Restcomm Media Server User Guide

Table of Contents

Preface

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 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.

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 to switch to the first virtual terminal. Press 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 menu:>[] shorthand used to indicate traversal through a menu and its sub-menus. This is to avoid the difficult-to-follow 'Select from the **Preferences** | **] sub-menu in the menu: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.

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"));
   }

}
```

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.

Provide feedback to the authors!

If you find a typographical error in this manual, or if you have thought of a way to make this manual better, we would love to hear from you! Please submit a report in the the {this-issue.tracker.ur}, against the product Restcomm Media Server` `, or contact the authors.

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

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

Chapter 1. Introduction to the Restcomm Media Server

1.1. Overview: the Reasoning and Need for Media Server

Media Gateways Bridge Multiple Technologies

Today, computers play an important role in modern communications. Widespread access to broadband Internet and the ubiquity of Internet Protocol (IP) enable the convergence of voice, data and video. Media gateways provide the ability to switch voice media between a network and its access point. Using Digital Subscriber Line (DSL) and fast-Internet cable technology, a media gateway converts, compresses and packetizes voice data for transmission back-and-forth across the Internet backbone for landline and wireless phones. Media gateways sit at the intersection of Public Switched Telephone Networks (PSTN) and wireless or IP-based networks.

Why Media Gateways for VoIP Is Needed

Multiple market demands are pushing companies to converge all of their media services using media gateways with Voice-over-IP (VoIP) capabilities. Some of the expected benefits of the architecture are as follows:

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 (APIs) accelerate development.

Handling multiple media types

Companies want solutions that are extensible and that will be ready to handle future needs like, video.

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.

What Is the Restcomm Media Server?

The Restcomm Media Server is an open source Media Server aimed at:

- Delivering competitive, complete, best-of-breed media gateway functionality of the highest quality.
- Meeting the demands of converged wireless and landline networks, DSL and cable broadband

access, and fixed-mobile converged —— networks from a singleand singularly-capablemedia gateway platform.

- Increasing flexibility with a media gateway that supports a wide variety of call control protocols, which possesses an architecture that can scale to meet the demands of small-carrier providers as well as large enterprises.

Because Restcomm 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.

Form version 4.0.0, the Restcomm Media Server is available only as Standalone.

1.2. Technical Specification and Capacity

The Restcomm Media Server is capable of

- Media and Codecs :
  - G711 (a-Law, u-Law)
  - GSM
  - Linear PCM(L16)
  - G729
  - DTMF(RFC 2833, INBAND)
- Media Files :
  - WAV
  - GSM
- Signaling and control :
  - MGCP
  - Java Media Control API(JSR-309)
- Capacity : Typical media sessions per server
  - G.711 , L16 @20ms – 500+ per cpu core
  - GSM @ 20ms – 95 GSM mixed with 380 G.711 , L16 , 475 overall per cpu core
  - G.729 @20ms – 45 GSM mixed with 180 G.711 , L16 , 225 overall per cpu core

    All benchmark tests where done on Amazon EC2 cloud instances.

1.3. Media Server Architecture

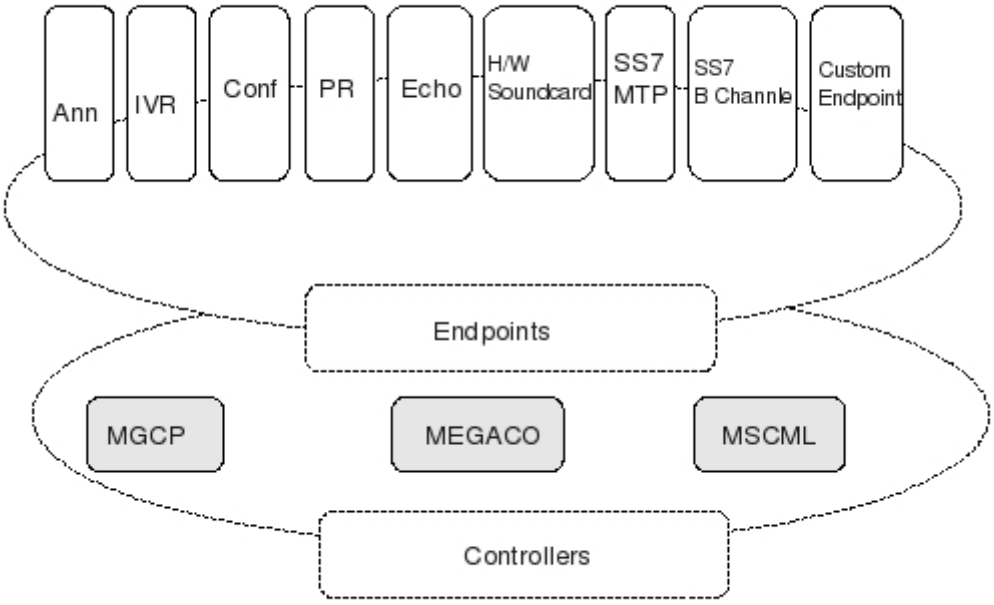
Media services have played an important role in the traditional Time Division Multiplexing (TDM)-based telephone network. As the network migrates to an IP-based environment, media services are

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 rapidly. This is carried out in a network architecture that supports the two concepts called *provisioning-on-demand* and *scaling-on-demand*.

1.3.1. 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 procedures 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 APIs.

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

Restcomm Media Server can also be controlled from Restcomm SIP Servlets or any other SIP

Servlets container using any of the above call control procedures or using the Restcomm JSR-309 Implementation. Restcomm JSR-309 Implementation internally leverages MGCP protocol to controll Media Server. Restcomm JSR-309 implementation details is out of scope of this document.

It is also possible to control the Restcomm 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 Restcomm Media Server.

Many key features of Restcomm 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.

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 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. From the points considered so far, the following basic endpoint types have been identified:

- 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 comprehensive. 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

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" that perform the conversion of media signal between circuit and packets, and at least one "signalling gateway" when connecting to an SS7 controlled network. The Call Agent can be distributed over several computer platforms.

Chapter 2. Installing the Media Server

The Restcomm Media Server distribution is available as Standalone

The 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 Media Server can be obtained by checking it out from its repository using the Git version control system. You can later run a build using Maven. The binary distribution is recommended for most users. Downloading 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 (): Installing, Configuring and Running

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

Restcomm Media Server either comes bundled with the JBoss Application Server or Standalone. This section details how to install the Restcomm Media Server that comes bundled with JBoss Application Server 5. For installation of Standalone Restcomm Media Server , refer to [_itms_binary_standalone_media_server_installing_configuring_and_running]

2.1.1. Pre-Install Requirements and Prerequisites

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

Hardware Requirements

Sufficient Disk Space

Once unzipped, the JBoss AS embedded Media Server binary release requires *at least* &MS_EMB_SIZE;of free disk space. Keep in mind that disk space requirements may change in future iterations.

Anything built for Java

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 ( ) 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.

2.1.2. Downloading

The latest version of the JBoss embedded Media Server is available from

http://www.mobicents.org/mms-downloads.html. The top row of the table holds the latest version. Click the Downloadlink to start the download.

2.1.3. 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.

Procedure: Installing the JBoss embedded Media Server BinaryDistribution 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 into which the files can be extracted. 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-3.0.0.FINAL.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.
   a. Java:

```
ms-<version>]$ jar -xvf “mms-3.0.0.FINAL.zip”
```

   b. Linux:

```
ms-<version>]$ unzip “mms-3.0.0.FINAL.zip”
```

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-3.0.0.FINAL.zip”
```

Procedure: Installing the JBoss embedded Media Server BinaryDistribution on Windows

1. For this procedure, it is assumed that the downloaded archive is saved in the *My Downloads* folder. You can also choose any convenient folder of your choice.
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.
   a. Move the downloaded zip file from *My Downloads* to the folder created in the previous step.
   b. Open the Windows Command Prompt and navigate to the folder that contains the archive using the `cd` command
   c. Execute the `jar -xvf` command to extract the archive contents into the current folder.

   ```
   C:\Users\<user>\My Downloads\ms-<version>\jar -xvf "mms-3.0.0.FINAL.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 Downloads\ms-<version>\delete "mms-3.0.0.FINAL.zip"
   ```

2.1.4. 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: Running the Media Server on Linux

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

   ```
   downloads]$ cd "mms-server"
   ```

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
```

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: 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 Startmenu and navigate to the correct folder:

```
C:\Users\<user>\My Downloads> cd "mms-server"
```

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

- *run.bat* batch file:

```
C:\Users\<user>\My Downloads\ms-<version>>bin\run.bat
```

- *run.jar* executable Java archive:

```
C:\Users\<user>\My Downloads\ms-<version>>java -jar bin\run.jar
```

2.1.5. 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. Directory Structure describes each directory, and the type of information contained within each location.

Table 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 Core Configuration File Set describes the location of the key configuration files, and provides a description of the

Table 2. 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.
conf/log4j.properties	Specifies the Apache log4j framework category priorities and appenders used by the Media Server.
conf/mgcp-conf.xml	Specifies the configuration for the MGCP controller.
conf/mediaserver.xml	Configuration file of the Media Server. For more information please see chapter 3.

File Name and Location	Description
deploy/server-beans.xml	Specified list of Java Beans necessary for bootstrapping the Media Server.

2.1.6. 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

Chapter 3. Configuring the Media Server

3.1. Network Configuration

Example 1. Network configuration

```
<network>
    <bindAddress>127.0.0.1</bindAddress>
    <externalAddress>127.0.0.1</externalAddress>
    <network>127.0.0.1</network>
    <subnet>255.255.255.255</subnet>
    <sbc>false</sbc>
</network>
```

Address Bindings

BindAddress is the address of the network interface to which Media Server is bound to. All RTP channels are open on this address.

ExternalAddress is the public address of the server. It is mainly used to patch SDP and to expose SRFLX candidates during ICE negotiation.

Network, Subnet and SBC

The Media server can work in two distinct modes: Standard or SBC.

When **sbc** is set to false, the Media Server will establish a channel based on address/port pair it receives from SDP response. However, if the RTP packet is used within a NAT context, data sent in SDP is the original data and not the NATted IP address/port (this is often the scenario). Furthermore, when **sbc** is set to false, data will be sent to invalid address and also will not be accepted since Media Server does not know the correct address of the UA.

In order to solve NAT issues the **sbc** option must be set to true. As result, Media Server will wait for first RTP packet; learn its remote IP address and port and only then it will send data based on the remote address and not on SDP response.

As consequence inter-server traffic will not work since both sides will be waiting for first packet. To solve this, you should set local network and local subnet to a range that will include all Media Servers in the same cluster. If the Media Server detects that the IP address and port in SDP are within the local IP network range, it will start sending packets immediately and not wait. This is similar to when **sbc** is set to false.

3.2. Controller Configuration

This configuration subset defines the default Media Server Controller, the core component that processes incoming requests and manages media resources.

The default controller is based on MGCP protocol. Enabling MGCP is always required as the JSR-309 driver is based on the current MGCP implementation as well.

Example 2. Controller Configuration

```
<controller protocol="mgcp">
    <address>127.0.0.1</address>
    <port>2427</port>
    <endpoints>
        <endpoint name="mobicents/bridge/"
class="org.mobicents.media.server.mgcp.endpoint.BridgeEndpoint" poolSize="50" />
        <endpoint name="mobicents/ivr/"
class="org.mobicents.media.server.mgcp.endpoint.IvrEndpoint" poolSize="50" />
        <endpoint name="mobicents/cnf/"
class="org.mobicents.media.server.mgcp.endpoint.ConferenceEndpoint" poolSize="50"
/>
    </endpoints>
    <configuration>mgcp-conf.xml</configuration>
    <poolSize>25</poolSize>
</controller>
```

3.2.1. Network

The **Address** parameter defines the address of the network interface to which the controller is bound to. The control channel is open on this address.

The **Port** parameter defines the port where the control channel will listen to for incoming requests. By default, it is set to 2427.

The **PoolSize** parameter defines the number of MGCP requests that Media Server can handle concurrently.

3.2.2. Endpoints

The **Endpoints** configuration allows you to configure all MGCP endpoint groups you want to use with the Media Server instance.

The **Configuration** parameter points to an XML file containing the definitions of MGCP packages, signals and packages mapping to endpoints.

The user can configure the initial **poolSize** for each endpoint type. This value will determine the number of endpoints that are preloaded on startup.

Endpoint Pooling

If all available endpoints are used and a request for additional endpoint is received, Media Server will allocate a new endpoint and store it in a resource pool.

The only exception to this rule is DS0 endpoint type. The DS0 endpoint can not be increased as it is directly related to the number of channels available on an E1/T1 card.

The **Name** parameter represents the name pattern to be used for MGCP requests that target a specific endpoint type. For example, the name pattern *mobicents/aap* will be created and accessed as *mobicents/aap/$* (where $ is an integer representing the endpoint ID).

The **Class** parameter represents the class of installer. Currently 2 types of installers are available:

- **VirtualEndpointInstaller** which allows you to install most endpoints
- **VirtualSS7EndpointInstaller** which is used for DS0 endpoints.

Endpoint class – defines the class which implements endpoints.

Currently, the following endpoint classes are available :

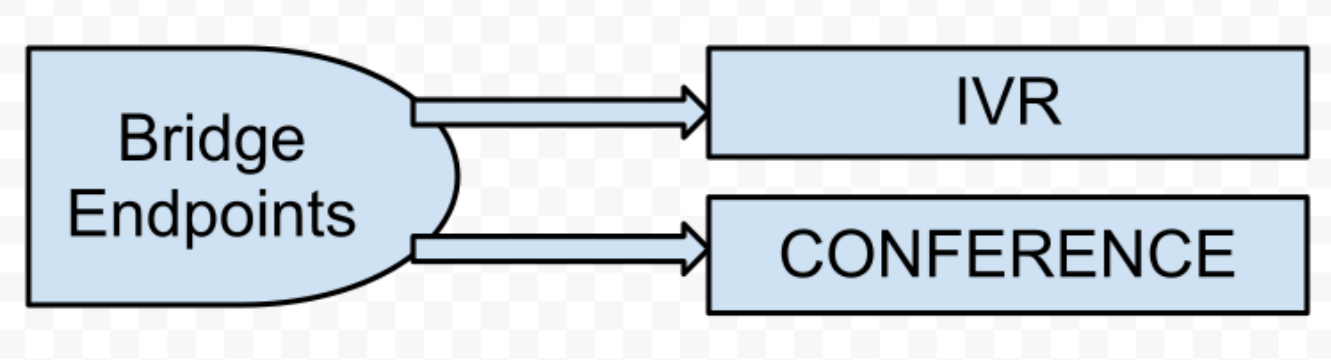
Table 3. Endpoint Classes

Endpoint Class	Available Resources	Connection Types	Relay Type
org.mobicents.media.server.mgcp.endpoint.AnnouncementEndpoint	Player	Local+RTP	Mixer
org.mobicents.media.server.mgcp.endpoint.IvrEndpoint	Player,Recorder,Dtmf Detector,Dtmf Generator	Local+RTP	Mixer
org.mobicents.media.server.mgcp.endpoint.ConferenceEndpoint	None	Local+RTP	Mixer
org.mobicents.media.server.mgcp.endpoint.BridgeEndpoint	None	Local+RTP	Splitter
org.mobicents.media.server.mgcp.endpoint.PacketRelayEndpoint	None	RTP	Mixer
org.mobicents.media.server.mgcp.endpoint.Ds0Endpoint	Signal Detector,Signal Generator	Local+RTP	Splitter

A Mixer endpoint will mix together data from both connections and resources. This implies that it will still be available even without the appropriate setting mode.

A Splitter endpoint is not a standard MGCP endpoint type, as it handles two different resources groups. The first group uses local connections whereas the second group uses RTP connections. This implies that non-RTP connections can communicate with RTP endpoints and none local connections can still access local endpoints.

The Bridge endpoint, a Splitter type, is useful in the scenarios shown below:



In this scenario, both IVR and Conference endpoints will be connected by a pair of Local Connections to the Bridge Endpoint. This is considered a good practice as there will be no cross over of IVR Resources (player , recorder , etc) to conference. The same applies to any conference traffic, it will not cross over to IVR. This is useful for recording RTP data for specific groups of users.

DS0 endpoint is a type Splitter. All connections, signal detector and signal generator are in group 1, while signalling channel SS7 is placed in group 2. That means that any SS7 channel data will be sent to any connection and signal detector, while data from signal generator and any connection will be sent only to SS7 channel.

Endpoint Groups

In order to configure multiple groups of endpoints of the same type per Media Server instance, you must change the name of each group.

3.3. Media Configuration

The media configuration contains definitions that have an impact on the media channels.

Example 3. Media Configuration

```
<media>
    <timeout>0</timeout>
    <lowPort>34534</lowPort>
    <highPort>65534</highPort>
    <jitterBuffer size="50" />
    <codecs>
        <codec name="l16" />
        <codec name="pcmu" />
        <codec name="pcma" />
        <codec name="gsm" />
        <codec name="g729" />
    </codecs>
</media>
```

3.3.1. RTP Channels

The **LowPort** and **HighPort** define the port range reserved for RTP channels. These values should be an even number, since odd ports are reserved for RTCP channels.

The **JitterBuffer** size parameter sets the maximum capacity of the jitter buffer, in milliseconds. Jitter Buffers are commonly configured to hold up to 50-60ms of audio.

3.3.2. RTP Timeout

Most SIP UA do not support any type of keep-alive between 200 OK and BYE. Therefore, in case the network goes down while a call is established, the call may hang forever. That is why RTP streaming should not be interrupted (exception to rule being *recvonly* and *inactive* modes). With the this in mind, the Media Server features the **RtpTimeout** parameter.

When **RtpTimeout** is set to greater than 0, the Media Server will monitor RTP traffic and if it finds period equal or greater then the RTP timeout (in seconds) it will delete the connection and notify the server that a connection was removed (by sending DLCX MGCP command). Consequently, any border server will receive a reliable notification that the call is still alive even when the communication network is no longer available.

When enabling RTP timeout, it is recommended that you do not set the mode to *inactive* or *sendonly* when you expect to receive data (after 180 or 200 OK) depending on your application

Example 4. Call Flow

```
Similar call flow may be like this

UA ----> INVITE ----> Control Server
Control Server ----> CRCX with mode inactive ----> Media Server
Control Server ----> INVITE ----> inside network or to other side
Inside network or other side ----> 183 with SDP ----> Control Server
Control Server ---> MDCX with mode sendonly ---> Media Server
Control Server ---> 183 with SDP ----> UA
Inside network or other side ----> 200 ----> Control Server
Control Server ---> MDCX with mode sendrecv ---> Media Server
Control Server ---> 200 ----> UA
```

In case of 180 or 183 without SDP response , intermediate MDCX is not required.

3.3.3. Codecs

Currently media server supports five codecs : G711 A/U, Linear PCM Raw, GSM, ILBC and G.729.

G.729 usage

Please note that a valid license is required to use G.729 , therefore you should purchase a license prior to enabling this codec.

If you decide to use a single codec for encoding or decoding data, you should leave one RAW or 2 Raw pair. This is useful only in case of a one way activity.

L16 usage

L16 codec is useful only in server to server communication where you have enough network bandwidth. It is not recommended to allow L16 codec for UA – server connections, this can lead to degradation of the signal quality due to increased jitter and packet loss.

3.4. Resources Configuration

In the current Media Server release, a global pool of resources is used to decrease garbage collection and allow for faster resource allocation.

Example 5. Resources Configuration

```
<resources>
    <localConnection poolSize="100" />
    <remoteConnection poolSize="50" />
    <player poolSize="50" />
    <recorder poolSize="50" />
    <dtmfDetector poolSize="50" dbi="-35" />
    <dtmfGenerator poolSize="50" toneVolume="-20" toneDuration="80" />
    <signalDetector poolSize="0" />
    <signalGenerator poolSize="0" />
</resources>
```

As seen above, default pool sizes are configured for each possible type of media components:

- **Local Connection** - Link between two MGCP Endpoints;
- **Remote Connection** - Link between an MGCP Endpoint and a remote peer;
- **Player** - Plays audio tracks;
- **Recorder** - Records audio streams;
- **DTMF Generator** - Generates out-of-band DTMF tones;
- **DTMF Detector** - Detects both inband and out-of-band tones;
- **Signal Detector** - Detects SS7 tones;
- **Signal Generator** - Generates SS7 tones.

About DTMF Detector Dbi

Audio data is mixed with DTMF inband tones often. As result, Media Server may detect false positive tones, or it may not detect tones which are sent.

By setting DTMF detector dbi parameter, the user can optimize tone detection by fine-tuning this parameter. However, default value has been tested and found to be generally appropriate.

Good practice mandates that inband tones should be used only in SS7/IP mixed network. IP-only networks should use out-of-band tones only and disable inband detection.

Signal Detector and Signal Generator are currently only used for connectivity tests for DS0 channel (COT isup signal), CO1, CO2, CT (Continuity Transport) and Loopback test modes.

For more information please see [_msep_ms_event_packages]

When the specified resource type is not available

Please note that a resource will be automatically allocated if the specified resource type is not available in the resource pool. This will require more memory allocation and in some cases may impact performance.

The more resources you have pre-configured on startup in the resource pool, the more memory the Media Server will require on startup. It is up to the user to decide the best trade-off for the setup (greater memory usage on startup vs slower response when new resources are required in runtime).

Chapter 4. Controlling and Programming the Restcomm Media Server

4.1. Restcomm Media Server Control Protocols

The Restcomm Media Server adopts a call control architecture where the call control “intelligence” is located outside of the Media Server itself, and is handled by external call control elements collectively known as Call State Control Function (CSCF).The media server assumes that these call control elements will synchronize with each other to send coherent commands and responses to the media servers under their control. Server Control Protocols is, in essence, an asynchronous master/slave protocol, where the Server Control Modules are expected to execute commands sent by CSCF. Each Server Control Module is implemented as a JSLEE application, and consists of a set of Service Building Blocks ()s, which are in charge of communicating with media server endpoints via SPI. Such an architecture avoids difficulties with programming concurrency, low-level transaction and state-management details, connection-pooling and other complex APIs.

4.1.1. Media Gateway Control Protocol Interface

The Media Gateway Control Protocol (MGCP) is a protocol for controlling media gateways (for example, the Media Server) from external call control elements such as media gateway controllers or Call Agents. The MGCP assumes that the Call Agents, will synchronize with each other to send coherent commands and responses to the gateways under their control.

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

- which events should be reported to the Call Agent;
- how endpoints should be connected; and,
- which signals should be played on which endpoints.

MGCP is, in essence, a master/slave protocol, where the gateways are expected to execute commands sent by the Call Agents. The general base architecture and programming interface is described in RFC 2805, and the current specific MGCP definition is located in RFC 3435.

4.1.2. JSR-309 Control protocol Interface

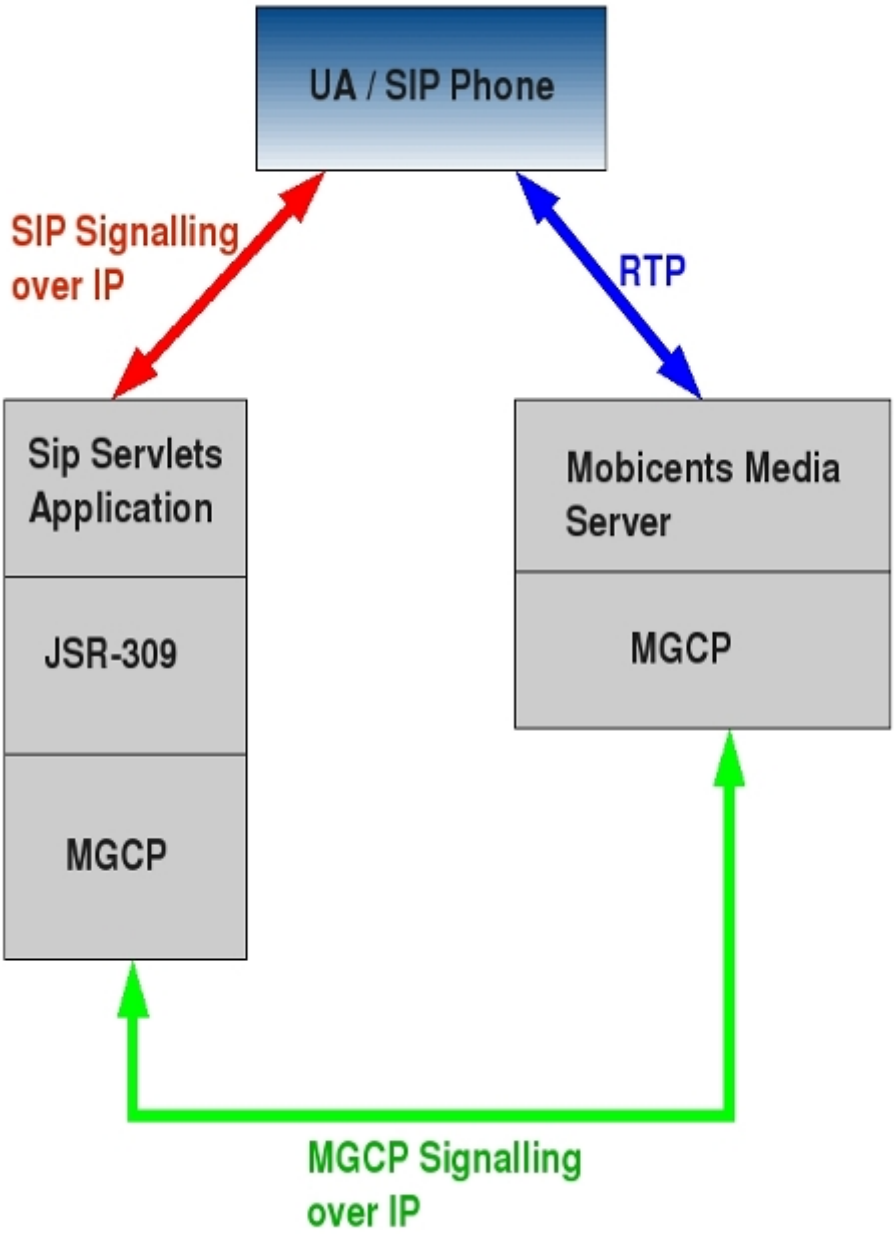
JSR-309 defines a programming model and object model for Media Server (MS) control independent of MS control protocols. JSR-309 API is not an API for a specific protocol. It will take advantage of the multiple and evolving Multimedia Server capabilities available in the industry today and also provide an abstraction for commonly used application functions like multi party conferencing, multimedia mixing and interaction dialogs.

Some of the commonly used MS control protocols are MGCP (RFC 3435), MEGACO (RFC 3525), Media Server Markup Language (MSML) (RFC 4722) and VoiceXML. The Restcomm implementation of JSR-309 API makes use of MGCP as MS control protocol.

The Restcomm JSR-309 Impl is first and only open source implementation of JSR-309 available as of today. To further understand the JSR-309 API, download specs from here

“The latest release of Restcomm JSR-309 Impl is part of binary media server release and can be found under clients folder”.

The diagram bellow shows the high-level architecture of how application can make use of JSR-309 over MGCP



Chapter 5. RMS: Event Packages

Announcement Package(A): `org.mobicents.media.server.mgcp.pkg.ann`

Table 4. Announcement Package(A):

Signal Name	Description
Ann	Play Announcement
oc	Operation Completed
of	Operation Failed

For more information please see: https://tools.ietf.org/html/rfc3660#section-2.12

`Advanced Audio Package(AU): org.mobicents.media.server.mgcp.pkg.au`

Table 5. Advanced Audio Package(AU):

Signal Name	Description
pa	Play Announcement
pc	Play Collect
pr	Play Record
es	End Signal
oc	Operation Completed
of	Operation Failed

For more information please see: https://tools.ietf.org/html/rfc2897

`Trunk Package(T): org.mobicents.media.server.mgcp.pkg.trunk`

Table 6. Trunk Package(T):

Signal Name	Description
co1	Continuity 1
co2	Continuity 2
lp	Loopback
ct	Continuity Transpoder
oc	Operation Completed
of	Operation Failed

For more information please see: https://tools.ietf.org/html/rfc3660#section-2.3

Dtmf Package(D): `org.mobicents.media.server.mgcp.pkg.dtmf`

Table 7. Dtmf Package(D):

Signal Name	Description
0	Tone 0
1	Tone 1
2	Tone 2
3	Tone 3
4	Tone 4
5	Tone 5
6	Tone 6
7	Tone 7
8	Tone 8
9	Tone 9
*	Tone *
#	Tone #
A	Tone A
B	Tone B
C	Tone C
D	Tone D
oc	Tone Operation Completed
oc	Operation Failed

For more information please see: https://tools.ietf.org/html/rfc3660#section-2.2

`Signal List Package (SL): org.mobicents.media.server.mgcp.pkg.sl `

Table 8. Signal List Package (SL):

Signal Name	Description
s	List of signals to be generated
oc	Operation Completed
of	Operation Failed

For more information please see: https://tools.ietf.org/html/rfc3660#section-2.8

5.1. List of supported packages per Endpoint

Configuration of supported packages and events can be done through the mgcp-conf.xml configuration file that is located in the *conf* directory.

The table belows shows supported packages and the related enabled endpoints

Table 9. List of Supported Packages per Endpoint

Package name	**Endpoints**
Ann	Play announcement
Announcement	IVR, Announcements
Advanced Audio	IVR
Trunk	DSO
Dtmf	IVR
Signal List	IVR

Chapter 6. Restcomm Media Server Demonstration Example

Restcomm Media Server 3.0.0.FINAL is a standalone media software. Control panels servers are required in order to make Restcomm Media Server part of your network architecture. Two controlling protocols (JSR-309 and MGCP) are currently enabled to work with Restcomm Media Server 3.0.0.FINAL.

Depending on your needs, Jain-slee , Sip Servlets or RestComm-Connect can be used to control the media server. The examples below are included in the &Restcomm Media Server_SHORT_NAME; software.

To get Jain-Slee examples please see following examples that are part of mobicents / telestax jain Slee release :

Table 10. Example for Jain-Slee

Jain-Slee Example List
call-controller2 which uses MGCP to control
mgcp-demo which uses MGCP to control
mscontrol-demo which uses JSR-309 to control

For more information please see Jain-Slee documentation that comes with this examples.

The following examples use Restcomm Media Server with Restcomm SIP Servlets :

Table 11. Examples for SIP Servlets

SIP Servlets Example List
conference-demo jsr-309
media-jsr-309-servlet
shopping-demo-jsr309

All servlets examples are using jsr-309 control protocol.

For more information please see SIP Servlets documentation that comes with these examples.

1. Using Restcomm Media Server with RestComm

with RestComm
RMS is required in order to successfully work with RestComm. Consequently, any example created to demonstrate the use of RestComm will incorporate Restcomm Media Server (RMS). RestComm use MGCP as the controlling protocol for RMS.

For more information please see RestComm-Connect documentation.

Chapter 7. Restcomm Media Server: Best Practices

7.1. Restcomm Media Server Best Practices

Note: these best practices apply to Restcomm Media Server version 5.0.0-SNAPSHOT and later

7.1.1. DTMF Detection Mode: RFC2833 versus Inband versus Auto

The Restcomm Media Server will block the resource depending on the DTMF detection mode configured in *jboss-service.xml* at start-up time. Inband is highly resource-intensive and must perform many more calculations in order to detect DTMF when compared to RFC2833. So if your application already knows that User Agents (UAs) support RFC2833, it is always better to configure DTMF mode as *RFC2833* rather than as *Inband* or *Auto*. Also, please note that *Auto* is even more resource-intensive because it does not know beforehand whether DTMF would be Inband or *RFC2833*, and hence both detection methods must be started. The default detection mode is *RFC2833*.

All of the Conference, Packet Relay and IVR endpoints have DTMF detection enabled; the mode can be configured using *jboss-service.xml*. We advise retaining the same mode for all three, but this is not a necessity.

7.1.2. Transcoding Is CPU-Intensive

Digital Signal Processing (DSP) is very costly and should be avoided as much as possible. By default, Announcement endpoints and IVR endpoints do not have DSP enabled. What this means is that your application needs to know beforehand which codecs are supported by your UA; you can then ask Announcement or IVR to play an audio file which has been pre-encoded in one of these formats. The onus of deciding which pre-encoded file to play lies with the application. For example, if I am writing a simple announcement application that would only play announcements to the end user, and I know that my end users have one of either the *PCMU* or *GSM* codecs, then I would make sure to have pre-encoded audio files such as *helloworld-pcmu.wav* and *helloworld-gsm.gsm*. Then, when the UA attempts to connect to the Media Server, my application knows which codecs the UA supports and can ask the Restcomm Media Server to play the respective file.

This strategy will work fine because, most of the time in the telecommunications world, applications have a known set of supported codecs, .However if this is not true, or if you are writing a simple demo application and need or want all codecs to be supported, you can put a Packet Relay endpoint in front of Announcement or IVR endpoint. This way, the Packet Relay will do all necessary digital signal processing, and your application need not bother about which audio file to play. The audio file in this case will be encoded in *Linear* format, and all UAs, irrespective of whether they support *PCMU*, *PCMA*, *Speex*, *G729* or *GSM* codecs, would be able to hear the announcement.

Appendix A: Understanding Digital Signal Processing and Streaming

The following information provides a basic introduction to Digital Signal Processing, and Streaming technologies. These two technologies are used extensively in the Media Server, therefore understanding these concepts will assist developers in creating customized media services for the Media Server.

Introduction to Digital Signal Processing

Digital Signal Processing, as the name suggests, is the processing of signals by digital means. A signal in this context can mean a number of different things. Historically the origins of signal processing are in electrical engineering, and a signal here means an electrical signal carried by a wire or telephone line, or perhaps by a radio wave. More generally, however, a signal is a stream of information representing anything from stock prices to data from a remote-sensing satellite. The term "digital" originates from the word "digit", meaning a number, therefore "digital" literally means numerical. This introduction to DSP will focus primary on two types digital signals: audio and voice.

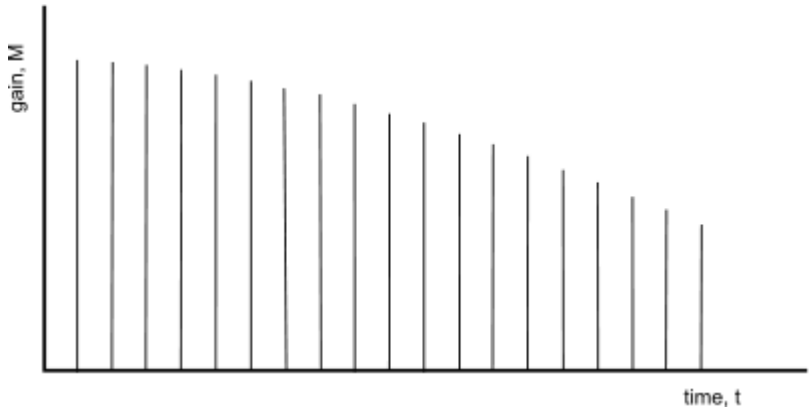
Analog and Digital Signals

Data can already be in a digital format (for example, the data stream from a Compact Disk player), and will not require any coversion. In many cases however, a signal is received in the form of an analog electrical voltage or current, produced by a microphone or other type of transducer. Before DSP techniques can be applied to an analog signal, it must be converted into digital form. Analog electrical voltage signals can be digitized using an analog-to-digital converter (ADC), which generates a digital output as a stream of binary numbers. These numbers represent the electrical voltage input to the device at each sampling instant.

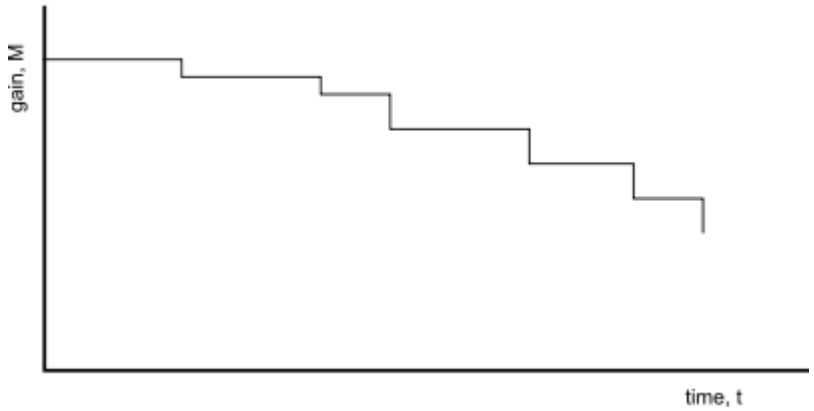
Discrete Signals

When converting a continuous analog signal to a digital signal, the analog signal must be converted to a signal format that computers can analyze and perform complex calculations on. Discrete Signals are easily stored and transmitted over digital networks and have the ability to be discrete in magnitude, time, or both.

Discrete-in-time values only exist at certain points in time. For example, if a sample of discrete-in-time data is taken at a point in time where there is no data, the result is zero.



Discrete-In-Magnitude values exist across a time range, however, the value of the datum in each time range consists of one constant result, rather than a variable set of results.



By converting continuous analog signals to discrete signals, finer computer data analysis is possible, and the signal can be stored and tranmitted efficiently over digital networks.

Sampling, Quantization, and Packetization

Sampling is the process of recording the values of a signal at given points in time. For ADCs, these points in time are equidistant, with the number of samples taken during one second dictating the called sample rate. It's important to understand that these samples are still analogue values. The mathematic description of the ideal sampling is the multiplication of the signal with a sequence of direct pulses.

Quantization is the process of representing the value of an analog signal by a fixed number of bits. The value of the analog signal is compared to a set of pre-defined levels. Each level is represented by a unique binary number, and the binary number that corresponds to the level closest to the analog signal value is chosen to represent that sample.

Sampling and quantization prepare digitized media for future processing or streaming. However, streaming and processing over individual samples is not effective for high volumes of data transferred via a network. The risk of data-loss is much higher when a large portion of data is transferred in a block. Networked media should be transmitted using media packets that carry several samples, thereby reducing the risk of data loss through the transmission process. This process is referred to as packetization.

Transfer Protocols

The Real-time Streaming Protocol (RTSP), Real-time Transport Protocol (RTP) and the Real-time Transport Control Protocol (RTCP) were specifically designed to stream media over networks. The latter two are built on top of UDP.

Real-time Transport Protocol

RTP provides end-to-end network transport functions suitable for applications transmitting real-time data, such as audio, video or simulation data, over multicast or unicast network services. RTP does not address resource reservation and does not guarantee quality-of-service for real-time services. The data transport is augmented by the Real-time Control Protocol (RTCP) to allow monitoring of the data delivery in a manner scalable to large multicast networks, and to provide minimal control and identification functionality. RTP and RTCP are designed to be independent of the underlying transport and network layers.

A RTP packet consists of a RTP header, followed by the data to send. In the RTP specification, this data is referred to as the payload. The header is transmitted in network byte order, just like the IP header. The Figure 5 shows the RTP header format.

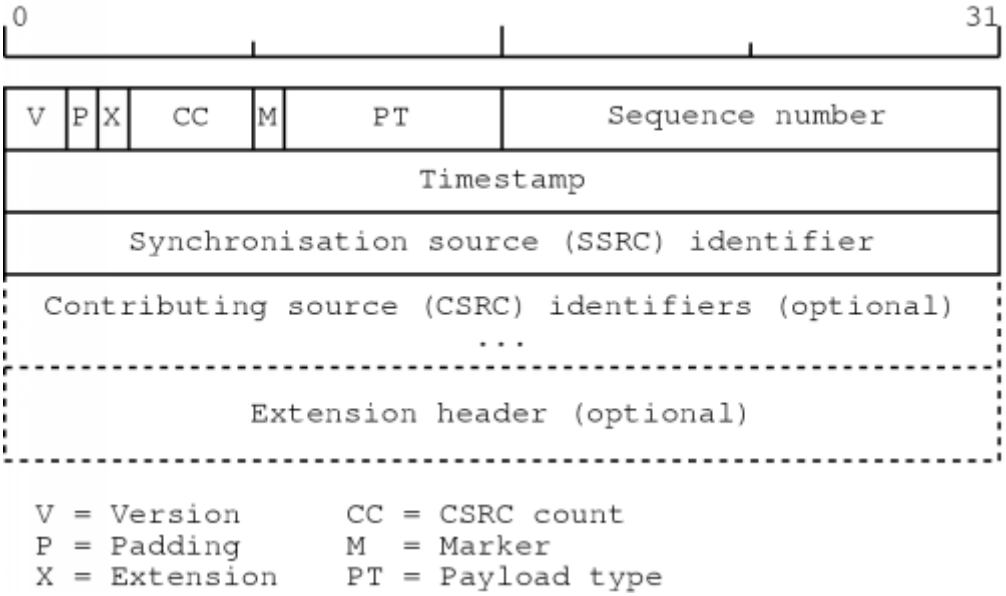


Table 12. Supported RTP Formats

Header Component	Description
V (Version)	Contains the version number of the RTP protocol. For example, the current version number is 2. This part of the header consumes 2 bits of the RTP packet.
P (Padding)	Contains padding bytes, which are excluded from the payload data count. The last padding byte contains the number of padding bytes present in the packet. Padding may be required for certain encryption algorithms that need the payload to be aligned on a multi-byte boundary.

Header Component	Description
X (Extension)	Specifies whether the header contains an Extension Header.
CC (CSRC Count)	Specifies how many contributing sources are specified in the header.
M (Marker)	Contains arbitrary data that can be interpreted by an application. The RTP specification does not limit the information type contained in this component of the header. For example, the Marker component might specify that media data is contained within the packet.
PT (Payload Type)	Specifies the type of data the packet contains, which determines how an application receiving the packet interprets the payload.
Sequence Number	Contains a unique numerical value, that can be used by applications to place received packets in the correct order. Video streams rely on the sequence number to order the packets for individual video frames received by an application. The starting number for a packet stream is randomized for security reasons.
Time Stamp	Contains the synchronization information for a stream of packets. The value specifies when the first byte of the payload was sampled. The starting number for the Time Stamp is also randomized for security reasons. For audio, the timestamp is typically incremented with the amount of samples in the packet so the receiving application can play the audio data at exactly the right time. For video, the timestamp is typically incremented per image. One image of a video will generally be sent in several packets, therefore the pieces of data will have the same Time Stamp, but use a different Sequence Number.
SSRC ID	Contains the packet Synchronization Source (SSRC) identifier of the sender. The information contained in this component of the header is used to correctly order multiple RTP streams contained in a packet. This scenario often occurs when an application sends both video and audio RTP streams in one packet. So the receiving application can correctly order and synchronize the data, the identifier is chosen randomly. This reduces the chance of a packet in both streams having the same identifier.

Header Component	Description
CSRC ID	Contains one (or more) Contributing Source (CSRC) identifiers for each RTP stream present in the packet. To assist audio streams re-assembly, the SSRC IDs can be appended to this packet component. The SSRC ID of the packet then becomes the source identifier for the forwarded packet.
Extension Header	Contains arbitrary information, specified by the application. The RTP defines the extension mechanism only. The extensions contained within the Extension Header are controlled by the application.

RTP headers do not contain a payload length field. The protocol relies on the underlying protocol to determine the end of the payload. For example, in the TCP/IP architecture, RTP is used on top of UDP, which does contain length information. Using this, an application can determine the size of the whole RTP packet and after its header has been processed, the application automatically knows the amount of data in its payload section.

Real-time Transport Control Protocol

The RTP is accompanied by a control protocol, the Real-time Transport Control Protocol (RTCP). Each participant of a RTP session periodically sends RTCP packets to all other participants in the session for the following reasons:

- To provide feedback on the quality of data distribution. The information can be used by the application to perform flow and congestion control functions, and be used for diagnostic purposes.
- To distribute identifiers that are used to group different streams together (for example, audio and video). Such a mechanism is necessary since RTP itself does not provide this information.
- To observe the number of participants. The RTP data cannot be used to determine the number of participants because participants may not be sending packets, only receiving them. For example, students listening to an on-line lecture.
- To distribute information about a participant. For example, information used to identify students in the lecturer's conferencing user-interface.

There are several types of RTCP packets that provide this functionality.

- Sender
- Receiver
- Source Description
- Application-specific Data

Sender reports (SR) are used by active senders to distribute transmission and reception statistics. If

a participant is not an active sender, reception statistics are still transmitted by sending receiver reports (RR).

Descriptive participant information is transmitted in the form of Source Description (SDES) items. SDES items give general information about a participant, such as their name and e-mail. However, it also includes a canonical name (CNAME) string, which identifies the sender of the RTP packets. Unlike the SSRC identifier, the SDES item stays constant for a given participant, is independent of the current session, and is normally unique for each participant. Thanks to this identifier it is possible to group different streams coming from the same source.

There is a packet type that allows application-specific data (APP) to be transmitted with RTP data. When a participant is about to leave the session, a goodbye (BYE) packet is transmitted.

The transmission statistics which an active sender distributes, include both the number of bytes sent and the number of packets sent. The statistics also include two timestamps: a Network Time Protocol (NTP) timestamp, which gives the time when this report was created, and a RTP timestamp, which describes the same time, but in the same units and with the same random offset of the timestamps in the RTP packets.

This is particularly useful when several RTP packet streams have to be associated with each other. For example, if both video and audio signals are distributed, there has to be synchronization between these two media types on playback, called inter-media synchronization. Since their RTP timestamps have no relation whatsoever, there has to be some other way to do this. By giving the relation between each timestamp format and the NTP time, the receiving application can do the necessary calculations to synchronize the streams.

A participant to a RTP session distributes reception statistics about each sender in the session. For a specific sender, a reception report includes the following information:

- Fraction of lost packets since the last report. An increase of this value can be used as an indication of congestion.
- Total amount of lost packets since the start of the session.
- Amount of inter-arrival jitter, measured in timestamp units. When the jitter increases, this is also a possible indication of congestion.
- Information used by the sender to measure the round-trip propagation time to this receiver. The round-trip propagation time is the time it takes for a packet to travel to this receiver and back.

Because the RTCP packets are sent periodically by each participant to all destinations, the packet broadcast interval should be reduced as much as possible. The RTCP packet interval is calculated from the number of participants and the amount of bandwidth the RTCP packets may occupy. To stagger the broadcast interval of RTCP packets to participants, the packet interval value is multiplied by a random number.

Jitter

The term Jitter refers to processing delays that occur at each endpoint, and are generally caused by packet processing by operating systems, codecs, and networks. Jitter affects the quality of the audio

and video stream when it is decoded by the receiving application.

End-to-end delay is caused by the processing delay at each endpoint, and may be caused in part by IP packets travelling through different network paths from the source to the destination. The time it takes a router to process a packet depends on its congestion situation, and this may also vary during the session.

Although a large overall delay can cause loss of interactivity, jitter may also cause loss of intelligibility. Though Jitter cannot be totally removed, the effects can be reduced by using a Jitter Buffer at the receiving end. The diagram below shows effect with media buffer and without media buffer

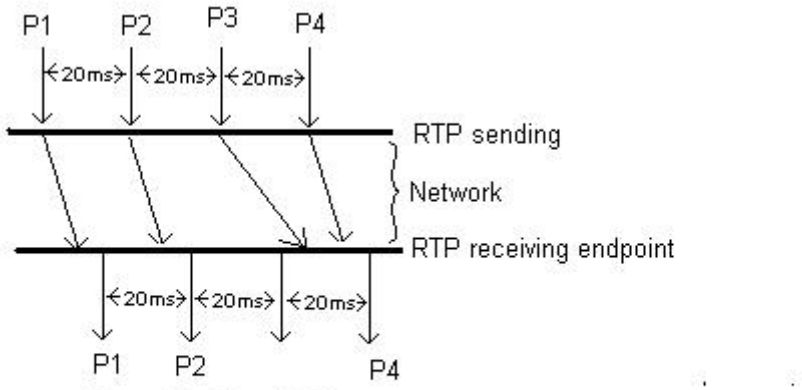


Fig a. No Jitter Buffer

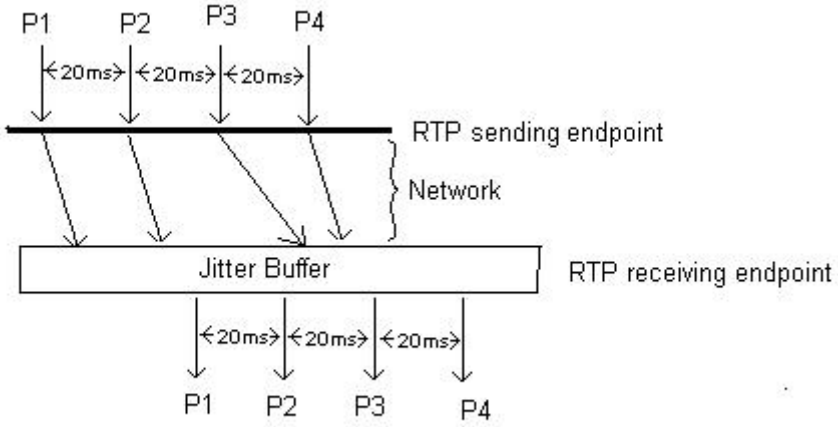


Fig b. With Jitter Buffer

Fig a. Shows that packet 3 is lost as it arrived late. Fig b uses Jitter buffer and hence arrived packets are stored in jitter and media components reads from Jitter once its half full. This way even if Packet 3 arrives little late, its read by the components.

Appendix B: Java Development Kit (): Installing, Configuring and Running

The [app]` Platform` is written in Java; therefore, before running any server, you must have a working Java Runtime Environment () or Java Development Kit () installed on your system. In addition, the JRE or JDK you are using to run [app] must be version 5 or higher [1: At this point in time, it is possible to run most servers, such as the JAIN SLEE, using a Java 6 JRE or JDK. Be aware, however, that presently the XML Document Management Server does not run on Java 6. We suggest checking the web site, forums or discussion pages if you need to inquire about the status of running the XML Document Management Server with Java 6.].

Should I Install the JRE or JDK?

Although you can run servers using the Java Runtime Environment, we assume that most users are developers interested in developing Java-based, [app]-driven solutions. Therefore, in this guide we take the tact of showing how to install the full Java Development Kit.

Should I Install the 32-Bit or the 64-Bit JDK, and Does It Matter?

Briefly stated: if you are running on a 64-Bit Linux or Windows platform, you should consider installing and running the 64-bit JDK over the 32-bit one. Here are some heuristics for determining whether you would rather run the 64-bit Java Virtual Machine (JVM) over its 32-bit cousin for your application:

- 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 gives virtually unlimited (1 exabyte) heap allocation. However large heaps 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. Barring memory issues, 64-bit hardware with the same relative clock speed and architecture is not likely to run Java applications faster than their 32-bit cousin.

Note that the following instructions detail how to download and install the 32-bit JDK, although the steps are nearly identical for installing the 64-bit version.

Downloading

You can download the Sun JDK 5.0 (Java 2 Development Kit) from Sun's website: http://java.sun.com/javase/downloads/index_jdk5.jsp. Click on the Download link next to "JDK 5.0 Update <x>`" (where [replaceable]<x>` is the latest minor version release number). On the next page, select your language and platform (both architecture—whether 32- or 64-bit—and operating system), read and agree to the Java Development Kit 5.0 License Agreement, and proceed to the download page.

The Sun website will present two download alternatives to you: one is an RPM inside a self-extracting file (for example, *jdk-1_5_0_16-linux-i586-rpm.bin*), and the other is merely a self-extracting file (e.g. *jdk-1_5_0_16-linux-i586.bin*). If you are installing the JDK on Red Hat Enterprise

Linux, Fedora, or another RPM-based Linux system, we suggest that you download the self-extracting file containing the RPM package, which will set up and use the SysV service scripts in addition to installing the JDK. We also suggest installing the self-extracting RPM file if you will be running [app]``` in a production environment.

Installing

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

Procedure: Installing the JDK on Linux

1. Regardless of which file you downloaded, you can install it on Linux by simply making sure the file is executable and then running it:

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

You Installed Using the Non-RPM Installer, but Want the SysV Service Scripts

If you download the non-RPM self-extracting file (and installed it), and you are running on an RPM-based system, you can still set up the SysV service scripts by downloading and installing one of the `-compat` packages from the JPackage project. Remember to download the `-compat` package which corresponds correctly to the minor release number of the JDK you installed. The compat packages are available from link:ftp://jpackage.hmdc.harvard.edu/JPackage/1.7/generic/RPMS.non-free/.

You do not need to install a `-compat` package in addition to the JDK if you installed the self-extracting RPM file! The `-compat` package merely performs the same SysV service script set up that the RPM version of the JDK installer does.

Procedure: Installing the JDK on Windows

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

Configuring

Configuring your system for the JDK consists in two tasks: setting the `JAVA_HOME` environment variable, and ensuring that the system is using the proper JDK (or JRE) using the `alternatives` command. Setting `JAVA_HOME` usually overrides the values for `java`, `javac` and `java_sdk_1.5.0` in `alternatives`, but we will set them all just to be safe and consistent.

Setting the `JAVA_HOME` Environment Variable on Generic Linux

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

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

As the root user, call `/usr/sbin/alternatives` with the `--config java` option to select between

JDKs and JREs installed on your system:

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

Finally, to make sure that you are using the correct JDK or Java version (5 or higher), and that the java executable is in your `PATH`, run the `java -version ` command in the terminal from your home directory:

```
~]$ 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

There is usually no reason (other than space concerns) to remove a particular JDK from your system, given that you can switch between JDKs and JREs easily using `alternatives`, and/or by setting `JAVA_HOME`.

Uninstalling the JDK on Linux

On RPM-based systems, you can 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 command, or use `Add/Remove Programs`.

Appendix C: Setting the JBOSS_HOME Environment Variable

The [app]` Platform` () is built on top of the [app]. You do not need to set the JBOSS_HOME environment variable to run any of the [app]` 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 Unix

At the command line, echo $JBOSS_HOME to see if it is currently defined in your environment:

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

The [app]` Platform` and most &THIS.PLATFORM; servers are built on top of the ([app]). When the [app]` Platform` or &THIS.PLATFORM; servers are built *from source*, then JBOSS_HOME *must* be set, because the &THIS.PLATFORM; files are installed into (or "over top of" if you prefer) a clean installation, and the build process assumes that the location pointed to by the JBOSS_HOME environment variable at the time of building is the [app] installation into which you want it to install the &THIS.PLATFORM; files.

This guide does not detail building the [app]` Platform` or any &THIS.PLATFORM; servers from source. It is nevertheless useful to understand the role played by JBoss AS and JBOSS_HOME in the &THIS.PLATFORM; 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 Unix and Windows

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 [app]` Platform` binary distribution.
- ...you have installed a &THIS.PLATFORM;server binary distribution *which bundles [app]``*.

You *MUST* set JBOSS_HOME if...

- ...you are installing the [app]` Platform` or any of the &THIS.PLATFORM; servers *from source*.
- ...you are installing the [app]` Platform` binary distribution, or one of the &THIS.PLATFORM; server binary distributions, which *do not* bundle [app]``.

Naturally, if you installed the [app]` Platform` or one of the &THIS.PLATFORM; server binary

releases which *do not* bundle `, yet requires it to run, then you should install before setting [var]`JBOSS_HOME or proceeding with anything else.

Setting the JBOSS_HOME Environment Variable on Unix

The JBOSS_HOME environment variable must point to the directory which contains all of the files for the [app]` Platform` or individual &THIS.PLATFORM; 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 Unix, 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: To Set JBOSS_HOME on Unix...

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 [2: Note that any other terminals which were opened prior to your having altered .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:

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

   ```
   ~]$ echo $JBOSS_HOME
   /home/silas/<path>/<to>/<install_directory>
   ```

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 &THIS.PLATFORM;Platform or individual &THIS.PLATFORM;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.

Appendix D: Revision History