JBoss Communications Media Server User Guide

by Oleg Kulikov, Amit Bhayani, Bartosz Baranowski,
Tom Wells, Jared Morgan, and Douglas Silas

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 modern VoIP world providers are offering highly customized services that combines media such as audio/video or IM. These services require dedicated and customized media processing capabilities. To achieve this the VoIP network separates the media processing functions into dedicated node which is responsible for media processing only, while all the intelligence is executed by the separate call controller. The node processing media streams is called as Media Server.

At the same time there are lots of media streams flows in legacy systems. It means that media handling node must be able to bridge the gap between traditional legacy systems and modern VoIP networks. This node typically called as Media gateway but normally media gateway acts as media server too.

1.2. What is JBoss Communications Media Server

The JBoss Communications Media Server is open source implementation of the VoIP network element responsible for media handling. The JBoss Communications media server supports IP and TDM interfaces and thus can act as media server and as media gateway. JBoss Communications Media server provides support for both distributed and centralized services including circuit switch voice/video, announcements, tones, etc.

The JBoss Communications Media Server is provided with telco standard MGCP interface. The JBoss Communications Media server can operate in pair with JBoss Communications Jain SLEE application server. However the fully standard MGCP interface allows to use JBoss Communications Media Server in pair with another call controller if so desired. JBoss Communications Media server is equipped with implementation of the media control API compliant to JSR-309 which allows to use Media server with JBoss Communications SIP Servlet Container.

The JBoss Communications Media Server includes embedded signaling gateway function which supports complete set of TDM and IP signaling protocols. Media server supports TDM access variants like ETSI ISUP, PRI. Media server supports signaling backhaul over IP with M3UA and SUA options.

The JBoss Communications Media Server is implemented using JBoss Microcontainer kernel which allows to archive maximum flexibility. It gives the ability to adopt media server for task specific and/or extend the functions of the media server by installing additional media processing components.

Installing the JBoss Communications Media Server

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

2.1. Java Development Kit: Installing, Configuring and Running

The JBoss Communications 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 version must be version 5 or higher.

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

JRE or JDK? Although JBoss Communications servers are capable of running on the Java Runtime Environment, this guide assumes the audience is mainly developers interested in developing Java-based, JBoss Communications-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 architecture, 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 JBoss Communications 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 2.1. 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 2.2. 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 JBoss Communications 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`.

2.2. Media Server Binary Distribution: Installing, Configuring and Running

This section details how to install the JBoss Communications Media Server.

2.2.1. 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 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 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 Media Server.

2.2.2. Downloading

The latest version of the 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.

2.2.3. Installing

Once the requirements and prerequisites have been met, the 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 2.3. Installing the 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.GA.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.GA.zip”
```

   - Linux:

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

Procedure 2.4. Installing the 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.GA.zip"
    ```

5. It is recommended that the folder holding the 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.GA.zip"
    ```

2.2.4. 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.GA Started ]]]]]]]]
```

Procedure 2.5. 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 "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 2.6. Running the Media Server on Windows

1. Using Windows Explorer, navigate to the `bin` subfolder in the installation directory.
2. The preferred way to start the 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 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
```

2.2.5. Start the Server With Alternate Configuration

Using `run.sh` without any arguments binds the server at `127.0.0.1`. To bind server to different ip, pass the ip address as value to `-b` command line option. For example to bind the server to `115.252.103.220`

```
ms-<version>]$ ./bin/run.sh -b 115.252.103.220
```

2.2.6. Using run.sh

The `run` script supports the following options:

```
usage: run.sh [options]
-h, --help                  Show help message
-b, --host=<host or ip>     Bind address for media server.
```

2.2.7. Stopping

Detailed instructions for stopping the 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 2.7. 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
   ```

Procedure 2.8. 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
```

2.2.8. 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 2.1, "Directory Structure"* describes each directory, and the type of information contained within each location.

Table 2.1. 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. It also contains the log4j.xml that defines the logging filter.
deploy	The deploy directory is the default location the deployment service looks to at start time for deployment content. This may be overridden through the *MainDeployer's path* attribute.
lib	Contains the startup JAR files used by the server.

Directory Name	Description
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. `log` directory is created automatically when the server is started.
mbrola	Contains the MBROLA binary and voices. The available voices are **The available MBROLA voices are** • us1: American English Female. Voice Name : mbrola_us1 • us2: American English Male. Voice Name : mbrola_us2 • us3: American English Male. Voice Name : mbrola_us3 Please note that MBROLA is not LGPL model. To understand license please look at *http://tcts.fpms.ac.be/synthesis/mbrola/mbrlicen.html* In addition to MBROLA, the free voices available are **Free Voice** • kevin: A low quality, unlimited domain, 8kHz diphone male voice. Voice Name : kevin • kevin16: A medium quality, unlimited domain, 16kHz diphone male voice. Voice Name : mbrola_us2 • alan: A high quality, limited domain, 16kHz cluster unit male voice. Voice Name : alan
media	The default directory for media files. The MediaPlayer, Recorder etc components uses

Directory Name	Description
	media directory as default directory to read from/write and play/record media files.
temp	`temp` is created automatically when the server is started. The *MainDeployer* creates a *temp/deployment-beans.xml* which contains all the beans defined in *-beans.xml file's from */deploy* directory.
native	Contains the native library to read/write from *dahdi* channels. As of now the native library is only for `linux os`.
ss7	The `ss7` directory contains the configuration file *ss7-beans.xml* for configuring the channels. One needs to configure the Channel and MTP beans and copy this file to `/deploy` to enable the *SS7*.

The Media Server uses a number of XML configuration files that control various aspects of the server.

2.2.9. Server File Set

The following example illustrates a truncated directory structure of the `mms-standalone-<version>` server files:

```
[user@localhost <MMS_HOME>]$ tree
|-- bin
|   |-- init_redhat.sh
|   |-- run.bat
|   |-- run.jar
|   `-- run.sh
|-- conf
|   |-- bootstrap-beans.xml
|   `-- log4j.xml
|-- deploy
|   |-- avprofile-beans.xml
|   |-- comp-beans.xml
|   |-- connection-beans.xml
|   |-- connection-states-beans.xml
|   |-- controllers
|   |   `-- mgcp
|   |       |-- mgcp-conf.xml
|   |       `-- packages
|   |           |-- au
|   |           |   |-- events.xml
```

```
|   |           |   |-- package
|   |           |   |   `-- package.xml
|   |           |   `-- signals.xml
|   |           `-- dtmf
|   |               |-- events.xml
|   |               |-- package
|   |               |   `-- package.xml
|   |               `-- signals.xml
|   |-- endpoint
|   |   |-- ann-beans.xml
|   |   |-- cnf-beans.xml
|   |   |-- ivr-beans.xml
|   |   |-- mgw-beans.xml
|   |   `-- packetrelay-beans.xml
|   |-- rtp-beans.xml
|   `-- server-beans.xml
|-- lib
|   |-- activation-1.1.jar
|   |-- cmu_time_awb-1.2.2.jar
|   |-- cmu_us_kal-1.2.2.jar
|   |-- cmudict04-1.2.2.jar
|   |-- cmulex-1.2.2.jar
|   |-- cmutimelex-1.2.2.jar
|   |-- dtdparser121-1.2.1.jar
|   |-- en_us-1.2.2.jar
|   |-- freetts-1.2.2.jar
|   |-- jain-mgcp-ri-1.0.jar
|   |-- jain-sip-ri-1.2.X-20100813.024234-297.jar
|   |-- java-getopt-1.0.9.jar
|   |-- jaxb-api-2.1.9-brew.jar
|   |-- jboss-common-core-2.2.14.GA.jar
|   |-- jboss-dependency-2.0.6.GA.jar
|   |-- jboss-kernel-2.0.6.GA.jar
|   |-- jboss-logging-spi-2.1.0.GA.jar
|   |-- jboss-mdr-2.0.1.GA.jar
|   |-- jboss-reflect-2.0.2.GA.jar
|   |-- jbossxb-2.0.1.GA.jar
|   |-- jspeex-0.9.7.jar
|   |-- log4j-1.2.14.jar
|   |-- mbrola-1.2.2.jar
|   |-- mgcp-impl-2.0.0.GA.jar
|   |-- mms-controllers-mgcp-2.1.0.BETA1-SNAPSHOT.jar
|   |-- mms-controllers-rtsp-2.1.0.BETA1-SNAPSHOT.jar
|   |-- mms-impl-2.1.0.BETA1-SNAPSHOT.jar
|   |-- mms-spi-2.1.0.BETA1-SNAPSHOT.jar
|   |-- mms-standalone-2.1.0.BETA1-SNAPSHOT.jar
|   |-- mtp-1.0.0.BETA3.jar
|   |-- netty-3.2.0.BETA1.jar
|   |-- stream-1.0.0.BETA1.jar
```

```
|   |-- stun4j-1.0.MOBICENTS.jar
|   |-- tritonus_gsm-0.3.6.jar
|   |-- tritonus_share-0.3.6.jar
|   `-- xml-apis-2.9.1.jar
|-- log
|   `-- server.log
|-- mbrola
|   |-- mbrola
|   |-- mbrola.exe
|   |-- readme.txt
|   |-- us1
|   |   |-- license.txt
|   |   |-- us1
|   |   |-- us1.txt
|   |   `-- us1mrpa
|   |-- us2
|   |   |-- license.txt
|   |   |-- us2
|   |   `-- us2.txt
|   `-- us3
|       |-- license.txt
|       |-- us3
|       `-- us3.txt
|-- media
|   |-- 8kalaw.wav
|   |-- 8kulaw.wav
|   |-- gwn44m.wav
|   |-- gwn44s.wav
|   |-- sample_100kbit.mov
|   |-- sample_100kbit.mp4
|   `-- sample_50kbit.3gp
|-- native
|   `-- zap-native-linux.so
|-- ss7
|   `-- ss7-beans.xml
`-- temp
    `-- deployment-beans.xml
```

2.2.10. Uninstalling

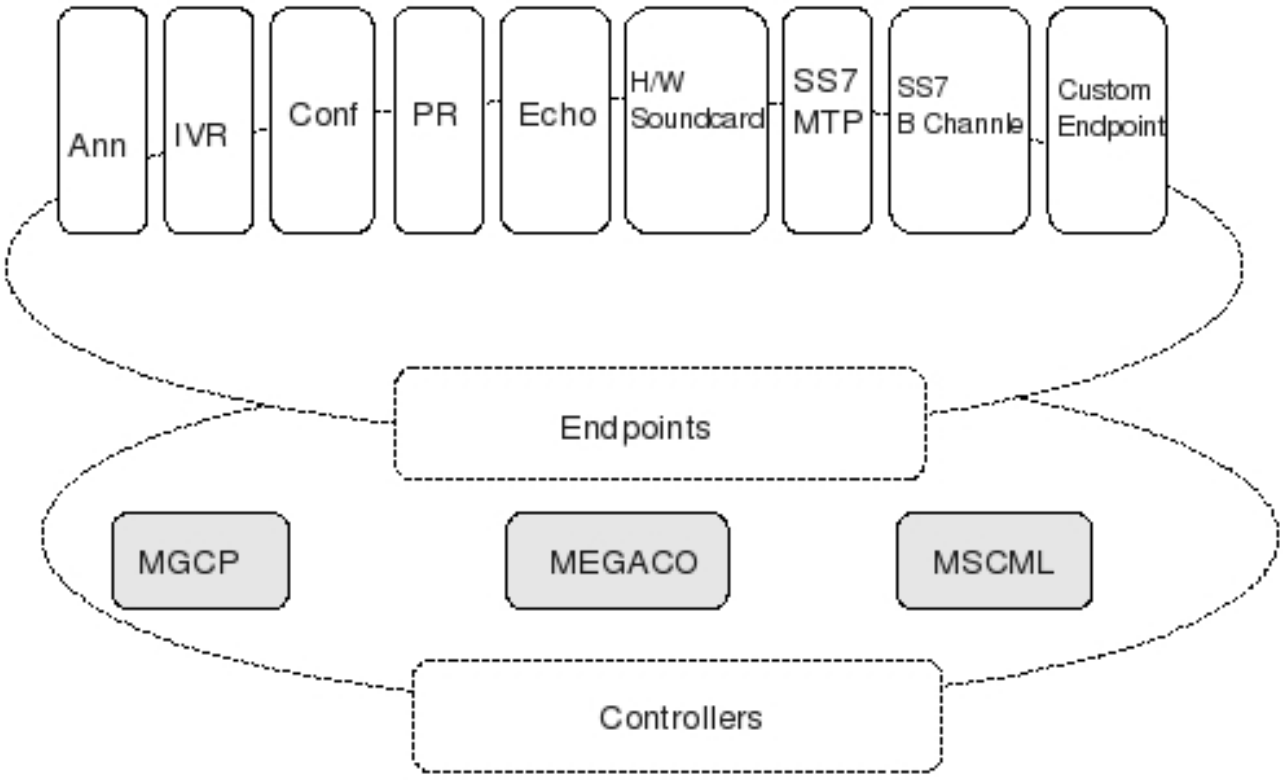
To uninstall the Media Server, delete the directory containing the extracted binary distribution.

Media Server Architecture

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 Media Server assumes a connection model where the basic constructs are endpoints and connections. Connections are grouped in calls. One or more connections can belong to one call. Connections and calls are set up at the initiative of one or several Call Agents.

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.



3.1. Endpoints

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

- Digital signal (DS0)
- Announcement server access point
- Conference bridge access point
- Packet relay
- Interactive Voice Response
- Sound card

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.

3.1.1. Digital Channel DSO

Digital channels provide an 8Khz*8bit service. Such channels are found in trunk and ISDN interfaces. They are typically part of digital multiplexes, such as T1, E1, T3 or E3 interfaces. Media gateways that support such channels are capable of translating the digital signals received on the channel, which may be encoded according to A or mu-law, using either the complete set of 8 bits or only 7 of these bits, into audio packets. When the media gateway also supports a NAS service, the gateway shall be capable of receiving either audio-encoded data (modem connection) or binary data (ISDN connection) and convert them into data packets.

In some cases, digital channels are used to carry signalling. This is the case for example of SS7 "F" links, or ISDN "D" channels. Media gateways that support these signalling functions shall be able to send and receive the signalling packets to and from a call agent, using the "back haul" procedures defined by the SIGTRAN working group of the IETF. Digital channels are sometimes used in conjunction with channel associated signalling, such as "MF R2".

3.1.2. Announcement Access Point

An announcement server endpoint provides acces to an announcement service. Under requests from the call agent, the announcement server will "play" a specified announcement. A given announcement endpoint is not supposed to support more than one connection at a time. If several connections were established to the same endpoint, then the same announcements would be played simultaneously over all the connections.

Connections to an announcement server are typically oneway, or "half duplex" -- the announcement server is not expected to listen the audio signals from the connection.

3.1.3. Conference bridge

A conference bridge endpoint is used to provide access to a specific conference.Media server establishes several connections between the endpoint and the packet networks, or between the endpoint and other endpoints in the same server instance. The precise number of connections that an endpoint support is a characteristic of the server's configuration, and may in fact vary according with the used hardware.

3.1.4. Packet Relay

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

3.1.5. Interactive Voice Response

An Interactive Voice Response (IVR) endpoint provides acces to an IVR service. Under requests from the call agent, the IVR server will "play" announcements and tones, and will "listen" to responses from the user.

A given IVR endpoint is not supposed to support more than one connection at a time. If several connections were established to the same endpoint, then the same tones and announcements would be played simultaneously over all the connections.

3.1.6. Soundcard

The sound card gives access to both Analogue-to-Digital Converter (ADC) and Digital-to-Analogue Converter (DAC) Data transmission over the Internet is done digitally so in order for voice to be transmitted it must be converted to digital using an ADC and be converted into analog again using a DAC so the voice it can be heard on the other end.

3.2. Endpoint local identifiers

The syntax of the local name depends on the type of endpoint being named. However, the local name for each of these types is naturally hierarchical, beginning with a term which identifies the physical gateway containing the given endpoint and ending in a term which specifies the individual endpoint concerned. With this in mind, the following rules for construction and interpretation of the local identifier for these entity types MUST be supported:

- The individual terms of the naming path MUST be separated by a single slash ("/", ASCII 2F hex).
- The individual terms are character strings composed of letters, digits or other printable characters, with the exception of characters used as delimitors ("/", "@"), characters used for wildcarding ("*", "$") and white spaces.

- Wild-carding is represented by square brakets and range using the following pattern: [a..b] where a,b - are integer numbers.
- In the ISUP protocol, trunks are grouped into trunk groups, identified by the SS7 point codes of the switches that the group connects. Circuits within a trunk group are identified by a circuit number (CIC in ISUP).

3.3. Calls and Connections

Connections are created on the call agent on each endpoint that will be involved in the "call." Connections may be either point to point or multipoint. A point to point connection is an association between two endpoints with the purpose of transmitting data between these endpoints. Once this association is established for both endpoints, data transfer between these endpoints can take place. A multipoint connection is established by connecting the endpoint to a multipoint session. Connections can be established over several types of bearer networks:

- Transmission of audio packets using RTP and UDP over a TCP/IP network.
- Transmission of packets over an internal connection, for example the TDM backplane or the interconnection bus of a gateway. This is used, in particular, for "hairpin" connections, connections that terminate in a gateway but are immediately rerouted over the telephone network.

In the classic example of a connection between two "DS0" endpoints (EP1 and EP2), the call agents controlling the end points will establish two connections (C1 and C2):

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

Once established, the connection parameters can be modified at any time by a "modify connection" command. The call agent may for example instruct the gateway to change the compression algorithm used on a connection, or to modify the IP address and UDP port to which data should be sent, if a connection is "redirected."

The call agent removes a connection by sending to the gateway a "delete connection" command. The gateway may also, under some circumstances, inform a gateway that a connection could not be sustained.

3.4. 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.

3.4.1. Media gateway control protocol

Media Gateway Control Protocol (MGCP) is used for controlling telephony gateways from external call control elements called media gateway controllers or call agents. A telephony gateway is a network element that provides conversion between the audio signals carried on telephone circuits and data packets carried over the Internet or over other packet networks.

MGCP assumes a call control architecture where the call control intelligence is outside the gateways and handled by external call control elements. The MGCP assumes that these call control elements, or Call Agents, will synchronize with each other to send coherent commands to the gateways under their control. MGCP is, in essence, a master/slave protocol, where the gateways are expected to execute commands sent by the Call Agents.

Capabilities of JBoss Communications Media Server

Each endpoint supports specific MGCP Packages and some or all of the Events/Signals defined in that package. Bellow table shows the capabilities of each endpoints defined in MMS

4.1. Announcement Endpoint

The Announcement Endpoint is identified by localname `/mobicents/media/aap/[1..10]` where number in square brackets indicates the range of endpoints configured and started by media server.

The MGCP Packages and corresponding Signals/Events supported by Announcement Endpoint is specified in bellow table. Line in *italics* is MGCP Signal Request/Signal Event Example

Table 4.1. Packages and corresponding Signals/Events supported by Announcement Endpoint

Supported Package		
Announcement (A)	**Supported Events**	**Parameters**
OperationComplete (oc)	NA *O: A/oc@1*	
OperationFailure (of)	NA *O: A/of@1*	
Supported Signals	**Parameters**	
Play Announcement (ann)	The PlayAnnouncement Signal must be qualified by a String of Parameter(s). If Play Announcemnet is not provided with a parameter specifying some form of playable audio or Text-to-Speech text an error is returned to application. The Play Announcement takes the URL of the media file to be played as parameter. The examples are • On remote server `http://127.0.0.1:8080/`	

`mgcpdemo/audio/RQNT-ULAW.wav`

S:A/ann@1(http://127.0.0.1:8080/mgcpdemo/audio/RQNT-ULAW.wav)

- On local server where media server is hosted `file:/home/mobicents/RQNT-ULAW.wav`

  S:A/ann@1(file:/home/mobicents/RQNT-ULAW.wav)

- In addition URL of txt file can also be passed and media server would play that text file as TTS with default voice. For ex `http://ann.example.net/hello.txt` or `file:/home/mobicents/hello.txt`

- Segment descriptor can also be used with the `ann` signal to make up an announcement. The only supported Segment descriptor is '`ts`'. '`ts`' Specifies a text string to be converted to speech. Optionally voice name '`vc`' can also be passed to direct media server to play this speech in given voice. For example `ts("Your text here") vc("mbrola_us1")`

  For supported voices look at *Table 2.1, "Directory Structure"*

	S:A/ann@6(ts(You have pressed One) vc(Mbrola_US1)) • The relative path of audio/txt file can also be passed for example `RQNT-ULAW.wav` In this case MMS will try to serach the corresponding file in `${mms.media.dir}` folder.	
Advance Announcement (AU)	**Supported Events**	**Parameters**
OperationComplete (oc)	NA *O: A/oc@1*	
OperationFailure (of)	NA *O: A/of@1*	
Supported Signals	**Parameters**	
Play Announcement (ann)	Same as Announcement Package's Play Announcement Signal	
Record (aupr)	The Record Signal must be qualified by a String of Parameter. If Record is not provided with a parameter specifying path (absolute or relative) and name of audio file, error is returned to application. The URL can be absolute URL or relative in which case recorded file will be placed at `${mms.media.dir}` folder. The examples are • On remote server `http://127.0.0.1:8080/mgcpdemo/audio/recorded.wav`	

	S:AU/ aupr@1(http://127.0.0.1:8080/ mgcpdemo/audio/ recorded.wav) • On local server where media server is hosted `file:/ home/mobicents/ recorded.wav` *S:AU/aupr@1(file:/home/ mobicents/recorded.wav)* • The relative path of file can also be passed for example `recorded.wav` In this case MMS will create recorded.wav in `${mms.media.dir}` folder.	
DTMF (D) :	**Supported Events**	**Parameters**
dtmf0	NA *O: D/dtmf0@1*	
dtmf1	NA *O: D/dtmf1@1*	
dtmf2	NA *O: D/dtmf2@1*	
dtmf3	NA *O: D/dtmf3@1*	
dtmf4	NA *O: D/dtmf4@1*	
dtmf5	NA *O: D/dtmf5@1*	
dtmf6	NA *O: D/dtmf6@1*	
dtmf7	NA *O: D/dtmf7@1*	

dtmf8	NA
	O: D/dtmf8@1
dtmf9	NA
	O: D/dtmf9@1
dtmfA	NA
	O: D/dtmfA@1
dtmfB	NA
	O: D/dtmfB@1
dtmfC	NA
	O: D/dtmfC@1
dtmfD	NA
	O: D/dtmfD@1
dtmfHash	NA
	O: D/dtmfHash@1
dtmfStar	NA
	O: D/dtmfStar@1
Supported Signals	**Parameters**
dtmf0	NA
	S:D/dtmf0@1
dtmf1	NA
	S:D/dtmf1@1
dtmf2	NA
	S:D/dtmf2@1
dtmf3	NA
	S:D/dtmf3@1
dtmf4	NA
	S:D/dtmf4@1
dtmf5	NA
	S:D/dtmf5@1
dtmf6	NA

	S:D/dtmf6@1
dtmf7	NA
	S:D/dtmf7@1
dtmf8	NA
	S:D/dtmf8@1
dtmf9	NA
	S:D/dtmf9@1
dtmfA	NA
	S:D/dtmfA@1
dtmfB	NA
	S:D/dtmfB@1
dtmfC	NA
	S:D/dtmfC@1
dtmfD	NA
	S:D/dtmfD@1
dtmfHash	NA
	S:D/dtmfHash@1
dtmfStar	NA
	S:D/dtmfStar@1

4.2. IVR Endpoint

The IVR Endpoint is identified by localname `/mobicents/media/IVR/[1..10]` where number in square brackets indicates the range of endpoints configured and started by media server.

The MGCP Packages and corresponding Signals/Events supported by IVR Endpoint is specified in bellow table. Line in *italics* is MGCP Signal Request/Signal Event Example

Table 4.2. Packages and corresponding Signals/Events supported by IVR Endpoint

Supported Package		
Announcement (A)	**Supported Events**	**Parameters**
OperationComplete (oc)	NA	
	O: A/oc@1	
OperationFailure (of)	NA	

O: A/of@1

Supported Signals	**Parameters**
Play Announcement (ann)	The PlayAnnouncement Signal must be qualified by a String of Parameter(s). If Play Announcemnet is not provided with a parameter specifying some form of playable audio or Text-to-Speech text an error is returned to application. The Play Announcement takes the URL of the media file to be played as parameter. The examples are • On remote server `http://127.0.0.1:8080/mgcpdemo/audio/RQNT-ULAW.wav` *S:A/ann@1(http://127.0.0.1:8080/mgcpdemo/audio/RQNT-ULAW.wav)* • On local server where media server is hosted `file:/home/mobicents/RQNT-ULAW.wav` *S:A/ann@1(file:/home/mobicents/RQNT-ULAW.wav)* • In addition URL of txt file can also be passed and media server would play that text file as TTS with default voice. For ex `http://ann.example.net/hello.txt` or `file:/home/mobicents/hello.txt`

	• Segment descriptor can also be used with the `ann` signal to make up an announcement. The only supported Segment descriptor is '`ts`'. '`ts`' Specifies a text string to be converted to speech. Optionally voice name '`vc`' can also be passed to direct media server to play this speech in given voice. For example `ts("Your text here") vc("mbrola_us1")` For supported voices look at *Table 2.1, "Directory Structure"* *S:A/ann@6(ts(You have pressed One) vc(Mbrola_US1))* • The relative path of audio/txt file can also be passed for example `RQNT-ULAW.wav` In this case MMS will try to serach the corresponding file in `${mms.media.dir}` folder.	
Advance Announcement (AU)	**Supported Events**	**Parameters**
OperationComplete (oc)	NA *O: A/oc@1*	
OperationFailure (of)	NA *O: A/of@1*	
Supported Signals	**Parameters**	
Play Announcement (ann)	Same as Announcement Package's Play Announcement Signal	
Record (aupr)	The Record Signal must be qualified by a String of Parameter. If Record is not	

	provided with a parameter specifying path (absolute or relative) and name of audio file, error is returned to application. The URL can be absolute URL or relative in which case recorded file will be placed at `${mms.media.dir}` folder. The examples are • On remote server `http://127.0.0.1:8080/mgcpdemo/audio/recorded.wav` *S:AU/aupr@1(http://127.0.0.1:8080/mgcpdemo/audio/recorded.wav)* • On local server where media server is hosted `file:/home/mobicents/recorded.wav` *S:AU/aupr@1(file:/home/mobicents/recorded.wav)* • The relative path of file can also be passed for example `recorded.wav` In this case MMS will create recorded.wav in `${mms.media.dir}` folder.	
DTMF (D) :	**Supported Events**	**Parameters**
dtmf0	NA *O: D/dtmf0@1*	
dtmf1	NA *O: D/dtmf1@1*	
dtmf2	NA	

	O: D/dtmf2@1
dtmf3	NA
	O: D/dtmf3@1
dtmf4	NA
	O: D/dtmf4@1
dtmf5	NA
	O: D/dtmf5@1
dtmf6	NA
	O: D/dtmf6@1
dtmf7	NA
	O: D/dtmf7@1
dtmf8	NA
	O: D/dtmf8@1
dtmf9	NA
	O: D/dtmf9@1
dtmfA	NA
	O: D/dtmfA@1
dtmfB	NA
	O: D/dtmfB@1
dtmfC	NA
	O: D/dtmfC@1
dtmfD	NA
	O: D/dtmfD@1
dtmfHash	NA
	O: D/dtmfHash@1
dtmfStar	NA
	O: D/dtmfStar@1
Supported Signals	**Parameters**
dtmf0	NA
	S:D/dtmf0@1
dtmf1	NA

	S:D/dtmf1@1
dtmf2	NA
	S:D/dtmf2@1
dtmf3	NA
	S:D/dtmf3@1
dtmf4	NA
	S:D/dtmf4@1
dtmf5	NA
	S:D/dtmf5@1
dtmf6	NA
	S:D/dtmf6@1
dtmf7	NA
	S:D/dtmf7@1
dtmf8	NA
	S:D/dtmf8@1
dtmf9	NA
	S:D/dtmf9@1
dtmfA	NA
	S:D/dtmfA@1
dtmfB	NA
	S:D/dtmfB@1
dtmfC	NA
	S:D/dtmfC@1
dtmfD	NA
	S:D/dtmfD@1
dtmfHash	NA
	S:D/dtmfHash@1
dtmfStar	NA
	S:D/dtmfStar@1

4.3. Conference Endpoint

The Conference Endpoint is identified by localname `/mobicents/media/cnf/[1..60]` where number in square brackets indicates the range of endpoints configured and started by media server.

The MGCP Packages and corresponding Signals/Events supported by Conference Endpoint is specified in bellow table. Line in *italics* is MGCP Signal Request/Signal Event Example

Table 4.3. Packages and corresponding Signals/Events supported by Conference Endpoint

Supported Package		
Announcement (A)	**Supported Events**	**Parameters**
OperationComplete (oc)	NA *O: A/oc@1*	
OperationFailure (of)	NA *O: A/of@1*	
Supported Signals	**Parameters**	
Play Announcement (ann)	The PlayAnnouncement Signal must be qualified by a String of Parameter(s). If Play Announcemnet is not provided with a parameter specifying some form of playable audio or Text-to-Speech text an error is returned to application. The Play Announcement takes the URL of the media file to be played as parameter. The examples are • On remote server `http://127.0.0.1:8080/mgcpdemo/audio/RQNT-ULAW.wav` *S:A/ann@1(http://127.0.0.1:8080/mgcpdemo/audio/RQNT-ULAW.wav)*	

- On local server where media server is hosted `file:/home/mobicents/RQNT-ULAW.wav`

  S:A/ann@1(file:/home/mobicents/RQNT-ULAW.wav)

- In addition URL of txt file can also be passed and media server would play that text file as TTS with default voice. For ex `http://ann.example.net/hello.txt` or `file:/home/mobicents/hello.txt`

- Segment descriptor can also be used with the `ann` signal to make up an announcement. The only supported Segment descriptor is '`ts`'. '`ts`' Specifies a text string to be converted to speech. Optionally voice name '`vc`' can also be passed to direct media server to play this speech in given voice. For example `ts("Your text here") vc("mbrola_us1")`

  For supported voices look at *Table 2.1, "Directory Structure"*

  S:A/ann@6(ts(You have pressed One) vc(Mbrola_US1))

- The relative path of audio/txt file can also be passed for example `RQNT-ULAW.wav`

	In this case MMS will try to serach the corresponding file in `${mms.media.dir}` folder.	
Advance Announcement (AU)	**Supported Events**	**Parameters**
OperationComplete (oc)	NA *O: A/oc@1*	
OperationFailure (of)	NA *O: A/of@1*	
Supported Signals	**Parameters**	
Play Announcement (ann)	Same as Announcement Package's Play Announcement Signal	
Record (aupr)	The Record Signal must be qualified by a String of Parameter. If Record is not provided with a parameter specifying path (absolute or relative) and name of audio file, error is returned to application. The URL can be absolute URL or relative in which case recorded file will be placed at `${mms.media.dir}` folder. The examples are • On remote server `http://127.0.0.1:8080/mgcpdemo/audio/recorded.wav` *S:AU/aupr@1(http://127.0.0.1:8080/mgcpdemo/audio/recorded.wav)* • On local server where media server is hosted `file:/`	

	`home/mobicents/recorded.wav` *S:AU/aupr@1(file:/home/mobicents/recorded.wav)* • The relative path of file can also be passed for example `recorded.wav` In this case MMS will create recorded.wav in `${mms.media.dir}` folder.	
DTMF (D) :	**Supported Events**	**Parameters**
dtmf0	NA *O: D/dtmf0@1*	
dtmf1	NA *O: D/dtmf1@1*	
dtmf2	NA *O: D/dtmf2@1*	
dtmf3	NA *O: D/dtmf3@1*	
dtmf4	NA *O: D/dtmf4@1*	
dtmf5	NA *O: D/dtmf5@1*	
dtmf6	NA *O: D/dtmf6@1*	
dtmf7	NA *O: D/dtmf7@1*	
dtmf8	NA *O: D/dtmf8@1*	
dtmf9	NA *O: D/dtmf9@1*	
dtmfA	NA	

	O: D/dtmfA@1
dtmfB	NA
	O: D/dtmfB@1
dtmfC	NA
	O: D/dtmfC@1
dtmfD	NA
	O: D/dtmfD@1
dtmfHash	NA
	O: D/dtmfHash@1
dtmfStar	NA
	O: D/dtmfStar@1
Supported Signals	**Parameters**
dtmf0	NA
	S:D/dtmf0@1
dtmf1	NA
	S:D/dtmf1@1
dtmf2	NA
	S:D/dtmf2@1
dtmf3	NA
	S:D/dtmf3@1
dtmf4	NA
	S:D/dtmf4@1
dtmf5	NA
	S:D/dtmf5@1
dtmf6	NA
	S:D/dtmf6@1
dtmf7	NA
	S:D/dtmf7@1
dtmf8	NA
	S:D/dtmf8@1
dtmf9	NA

	S:D/dtmf9@1
dtmfA	NA
	S:D/dtmfA@1
dtmfB	NA
	S:D/dtmfB@1
dtmfC	NA
	S:D/dtmfC@1
dtmfD	NA
	S:D/dtmfD@1
dtmfHash	NA
	S:D/dtmfHash@1
dtmfStar	NA
	S:D/dtmfStar@1

4.4. PacketRelay Endpoint

The PacketRelay Endpoint is identified by localname `/mobicents/media/packetrelay/[1..10]` where number in square brackets indicates the range of endpoints configured and started by media server.

The MGCP Packages and corresponding Signals/Events supported by PacketRelay Endpoint is specified in bellow table. Line in *italics* is MGCP Signal Request/Signal Event Example

Table 4.4. Packages and corresponding Signals/Events supported by PacketRelay Endpoint

Supported Package		
Announcement (A)	**Supported Events**	**Parameters**
OperationComplete (oc)	NA	
	O: A/oc@1	
OperationFailure (of)	NA	
	O: A/of@1	
Supported Signals	**Parameters**	
Play Announcement (ann)	The PlayAnnouncement Signal must be qualified by a String of Parameter(s). If Play Announcemnet is not provided	

with a parameter specifying some form of playable audio or Text-to-Speech text an error is returned to application.

The Play Announcement takes the URL of the media file to be played as parameter. The examples are

- On remote server `http://127.0.0.1:8080/mgcpdemo/audio/RQNT-ULAW.wav`

  S:A/ann@1(http://127.0.0.1:8080/mgcpdemo/audio/RQNT-ULAW.wav)

- On local server where media server is hosted `file:/home/mobicents/RQNT-ULAW.wav`

  S:A/ann@1(file:/home/mobicents/RQNT-ULAW.wav)

- In addition URL of txt file can also be passed and media server would play that text file as TTS with default voice. For ex `http://ann.example.net/hello.txt` or `file:/home/mobicents/hello.txt`

- Segment descriptor can also be used with the `ann` signal to make up an announcement. The only supported Segment descriptor is '`ts`'. '`ts`' Specifies a text string to

	be converted to speech. Optionally voice name 'vc' can also be passed to direct media server to play this speech in given voice. For example `ts("Your text here") vc("mbrola_us1")` For supported voices look at *Table 2.1, "Directory Structure"* *S:A/ann@6(ts(You have pressed One) vc(Mbrola_US1))* • The relative path of audio/txt file can also be passed for example `RQNT-ULAW.wav` In this case MMS will try to serach the corresponding file in `${mms.media.dir}` folder.	
Advance Announcement (AU)	**Supported Events**	**Parameters**
OperationComplete (oc)	NA *O: A/oc@1*	
OperationFailure (of)	NA *O: A/of@1*	
Supported Signals	**Parameters**	
Play Announcement (ann)	Same as Announcement Package's Play Announcement Signal	
Record (aupr)	The Record Signal must be qualified by a String of Parameter. If Record is not provided with a parameter specifying path (absolute or relative) and name of audio file, error is returned to application.	

	The URL can be absolute URL or relative in which case recorded file will be placed at `${mms.media.dir}` folder. The examples are • On remote server `http://127.0.0.1:8080/mgcpdemo/audio/recorded.wav` *S:AU/aupr@1(http://127.0.0.1:8080/mgcpdemo/audio/recorded.wav)* • On local server where media server is hosted `file:/home/mobicents/recorded.wav` *S:AU/aupr@1(file:/home/mobicents/recorded.wav)* • The relative path of file can also be passed for example `recorded.wav` In this case MMS will create recorded.wav in `${mms.media.dir}` folder.	
DTMF (D) :	**Supported Events**	**Parameters**
dtmf0	NA *O: D/dtmf0@1*	
dtmf1	NA *O: D/dtmf1@1*	
dtmf2	NA *O: D/dtmf2@1*	
dtmf3	NA *O: D/dtmf3@1*	
dtmf4	NA	

	O: D/dtmf4@1
dtmf5	NA
	O: D/dtmf5@1
dtmf6	NA
	O: D/dtmf6@1
dtmf7	NA
	O: D/dtmf7@1
dtmf8	NA
	O: D/dtmf8@1
dtmf9	NA
	O: D/dtmf9@1
dtmfA	NA
	O: D/dtmfA@1
dtmfB	NA
	O: D/dtmfB@1
dtmfC	NA
	O: D/dtmfC@1
dtmfD	NA
	O: D/dtmfD@1
dtmfHash	NA
	O: D/dtmfHash@1
dtmfStar	NA
	O: D/dtmfStar@1
Supported Signals	**Parameters**
dtmf0	NA
	S:D/dtmf0@1
dtmf1	NA
	S:D/dtmf1@1
dtmf2	NA
	S:D/dtmf2@1
dtmf3	NA

	S:D/dtmf3@1
dtmf4	NA
	S:D/dtmf4@1
dtmf5	NA
	S:D/dtmf5@1
dtmf6	NA
	S:D/dtmf6@1
dtmf7	NA
	S:D/dtmf7@1
dtmf8	NA
	S:D/dtmf8@1
dtmf9	NA
	S:D/dtmf9@1
dtmfA	NA
	S:D/dtmfA@1
dtmfB	NA
	S:D/dtmfB@1
dtmfC	NA
	S:D/dtmfC@1
dtmfD	NA
	S:D/dtmfD@1
dtmfHash	NA
	S:D/dtmfHash@1
dtmfStar	NA
	S:D/dtmfStar@1

Configuring the JBoss Communications Media Server

The JBoss Communications 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.

5.1. MainDeployer

The configurable aspects of MainDeployer are:

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

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

5.2. Server instance

The modularization effort allows to construct and start more then one instance of the media server in the single JBoss microcontainer. The single server instance is defined by the following description

Example 5.1. The Media Server instance Declaration

```
<bean name="MediaServer" class="org.mobicents.media.Server">
<property name="rtpManager"><inject bean="RTPFactory"/></property>
   <incallback method="addFactory" />
   <uncallback method="removeFactory" />
</bean>
```

The configurable aspects of the server instance are:

rtpManager
: Specifies the relationship between the Real Time Transmission Protocol Manager and Server instance

Thus it is possible to run Media several media server instances with different RTP configuration. Such configuration is used when Media server acts as gateway at the boundaries of the different networks.

5.3. Media types definition

Each media type to be processed must be declared. The following listing shows declaration of Audio and Video media types

```
<bean name="MediaType.audio" class="org.mobicents.media.server.spi.MediaType">

<constructor factoryClass="org.mobicents.media.server.spi.MediaType" factoryMethod="getInstance">
        <parameter>audio</parameter>
    </constructor>
</bean>

<bean name="MediaType.Video" class="org.mobicents.media.server.spi.MediaType">

<constructor factoryClass="org.mobicents.media.server.spi.MediaType" factoryMethod="getInstance">
        <parameter>video</parameter>
    </constructor>
</bean>
```

The configurable aspects of the Media type object are:

5.4. Media format definition

Format specifies a particular arrangement of data in a media stream. By examing the information stored in the format, components can discover how to interpret the bits in the binary sound data. Format accommodates a number of common encoding techniques, including pulse-code modulation (PCM), mu-law encoding or a-law encoding. These encoding techniques are predefined, but user can create new encoding types.

In addition to the encoding, the format includes other properties that further specify the exact arrangement of the data which are specific for each media type.

5.4.1. Audio format definition

Audio specific parameters include the number of channels, sample rate, sample size, byte order. Sounds may have different numbers of audio channels: one for mono, two for stereo. The sample rate measures how many "snapshots" (samples) of the sound pressure are taken per second, per channel. (If the sound is stereo rather than mono, two samples are actually measured at each instant of time: one for the left channel, and another for the right channel; however, the sample rate still measures the number per channel, so the rate is the same regardless of the number of

channels. This is the standard use of the term.) The sample size indicates how many bits are used to store each snapshot; 8 and 16 are typical values. For 16-bit samples (or any other sample size larger than a byte), byte order is important; the bytes in each sample are arranged in either the "little-endian" or "big-endian" style.

Example 5.2. Audio format definition

```
<bean name="PCMU" class="org.mobicents.media.format.AudioFormat">
      <constructor>
        <parameter>ULAW</parameter>
        <parameter>8000</parameter>
        <parameter>8</parameter>
        <parameter>1</parameter>
      </constructor>
   </bean>
```

The respective parameters of the audio format are:

- Encoding name
- Sample rate
- Sample size in bits
- Number of channels

5.5. Codec definition

A codec is a component capable of encoding and/or decoding a digital data stream or signal

Example 5.3. Codec definition

```
<bean name="G711.ulaw.encoder"
      class="org.mobicents.media.server.impl.dsp.audio.g711.ulaw.EncoderFactory" />
```

Codecs can be grouped into processors like depicted at the following example

Example 5.4. Codec definition

```
<bean name="DSP"
      class="org.mobicents.media.server.impl.dsp.DspFactory">
      <property name="name">dsp</property>
      <property name="codecFactories">
        <list>
          <inject bean="G711.ulaw.encoder" />
          <inject bean="G711.ulaw.decoder" />
        </list>
      </property>
  </bean>
```

5.6. RTP Audio Video profile

Real-time audio and video conferencing and communication applications that use the Real-time Transport Protocol (RTP) employ a standardized description format (Session Description Protocol, SDP) to describe the media streams carried in a multi-media session. This description format specifies the technical parameters of the media streams. Such a set of RTP parameters of the media stream and its compression or encoding methods is known as a media profile, or RTP audio video profile (RTP/AVP). Each profile is identified by a standardized payload type identifier (RFC 3551 and others)

Table 5.1. RTP/AVP audio and video payload types

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

Payload identifiers 96–127 are reserved for payloads defined dynamically during a session. The minimum payload support is defined as 0 (PCMU) and 5 (DVI4). The document recommends

dynamically assigned port numbers, although 5004 and 5005 have been registered for use of the profile and can be used instead. The standard also describes the process of registering new payload types with IANA.

Dynamic payloads can be configured using Format description and A/V Profile description

5.7. RTP Manager

This is the starting point for creating, maintaining and closing an RTP session. RTP Manager supports unicast session only.

The configurable aspects of the RTPFactory are:

codecs
: List of encoders/decoders that this `RtpFactory` is capable of transcoding.

bindAddress
: Specifies the IP address which will be used for RTP session

lowPort
: Lowest port (in the range of lowest to highest port's) available for creating RTP session. The first free port in the given range is assigned to the session.

highPort
: Highest port (in the range of lowest to highest port's) available for creating RTP session. The first free port in the given range is assigned to the session.

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

Example 5.5. RTP Manager definition

```
<bean name="RTPFactory" class="org.mobicents.media.server.impl.rtp.RtpFactory">
      <property name="bindAddress">${mms.bind.address}</property>
      <property name="jitter">100</property>
      <property name="lowPort">1024</property>
      <property name="highPort">65535</property>
      <property name="codecs">

 <map class="java.util.Hashtable" keyClass="org.mobicents.media.server.spi.MediaType" valueClass="java.util.L
            <entry>
               <key><inject bean="MediaType.audio"></inject></key>
               <value>
                  <list>
                     <inject bean="G711.ulaw.encoder" />
```

```
                    <inject bean="G711.ulaw.decoder" />
                </list>
            </value>
        </entry>
    </map>
</property>
</bean>
```

5.8. Dual-tone multi-frequency (DTMF) tones

Dual-tone multi-frequency (DTMF) signaling is used for telephone signaling over the line in the voice-frequency band to the call switching center. The version of DTMF used for telephone tone dialing is known by the trademarked term Touch-Tone, and is standardised by ITU-T Recommendation Q.23. Other multi-frequency systems are used for signaling internal to the telephone network

5.8.1. DTMF Detector

DTMF detector is defined by the following component:

Example 5.6. DTMF detector

```
<bean name="dtmf.detector" class="org.mobicents.media.server.impl.resource.dtmf.DetectorFactory">
    <property name="name">dtmf.detector</property>
    <property name="duration">40</property>
    <property name="interdigitInterval">150</property>
</bean>
```

Configurable aspects of the DTMF detector are

name
: The name of the detector

duration
: The minimal duration of tone in milliseconds

interDigitInterval
: The minimal interval between two tones in milliseconds

5.8.2. DTMF Generator

DTMF generator is defined by the following component:

Example 5.7. DTMF generator

```
<bean name="dtmf.generator" class="org.mobicents.media.server.impl.resource.dtmf.GeneratorFactory">
      <property name="name">dtmf.generator</property>
      <property name="duration">40</property>
   </bean>
```

Configurable aspects of the DTMF generator are

name
: The name of the detector

duration
: The minimal duration of tone in milliseconds

5.9. Text-to-Speech engine

Important

MBrola is free only for non comercial applications!

Changing text to speech is a capability to stream speach generated from text. JBoss Communications Media Server supports two TTS engines:

- FreeTTS
- MBrola

By default Server provides following set of voices:

Table 5.2. TTS Voices

Voice	Engine
alan	FreeTTS
kevin	FreeTTS
kevin16	FreeTTS
mbrola_us1	MBrola
mbrola_us2	MBrola

Voice	Engine
mbrola_us3	MBrola

5.9.1. MBrola configuration

MBrola engine picks up list of available voices from directory. It is specified at runtime with following switch: `-Dmbrola.base` . Following value is equal to default:

```
-Dmbrola.base="$MMS_HOME/mbrola"
```

5.9.2. FreeTTS configuration

Does not requrie any configuration.

5.9.3. Voice pool configuration

JBoss Communications Media Server has pool of voices. At startup it creates desired number of voice generating objects. This reduces runtime overhead and delays in response from Server. Size of pool, for each voice can be configured via media player configuration property: `voices`.

`Voices` property is of `Map` type. Table below describes values stored in this propert:

Table 5.3. Voices Map property

Attribute	Type	Description
key	String	Defines pool name. It must match correct voice name, ie: alan
value	Integer	Defines size of pool for voice. It must be positive integer.

Example configuration looks as follows:

```
<bean name="media.audio"
   class="org.mobicents.media.server.impl.resource.mediaplayer.audio.AudioPlayerFactory">
   <property name="name">media.player</property>
   <property name="audioMediaDirectory">${mms.media.dir}</property>

   <property name="voices">
      <map class="java.util.Hashtable" keyClass="java.lang.String"
         valueClass="java.lang.Integer">

         <entry>
```

```
            <key>alan</key>
            <value>1</value>
         </entry>
         <entry>
            <key>kevin16</key>
            <value>1</value>
         </entry>
         <entry>
            <key>kevin</key>
            <value>1</value>
         </entry>
         <entry>
            <key>mbrola_us3</key>
            <value>1</value>
         </entry>
         <entry>
            <key>mbrola_us2</key>
            <value>1</value>
         </entry>
         <entry>
            <key>mbrola_us1</key>
            <value>1</value>
         </entry>

      </map>
   </property>

</bean>
```

5.10. Connection state manager

The `Connection` can be in one of the following four operational states.

- NULL

  The `Connection` has been installed successfully and is ready to be activated. The `Connection` is not running, i.e., its not receiving nor transmitting media

- IDLE

  The receiving or/and transmitting `Channel` of `Connection` is/are connected to `Sink` or/and `Source` of `Endpoint`. However `Connection` is still not running, i.e., its not receiving nor transmitting media. `Connection` transits back to NULL when deleted in this stage.

- HALF_OPEN

  The `Connection` is bound and can only receive media. `Connection` transits back to NULL when deleted in this stage.

- OPEN

  The remote `SDP` is set and now `Connection` can receive as well as transmit media. `Connection` transits back to NULL when deleted in this stage.

There is no direct control over the states of `Connection` but only through `Controller` module

The `ConnectionStateManager` maps various state's of `Connection` with life-time (in milli seconds) of `Connection` in that state

ConnectionStateManager is defined by the following component:

```
<bean name="ConnectionStateManager" class="org.mobicents.media.server.ConnectionStateManager">
   <constructor>
      <parameter>

 <map class="java.util.Hashtable" keyClass="org.mobicents.media.server.spi.ConnectionState" valueClass="java
            <entry>
               <key><inject bean="ConnectionState.NULL"></inject></key>
               <value>0</value>
            </entry>
            <entry>
               <key><inject bean="ConnectionState.IDLE"></inject></key>
               <value>30000</value>
            </entry>
            <entry>
               <key><inject bean="ConnectionState.HALF_OPEN"></inject></key>
               <value>3600000</value>
            </entry>
            <entry>
               <key><inject bean="ConnectionState.OPEN"></inject></key>
               <value>3600000</value>
            </entry>
            <entry>
               <key><inject bean="ConnectionState.CLOSED"></inject></key>
               <value>0</value>
            </entry>
         </map>
      </parameter>
   </constructor>
```

```
</bean>
```

Irrespective if `Connection` is running or not, the `Connection` will be deleted after timeout (time starting from transition to this state) value specified for that state

5.11. MGCP controller configuration

MGCP Controller is implemented by the following component

```
<bean name="MgcpController"
      class="org.mobicents.media.server.ctrl.mgcp.MgcpController">
      <property name="server">
         <inject bean="MediaServer" />
      </property>
      <property name="defaultNotifiedEntity">client@localhost:2727</property>
      <property name="bindAddress">${mms.bind.address}</property>
      <property name="port">2427</property>
      <incallback method="addPackage" />
      <uncallback method="removePackage" />
   </bean>
```

Configurable aspects of the MGCP controller are

server
: The server instance which is under control

bindAddress
: The IP address to which controller is bound

port
: The port number used by controller

defaultNotifiedEntity
: Defines default notified entity value.

Advanced custom configuration and extension

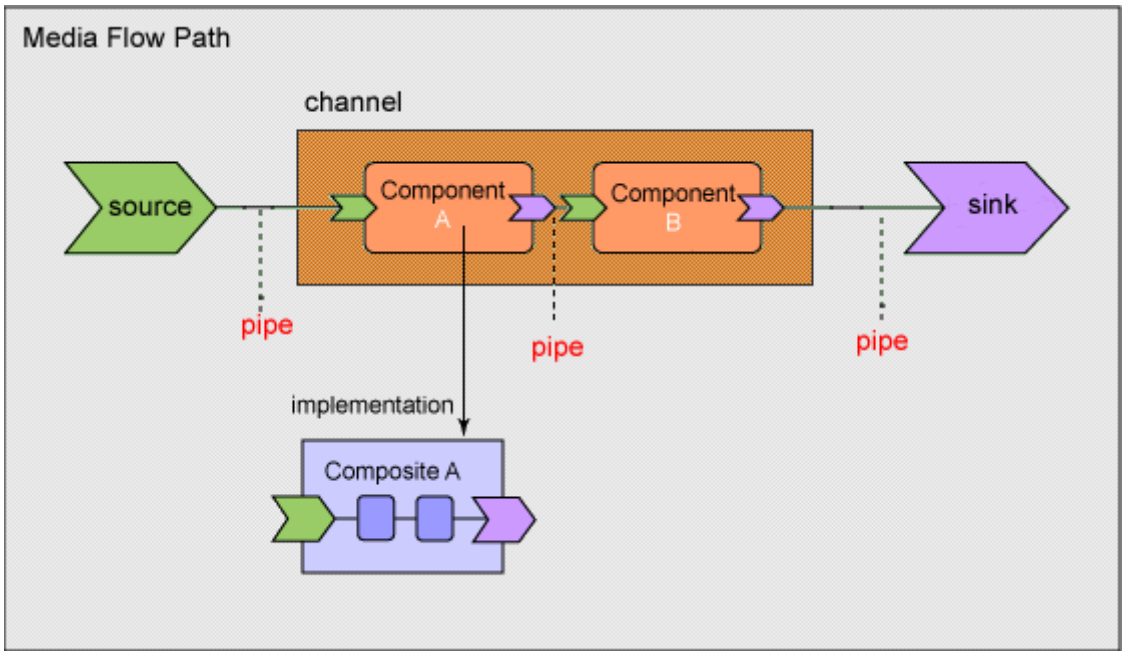
A common theme for JBoss Communications Media Server is the breaking out of internal fixed subsystems into stand-alone components.JBoss Communications Media Server strategy for making available the various voice/video services as independent components, so that they can be wired-together on demand

JBoss Communications Media server architecture promotes the usage of Service Objects to represent the media flow path. The component architecture divides the process of constructing media services into two major parts:

- Implementing components that generate, or consume, media data.
- Assembling media component chains to build a media flow path.

6.1. Channel

The role of channel is to construct media flow path by joining components using pipes. Channel can be connected to any source/sink or to other channel. The following diagram explains channel structure.



Respective channel declaration:

Example 6.1. Channel definition

```
<bean name="tx-channel" class="org.mobicents.media.server.resource.ChannelFactory">
      <property name="components">
         <list>
            <inject bean="audio.mixer" />
            <inject bean="media.player" />
            <inject bean="dtmf.generator" />
         </list>
      </property>

      <property name="pipes">
         <list>
            <inject bean="tx-pipe-1" />
            <inject bean="tx-pipe-2" />
            <inject bean="tx-pipe-3" />
            <inject bean="tx-pipe-4" />
         </list>
      </property>
   </bean>
```

Configurable aspects of the channel are:

components
: The list of components used by the channel

pipes
: The list of pipes wich defines the actual media flow path

6.1.1. Pipe

Pipe is used to join two components inside channel. 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.

The definition of the Pipe:

Example 6.2.

```
<bean name="tx-pipe-1"
      class="org.mobicents.media.server.resource.PipeFactory">
      <property name="outlet">media.player</property>
      <property name="outlet">audio.mixer</property>
      <property name="valve"><inject bean="Valve.open"></inject></property>
```

```
  </bean>
```

Configurable aspects of the pipe are:

inlet
: Identifies the component connected to the input of the pipe

oulet
: Identifies the component connected to the output of the pipe

valve
: Valve shows the default state of the pipe. By default pipe is always closed.

6.1.2. Valve

The valve defines the initial state of the pipe and is defined as

Example 6.3.

```
<bean name="Valve.open" class="org.mobicents.media.server.spi.Valve">

 <constructor factoryClass="org.mobicents.media.server.spi.Valve" factoryMethod="getInstance">
     <parameter<open</parameter>
  </constructor>
  </bean>

  <bean name="Valve.close" class="org.mobicents.media.server.spi.Valve">

 <constructor factoryClass="org.mobicents.media.server.spi.Valve" factoryMethod="getInstance">
     <parameter<close</parameter>
  </constructor>
  </bean>
```

6.1.3. Components

To achieve the modularization every media component's in JBoss Communications Media Server are identified as either MediaSource or MediaSink. As name suggests MediaSource is the one that has capability to generate media while MediaSink is the one that consumes media.

Several MediaSource of different MediaType can be grouped together in one MultiMediaSource. For example media player is source of the audio and video content

ResourceGroup is collection of MediaSource and MediaSink. These MediaSource and MediaSink can be of different MediaTypes

Some of the components are not itself media sink or source but they provide access to the actual media source or sink. Such components are called Inlet and Outlet respectively.

6.2. 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.

6.3. Virtual Endpoint Composition

If the endpoint is implemented by software components only (virtual endpoint) then in this case endpoint can be constucted dynamic

Example 6.4.

```
<bean name="PacketRelayEndpoint" class="org.mobicents.media.server.EndpointFactoryImpl">
      <property name="localName">/mobicents/media/packetrelay/[1..10] </property>
      <property name="connectionFactory">
         <inject bean="default-connection" />
      </property>
      <property name="groupFactory">
         <inject bean="PacketRelayBridgeFactory" />
      </property>
   </bean>
```

Configurable aspects of the virtual endpoint are:

connectionFactory
: Specifies factory used to construct connection.

groupFactory
: Specified resource group.

mediaSource
: Specifies media source component

mediaSink
: Specifies media sink component

6.4. Connection Composition

Connection consits from collection of channels specific for direction and media type:

Example 6.5.

```
  <bean name="default-connection" class="org.mobicents.media.server.ConnectionFactory">
        <property name="txChannelFactory">\>

 <map class="java.util.Hashtable" keyClass="java.lang.String" valueClass="org.mobicents.media.server.resource
              <entry><key>audio</key><value><inject bean="tx-channel"/></value></entry>
            </map>
        </property>
        <property name="rxChannelFactory">

 <map class="java.util.Hashtable" keyClass="java.lang.String" valueClass="org.mobicents.media.server.resource
              <entry><key>audio</key><value><inject bean="rx-channel"/></value></entry>
            </map>
        </property>
      <property name="connectionStateManager"><inject bean="ConnectionStateManager"/></
property>
   </bean>
```

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.