JBoss Communications Media Server User Guide

by Oleg Kulikov, Amit Bhayani, Bartosz Baranowski, Jared Morgan, Douglas Silas, Ivelin Ivanov, Vladimir Ralev, Eduardo Martins, Jean Deruelle, Luis Barreiro, Alexandre Mendonça, and Pavel Šlégr

1. Document Conventions

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

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

1.1. Typographic Conventions

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

`Mono-spaced Bold`

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

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

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

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

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

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

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

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

Proportional Bold

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

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

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

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

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

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

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

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

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

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

1.2. Pull-quote Conventions

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

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

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

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

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

import javax.naming.InitialContext;

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

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

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

}
```

1.3. Notes and Warnings

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

Note

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

Important

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

Warning

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

2. Provide feedback to the authors!

If you find a typographical error in this manual, or if you have thought of a way to make this manual better, we would love to hear from you! Please submit a report in Bugzilla: *http://bugzilla.redhat.com/bugzilla/* against the product **${product.name}**, or contact the authors.

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

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

Introduction to the JBoss Communications Media Server

1.1. Introduction

In the world of telephony, a media server is the name given to the computing component that processes the audio and/or video streams associated with telephone calls or connections. Conference services are a particular example of how media servers can be used, as a special 'engine' is needed to mix audio streams together so that conference participants can hear all of the other participants. Conferencing servers may also need other specialized functions like "loudest talker" detection, or transcoding of audio streams, and also interpreting DTMF tones used to navigate menus also known as Interactive Voice Response (IVR). For video processing, it may be needed to change video streams, for example transcode from one video codec to another or rescale (transrate) a picture from one size to another. This media processing functions are the core responsibility of a media server. Few of the real Media Server Use Cases are defined here.

Media Server can be Hardware Media Server or Software Based. The Hardware Media Server uses the hardware components for processing the audio/video. Depending on which hardware media server, there would be individual hardware dedicated to do specific job for example reduction of echo etc. In case of Software Media Server all the audio/video processing is done by a software, no specifc hardware is used.

1.2. The Justification for S/w Media Server

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 Servers 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 software media server converts, compresses and packetizes voice data for transmission back-and-forth across the Internet backbone for landline and wireless phones. Media gateways sit at the intersection of Public Switched Telephone Networks (PSTNs) and wireless or IP-based networks.

Multiple market demands are pushing companies to converge all of their media services using media gateways with Voice-over-IP (VoIP) capabilities. Companies have expectations for such architectures, which include:

Lowering initial costs
: Capital investment is decreased because low-cost commodity hardware can be used for multiple functions.

Lowering development costs
: Open system hardware and software standards with well-defined applications reduce costs, and Application Programming Interfaces (API s) accelerate development.

Handling multiple media types
: Companies want VoIP solutions today, but also need to choose extensible solutions that will handle video in the near future.

Lowering the costs of deployment and maintenance
: Standardized, modular systems reduce training costs and maintenance while simultaneously improving uptime.

Enabling rapid time-to-market
: Early market entry hits the window of opportunity and maximizes revenue.

1.3. What is JBoss Communications Media Server

With telephony networks moving more towards VoIP technology, and using Session Initiation Protocol (SIP), the idea of media servers has started to gain some traction. Typically, an application (the 'application server') has the controlling logic, and controls a remote media server (or multiple servers) over an IP connection using various industry recognized standard protocols like MGCP, MEGACO (H.248), MSML, VoiceXML etc. The JBoss Communications Media Server is first and only open source media server available as of today that has support for MGCP. The Mobicents also has JSR-309 Standard Api implementation to control media servers irrespective of underlying protocol used. Hence Application's developed on any platform can control JBoss Communications Media Server using JSR-309 API.

The Mobicents Media Server core is based on software components only. This property allows to easy scale from development environment on a single server to production deployment in a distributed network environment. Mobicents Media Server binary is distributed in two forms: standalone server which is recommended for production deployments and integrated with JBoss 5 Application server which is more preferred for development purposes.

Because Mobicents Media Server is Java based, it is cross platform, easy to install and run on any operating system that supports Java. The available source code is a powerful tool to debug the server and understand processing logic. It also gives you the flexibility to create customized components and configurations for your personal or business use.

1.4. Media Server Use Case

Media Server's are heavily used in conventional applications across various market segments for example the Mobile / PSTN service provider have media server's to play announcements like "The user is busy, please call after some time" or to record the voice message when the called party is busy and to re-play the same message when asked for.

The IVR Applications used by various segments of industry, for example IVR system used in banks to let their customers know their balance by calling a Toll-Free-Number or do other banking transactions.

Developing a converged application has now become very easy with availability of Open Source JBoss Communications Media Server. Appart from conventional Applications like IVR,

Conferencing, various Tone (like busy, congestion etc) generation/detection below are few Use Cases that can be developed using Mobicents Media Server and Mobicents Platform.

Developing Converged Applications

A Converged Application delivers data, voice, video to end user on any device like Public Switch Telephone Network (land line/mobile) or VoIP Phones or any other device on any network like SS7, IP etc. There is no limitation on what kind of Converged Application can be developed by using the Mobicents Media Server.

- Imagine a 'Click-to-Call' application on your business portal where customer can reach your Customer Executive by clicking a link on your site.
- A virtual class room where every student is logged to a session using their personal SIP/ Mobile phones and tutor is taking class using his/her own phone.
- Imagine a call routing to your mobile phone as soon as you swipe the Credit Card and IVR system asking for your password (DTMF) to authorize the transaction. Now somebody who wants to make illicit use of your Credit Card needs to have not only your mobile but your password too.
- A online Furniture (or any goods that has high cost of delivering to door step) Shopping Portal makes a call to customer a day before delivery intimating delivery date and time and gives an option to Customer to change it either via Sopping Portal or directly through IVR (DTMF) such that there is almost zero probability to find the door locked once your truck reaches Customer door step.

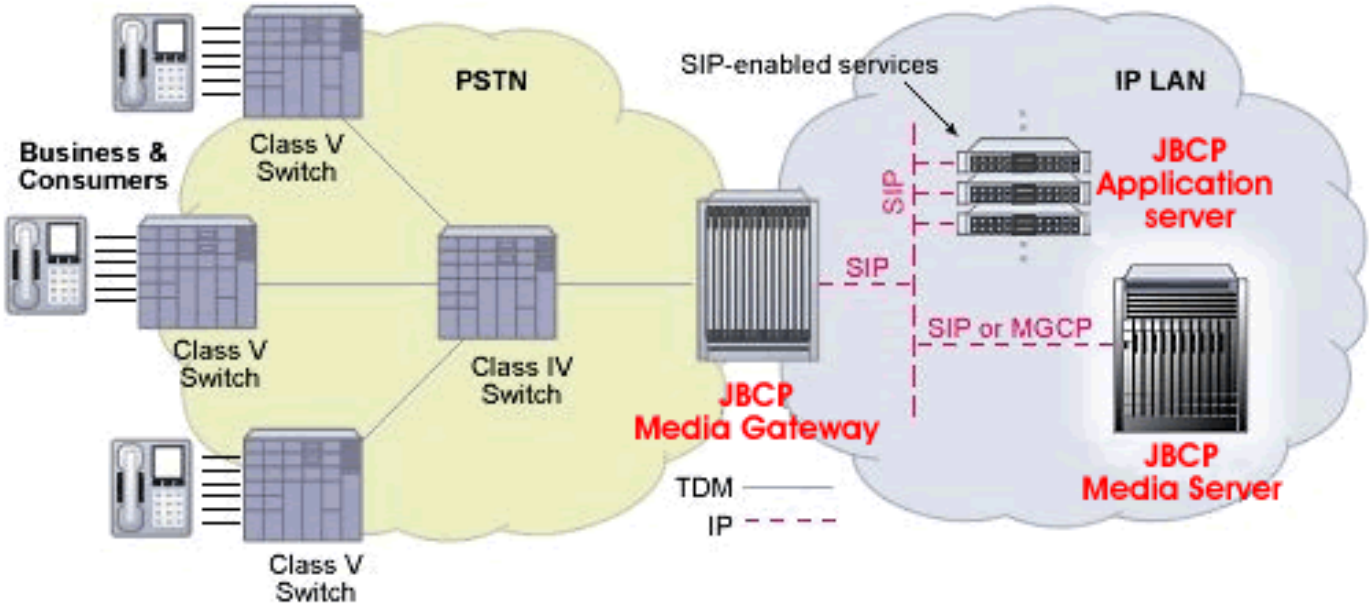
Innovation in Traditional Telco Value Added Service

As explained in above paragraphs, traditional telco value added service's like Ring Back Tone (RBT), Voice Message, Call Forwarding, Call Blocking are quite common terms now. But to subscribe/unsbscribe for each of these service, to manage these service is hassle as each of them are different service in it-self. How about a Portal where customer can log in and manage all its service that he/she is subscribed to? The Portal also gives an Rule based call routing option to let user decide where the call gets routed (his/her mobile,land phone,business phone,home phone etc) depending on which time of day the call arrives. How about letting the user record the converstaion by press of a button and latter through Portal user can listen, download, forward mail as attachment. Let user upload his/her favorite mp3 songs on portal which then can be used as personalized RBT for individual callers calling the user. Let user record the message like "This Phone is out of service for next 6 months. Please call after 6 months" and play it when any unwanted caller calls ;). Basically its a call-blocking service but with much more flexibility! Even Presence service can be leveraged to develop more interesting apps.

Capitalize on next Generation Netwrok - IMS

In order to create innovative, cost effective and felxible applications and to reduce time to market and meet customer demands, the transition should happen from legacy SS7 network to IP based Multimedia Subsystem (IMS). Mobicents Media Server can act as IMS - Media Resource Function (MRF) that caters to media needs within IP based home netwroks.

Mobicents Media Server can also be deployed as PSTN gateway that converst the SIP signalling to SS7 signals (ISUP etc). On one side the Media Server is able to send and receive IMS media over the Real-Time Protocol (RTP). On the other side the Media Server uses one or more PCM (Pulse Code Modulation) time slots to connect to the SS7 network



Technical Specification and Capacity of JBoss Communications Media Server

2.1. Technical Specification of JBoss Communications Media Server

- Media and Coders :
  - G711 (a-Law, u-Law)
  - GSM
  - SPEEX
  - G729
  - DTMF(RFC 2833, INBAND)
- Media Files :
  - WAV
  - SPX
  - GSM
  - MP3
- Signaling and control :
  - MGCP
  - Java Media Control API(JSR-309)
- Audio processing :
  - Audio transcoding, bridging and conference
  - DTMF detection and generation.Both Inband as well as RFC2833
  - Inband generation/detection of other Tones like busy, congestion etc possible
  - Text-To-Speech Support
  - Capability to generate silence/comfort noise

2.2. Capacity of JBoss Communications Media Server

- Capacity : Typical media sessions per server
  - G.711 @ 20 ms - upto 500 with dual-core 2.33GHz CPU 4GB RAM

Installing the JBoss Communications Media Server

The JBoss Communications Media Server distribution is available in two forms: Standalone and Embedded in JBoss Application Server 5. The standalone version is more preferred for carrier production deployment while JBoss Application Server 5 version is most useful for development

The JBoss Communications Media Server is available in both binary and source code distributions. The simplest way to get started with the Media Server is to download the ready-to-run binary distribution. Alternatively, the source code for the JBoss Communications Media Server can be obtained by checking it out from its repository using the Subversion version control system (SVN), and then built using the Maven build system. Whereas installing the binary distribution is recommended for most users, obtaining and building the source code is recommended for those who want access to the latest revisions and Media Server capabilities.

Installing the Java Development Kit

3.1. Java Development Kit: Installing, Configuring and Running

The **MobicentsJBCP** platform is written in Java. A working Java Runtime Environment (JRE) or Java Development Kit (JDK) must be installed prior to running the server. The required versiom must be version 5 or higher.

It is possible to run most **MobicentsJBCP** servers, such as the JAIN SLEE Server, using a Java 6 JRE or JDK.

JRE or JDK? Although **MobicentsJBCP** servers are capable of running on the Java Runtime Environment, this guide assumes the audience is mainly developers interested in developing Java-based, **MobicentsJBCP**-driven solutions. Therefore, installing the Java Development Kit is covered due to the anticipated audience requirements.

32-Bit or 64-Bit JDK. If the system uses 64-Bit Linux or Windows architecure, the 64-bit JDK is strongly recommended over the 32-bit version. The following heuristics should be considered in determining whether the 64-bit Java Virtual Machine (JVM) is suitable:

- Wider datapath: the pipe between RAM and CPU is doubled, which improves the performance of memory-bound applications when using a 64-bit JVM.
- 64-bit memory addressing provides a virtually unlimited (1 exabyte) heap allocation. Note that large heaps can affect garbage collection.
- Applications that run with more than 1.5 GB of RAM (including free space for garbage collection optimization) should utilize the 64-bit JVM.

- Applications that run on a 32-bit JVM and do not require more than minimal heap sizes will gain nothing from a 64-bit JVM. Excluding memory issues, 64-bit hardware with the same relative clock speed and architecture is not likely to run Java applications faster than the 32-bit version.

Note

The following instructions describe how to download and install the 32-bit JDK, however the steps are nearly identical for installing the 64-bit version.

Downloading. Download the Sun JDK 5.0 (Java 2 Development Kit) from Sun's website: *http://java.sun.com/javase/downloads/index_jdk5.jsp*. Click the **Download** link next to "JDK 5.0 Update <*x*>" (where <*x*> is the latest minor version release number).

The Sun website offers two download options:

- A self-extracting RPM (for example, `jdk-1_5_0_16-linux-i586-rpm.bin`)
- A self-extracting file (e.g. `jdk-1_5_0_16-linux-i586.bin`)

If installing the JDK on Red Hat Enterprise Linux, Fedora, or another RPM-based Linux system, it is recommended that the self-extracting file containing the RPM package is selected. This option will set up and use the SysV service scripts in addition to installing the JDK. The RPM option is also recommended if the **MobicentsJBCP** platform is being set up in a production environment.

Installing. The following procedures detail how to install the Java Development Kit on both Linux and Windows.

Procedure 3.1. Installing the JDK on Linux

- Ensure the file is executable, then run it:

```
~]$ chmod +x "jdk-1_5_0_<minor_version>-linux-<architecture>-rpm.bin"
~]$ ./"jdk-1_5_0_<minor_version>-linux-<architecture>-rpm.bin"
```

Setting up SysV Service Scripts for Non-RPM Files

If the non-RPM self-extracting file is selected for an RPM-based system, the SysV service scripts can be configured by downloading and installing one of the `-compat` packages from the JPackage project. Download the `-compat` package that corresponds correctly to the minor release number of the installed JDK. The compat packages are available from *ftp://jpackage.hmdc.harvard.edu/JPackage/1.7/generic/RPMS.non-free/*.

Important

A `-compat` package is not required for RPM installations. The `-compat` package performs the same SysV service script set up that the RPM version of the JDK installer does.

Procedure 3.2. Installing the JDK on Windows

- Using Explorer, double-click the downloaded self-extracting installer and follow the instructions to install the JDK.

Configuring. Configuring the system for the JDK consists of two tasks: setting the `JAVA_HOME` environment variable, and ensuring the system is using the proper JDK (or JRE) using the `alternatives` command. Setting `JAVA_HOME` generally overrides the values for `java`, `javac` and `java_sdk_1.5.0` in `alternatives`, however it is recommended to specify the value for consistency.

Setting the `JAVA_HOME` Environment Variable on Generic Linux

After installing the JDK, ensure the `JAVA_HOME` environment variable exists and points to the location of the JDK installation.

Setting the `JAVA_HOME` Environment Variable on Linux. Determine whether `JAVA_HOME` is set by executing the following command:

```
~]$ echo $JAVA_HOME
```

If `JAVA_HOME` is not set, the value must be set to the location of the JDK installation on the system. This can be achieved by adding two lines to the `~/.bashrc` configuration file. Open `~/.bashrc` (or create it if it does not exist) and add a line similar to the following one anywhere inside the file:

```
export JAVA_HOME="/usr/lib/jvm/jdk1.5.0_<version>"
```

The changes should also be applied for other users who will be running the **MobicentsJBCP** on the machine (any environment variables `exported` from `~/.bashrc` files are local to that user).

Setting `java`, `javac` and `java_sdk_1.5.0` using the `alternatives` command

Selecting the Correct System JVM on Linux using `alternatives`. On systems with the `alternatives` command, including Red Hat Enterprise Linux and Fedora, it is possible to

choose which JDK (or JRE) installation to use, as well as which `java` and `javac` executables should be run when called.

As the superuser, call `/usr/sbin/alternatives` with the `--config java` option to select between JDKs and JREs installed on your system:

```
home]$ sudo /usr/sbin/alternatives --config java

There are 3 programs which provide 'java'.

  Selection    Command
-----------------------------------------------
   1           /usr/lib/jvm/jre-1.5.0-gcj/bin/java
   2           /usr/lib/jvm/jre-1.6.0-sun/bin/java
*+ 3           /usr/lib/jvm/jre-1.5.0-sun/bin/java

Enter to keep the current selection[+], or type selection number:
```

The Sun JDK, version 5, is required to run the `java` executable. In the `alternatives` information printout above, a plus (+) next to a number indicates the option currently being used. Press **Enter** to keep the current JVM, or enter the number corresponding to the JVM to select that option.

As the superuser, repeat the procedure above for the `javac` command and the `java_sdk_1.5.0` environment variable:

```
home]$ sudo /usr/sbin/alternatives --config javac
```

```
home]$ sudo /usr/sbin/alternatives --config java_sdk_1.5.0
```

Setting the `JAVA_HOME` Environment Variable on Windows

For information on how to set environment variables in Windows, refer to *http://support.microsoft.com/kb/931715*.

Testing. To ensure the correct JDK or Java version (5 or higher), and that the java executable is in the `PATH` environment variable, run the `java -version` command in the terminal from the home directory:

```
home]$ java -version
java version "1.5.0_16"
Java(TM) 2 Runtime Environment, Standard Edition (build 1.5.0_16-b03)
```

```
Java HotSpot(TM) Client VM (build 1.5.0_16-b03, mixed mode, sharing)
```

Uninstalling. It is not necessary to remove a particular JDK from a system, because the JDK and JRE version can be switched as required using the `alternatives` command, and/or by setting `JAVA_HOME`.

Uninstalling the JDK on Linux. On RPM-based systems, uninstall the JDK using the `yum remove <jdk_rpm_name>` command.

Uninstalling the JDK on Windows. On Windows systems, check the JDK entry in the `Start` menu for an uninstall option, or use `Add/Remove Programs`.

3.2. JBoss Application Server 5.x.y embedded Media Server Binary Distribution: Installing, Configuring and Running

The JBoss Communications Media Server either comes bundled with the JBoss Application Server or Standalone. This section details how to install the JBoss Communications Media Server that comes bundled with JBoss Application Server 5. For installation of Standalone JBoss Communications Media Server, refer to *Section 3.3, " Standalone Media Server Binary Distribution: Installing, Configuring and Running "*

3.2.1. Pre-Install Requirements and Prerequisites

Ensure that the following requirements have been met before continuing with the install.

Hardware Requirements

Sufficient Disk Space
: Once unzipped, the JBoss AS embedded Media Server binary release requires *at least* 110 Mb of free disk space. Keep in mind that disk space requirements may change from release to release.

Anything Java Itself Will Run On
: The JBoss embedded Media Server and its bundled servers, JBoss, are 100% Java. The Media Server will run on the same hardware that the JBoss Application Server runs on.

Software Prerequisites

JDK 5 or Higher
: A working installation of the Java Development Kit (JDK) version 5 or higher is required in order to run the JBoss embedded Media Server. Note that the JBoss Application Server is a runtime dependency of the Media Server and, as mentioned, comes bundled with the binary distribution.

3.2.2. Downloading

The latest version of the JBoss embedded Media Server is available from *http://www.mobicents.org/mms/mms-downloads.html* . The top row of the table holds the latest version. Click the `Download` link to start the download.

3.2.3. Installing

Once the requirements and prerequisites have been met, the JBoss embedded Media Server can be installed onto the system. Follow the instructions below for the operating system on which the server will reside.

Version Numbers

For clarity, the command line instructions presented in this chapter use specific version numbers and directory names. Ensure this information is substituted with the binary distribution's version numbers and file names.

Procedure 3.3. Installing the JBoss embedded Media Server Binary Distribution on Linux

It is assumed that the downloaded archive is saved in the home directory, and that a terminal window is open displaying the home directory.

1. Create a subdirectory to extract the files into. For ease of identification, it is recommended that the version number of the binary is included in this directory name.

```
~]$ mkdir “ms-<version>”
```

2. Move the downloaded zip file into the directory:

```
~]$ mv “mms-jboss-5.1.0.GA-2.0.0.BETA3.zip” “ms-<version>”
```

3. Move into the directory:

```
~]$ cd “ms-<version>”
```

4. Extract the files into the current directory by executing one of the following commands.

   - Java:

```
ms-<version>]$ jar -xvf “mms-jboss-5.1.0.GA-2.0.0.BETA3.zip”
```

- Linux:

```
ms-<version>]$ unzip "mms-jboss-5.1.0.GA-2.0.0.BETA3.zip"
```

Note

Alternatively, use `unzip` -d <unzip_to_location> to extract the zip file's contents to a location other than the current directory.

5. Consider deleting the archive, if free disk space is an issue.

```
ms-<version>]$ rm “mms-jboss-5.1.0.GA-2.0.0.BETA3.zip”
```

Procedure 3.4. Installing the JBoss embedded Media Server Binary Distribution on Windows

1. For this procedure, it is assumed that the downloaded archive is saved in the `My Downloads` folder.
2. Create a subfolder in `My Downloads` to extract the zip file's contents into. For ease of identification, it is recommended that the version number of the binary is included in the folder name. For example, `ms-<version>` .
3. Extract the contents of the archive, specifying the destination folder as the one created in the previous step.
4. Alternatively, execute the `jar -xvf` command to extract the binary distribution files from the zip archive.

    1. Move the downloaded zip file from `My Downloads` to the folder created in the previous step.
    2. Open the Windows Command Prompt and navigate to the folder that contains the archive using the `cd` command
    3. Execute the `jar -xvf` command to extract the archive contents into the current folder.

    ```
    C:Users<user>My Downloadsms-<version>jar -xvf "mms-
    jboss-5.1.0.GA-2.0.0.BETA3.zip"
    ```

5. It is recommended that the folder holding the JBoss embedded Media Server files (in this example, the folder named `ms- <version>`) is moved to a user-defined location for storing executable programs. For example, the `Program Files` folder.

6. Consider deleting the archive, if free disk space is an issue.

```
C:Users<user>My Downloadsms-<version>delete "mms-
jboss-5.1.0.GA-2.0.0.BETA3.zip"
```

3.2.4. Setting the JBOSS_HOME Environment Variable

The **Mobicents Platform** (**Mobicents**) is built on top of the **JBoss Application Server** (**JBoss AS**). You do not need to set the `JBOSS_HOME` environment variable to run any of the **Mobicents Platform** servers *unless* `JBOSS_HOME` is *already* set.

The best way to know for sure whether `JBOSS_HOME` was set previously or not is to perform a simple check which may save you time and frustration.

Checking to See If JBOSS_HOME is Set on Linux. At the command line, `echo` **`$JBOSS_HOME`** to see if it is currently defined in your environment:

```
~]$ echo $JBOSS_HOME
```

The **Mobicents Platform** and most Mobicents servers are built on top of the **JBoss Application Server** (**JBoss AS**). When the **Mobicents Platform** or Mobicents servers are built *from source*, then `JBOSS_HOME` *must* be set, because the Mobicents files are installed into (or “over top of” if you prefer) a clean **JBoss AS** installation, and the build process assumes that the location pointed to by the `JBOSS_HOME` environment variable at the time of building is the **JBoss AS** installation into which you want it to install the Mobicents files.

This guide does not detail building the **Mobicents Platform** or any Mobicents servers from source. It is nevertheless useful to understand the role played by **JBoss AS** and `JBOSS_HOME` in the Mobicents ecosystem.

The immediately-following section considers whether you need to set `JBOSS_HOME` at all and, if so, when. The subsequent sections detail how to set `JBOSS_HOME` on Linux and Windows

Important

Even if you fall into the category below of *not needing* to set `JBOSS_HOME`, you may want to for various reasons anyway. Also, even if you are instructed that you do *not need* to set `JBOSS_HOME`, it is good practice nonetheless to check and make sure that `JBOSS_HOME` actually *isn't* set or defined on your system for some reason. This can save you both time and frustration.

You *DO NOT NEED* to set `JBOSS_HOME` if...

- ...you have installed the **Mobicents Platform** binary distribution.

- ...you have installed a Mobicents server binary distribution *which bundles* ***JBoss AS***.

You *MUST* set `JBOSS_HOME` if you are:

- installing the **Mobicents Platform** or any of the Mobicents servers *from source*.
- installing the **Mobicents Platform** binary distribution, or one of the Mobicents server binary distributions, which *do not* bundle **JBoss AS**.

Naturally, if you installed the **Mobicents Platform** or one of the Mobicents server binary releases which *do not* bundle **JBoss AS**, yet requires it to run, then you should *install* ***JBoss AS*** [http://www.jboss.org/file-access/default/members/jbossas/freezone/docs/Installation_Guide/4/html/index.html] before setting `JBOSS_HOME` or proceeding with anything else.

Setting the JBOSS_HOME Environment Variable on Linux. The `JBOSS_HOME` environment variable must point to the directory which contains all of the files for the **Mobicents Platform** or individual Mobicents server that you installed. As another hint, this topmost directory contains a `bin` subdirectory.

Setting `JBOSS_HOME` in your personal `~/.bashrc` startup script carries the advantage of retaining effect over reboots. Each time you log in, the environment variable is sure to be set for you, as a user. On Linux, it is possible to set `JBOSS_HOME` as a system-wide environment variable, by defining it in `/etc/bashrc`, but this method is neither recommended nor detailed in these instructions.

Procedure 3.5. To Set JBOSS_HOME on Linux

1. Open the `~/.bashrc` startup script, which is a hidden file in your home directory, in a text editor, and insert the following line on its own line while substituting for the actual install location on your system:

```
export JBOSS_HOME="/home/<username>/<path>/<to>/<install_directory>"
```

2. Save and close the `.bashrc` startup script.
3. You should `source` the `.bashrc` script to force your change to take effect, so that `JBOSS_HOME` becomes set for the current session. Note that any other terminals which were opened prior to altering `.bashrc` will need to `source~/.bashrc` as well should they require access to `JBOSS_HOME`.

```
~]$ source ~/.bashrc
```

4. Finally, ensure that `JBOSS_HOME` is set in the current session, and actually points to the correct location:

Note

The command line usage below is based upon a binary installation of the **Mobicents Platform**. In this sample output, `JBOSS_HOME` has been set correctly to the *`topmost_directory`* of the **Mobicents** installation. Note that if you are installing one of the standalone **Mobicents** servers (with **JBoss AS** bundled!), then `JBOSS_HOME` would point to the *`topmost_directory`* of your server installation.

```
~]$ echo $JBOSS_HOME
/home/user/
```

Setting the JBOSS_HOME Environment Variable on Windows. The `JBOSS_HOME` environment variable must point to the directory which contains all of the files for the Mobicents Platform or individual Mobicents server that you installed. As another hint, this topmost directory contains a `bin` subdirectory.

For information on how to set environment variables in recent versions of Windows, refer to *http://support.microsoft.com/kb/931715*.

3.2.5. Running

In the Linux terminal or Windows command prompt, the JBoss embedded Media Server has started successfully if the last line of output is similar to the following (ending with “Started in 23s:648ms”):

```
11:27:34,663 INFO [ServerImpl] JBoss (Microcontainer) [5.1.0.GA (build: SVNTag=JBoss_5_1_0_GA date=200905221053)] Started in 37s:637ms
```

Procedure 3.6. Running the Media Server on Linux

1. Change the working directory to installation directory (the one in which the zip file's contents was extracted to)

   ```
   downloads]$ cd "ms-<version>"
   ```

2. (Optional) Ensure that the `bin/run.sh` start script is executable.

   ```
   ms-<version>]$ chmod +x bin/run.sh
   ```

3. Execute the `run.sh` Bourne shell script.

```
ms-<version>]$ ./bin/run.sh
```

Note

Instead of executing the Bourne shell script to start the server, the `run.jar` executable Java archive can be executed from the `bin` directory:

```
ms-<version>]$ java -jar bin/run.jar
```

Procedure 3.7. Running the JBoss embedded Media Server on Windows

1. Using Windows Explorer, navigate to the `bin` subfolder in the installation directory.
2. The preferred way to start the JBoss embedded Media Server is from the Command Prompt. The command line interface displays details of the startup process, including any problems encountered during the startup process.

   Open the Command Prompt via the **Start** menu and navigate to the correct folder:

```
C:Users<user>My Downloads> cd "ms-<version>"
```

3. Start the JBoss Application Server by executing one of the following files:

   - `run.bat` batch file:

```
C:Users<user>My Downloadsms-<version>>binrun.bat
```

   - `run.jar` executable Java archive:

```
C:Users<user>My Downloadsms-<version>>java -jar binrun.jar
```

3.2.6. Stopping

Detailed instructions for stopping the JBoss Application Server are given below, arranged by platform. If the server is correctly stopped, the following three lines are displayed as the last output in the Linux terminal or Command Prompt:

```
[Server] Shutdown complete Shutdown complete Halting VM
```

Procedure 3.8. Stopping the Media Server on Linux

1. Change the working directory to the binary distribution's install directory.

```
~]$ cd "ms-<version>"
```

2. (Optional) Ensure that the bin/shutdown.sh start script is executable:

```
ms-<version>]$ chmod +x bin/shutdown.sh
```

3. Run the `shutdown.sh` executable Bourne shell script with the `-S` option (the short option for `--shutdown`) as a command line argument:

```
ms-<version>]$ ./bin/shutdown.sh -S
```

Note

The `shutdown.jar` executable Java archive with the `-S` option can also be used to shut down the server:

```
ms-<version>]$ java -jar bin/shutdown.jar -S
```

Procedure 3.9. Stopping JBoss embedded Media Server on Windows

- Stopping the JBoss Application Server on Windows consists of executing either the `shutdown.bat` or the `shutdown.jar` executable file in the `bin` subfolder of the MMS for JBoss binary distribution. Ensure the `-S` option (the short option for `--shutdown`) is included in the command line argument.

```
C:Users<user>My Downloadsms-<version>>binshutdown.bat -S
```

  - The `shutdown.jar` executable Java archive with the `-S` option can also be used to shut down the server:

```
C:Users<user>My Downloadsms-<version>>java -jar binshutdown.jar -S
```

3.2.7. Server Structure

Now the server is installed, it is important to understand the layout of the server directories. An understanding of the server structure is useful when deploying examples, and making configuration changes. It is also useful to understand what components can be removed to reduce the server boot time.

The directory structure in the JBoss embedded Media Server installation directory is named using a standard structure. *Table 3.1, "Directory Structure"* describes each directory, and the type of information contained within each location.

Table 3.1. Directory Structure

Directory Name	Description
bin	Contains the entry point JARs and start-up scripts included with the Media Server distribution.
conf	Contains the core services that are required for the server. This includes the bootstrap descriptor, log files, and the default bootstrap-beans.xml configuration file.
deploy	Contains the dynamic deployment content required by the hot deployment service. The deploy location can be overridden by specifying a location in the URL attribute of the URLDeploymentScanner configuration item.
lib	Contains the startup JAR files used by the server.
log	Contains the logs from the bootstrap logging service. The `log` directory is the default directory into which the bootstrap logging service places its logs, however, the location can be overridden by altering the log4j.xml configuration file. This file is located in the `/conf` directory.

The Media Server uses a number of XML configuration files that control various aspects of the server. In case of embedded Media Server all the files related Media Server are placed in mms-jboss-5.1.0.GA-<version>/jboss-5.1.0.GA/server/default/deploy/mobicents-media-server *Table 3.2, "Core Configuration File Set"* describes the location of the key configuration files, and provides a description of the

Table 3.2. Core Configuration File Set

File Name and Location	Description
mobicents-media-server.sar/META-INF/jboss-beans.xml	Specifies which additional microcontainer deployments are loaded as part of the bootstrap phase. bootstrap-beans.xml references other configuration files contained in the `/conf/bootstrap/` directory. For

File Name and Location	Description
	a standard configuration, the bootstrap configuration files require no alteration.
mobicents-media-server/ann-beans.xml	Specifies the configuration for announcement access points.
mobicents-media-server/ivr-beans.xml	Specifies the configuration for Interactive Voice Response (IVR) endpoints.
mobicents-media-server/prelay-beans.xml	Specifies the configuration for Packet Relay endpoints.
mobicents-media-server/cnf-beans.xml	Specifies the configuration for Conference endpoints.
mobicents-media-server/test-beans.xml	Specifies the endpoint for test capabilities.
mgcp-controller-service.sar/META-INF/jboss-beans.xml	Specifies the configuration for the MGCP controller.

3.2.8. Testing

For information on testing the Media Server, refer to *Section 3.4, " Writing and Running Tests Against the Media Server "* .

3.2.9. Uninstalling

To uninstall the Media Server, delete the directory containing the extracted binary distribution.

3.3. Standalone Media Server Binary Distribution: Installing, Configuring and Running

The Mobicents Media Server either comes bundled with the JBoss Application Server or Standalone. This section details how to install the Standalone Mobicents Media Server. For installation of JBoss embedded Mobicents Media Server, refer to *Section 3.2, " JBoss Application Server 5.x.y embedded Media Server Binary Distribution: Installing, Configuring and Running "*

3.3.1. Pre-Install Requirements and Prerequisites

Ensure that the following requirements have been met before continuing with the install.

Hardware Requirements

Sufficient Disk Space
: Once unzipped, the Standalone Media Server binary release requires *at least* 5 Mb of free disk space. Keep in mind that disk space requirements may change from release to release.

Anything Java Itself Will Run On
: The Standalone Media Server is 100% Java.

Software Prerequisites

JDK 5 or Higher
: A working installation of the Java Development Kit (JDK) version 5 or higher is required in order to run the Standalone Media Server.

3.3.2. Downloading

The latest version of the Standalone Media Server is available from *http://www.mobicents.org/mms-downloads.html* . The top row of the table holds the latest version. Click the `Download` link to start the download.

3.3.3. Installing

Once the requirements and prerequisites have been met, the Standalone Media Server can be installed onto the system. Follow the instructions below for the operating system on which the server will reside.

Version Numbers

For clarity, the command line instructions presented in this chapter use specific version numbers and directory names. Ensure this information is substituted with the binary distribution's version numbers and file names.

Procedure 3.10. Installing the Standalone Media Server Binary Distribution on Linux

It is assumed that the downloaded archive is saved in the home directory, and that a terminal window is open displaying the home directory.

1. Create a subdirectory to extract the files into. For ease of identification, it is recommended that the version number of the binary is included in this directory name.

```
~]$ mkdir “ms-<version>”
```

2. Move the downloaded zip file into the directory:

```
~]$ mv “mms-standalone-2.0.0.BETA3.zip” “ms-<version>”
```

3. Move into the directory:

```
~]$ cd “ms-<version>”
```

4. Extract the files into the current directory by executing one of the following commands.

   - Java:

     ```
     ms-<version>]$ jar -xvf “mms-standalone-2.0.0.BETA3.zip”
     ```

   - Linux:

     ```
     ms-<version>]$unzip "mms-standalone-2.0.0.BETA3.zip"
     ```

   Note

   Alternatively, use `unzip -d <unzip_to_location>` to extract the zip file's contents to a location other than the current directory.

5. Consider deleting the archive, if free disk space is an issue.

   ```
   ms-<version>]$ rm “mms-standalone-2.0.0.BETA3.zip”
   ```

Procedure 3.11. Installing the Standalone Media Server Binary Distribution on Windows

1. For this procedure, it is assumed that the downloaded archive is saved in the `My Downloads` folder.
2. Create a subfolder in `My Downloads` to extract the zip file's contents into. For ease of identification, it is recommended that the version number of the binary is included in the folder name. For example, `ms-<version>` .
3. Extract the contents of the archive, specifying the destination folder as the one created in the previous step.
4. Alternatively, execute the `jar -xvf` command to extract the binary distribution files from the zip archive.

   1. Move the downloaded zip file from `My Downloads` to the folder created in the previous step.
   2. Open the Windows Command Prompt and navigate to the folder that contains the archive using the `cd` command

3. Execute the `jar -xvf` command to extract the archive contents into the current folder.

```
C:Users<user>My Downloadsms-<version>jar -xvf "mms-
standalone-2.0.0.BETA3.zip"
```

5. It is recommended that the folder holding the Standalone Media Server files (in this example, the folder named `mms-standalone-<version>`) is moved to a user-defined location for storing executable programs. For example, the `Program Files` folder.

6. Consider deleting the archive, if free disk space is an issue.

```
C:Users<user>My Downloadsms-<version>delete "mms-
standalone-2.0.0.BETA3.zip"
```

3.3.4. Running

In the Linux terminal or Windows command prompt, the Standalone Media Server has started successfully if the last line of output is similar to the following

```
2100 [main] INFO org.mobicents.media.server.bootstrap.MainDeployer - [[[[[[[[ Mobicents Media
Server: release.version=2.0.0.BETA3 Started ]]]]]]]]
```

Procedure 3.12. Running the Standalone Media Server on Linux

1. Change the working directory to installation directory (the one in which the zip file's contents was extracted to)

```
downloads]$ cd "mms-standalone-<version>"
```

2. (Optional) Ensure that the `bin/run.sh` start script is executable.

```
ms-<version>]$ chmod +x bin/run.sh
```

3. Execute the `run.sh` Bourne shell script.

```
ms-<version>]$ ./bin/run.sh
```

Note

Instead of executing the Bourne shell script to start the server, the `run.jar` executable Java archive can be executed from the `bin` directory:

```
mms-standalone-<version>]$ java -jar bin/run.jar
```

Procedure 3.13. Running the Standalone Media Server on Windows

1. Using Windows Explorer, navigate to the `bin` subfolder in the installation directory.
2. The preferred way to start the Standalone Media Server is from the Command Prompt. The command line interface displays details of the startup process, including any problems encountered during the startup process.

   Open the Command Prompt via the **Start** menu and navigate to the correct folder:

   ```
   C:Users<user>My Downloads>cd "mms-standalone-<version>"
   ```

3. Start the Standalone Server by executing one of the following files:

   - `run.bat` batch file:

     ```
     C:Users<user>My Downloadsmms-standalone<version>>binrun.bat
     ```

   - `run.jar` executable Java archive:

     ```
     C:Users<user>My Downloadsmms-standalone-<version>>java -jar binrun.jar
     ```

3.3.5. Stopping

Detailed instructions for stopping the Standalone Media Server are given below, arranged by platform. If the server is correctly stopped, the following three lines are displayed as the last output in the Linux terminal or Command Prompt:

```
[Server] Shutdown complete Shutdown complete Halting VM
```

Procedure 3.14. Stopping the Standalone Media Server on Linux

1. Change the working directory to the binary distribution's install directory.

   ```
   
   ```

```
~]$ cd "mms-standalone-<version>"
```

2. (Optional) Ensure that the bin/shutdown.sh start script is executable:

```
mms-standalone-<version>]$ chmod +x
bin/shutdown.sh
```

3. Run the `shutdown.sh` executable Bourne shell script with the `-S` option (the short option for `--shutdown`) as a command line argument:

```
mms-standalone-<version>]$./bin/shutdown.sh -S
```

Note

The `shutdown.jar` executable Java archive with the `-S` option can also be used to shut down the server:

```
mms-standalone-<version>]$ java -jar bin/shutdown.jar -S
```

Procedure 3.15. Stopping Standalone Media Server on Windows

- Stopping the Standalone Media Server on Windows consists of executing either the `shutdown.bat` or the `shutdown.jar` executable file in the `bin` subfolder of the MMS for JBoss binary distribution. Ensure the `-S` option (the short option for `--shutdown`) is included in the command line argument.

```
C:Users<user>My Downloadsmms-standalone-<version>>binshutdown.bat -S
```

  - The `shutdown.jar` executable Java archive with the `-S` option can also be used to shut down the server:

```
C:Users<user>My Downloadsmms-standalone-<version>>java -jar
 binshutdown.jar -S
```

3.3.6. Server Structure

Now the server is installed, it is important to understand the layout of the server directories. An understanding of the server structure is useful when deploying examples, and making configuration changes. It is also useful to understand what components can be removed to reduce the server boot time.

The directory structure in the Standalone Media Server installation directory is named using a standard structure. *Table 3.3, "Directory Structure"* describes each directory, and the type of information contained within each location.

Table 3.3. Directory Structure

Directory Name	Description
bin	All the entry point JARs and start scripts included with the Media Server distribution are located in the bin directory
conf	The conf directory contains the bootstrap descriptor, bootstrap-beans.xml by default, file for a given server configuration. This defines the core services that are fixed for the lifetime of the server.
deploy	The deploy directory is the default location the hot deployment service looks to for dynamic deployment content. This may be overridden through the URLDeploymentScanner URLs attribute.
lib	Contains the startup JAR files used by the server.
log	Contains the logs from the bootstrap logging service. The `log` directory is the default directory into which the bootstrap logging service places its logs, however, the location can be overridden by altering the log4j.xml configuration file. This file is located in the `/conf` directory.

The Standalone Media Server uses a number of XML configuration files that control various aspects of the server. *Table 3.4, "Core Configuration File Set"* describes the location of the key configuration files, and provides a description of the

Table 3.4. Core Configuration File Set

File Name and Location	Description
conf/bootstrap-beans.xml	Specifies which additional microcontainer deployments are loaded as part of the bootstrap phase. bootstrap-beans.xml references other configuration files contained in the `/conf/bootstrap/` directory. For a standard configuration, the bootstrap configuration files require no alteration.

File Name and Location	Description
conf/log4j.properties	Specifies the Apache `log4j` framework category priorities and appenders used by the Media Server.
deploy/ann-beans.xml	Specifies the configuration for announcement access points.
deploy/ivr-beans.xml	Specifies the configuration for Interactive Voice Response (IVR) endpoints.
deploy/prelay-beans.xml	Specifies the configuration for Packet Relay endpoints.
deploy/cnf-beans.xml	Specifies the configuration for Conference endpoints.
deploy/test-beans.xml	Specifies the endpoint for test capabilities.
deploy/mgcp-conf.xml	Specifies the configuration for the MGCP controller.

3.3.7. Testing

For information on testing the Media Server, refer to *Section 3.4, " Writing and Running Tests Against the Media Server "* .

3.3.8. Uninstalling

To uninstall the Media Server, delete the directory containing the extracted binary distribution.

3.4. Writing and Running Tests Against the Media Server

For information about the different kinds of tests that the Media Server provides, refer to *Writing and Running Tests Against MMS* [http://groups.google.com/group/mobicents-public/web/mobicents-ms-tests] .

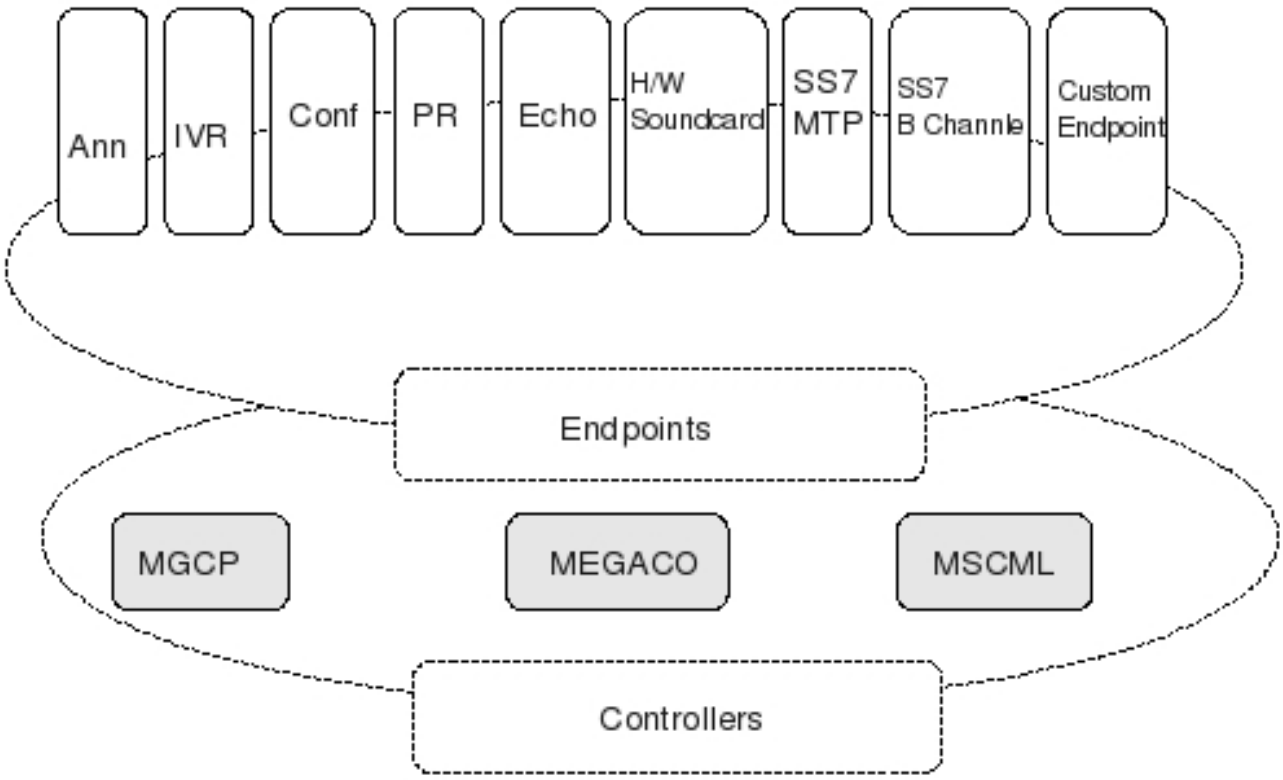
Media Server Architecture

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 called *provisioning-on-demand* and *scaling-on-demand* .

4.1. High level components

The Media Server's high degree of modularity benefits the application developer in several ways. The already-tight code can be further optimized to support applications that require small footprints. For example, if PSTN interconnection is unnecessary in an application, then the D-channel feature can be removed from the Media Server. In the future, if the same application is deployed within a Signaling System 7 (SS7) network, then the appropriate endpoint can be enabled, and the application is then compatible.



The Media Server architecture assumes that call control intelligence lies outside of the Media Server, and is handled by an external entity. The Media Server also assumes that call controllers will use control procedure described by protocols such as MGCP , MEGACO or MSML , among others. Each specific control module can be plugged in directly to the server as a standard deployable unit. Utilizing the JBoss Microcontainer for the implementation of control protocol-specific communication logic allows for simple deployment. It is therefore unnecessary for developers to configure low-level transaction and state management details, multi-threading, connection-pooling and other low-level details and API s.

The Mobicents Media Server call control intelligence can be a JSLEE Application deployed on Mobicents JAIN SLEE Server or any other JAIN SLEE container. In case of Mobicents JSLEE Server there is already MGCP Resource Adaptor available.

Mobicents Media Server can also be controlled from Mobicents SIP Servlets or any other SIP Servlets container using any of the above call control procedures or using the Mobicents JSR-309 Implementation. Mobicents JSR-309 Implementation internally leverages MGCP protocol to controll media server. Mobicents JSR-309 implementation details is out of scope of this document.

It is also possible to control the Mobicents Media Server from any third party Java application (including standalone Java apps) or other technologies like .NET etc as far as they follow standrad protocols like MGCP, MEGACO etc. There is no dependency on call controller but the protocol used between the call controller and Mobicents Media Server.

Many key features of Mobicents Media Server are provided by integrating individual components operating using generic Service Provider Interface. There are two of types of high level components: Endpoints and Controllers.

4.1.1. 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 either physical or virtual. Physical endpoint creation requires hardware installation, while software is sufficient for creating virtual endpoints. An interface on a gateway that terminates at a trunk connected to a PTSN switch would be an example of a physical endpoint. An audio source in an audio content server would be 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 access point
- packet relay

- Asynchronous Transfer Mode (ATM) "trunk side" interface

This list is not final: other endpoint types may be defined in the future, such as test endpoints which 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.

Descriptions of Various Access Point Types

Announcement Server Access Point

An announcement server endpoint provides access, intuitively, to an announcement server. Upon receiving requests from the call agent, the announcement server “plays” 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; they are “half-duplex” : the announcement server is not expected to listen to audio signals from the connection. Announcement access points are capable of playing announcements; however, these endpoints do not have the capability of transcoding. To achieve transcoding, a Packet Relay must be used. Also note that the announcement server endpoint can generate tones, such as dual-tone multi-frequency (DTMF).

Interactive Voice Response Access Point

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” for 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. Similarly to announcement endpoints, IVR endpoints do not possess media-transcoding capabilities. IVR plays and records in the format in which the media was stored or received.

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 packet networks, or between the endpoint and other endpoints in the same gateway. The signals originating from these connections are 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 allocation of resources within the gateway.

Packet Relay Endpoint

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 and gateways which operate over different transmission networks, such as IP or ATM.

Echo Endpoint

An echo—or loopback—endpoint is a test endpoint that is used for maintenance and/or continuity testing. The endpoint returns the incoming audio signal from the endpoint back to that same endpoint, thus creating an echo effect

4.1.2. Controller Modules

Controller Modules allows external interfaces to be implemented for the Media Server. Each controller module implements an industry standard control protocol, and uses a generic SPI to control processing components or endpoints.

One such controller module is the Media Gateway Control Protocol (MGCP). MGCP is designed as an internal protocol within a distributed system that appears to outside as a single VoIP gateway. The MGCP is composed of a Call Agent, and set of gateways including at least one "media gateway" which performs the conversion of media signal between circuit and packets, and atleast one "signalling gateway" when connected to SS7 controlled network. The Call Agent can be distributed over several computer platforms.

4.2. Design Overview

The Mobicents Media Server is developed on top of existing Java technologies. The Java platform is ideal for network computing. It offers single, unified-and-unifying programming model that can connect all elements of a business infrastructure. The modularization effort is supported by use of the JBoss Microcontainer which allows to deploy services written as Plain Java Objects into a Standard Java SE runtime environment in controlled manner and achieve great level of customization. Dependencies are fully managed to ensure that new services cannot be deployed until services they depend on have first been deployed. Where it makes sense to do so you can even re-deploy services at runtime providing that you access them via the microcontainer bus. Undeploying a service causes all dependent services to be undeployed first in order to maintain the integrity of the system.

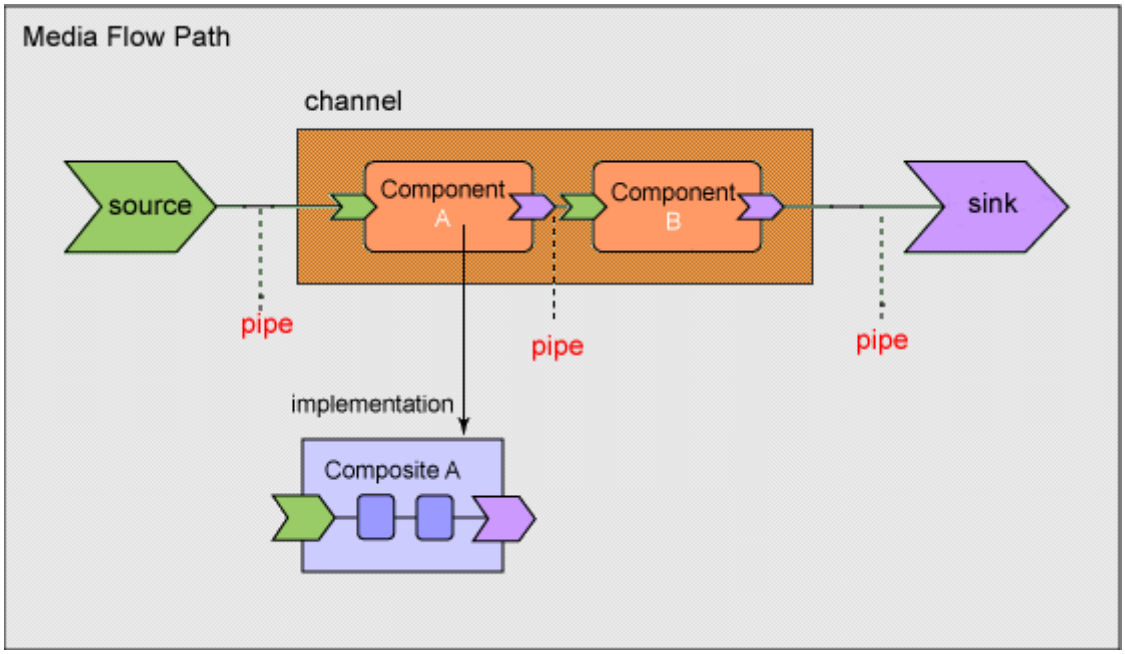
Media Source and Media Sink. To achieve the modularization every media component's in Mobicents Media Server (like AudioPlayer, Recorder, DTMF Detector/Generator) are identified as either MediaSource or MediaSink. As name suggests MediaSource is the one that has capability to generate media (AudioPlayer) while MediaSink (Recorder) is the one that consumes media.

Components and Factories. For creating any component Media Server uses suitable Factory. Each component has its unique identifier and name defined by its factory. Component identifier is unique within the entire server implementation. The name of component in opposite way is shared across component produced by same factory.

Endpoint Composition. Each of the Endpoints declared in Mobicents Media Server are composition of these Media Source/Sink and also depends on how each of these media components are ordered. For example which media component is first in line to consume/produce media. The transition of media through this ordered media components is achieved by Channels.

Channel. Channel is not a media component but it is able to join with Media Source and Media Sink or join with other channel. The role of channel is to construct media flow path by joining components using pipes.

Pipe. Each Pipe has either inlet or outlet or both defined. A Pipe with only inlet defined acts as exhaust for a channel while Pipe with only outlet acts as intake for a Channel. If a Pipe has both inlet and outlet defined, it means its an internal pipe joining two components.



For example in diagram above Pipe1 (joining source and Component A) is the one with only outlet defined and Pipe3 (joinig sink and Component B) is the one with only inlet defined while Pipe2 (joining Component A and Component B) has both inlet and outlet defined and hence acts as internal pipe joining Component A and Component B.

Endpoints may only decalre Channel to receive media (Rx-Channel) or Channel to send media (Tx-Channel) or Channel for both receiving as well as sending media.

In addition to Channels, each Endpoints also has either MediaSource or MediaSink or both acting as primary source/sink of Media. If Endpoint doesn't have primary MediaSource or MediaSink it needs to declare the ResourceGroup which is a container for MediaSource and MediaSink. For exampl IVR Endpoint has both MediaSource and MediaSink while Conference Endpoint has ResourceGroup (ConferenceBridge)

Table 4.1. Component Definition

Component	Media Source	Media Sink	Component Factory	Description
AudioPlayer	yes	no	org.mobicents.media.server.impl.resource.audio.AudioPlayerFactory	AudioPlayer is capable of playing pcma, pcmu, speex, gsm, linear, linear 44100 mono, linear

Component	Media Source	Media Sink	Component Factory	Description
				44100 stero encoded files
Recorder	no	yes	org.mobicents.mediaserver.impl.resource.audio.RecorderFactory	Recorder is capable of recording in pcma, pcmu, linear formats
Rfc2833Detector	no	yes	org.mobicents.mediaserver.impl.resource.dtmf.Rfc2833DetectorFactory	Rfc2833Detector is capable of detecting RFC2833 RTP Events. Basically used for DTMF detection.
Rfc2833Generator	yes	no	org.mobicents.mediaserver.impl.resource.dtmf.Rfc2833GeneratorFactory	Rfc2833Generator is capable of generating RFC2833 RTP Events. Basically used for DTMF generation.
InbandDetector	no	yes	org.mobicents.mediaserver.impl.resource.dtmf.InbandDetectorFactory	InbandDetector is capable of detecting inband DTMF. InbandDetector is mostly used when detecting DTMF from conventional SS7 line where as Rfc2833Detector is used only for IP netwrok
InbandGenerator	yes	no	org.mobicents.mediaserver.impl.resource.dtmf.InbandGeneratorFactory	InbandGeneratorFactory is capable of generating inband DTMF. InbandGenerator is mostly used

Component	Media Source	Media Sink	Component Factory	Description
				when generating DTMF on conventional SS7 line where as Rfc2833Generator is used only for IP netwrok
Player	yes	no	org.mobicents.media.server.impl.resource.audio.soundcard.PlayerFactory	This is special kind of component to play the media directly on sound hardware installed on Media Server. The sound hardware is any hardware that can understand the Format of media arriving. For example to directly play media on audio card of computer where media server is running
Demultiplexer	yes	no	org.mobicents.media.server.impl.resource.DemuxFactory	A Demultiplexer is one that has one media stream as input but can produce many media stream as output. For example Demultiplexer can be in path of IVR endpoint RxChannel and

Component	Media Source	Media Sink	Component Factory	Description
				output's from Demultiplexer can be connected to InbandDetector as well as Rfc2833Detector using Pipe.
Multiplexer	no	yes	org.mobicents.media.server.impl.resource.MuxFactory	A Multiplexer is one that has many media stream's as input but will produce only one media stream as output. For example Multiplexer can be in path of IVR endpoint TxChannel and input's of Multiplexer can be connected to InbandGenerator as well as Rfc2833Generator using Pipe.

Configuring the JBoss Communications Media Server

All endpoints are plugged as POJO service in JBoss Microcontainers. To create a component for the JBoss Communications Media Server, the appropriate component Factory must be used. Each component within a factory has an identifier and name that is unique across the server implementation. Because each component is unique in the Media Server, it can be referenced and pulled into other applications.

5.1. Timer

The Timer provides a time source, and functions similar to a crystal oscillator. This endpoint can be configured to specify the millisecond interval between two oscillations.

The configurable aspect of the Timer is:

heartBeat
: Time interval (in milliseconds) between two subsequent oscillations.

5.2. MainDeployer

The MainDeployer endpoint manages hot deployment of components and enpoints. Hot-deployable components and endpoints are defined as those that can be added to or removed from the running server.

MainDeployer scans the `/deploy` directory, looking for configuration files that have changed since the last scan. When MainDeployer detects any changes to the directory, any changes resulting from the removed configuration file are processed. This includes re-deploying changed beans, adding new beans, or removing beans that are no longer required.

To understand the functionality of the MainDeployer endpoint, experiment by removing the `ann-beans.xml` configuration file from the `/deploy` directory while the server is running. Observe how the server behaves once the file is removed from the folder.

The configurable aspects of MainDeployer are:

path
: Specifies the location of the configuration XML files. Generally, this is the /deploy directory.

scanPeriod
: Specifies the time (in milliseconds) that MainDeployer checks the specified path for changes to the directory.

fileFilter
: Specifies the file extensions that will be deployed or monitored. Supported file extensions are -beans.xml and -conf.xml

5.3. RTPFactory

`RTPFactory` is responsible for managing the actual RTP Socket. The reference of `RTPFactory` is passed to each endpoint which, in turn, leverage the `RTPFactory` to create Connections and decide on supported codecs.

The configurable aspects of the RTPFactory are:

formatMap
: Specifies the relationship between the RTP payload type and format. *Table 5.1, "Supported RTP Formats"* describes the payload types and their supported formats.

bindAddress
: Specifies the IP address to which the RTP socket is bound.

portRange
: Specifies the port range within which the RTP socket will be created. The first free port in the given range is assigned to the socket.

jitter
: Specifies the size of the jitter buffer (in milliseconds) for incoming packets.

timer
: Specifies the timer instance from which reading process is synchronized.

stunAddress
: Specifies the location of the STUN server to use. For more information regarding STUN, refer to *Section 5.12, "MMS STUN Support"* .

Supported RTP Formats. The `RTPFactory` is able to receive the following RTP media types:

Table 5.1. Supported RTP Formats

Payload Type	Format	Specification	Description
0	PCMU	*RFC 1890* [http://www.ietf.org/rfc/rfc1890.txt]	ITU G.711 U-law audio
3	GSM	*RFC 1890* [http://www.ietf.org/rfc/rfc1890.txt]	GSM full-rate audio
8	PCMA	*RFC 1890* [http://www.ietf.org/rfc/rfc1890.txt]	ITU G.711 A-law audio
18	G729	N/A	G.729 audio
31	H.261	N/A	Video
97	SPEEX	N/A	Speex narrow band audio

Payload Type	Format	Specification	Description
101	DTMF	*RFC 2893* [http://www.ietf.org/rfc/rfc2893.txt]	Dual-tone Multi-frequency (DTMF) Events

5.4. Digital Signal Processor (DSP)

The configurable aspect of the DspFactory are:

name
: The name of the processor

CodecFactories
: The list of codecs

5.5. Audio Player

The configurable aspect of the AudioPlayerFactory are:

name
: The name of the Audio Player

5.6. Audio Recorder

The configurable aspect of the RecorderFactory are:

name
: The name of the Audio Recorder

recordDir
: The location of recrodDir will be considered as parent and all the audio files recorded will go in this parent directory. The location specified by recordDir should be present in folder structure else Recorder will fail. It can be relative like '${mms.home.dir}' in which case all the recorded files will be stored in MMS_HOME or user can specify absolute value like '/home/user/workarea/myapp/recordedfiles' on linux and 'c:/workarea/myapp/recordedfiles' on windows

5.7. DTMF

Two different types of components are used to handle inband and rfc2833 mode of detecting and generating DTMF tones.

5.7.1. Rfc2833 Detector

The configurable aspects of the Rfc2833DetectorFactory are:

name
: The name of the detector

5.7.2. Inband Detector

The configurable aspects of the InbandDetectorFactory are:

name
: The name of the detector

5.7.3. Rfc2833 Generator

The configurable aspects of the Rfc2833GeneratorFactory are:

name
: The name of the generator

5.7.4. Inband Generator

The configurable aspects of the InbandGeneratorFactory are:

name
: The name of the generator

5.8. Announcement Server Access Points

An Announcement Server endpoint provides access to an announcement service. Upon receiving 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 (“half-duplex”), therefore, the Announcement Server is not expected to listen to audio signals from the connection.

Announcement endpoints do not transcode announced media; in order to achieve this, the application must use Packet Relay endpoints on the media path. Also note that the announcement server endpoint can generate a tones such as DTMF, Busy, Congestion etc.

Example 5.1. The Announcement Endpoint Declaration

```
  <bean name="Ann-TxChannelFactory"
    class="org.mobicents.media.server.resource.ChannelFactory" />


<bean name="AnnConnectionFactory" class="org.mobicents.media.server.ConnectionFactory">
      <property name="txChannelFactory"><inject bean="Ann-TxChannelFactory"/></property>
    </bean>

  <!-- ANNOUNCEMENT  -->
  <bean name="Announcement-Access-Point"
```

```
    class="org.mobicents.media.server.EndpointImpl">
    <property name="localName">
      /mobicents/media/aap/[1..10]
    </property>
    <property name="timer">
      <inject bean="Timer" />
    </property>
    <property name="sourceFactory">
      <inject bean="AudioPlayerFactory" />
    </property>
    <property name="rtpFactory">
      <map class="java.util.Hashtable" keyClass="java.lang.String"
        valueClass="org.mobicents.media.server.impl.rtp.RtpFactory">
        <entry>
          <key>audio</key>
          <value>
            <inject bean="RTPAudioFactory" />
          </value>
        </entry>
      </map>
    </property>
    <property name="connectionFactory">
      <inject bean="AnnConnectionFactory" />
    </property>
  </bean>
```

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

localName
: Specifies the name under which the endpoint is to be bound.

  This parameter allows a set of enpoints to be specified, which are then created and bound automatically by the Announcement Server. Consider the scenario where a total of 10 endpoints are required. To specify this in the attribute, the following path is provided: `/media/aap/[1..10]` . The `[1..10]` in the directory path tells the Announcement Server to create a set of 10 endpoints in the `/aap` directory, named according to the endpoint number, which start at one and finish at ten. For example, `/media/aap/1, media/aap/2, ... media/aap/10` .

timer
: Specifies the timer instance from which reading process is synchronized.

sourceFactory
: Specifies the Java bean responsible for generating the source media.

rtpFactory
: Specifies the location of the RTP Factory. For more information about the RTP Factory, refer to *Section 5.3, "RTPFactory"*

connectionFactory
: Specifies the instance of ConnectionFactory that wraps the custom transmission channel factory.

Customization. The Announcement Endpoint by default is configured to only play audio files. Its also possible to generate Tones like DTMF using either Rfc2833Generator or InbandDetector or both. To use these Generators you also need to declare a Multiplexer that multiplexe's the media stream from AudioPlayer and DTMF Generatot to one stream. Bellow shown is example of how Rfc2833Generator can be used.

```
<bean name="MuxFactory"
 class="org.mobicents.media.server.impl.resource.MuxFactory">
 <constructor>
  <parameter>Mux</parameter>
 </constructor>
</bean>

<bean name="Rfc2833GeneratorFactory"

class="org.mobicents.media.server.impl.resource.dtmf.Rfc2833GeneratorFactory">
 <property name="name">Rfc2833GeneratorFactory</property>
</bean>


<bean name="ann-Pipe-1"
 class="org.mobicents.media.server.resource.PipeFactory">
 <property name="outlet">Mux</property>
</bean>
<bean name="ann-Pipe-2"
 class="org.mobicents.media.server.resource.PipeFactory">
 <property name="inlet">Rfc2833GeneratorFactory</property>
 <property name="outlet">Mux</property>
</bean>
<bean name="ann-Pipe-3"
 class="org.mobicents.media.server.resource.PipeFactory">
 <property name="inlet">Mux</property>
 <property name="outlet">audio.processor</property>
</bean>
<bean name="ann-Pipe-4"
 class="org.mobicents.media.server.resource.PipeFactory">
 <property name="inlet">audio.processor</property>
</bean>

<bean name="TxChannelFactory"
```

```
  class="org.mobicents.media.server.resource.ChannelFactory">
  <property name="components">
   <list>
    <inject bean="MuxFactory" />
    <inject bean="Rfc2833GeneratorFactory" />
    <inject bean="AudioProcessorFactory" />
   </list>
  </property>
  <property name="pipes">
   <list>
    <inject bean="ann-Pipe-1" />
    <inject bean="ann-Pipe-2" />
    <inject bean="ann-Pipe-3" />
    <inject bean="ann-Pipe-4" />
   </list>
  </property>
 </bean>

       <bean name="AnnConnectionFactory"
 class="org.mobicents.media.server.ConnectionFactory">
           <property name="txChannelFactory"><inject
 bean="TxChannelFactory"/></property>
       </bean>
```

5.9. 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. IVR endpoints do not posses the capability of transcoding played or recorded media streams. IVRs record or play in the format that the data was delivered.

Example 5.2. The IVREndpointManagement MBean

```
<bean name="IVR-TxChannelFactory"
   class="org.mobicents.media.server.resource.ChannelFactory" />

<bean name="IVR-Pipe-1"
   class="org.mobicents.media.server.resource.PipeFactory">
   <property name="outlet">audio.processor</property>
</bean>
```

```
<bean name="IVR-Pipe-2"
   class="org.mobicents.media.server.resource.PipeFactory">
   <property name="inlet">audio.processor</property>
   <property name="outlet">DeMux</property>
</bean>
<bean name="IVR-Pipe-3"
   class="org.mobicents.media.server.resource.PipeFactory">
   <property name="inlet">DeMux</property>
   <property name="outlet">Rfc2833DetectorFactory</property>
</bean>
<bean name="IVR-Pipe-4"
   class="org.mobicents.media.server.resource.PipeFactory">
   <property name="inlet">DeMux</property>
</bean>


<bean name="IVR-RxChannelFactory"
   class="org.mobicents.media.server.resource.ChannelFactory">
   <property name="components">
      <list>
         <inject bean="DeMuxFactory" />
         <inject bean="Rfc2833DetectorFactory" />
         <inject bean="AudioProcessorFactory" />
      </list>
   </property>
   <property name="pipes">
      <list>
         <inject bean="IVR-Pipe-1" />
         <inject bean="IVR-Pipe-2" />
         <inject bean="IVR-Pipe-3" />
         <inject bean="IVR-Pipe-4" />
      </list>
   </property>
</bean>


<bean name="IVRConnectionFactory" class="org.mobicents.media.server.ConnectionFactory">
      <property name="txChannelFactory"><inject bean="IVR-TxChannelFactory"/></property>
      <property name="rxChannelFactory"><inject bean="IVR-RxChannelFactory"/></property>
   </bean>

<!-- IVR  -->
<bean name="IVREndpoint"
   class="org.mobicents.media.server.EndpointImpl">
```

```
    <property name="localName">
      /mobicents/media/IVR/[1..10]
    </property>
    <property name="timer">
      <inject bean="Timer" />
    </property>
    <property name="sourceFactory">
      <inject bean="AudioPlayerFactory" />
    </property>
    <property name="sinkFactory">
      <inject bean="RecorderFactory" />
    </property>
    <property name="rtpFactory">
      <map class="java.util.Hashtable" keyClass="java.lang.String"
        valueClass="org.mobicents.media.server.impl.rtp.RtpFactory">
        <entry>
          <key>audio</key>
          <value>
            <inject bean="RTPAudioFactory" />
          </value>
        </entry>
      </map>
    </property>
    <property name="connectionFactory">
      <inject bean="IVRConnectionFactory" />
    </property>
  </bean>
```

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

localName
: Specifies the name under which the endpoint is to be bound.

  This parameter allows a set of enpoints to be specified, which are then created and bound automatically by the Media Server. Consider the scenario where a total of 10 endpoints are required. To specify this in the attribute, the following path is provided: `/mobicents/media/IVR/[1..10]` . The `[1..10]` in the directory path tells the Media Server to create a set of 10 endpoints in the `/IVR` directory, named according to the endpoint number, which start at one and finish at ten. For example, `/mobicents/media/IVR/1, /mobicents/media/IVR/2, ... /mobicents/media/IVR/10` .

timer
: Specifies the timer instance from which reading process is synchronized.

sourceFactory
: Specifies the Java bean responsible for generating the source media.

sinkFactory
: Specifies the Java bean responsible for using the source media generated by the `sourceFactory` bean.

rtpFactory
: Specifies the location of the RTP Factory. For more information about the RTP Factory, refer to *Section 5.3, "RTPFactory"*

connectionFactory
: Specifies the instance of ConnectionFactory that wraps the custom transmission and receiving channel factory.

Customization. The IVR by default detects only RFC 2833 DTMF events. However if you want to use Inband detector instead of RFC2833, replace Rfc2833DetectorFactory with InbandDetectorFactory. You will have to declare the InbandDetectorFactory bean as shown

```
<bean name="InbandDetectorFactory"

class="org.mobicents.media.server.impl.resource.dtmf.InbandDetectorFactory">
 <property name="name">InbandDetectorFactory</property>
</bean>
```

It is also possible to have RFC2833 and Inband detector both working at same time. All you need to do is declare InbandDetectorFactory as explained above and have one more pipe that connects this InbandDetectorFactory with already declared DeMux.

```
<bean name="InbandDetectorFactory"

class="org.mobicents.media.server.impl.resource.dtmf.InbandDetectorFactory">
 <property name="name">InbandDetectorFactory</property>
</bean>

<bean name="IVR-TxChannelFactory"
 class="org.mobicents.media.server.resource.ChannelFactory" />

<bean name="IVR-Pipe-1"
 class="org.mobicents.media.server.resource.PipeFactory">
 <property name="outlet">audio.processor</property>
</bean>
<bean name="IVR-Pipe-2"
 class="org.mobicents.media.server.resource.PipeFactory">
 <property name="inlet">audio.processor</property>
```

```
  <property name="outlet">DeMux</property>
 </bean>
 <bean name="IVR-Pipe-3"
  class="org.mobicents.media.server.resource.PipeFactory">
  <property name="inlet">DeMux</property>
  <property name="outlet">Rfc2833DetectorFactory</property>
 </bean>
 <bean name="IVR-Pipe-4"
  class="org.mobicents.media.server.resource.PipeFactory">
  <property name="inlet">DeMux</property>
  <property name="outlet">InbandDetectorFactory</property>
 </bean>
 <bean name="IVR-Pipe-5"
  class="org.mobicents.media.server.resource.PipeFactory">
  <property name="inlet">DeMux</property>
 </bean>


 <bean name="IVR-RxChannelFactory"
  class="org.mobicents.media.server.resource.ChannelFactory">
  <property name="components">
   <list>
    <inject bean="DeMuxFactory" />
    <inject bean="Rfc2833DetectorFactory" />
    <inject bean="InbandDetectorFactory" />
    <inject bean="AudioProcessorFactory" />
   </list>
  </property>
  <property name="pipes">
   <list>
    <inject bean="IVR-Pipe-1" />
    <inject bean="IVR-Pipe-2" />
    <inject bean="IVR-Pipe-3" />
    <inject bean="IVR-Pipe-4" />
    <inject bean="IVR-Pipe-5" />
   </list>
  </property>
 </bean>
```

5.10. Packet Relay Endpoint

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 (for example, gateways which do not support compatible compression algorithms, or gateways which operate over different transmission networks such as IP or ATM).

Example 5.3. The PREndpointManagement MBean

```
<bean name="PR-Pipe1"
   class="org.mobicents.media.server.resource.PipeFactory">
   <property name="outlet">audio.processor</property>
</bean>
<bean name="PR-Pipe2"
   class="org.mobicents.media.server.resource.PipeFactory">
   <property name="inlet">audio.processor</property>
</bean>

<bean name="PR-RxChannelFactory"
   class="org.mobicents.media.server.resource.ChannelFactory">
   <property name="components">
      <list>
         <inject bean="AudioProcessorFactory" />
      </list>
   </property>
   <property name="pipes">
      <list>
         <inject bean="PR-Pipe1" />
         <inject bean="PR-Pipe2" />
      </list>
   </property>
</bean>

<bean name="PR-TxChannelFactory"
   class="org.mobicents.media.server.resource.ChannelFactory">
   <property name="components">
      <list>
         <inject bean="AudioProcessorFactory" />
      </list>
   </property>
   <property name="pipes">
      <list>
         <inject bean="PR-Pipe1" />
         <inject bean="PR-Pipe2" />
      </list>
   </property>
</bean>
```

```
<bean name="PacketRelayConnectionFactory" class="org.mobicents.media.server.ConnectionFactory">
      <property name="txChannelFactory"><inject bean="PR-TxChannelFactory"/></property>
      <property name="rxChannelFactory"><inject bean="PR-RxChannelFactory"/></property>
    </bean>

  <bean name="PacketRelayBridgeFactory"
    class="org.mobicents.media.server.impl.resource.prelay.BridgeFactory">
    <property name="name">packet.relay</property>
  </bean>

  <bean name="PacketRelayEndpoint"
    class="org.mobicents.media.server.EndpointImpl">
    <property name="localName">
      /mobicents/media/packetrelay/[1..10]
    </property>
    <property name="timer">
      <inject bean="Timer" />
    </property>
    <property name="rtpFactory">
      <map class="java.util.Hashtable" keyClass="java.lang.String"
        valueClass="org.mobicents.media.server.impl.rtp.RtpFactory">
        <entry>
          <key>audio</key>
          <value>
            <inject bean="RTPAudioFactory" />
          </value>
        </entry>
      </map>
    </property>
    <property name="connectionFactory">
      <inject bean="PacketRelayConnectionFactory" />
    </property>
    <property name="groupFactory">
      <inject bean="PacketRelayBridgeFactory" />
    </property>
  </bean>
```

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

localName
: Specifies the name under which the endpoint is to be bound.

  This parameter allows a set of enpoints to be specified, which are then created and bound automatically by the Media Server. Consider the scenario where a total of 10 endpoints are required. To specify this in the attribute, the following path is provided: `/mobicents/media/packetrelay/[1..10]` . The `[1..10]` in the directory path tells the Media Server to create a set of 10 endpoints in the `/packetrelay` directory, named according to the endpoint number, which start at one and finish at ten. For example, `/mobicents/media/packetrelay/1, /mobicents/media/packetrelay/2, ... /mobicents/media/packetrelay/10` .

timer
: Specifies the timer instance from which reading process is synchronized.

rtpFactory
: Specifies the location of the RTP Factory. For more information about the RTP Factory, refer to *Section 5.3, "RTPFactory"*

connectionFactory
: Specifies the instance of ConnectionFactory that wraps the custom transmission and receiving channel factory.

groupFactory
: Specifies the instance of BridgeFactory that wraps the source and sink.

5.11. 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" . 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. The conf endpoint can play an announcement directly on connections and hence only for the participant listening to an announcement, and can even detect DTMF for connection.

Example 5.4. The ConfEndpointManagement MBean

```
<bean name="Cnf-DefaultChannelFactory"
   class="org.mobicents.media.server.resource.ChannelFactory" />

<bean name="Cnf-Pipe-1"
   class="org.mobicents.media.server.resource.PipeFactory">
   <property name="outlet">audio.processor</property>
```

```
    </bean>
    <bean name="Cnf-Pipe-2"
        class="org.mobicents.media.server.resource.PipeFactory">
        <property name="inlet">audio.processor</property>
    </bean>

    <bean name="Cnf-Dsp-ChannelFactory"
        class="org.mobicents.media.server.resource.ChannelFactory">
        <property name="components">
            <list>
                <inject bean="AudioProcessorFactory" />
            </list>
        </property>
        <property name="pipes">
            <list>
                <inject bean="Cnf-Pipe-1" />
                <inject bean="Cnf-Pipe-2" />
            </list>
        </property>
    </bean>

    <bean name="CnfBridgeFactory"
        class="org.mobicents.media.server.impl.resource.cnf.CnfBridgeFactory">
        <property name="name">cnf.bridge</property>
    </bean>


  <bean name="CnfConnectionFactory" class="org.mobicents.media.server.ConnectionFactory">
                <property  name="txChannelFactory"><inject  bean="Cnf-Dsp-ChannelFactory"/></
property>
                <property  name="rxChannelFactory"><inject  bean="Cnf-Dsp-ChannelFactory"/></
property>
        </bean>

        <!-- Conference with RTP and DSP  -->
    <bean name="CnfEndpoint-1"
        class="org.mobicents.media.server.EndpointImpl">
        <property name="localName">
            /mobicents/media/cnf/[1..10]
        </property>
        <property name="timer">
            <inject bean="Timer" />
        </property>
```

```
            <property name="groupFactory">
          <inject bean="CnfBridgeFactory" />
       </property>

       <property name="rtpFactory">
          <map class="java.util.Hashtable" keyClass="java.lang.String"
             valueClass="org.mobicents.media.server.impl.rtp.RtpFactory">
             <entry>
                <key>audio</key>
                <value>
                   <inject bean="RTPAudioFactory" />
                </value>
             </entry>
          </map>
       </property>

       <property name="connectionFactory">
          <inject bean="CnfConnectionFactory" />
       </property>
    </bean>


  <bean name="CnfLocalConnectionFactory" class="org.mobicents.media.server.ConnectionFactory">
             <property name="txChannelFactory"><inject bean="Cnf-DefaultChannelFactory"/></
property>
             <property name="rxChannelFactory"><inject bean="Cnf-DefaultChannelFactory"/></
property>
       </bean>

       <!-- Conference local bridge  -->
    <bean name="CnfEndpoint-local"
       class="org.mobicents.media.server.EndpointImpl">
       <property name="localName">
          /mobicents/media/cnf/local/[1..10]
       </property>
       <property name="timer">
          <inject bean="Timer" />
       </property>

             <property name="groupFactory">
          <inject bean="CnfBridgeFactory" />
       </property>
       <property name="connectionFactory">
          <inject bean="CnfLocalConnectionFactory" />
```

```
    </property>

  </bean>
```

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

localName
: Specifies the name under which the endpoint is to be bound.

  This parameter allows a set of enpoints to be specified, which are then created and bound automatically by the Media Server. Consider the scenario where a total of 10 endpoints are required. To specify this in the attribute, the following path is provided: `/mobicents/media/cnf/local/[1..10]` . The `[1..10]` in the directory path tells the Media Server to create a set of 10 endpoints in the `/cnf` directory, named according to the endpoint number, which start at one and finish at ten. For example, `/mobicents/media/cnf/1, /mobicents/media/cnf/2, ... /mobicents/media/cnf/10` .

timer
: Specifies the timer instance from which reading process is synchronized.

connectionFactory
: Specifies the instance of ConnectionFactory that wraps the custom transmission and receiving channel factory.

groupFactory
: Specifies the instance of BridgeFactory that wraps the source and sink.

5.12. MMS STUN Support

When using Mobicents Media Server behind a routing device performing Network Address Translation, you may need to employ the Simple Traversal of User Datagram Protocol through Network Address Translators (abbreviated: STUN) protocol in order for the server to operate correctly. In general, it is recommended to avoid deploying the MMS behind a NAT, since doing so can incur significant performance penalties and failures. Nevertheless, the current MMS implementation does work with a static NAT, a.k.a. a one-to-one (1-1) NAT, in which no port-mapping occurs. Full Cone NAT should also work with Address-Restricted NAT.

For more information STUN NAT classification, refer to chapter 5 of *RFC3489 - STUN - Simple Traversal of User Datagram Protocol (UDP)* [http://www.faqs.org/rfcs/rfc3489.html] .

MMS STUN Configuration. Each RTPFactory in the Media Server can have its own STUN preferences. The STUN options are specified in the configuration file `mobicents-media-server/mobicents-media-server.sar/META-INF/jboss-service.xml` for embedded Media

Server and `/conf/bootstrap-beans.xml` for standalone Media Server Here is an example of an RTPFactory bean with static NAT configuration:

Example 5.5. Static NAT configuration of an RTPFactory

```
<bean name="RTPAudioFactory"
   class="org.mobicents.media.server.impl.rtp.RtpFactory">
   <property name="formatMap">
      <map class="java.util.Hashtable"
         keyClass="java.lang.Integer"
         valueClass="org.mobicents.media.Format">
         <entry>
            <key>0</key>
            <value>
               <inject bean="PCMU" />
            </value>
         </entry>
         <entry>
            <key>8</key>
            <value>
               <inject bean="PCMA" />
            </value>
         </entry>
         <entry>
            <key>3</key>
            <value>
               <inject bean="GSM" />
            </value>
         </entry>
         <entry>
            <key>97</key>
            <value>
               <inject bean="SPEEX" />
            </value>
         </entry>
         <entry>
            <key>101</key>
            <value>
               <inject bean="DTMF" />
            </value>
         </entry>
      </map>
   </property>
```

```
    <property name="bindAddress">10.65.193.65</property>
    <property name="localPort">9200</property>
    <property name="jitter">60</property>
    <property name="timer">
      <inject bean="Timer" />
    </property>
    <property name="stunAddress">stun.ekiga.net:3478</property>

  </bean>
```

In order to use stun configure stunAddress property and point to STUN server : port. If no port is specified by default it will take 3478.

Appendix A. Revision History

Revision History

Revision 3.0 Thu Jun 11 2009 JaredMorgan<jmorgan@redhat.com>

Second release of the "parameterized" documentation.

Revision 2.0 Fri Mar 06 2009 DouglasSilas<dhensley@redhat.com>

First release of the "parameterized", and much-improved JBCP documentation.