from the connection.

Configuration of an Announcement Server Access Point. The configurable attributes of the Announcement Server are as follows:

JndiName	The Java Naming and Directory Interface (JNDI) name under which the endpoint is to be bound.
BindAddress	The IP address to which this endpoint is bound.
Jitter	The size of jitter buffer in milliseconds.
PacketizationPeriod	The period of media stream packetization in milliseconds.
PCMA	Enable or disable G711 (A-law) codec support.
PCMU	Enable or disable G711 (U-law) codec support.

Supported Media Types and Formats. The supported media types and formats are listed as follows:

WAVE (.wav)	16-bit mono/stereo linear

Supported RTP Formats. This endpoint is able to receive the following RTP media types:

Media Type	Payload Number
Audio: G711 (A-law) 8bit, 8kHz	8
Audio: G711 (U-law) 8bit, 8kHz	0

Supported Packages. The supported packages are as follows:

- `org.mobicents.media.server.spi.events.Announcement`

3.2. Interactive Voice Response

An Interactive Voice Response (IVR) endpoint provides access to an IVR service. Upon requests from the Call Agent, the IVR server “plays” announcements and tones, and “listens” to voice messages from the user. A given IVR endpoint is not expected to support more than one connection at a time. For example, if several connections were established to the same endpoint, then the same tones and announcements would be played simultaneously over all connections.

```
<mbean
 code="org.mobicents.media.server.impl.ivr.IVREndpointManagement"
 name="media.mobicents:endpoint=ivr">
 <attribute
  name="JndiName">media/endpoint/IVR</attribute>
 <attribute
  name="BindAddress">${jboss.bind.address}</attribute>
 <attribute
  name="Jitter">60</attribute>
 <attribute
  name="PacketizationPeriod">20</attribute>
 <attribute
  name="PortRange">1024-65535</attribute>
 <attribute
  name="PCMU">true</attribute>
 <attribute
  name="PCMA">false</attribute>
 <attribute
  name="MediaType">audio.x_wav</attribute>
 <attribute
  name="RecordDir">${jboss.server.data.dir}</attribute>
 <attribute
  name="DTMF">detector.mode=AUTO</attribute>
</mbean>
```

Example 8.2. The IVREndpointManagement MBean

Configuration of the Interactive Voice Response Endpoint. The configurable attributes of the Interactive Voice Response endpoint are as follows:

JndiName
: The Java Naming and Directory Interface (JNDI) name under which endpoint is to be bound.

BindAddress
: The IP address to which this endpoint is to be bound.

Jitter
: The size of jitter buffer in milliseconds.

PacketizationPeriod
: The period of media stream packetization in milliseconds.

PCMA
: Enable or disable G711 (A-law) codec support.

PCMU
: Enable or disable G711 (U-law) codec support.

RecordDir
: The directory where the recorded files should be created and stored.

DTMF
: The dual-tone multi-frequency (DTMF) type supported. By default it is set to AUTO, but you can also specify INBAND or RFC2833. Note that if you select RFC2833, you *also* need to specify the DTMF Paylod property. For example:

```
<attribute
 name="DTMF">
detector.mode=INBAND
dtmf.payload = 101
</attribute>
```

detector.mode
: Configures DTMF detector mode. Possible values are AUTO, INBAND or RFC2833.

dtmf.payload
: Configures the payload number if RFC2833 mode is specified.

Supported Media Types and Formats. The supported media types and formats are listed as follows:

WAVE (.wav)
: 16-bit mono/stereo linear

Supported RTP Formats. The endpoint is able to receive the follwing RTP media types:

Media Type	Payload Number
Audio: G711 (A-law) 8bit, 8kHz	8
Audio: G711 (U-law) 8bit, 8kHz	0

Record Directory Configuration. You can specify the common directory where all the recorded files should be stored, or simply omit this attribute, in which case the default directory is null, and the application needs to pass an absolute directory path to record to.

Supported Packages. The supported packages are as follows:

- `org.mobicents.media.server.spi.events.Announcement`

- `org.mobicents.media.server.spi.events.Basic`
- `org.mobicents.media.server.spi.events.AU`

3.3. Packet Relay Endpoint

A packet relay endpoint is a specific form of conference bridge that typically only supports two connections. Packets relays can be found in firewalls between a protected and an open network, or in transcoding servers used to provide interoperation between incompatible gateways (for example, gateways which do not support compatible compression algorithms, or gateways which operate over different transmission networks such as IP or ATM).

Configuration of the Packet Relay Endpoint. The configurable attributes of the Packet Relay endpoint are as follows:

JndiName
: The JNDI name under which endpoint is to be bound.

BindAddress
: The IP address to which this endpoint is bound.

Jitter
: The size of jitter buffer in milliseconds.

PacketizationPeriod
: The period of media stream packetization in milliseconds.

PCMA
: Enable or disable G711 (A-law) codec support.

PCMU
: Enable or disable G711 (U-law) codec support.

Supported RTP Formats. This endpoint is able to receive the follwing RTP media types:

Media Type	Payload Number
Audio: G711 (A-law) 8bit, 8kHz	8
Audio: G711 (U-law) 8bit, 8kHz	0

DTMF Configuration. The dual-tone multi-frequency (DTMF) configuration is determined by the DTMF attribute. The properties are as follows:

detector.mode
: Configures DTMF detector mode. Possible values are AUTO, INBAND or RFC2833.

dtmf.payload
: Configures the payload number *if* RFC2833 mode is *also* specified.

Supported Packages. The supported packages are as follows:

- `org.mobicents.media.server.spi.events.Basic`

3.4. Conference Bridge Endpoint

The Mobicents Media Server should be able to establish several connections between the endpoint and packet networks, or between the endpoint and other endpoints in the same gateway. The signals originating from these connections shall be mixed according to the connection “mode”, as specified later in this document. *TBD: prev. sentence: where?* The precise number of connections an endpoint supports is a characteristic of the gateway, and may in fact vary according with the allocation of resources within the gateway.

Configuration of the Conference Bridge Endpoint. The configurable attributes of the Conference Bridge endpoint are as follows:

JndiName
: The JNDI name under which endpoint is to be bound.

BindAddress
: The IP address to which this endpoint will be bound.

Jitter
: The size of jitter buffer in milliseconds.

PacketizationPeriod
: The period of media stream packetization in milliseconds.

PCMA
: Enable or disable G711 (A-law) codec support.

PCMU
: Enable or disable G711 (U-law) codec support.

Supported RTP Formats. This endpoint is able to receive the follwing RTP media types:

Media Type	Payload Number
Audio: G711 (A-law) 8bit, 8kHz	8
Audio: G711 (U-law) 8bit, 8kHz	0

DTMF Configuration. The dual-tone multi-frequency (DTMF) configuration is determined by DTMF attribute. The properties are as follows:

detector.mode
: Configures DTMF detector mode. Possible values are AUTO, INBAND or RFC2833.

`dtmf.payload`
: Configures DTMF detector mode. Possible values are AUTO, INBAND and RFC2833.

Supported Packages. The supported packages are as follows:

- `org.mobicents.media.server.spi.events.Basic`

4. MMS Control Protocols

Media Server assumes a call control architecture where the call control “intelligence” is outside the Media Server and handled by external call control elements known as Call State Control Function.The media server assumes that these call control elements, or CSCF,will synchronize with each other to send coherent commands and responses to the media servers under their control. Server Control Protocols is, in essence, a master/slave asynchronous protocol, where the Server Control Modules are expected to execute commands sent by CSCF. Each Server Control Module is implemented as JSLEE application and consist of a set of SBBs which are in charge to communicate with media server's endpoints via SPI. It allows to avoid difficulties with programming concurrency, low level transaction and state management details, connection pooling and other complex-level APIs.

The MGCP Module. The MGCP module is included in default binary distribution. The Call Agent uses MGCP to tell the Media Server:

- what events should be reported to the Call Agent
- how endpoints should be connected together
- what signals should be played on endpoints

Configuring the MGCP Module. The default port for MGCP is 2728. You can mobicents management console to override the default value.

5. MMS Control API

The main objective of the Media Server control API is to provide multimedia application developers with a Media Server abstraction interface.

5.1. Basic Components of the MMS API

In this section we describes the basic objects of the API as well as the common design patterns.

The API components consist of a related set of interfaces, classes, operations, events, capabilities, and exceptions. The API provides seven key objects, which are common MS and more advanced packages. We provide a very brief description of the API in this overview document; The seven key objects are:

MsProvider
: Represents the “window” through which an application views the call processing.

MsSession
: Represents a call and is a dynamic “collection of physical and logical entities” that bring two or more endpoints together.

MsEndpoint
: Represents a logical endpoint (e.g., announcement access server, IVR).

MsConnection
: Represents the dynamic relationship between a MsSession and an UA.

MsLink
: Represent the dynamic relationship between a two endpoint located on same Media server

MsSignalGenerator
: Represents the media resource which is responsible for generating media

MsSignalDetector
: Represents the media resource which is responsible for detecting media.

The purpose of a MsConnection object is to describe the relationship between a MsSession object and an UA. A MsConnection object exists if the UA is a part of the media session. MsConnection objects are immutable in terms of their MsSession and UA references. In other words, the MsSession and UA object references do not change throughout the lifetime of the MsConnection object instance. The same MsConnection object may not be used in another MsSession.

MsProvider can be used to create the MsSession as well as SignalGenerator and SignalDetector.

MsSession is a transient association of (zero or more) connection for the purposes of engaging in a real-time communications interchange. The session and its associated connection objects describe the control and media flows taking place in a communication network. Applications create instances of a MsSession object with the MsProvider.createSession() method, which returns a MsSession object that has zero connections and is in the IDLE state. The MsProvider object instance does not change throughout the lifetime of the MsSession object. The MsProvider associated with a MsSession is obtained via the getProvider() method.

Application creates instance of MsConnection object with the MsSession.createNetworkConnection(String endpointName). The Application calls MsConnection.modify(String localDesc, String remoteDesc) passing the local SDP and remote SDP. The MsConnection at this time will find out corresponding EndPoint from JNDI using endPointName passed to it and call createConnection(int mode) to create instance of Connection. This Connection creates the instance of RtpSocketAdaptorImpl which opens up the socket for RTP data transfer. The transfer of data has not yet started.

Application creates instance of MsLink object with the MsSession.createLink(MsLinkMode mode). The Application calls MsLink.join(String endpointName1, String endpointName2) passing the endpoint names of two local endpoint to be joined. The MsLink at this time will find out corresponding EndPoint's from JNDI using endPointName passed to it and call createConnection(int mode) to create instance of Connection. This Connection creates the instance of RtpSocketAdaptorImpl which opens up the socket for RTP data transfer. The transfer of data has not yet started. As soon as Connections are created for both the Endpoint's, setOtherParty(Connection other) is called on each Connection passing the other Connection which starts the data transfer between two Connection's.

The Application creates instance of MsSignalGenerator using MsProvider.getSignalGenerator(endpointName) passing endPointName. Application calls MsGenerator.apply(Announcement.PLAY, new String[] { Audio file URL }) to play the audio file. At this point MsGenerator looks up the EndPoint corresponding the endpointName passed and simply calls play on it to start the transmission of audio via RTP.

5.2. Basic API Patterns: Listeners

The basic programming pattern of the API is that applications (which reside “above” the API) make synchronous calls to API methods. The platform or network element implementing the API can inform the application of underlying events (e.g. the arrival of incoming calls) by means of Java events. The application provides Listener objects corresponding to the events that it is interested in obtaining.

Listeners

MsSessionListener
: For underlying application that is interested in receiving events for state change of MsSession should implement MsSessionListener.

MsConnectionListener
: For underlying application that is interested in receiving events for state change of MsConnection should implement MsConnectionListener.

MsLinkListener

For underlying application that is interested in receiving events for state change of MsLink should implement MsLinkListener.

MsResourceListener

For underlying application that is interested in receiving events for state change of MsSignalDetector or MsSignalGenerator should implement MsResourceListener.

5.3. Events

Each of the listener's defined above are listening to different type's of events that are fired by API.

Events related to MsSession. MsSessionListener is listening for MsSessionEvent which carries the MsSessionEventID that represents state change of MsSession. The following table shows the different types of MsSessionEventID, when these events would be fired and corresponding methods of MsSessionListener that will be called.

MsSessionEventID	Description	MsSessionListener Method Called
SESSION_CREATED	Fired when MsProvider.createSession() is called and a new MsSession is created	public void sessionCreated(MsSessionEvent evt)
SESSION_CREATED	Fired when MsProvider.createSession() is called and a new MsSession is created	public void sessionCreated(MsSessionEvent evt)
SESSION_ACTIVE	When the MsConnection or MsLink is created on MsSession for first time it transitions to ACTIVE state and SESSION_ACTIVE is fired. After this the state remains ACTIVE even if application creates more MsConnections or MsLinks	public void sessionActive(MsSessionEvent evt)
SESSION_INVALID	When all the MsConnection or MsLink objects are disassociated from MsSession, it transitions to INVALID state and	public void sessionInvalid(MsSessionEvent evt)

MsSessionEventID	Description	MsSessionListener Method Called
	SESSION_INVALID is fired	

Events Related to MsConnection. MsConnectionListener is listening for MsConnectionEvent which carries the MsConnectionEventID that represents state change of MsConnection. The following table shows the different types of MsConnectionEventID and when these events would be fired and corresponding methods of MsConnectionListener that will be called.

MsConnectionEventID	Description	MsConnectionListener Method Called
CONNECTION_INITIALIZED	As soon as new MsConnection is created by calling MsSession.createNetworkConnection(String endpointName) CONNECTION_INITIALIZED is fired	public void connectionInitialized(MsConnection event)
CONNECTION_CREATED	As soon as creation of RTP connection (by calling MsConnection.modify(String localDesc, String remoteDesc)) is successful, CONNECTION_CREATED is fired. When modify() is called, MsConnection checks if there is Endpoint associated with it and if there is no Endpoint means this is creation of RTP connection request	public void connectionCreated(MsConnectionE event)
CONNECTION_MODIFIED	As soon as modification of MsConnection (by calling MsConnection.modify(String localDesc, String remoteDesc)) is successful CONNECTION_MODIFIED is fired. When modify() is called, MsConnection checks if there is Endpoint associated it and if there is Enpoint means this is modification request.	public void connectionModifed(MsConnectionE event)
CONNECTION_DELETED	As soon as MsConnection is successfully released	public void connectionDeleted(MsConnectionE event)

MsConnectionEventID	Description	MsConnectionListener Method Called
	(MsConnection.release()) is fired	CONNECTION_DELETED
@Deprecated TX_FAILED	When ever creation of new RTP connection or modification of existing MsConnection fails, TX_FAILED is fired	public void txFailed(MsConnectionEvent event)

Events Related to MsLink. MsLinkListener is listening for MsLinkEvent which carries the MsLinkEventID that represents state change of MsLink. The following table shows the different types of MsLinkEventID and when these events would be fired and corresponding methods of MsLinkListener that will be called.

MsLinkEventID	Description	MsLinkListener Method Called
LINK_CREATED	As soon as new MsLink is created by calling MsSession.createLink(MsLinkMode mode) LINK_CREATED is fired	public void linkCreated(MsLinkEvent evt)
LINK_JOINED	Fired as soon as join(String a, String b) operation of MsLink is successful	public void linkJoined(MsLinkEvent evt)
INK_DROPPED	Fired as soon as release() operation of MsLink is successful	public void linkDropped(MsLinkEvent evt)
LINK_FAILED	Fired as soon as join(String a, String b) operation of MsLink fails	public void linkFailed(MsLinkEvent evt)

5.4. MSC API Objects: Finite State Machines

MsSessionState Finite State Machine. The behavior of the MsSession is specified in terms of Finite State Machines (FSMs) represented by MsSessionState, shown below.

IDLE
: This state indicates the Session has zero connections or links.

ACTIVE
: This state indicates the Session has one or more connections or links.

INVALID
: This state indicates the Session has lost all of its connections or links.

MsConnection Finite State Machine. MsConnection state is represented by MsConnectionState enum

IDLE
: This state indicates that the MsConnection is just created and has no resources attached to it

HALF_OPEN
: This state indicates that the MsConnection has created the RTP Socket but doesn't yet have any information of Remote SDP to send the RTP Packets. MsConnection is still usable in HALF_OPEN state if its only receiving the RTP Packets but doesn't have to send any

OPEN
: state indicates that the MsConnection now has information of remote SDP and can send RTP Packates to the remote IP (for example UA)

FAILED
: This state indicates that creation or modification of MsConnection failed and is not reusable anymore

CLOSED
: state indicates that MsConnection has released all its resources and closed the RTP sockets. Its not usable any more.

MsLink Finite State Machine. MsLink state is represented by MsLinkState enum:

IDLE
: This state indicates that the MsLink is created and has no endpoints associated with it

JOINED
: This state indicates that the Connection's from both the Endpoint's are created and data transfer has started

FAILED
: This state indicates that the creation of MsLink failed and is not usable anymore

INVALID
: state indicates that MsLink has closed Connection's of both the Endpoint's and is not usable anymore

5.5. API Methods and Usage

So far we have specified the key objects as well as their FSMs. To understand operationally how these objects are used and the methods they offer, we refer the user to the UML sequence diagram examples. The following call flow depicts the simple announcement.

TBD: Replace this orderedlist with a callout list once the graphic is remade.

1. application receives underlying signaling message
2. application registers listeners
3. application registers listeners
4. application registers listeners
5. application creates MsSession object.
6. application creates MsConnection object using MsSession object.
7. application modify MsConnection passing SDP descriptor received on signaling channel.
8. MsConnection implementation sends request to the Media Server using one of the control protocol.
9. Server responds that connection on the Media server is created
10. Application receive ConnectionEvent.CONNECTION_CREATED
11. application obtains server's SDP and sends response to the UA. Media conversation started.
12. Application creates SignalGenerator and ask it to play announcement
13. Application creates SignalGenerator and ask it to play announcement
14. Application creates SignalGenerator and ask it to play announcement
15. Server report that announcement complete.
16. Server report that announcement complete.

```
package org.mobicents.mscontrol;

import org.mobicents.media.msc.common.MsLinkMode;
import org.mobicents.media.server.impl.common.events.EventID;
import org.mobicents.mscontrol.impl.MsProviderImpl;

/**
 * This is just a psuedocode to show how to use the MSC Api. This
 example uses
 * the Packet Relay Endpoint and Announcement Endpoint as follows
 *
 * UA <----> RTP Connection <---- (one side) Packet Relay Endpoint
 (other side) <-----> MsLink  <----> Announcement Endpoint
 *
 * @author amit bhayani
 *
 */
public class MyMSCTest implements MsSessionListener,
 MsConnectionListener, MsLinkListener {

    private MsSession session;
    private MsProvider msProvider;

    public void startMedia() {

        // Creating the provider
        MsProvider provider = new MsProviderImpl();

        // Registering the Listeners
        provider.addSessionListener(this);
        provider.addConnectionListener(this);
        provider.addLinkListener(this);

        // Creating the Session
        session = provider.createSession();

        // Creating the connection passing the Endpoint Name. Here
 we are
        // creating Packet Relay Endpoint which has only two
 connections. One is
        // connected to UA and other we can connect to Announcement
 Endpoint
        MsConnection connection =
 session.createNetworkConnection("media/trunk/PacketRelay/$");

        // Get the Remote SDP here and pass it to connection. If
 creation of
        // connection is successful connectionCreated method will
 be called
        connection.modify("$", remoteDesc);
    }

    public void sessionActive(MsSessionEvent evt) {

    }
```

Example 8.3. Example Code

6. MMS Event Packages

The Basic Package. Package name: `org.mobicents.media.server.spi.events.Basic`

Event ID	Description	Type	Duration
`org.mobicents.media.server.spi.events.Basic.DTMF`	DTMF Event	BR	

The Announcement Package. Package name: `org.mobicents.media.server.spi.event.Announcement`

Event ID	Description	Type	Duration
`org.mobicents.media.server.spi.event.Announcement.PLAY`	play an announcement	TO	variable
`org.mobicents.media.server.spi.event.Announcement.COMPLETED`			
`org.mobicents.media.server.spi.event.Announcement.FAILED`			

The announcement action is qualified by an URL name and by a set of initial parameters. The “operation complete” event will be detected when the announcement is played out. If the announcement cannot be played out, an operation failure event can be returned. The failure may be explained by a commentary.

The Advanced Audio Package. Package name: `org.mobicents.media.server.spi.events.AU`

Event ID	Description	Type	Duration
`org.mobicents.media.server.spi.event.AU.PLAY_RECORD`	play an announcement	TO	variable
`org.mobicents.media.server.spi.event.AU.PROMPT_AND_COLLECT`			
`org.mobicents.media.server.spi.event.Announcement.FAILED`			

Table 8.1. The Advanced Audio Package

PLAY_RECORD Plays a prompt and records user speech. If the user does not speak, the user may be reprompted and given another chance to record. By default PlayRecord does not play an initial prompt, makes only one attempt to record, and therefore functions as a simple Record operation

7. MMS Demonstration Example

The motive of this example is to demonstrate the capabilities of new Media Server (MS) and Media Server Resource Adaptors (MSC-RA).

The example demonstrates usage of

- the Announcement Endpoint
- the Packet Relay Endpoint
- The Loop Endpoint
- The Conference Endpoint
- The IVR Endpoint

For more information on each of these types of endpoints, refer to *Section 2, “Mobicents Media Server Architecture”*.

Where is the Code? Check out the 'mms-demo' example from *http://mobicents.googlecode.com/svn/trunk/servers/jain-slee/examples/mms-demo*.

Install and Run. Start the Mobicents Server (this will also start Media Server). Make sure you have `server/default/deploy/mobicents.sar` and `server/default/deploy/mediaserver.sar` in your Mobicents Server

From Binary. Go to `/examples/mms-demo` and call 'ant deploy-all'. This will deploy the SIP RA, MSC RA, the mms-demo example and also mms-demo-audio.war. The war file contains the audio *.wav files that are used by mms-demo example.

From Source Code. If you are deploying from source code, you may deploy each of the resource adaptors individually

- make sure `JBOSS_HOME` is set and server is running
- Call mvn install from `servers/jain-slee/resources/sip` to deploy SIP RA
- Call mvn install from `servers/jain-slee/resources/media` to deploy media RA
- Call mvn install from `servers/jain-slee/examples/mms-demo` to deploy example

Once the example is deployed, make a call from your SIP Phone to TBD.

1010: Loop Endpoint Usage Demonstration. As soon as the call is established CallSbb creates a Connection using PREndpointImpl. PREndpointImpl has two Connections, one connected to calling UA by calling msConnection.modify("$", sdp). Once the connection is established CallSbb creates child LoopDemoSbb and calls startDemo() on it passing the PREndpoint name as argument. LoopDemoSbb creates child AnnouncementSbb which uses the AnnEndpointImpl to make an announcement. The other Connection of PREndpointImpl is connected to Connection from AnnEndpointImpl using the MsLink.

```
MsLink link = session.createLink(MsLink.MODE_FULL_DUPLEX);
```

```
....
...
link.join(userEndpoint, ANNOUNCEMENT_ENDPOINT);
```

Once the link is created (look at onLinkCreated()) AnnouncementSbb creates the instance of MsSignalGenerator and attaches the AnnouncementSbb to ActivityContextInterface created passing MsSignalGenerator. MsSignalGenerator is used to play the announcement as shown below:

```
MsSignalGenerator generator =
 msProvider.getSignalGenerator(getAnnouncementEndpoint());
ActivityContextInterface generatorActivity =
 mediaAcif.getActivityContextInterface(generator);
generatorActivity.attach(sbbContext.getSbbLocalObject());
generator.apply(Announcement.PLAY, new String[]{url});
```

As soon as announcement is over LoopDemoSbb creates child LoopbackSbb and calls startConversation() on it passing the PREndpoint name as argument. LoopbackSbb uses MsLink to associate the other Connection of PREndpointImpl to LoopEndpointImpl. LoopEndpointImpl simply forwards the voice packet received from caller back to caller.

```

 MsLink link = session.createLink(MsLink.MODE_FULL_DUPLEX);
.......
...
link.join(endpointName, LOOP_ENDPOINT);
```

Figure 8.3. The SBB Child Relation Diagram

1011: DTMF Usage Demonstration. As soon as the call is established CallSbb creates a Connection using PREndpointImpl. PREndpointImpl has two Connections, one connected to calling UA by calling msConnection.modify("$", sdp). Once the connection is established CallSbb creates child DtmfDemoSbb and calls startDemo() on it passing the PREndpoint name as argument. DtmfDemoSbb creates child AnnouncementSbb which uses the AnnEndpointImpl to make an announcement. The other Connection of PREndpointImpl is connected to Connection from AnnEndpointImpl using the MsLink.

```
....
```

```
MsLink link = session.createLink(MsLink.MODE_FULL_DUPLEX);
....
...
link.join(userEndpoint, ANNOUNCEMENT_ENDPOINT);
```

Once the link is created (look at onLinkCreated()) AnnouncementSbb creates the instance of MsSignalGenerator and attaches the AnnouncementSbb to ActivityContextInterface created passing MsSignalGenerator. MsSignalGenerator is used to play the announcement as shown below.

```
MsSignalGenerator generator =
 msProvider.getSignalGenerator(getAnnouncementEndpoint());
ActivityContextInterface generatorActivity =
 mediaAcif.getActivityContextInterface(generator);
generatorActivity.attach(sbbContext.getSbbLocalObject());
generator.apply(Announcement.PLAY, new String[]{url});
```

As soon as announcement is over DtmfDemoSbb creates instance of MsSignalDetector and attaches the DtmfDemoSbb to ActivityContextInterface created passing MsSignalDetector. This is necessary so that DtmfDemoSbb receives the DTMF event, look at onDtmf() method. In fact the SBB needs to be attached to this ACI everytime as the activity is over as soon as the DTMf is received.

```
MsSignalDetector dtmfDetector =
 msProvider.getSignalDetector(this.getUserEndpoint());
ActivityContextInterface dtmfAci =
mediaAcif.getActivityContextInterface(dtmfDetector);
dtmfAci.attach(sbbContext.getSbbLocalObject());
dtmfDetector.receive(Basic.DTMF, connection, new String[]{});
```

On every DTMF received DtmfDemoSbb plays corresponding wav file using the AnnouncementSbb as explained above.

The SBB Child Relation Diagram

Figure 8.4.

1012: ConfEndpointImpl Usage Demonstration. As soon as the call is established CallSbb creates a Connection using PREndpointImpl. PREndpointImpl has two Connections, one connected to calling UA by calling msConnection.modify("$", sdp). Once the connection is established CallSbb creates child ConfDemoSbb and calls startDemo() on it passing the PREndpoint name as argument. ConfDemoSbb creates child AnnouncementSbb which uses the AnnEndpointImpl to make an announcement. The other Connection of PREndpointImpl is connected to Connection from AnnEndpointImpl using the MsLink.

```
....
MsLink link = session.createLink(MsLink.MODE_FULL_DUPLEX);
....
...
link.join(userEndpoint, ANNOUNCEMENT_ENDPOINT);
```

Once the link is created (look at onLinkCreated()) AnnouncementSbb creates the instance of MsSignalGenerator and attaches the AnnouncementSbb to ActivityContextInterface created passing MsSignalGenerator. MsSignalGenerator is used to play the announcement as shown below

```
MsSignalGenerator generator =
 msProvider.getSignalGenerator(getAnnouncementEndpoint());
ActivityContextInterface generatorActivity =
 mediaAcif.getActivityContextInterface(generator);
generatorActivity.attach(sbbContext.getSbbLocalObject());
generator.apply(Announcement.PLAY, new String[]{url});
```

As soon as announcement is over ConfDemoSbb creates child ForestSbb and calls enter() on it passing the PREndpoint name as argument. ForestSbb uses MsLink to associate the other Connection of PREndpointImpl to ConfEndpointImpl:

```
MsLink link = session.createLink(MsLink.MODE_FULL_DUPLEX);
link.join(endpointName, CNF_ENDPOINT);
```

Once the link is established (Look at onConfBridgeCreated()) ForestSbb creates many child AnnouncementSbb each responsible for unique announcement (in this case playing crickets.wav and mocking.wav). So now UA is actually listening to many announcements at same go.

The SBB Child Relation Diagram

Figure 8.5.

Appendix A. Revision History

Revision History

Revision 1.0	Mon Feb 19 2008	Douglas Silas
Initial Release		