Avaya Solution & Interoperability Test Lab

Avaya IP Telephone JMF Based Stand-alone RTP Streaming Sample Application – Issue 1.0

Abstract

This document describes the installation and operation of a sample application for the Audio Push and Transmit Audio features of Avaya 4600 and 9600 Series IP Telephones. The sample application demonstrates how to use Sun Microsystems' Java Media Framework (JMF) classes to implement the RTP audio streaming capabilities required by the Audio Push and Transmit Audio features. The Audio Push application will work on Avaya 4600 and 9600 Series IP Telephones. The Transmit Audio application will work only on Avaya 9600 Series IP Telephones.

1. Introduction

Avaya 4600 and 9600 Series IP Telephones have an embedded Push API that can be used to stream audio announcements to the phone. Audio Push is the ability to transmit RTP streams to the endpoint. This Push type provides additional capabilities such as start and stop to control audio streams.

Avaya 9600 Series IP Telephones also support an additional type of Push Message known as a Transmit Audio Push that allows the end user to send an audio stream from the phone to a server based application.

The sample application described in this document uses Sun Microsystems' Java Media Framework (JMF) classes to receive an Audio Push or Transmit Audio request from an Avaya IP Telephone and process the RTP stream.

2. Equipments and Softwares Used

Equipment	**Software**
Avaya™ 4620 IP Telephone	Firmware R2.8 H.323 release for the 4600 IP Telephones
Avaya™ 9630 IP Telephone	Firmware R1.5 H.323 release for the 9600 IP Telephones
PC	Windows XP Professional JRE version 1.5.0_11 JMF version 2.1.1

3. Configuration and Prerequisites

The audio push application requires Sun's Java Media Framework (JMF) installation to stream a WAV file to a given IP and port.

See http://java.sun.com/products/java-media/jmf/2.1.1/download.html for download and installation details for JMF.

A Java development environment is required to compile and run the application. Download and install Sun's Java Development Kit (JDK). Installing the JDK will install the Java Runtime Environment (JRE) on the system. See http://java.sun.com/javase/downloads/index_jdk5.jsp to download and install the Java Development Kit (JDK).

Modify the CLASSPATH environment variable in order to execute the sample application. The CLASSPATH should contain an entry that points to the Java classes

produced from the sample application source code. The CLASSPATH must also include the location of the JMF jar files.

The IP Address of the machine executing the sample application should be listed in the TPS (Trusted Push Server) list of the IP Telephone.

4. Installation and contents of sample application

Installation of the sample application is a two-step process. The first step involves extracting the .zip file to a user-defined location and the second step involves setting up the CLASSPATH variable to point to this user-defined location.

This sample code can also be installed in any suitable Java IDE such as Eclipse, NetBeans, etc.

4.1 Java Source

The source is divided into two packages:

1) **com.avaya.ipphones.samples** containing the server files and one client file.
2) **com.avaya.ipphones.exceptions** containing the exception specific files for push and transmit audio respectively.

Package **com.avaya.ipphones.samples** contains the following Java files.

1) **PushAudioServer.java:** This is a server side application that will receive the request from the IP phone for the Audio Push content and stream the audio file to the IP Telephone. The file to stream is specified as part of the request. Currently the application supports only '.wav' files and will only process HTTP GET requests (as opposed to POSTs) from the IP Telephone.

2) **TransmitAudioServer.java:** This is a server side application that will receive the request from the IP phone for the Transmit Audio Push content, process the audio stream from the IP Telephone, and store it to a file specified in the input request. Currently the application supports only '.wav' files and will only process HTTP GET requests (as opposed to POSTs) from the IP Telephone.

3) **ConfigStrings.java:** This file contains code related to configurations and errors. This is used within the servers to display information to the user in case an error occurs.

4) **MediaClientImpl.java:** This file implements a simple client. Its only use is to drive the server application. It generates the initial Push Request to the Avaya IP telephone and uses the Push Request post field parameters to pass the file name to the server to stream (Push Audio) or create (Transmit Audio Push). It requires

three parameters for successful execution: the type of push to perform (“push” for Audio Push or “transmit” for Transmit Audio Push), the IP address of the server running the sample server applications, and the IP Address of the Avaya IP Telephone to push the audio to or receive the audio from.

Package **com.avaya.ipphones.exceptions** contains the following Java files.

1) **AudioStreamException.java:** Class to display exceptions encountered while executing the PushAudioServer.

2) **AudioTransmitException.java:** Class to display exceptions encountered while executing the TransmitAudioServer.

The following variables present in the sample application are hard coded. These values should be modified to execute the sample application properly in the local environment.

Variable	File in which the variable is present	Description	Example
m_port	PushAudioServer.java	This variable specifies the port on which the audio server listen’s for requests from the IP Telephone.	m_port=8989
port	TransmitAudioServer.java	This variable specifies the port on which the Transmit audio server listen’s for requests from the IP Telephone.	Port=8990
pushAudioFile	MediaClientImpl.java	This variable specifies the location of the audio file to be streamed to the phone during an audio push request.	pushAudioFile= "C:/TransmitAudio/ music.wav"
transmitStore	MediaClientImpl.java.	This variable specifies the location where the audio file would be stored on the server during Transmit audio push	transmitStore= “C:/TransmitAudio”

Note: After modifying the variables, compile the sample application in any suitable Java Development environment.

5. Running the sample application

The sample application has two parts, one to push the audio to the IP Telephone and the other to receive audio from an IP Telephone and store it. Hence there are two servers, one server for each.

5.1 Audio Push

File Name: **PushAudioServer**

At the command prompt, navigate to the directory where the .zip file is extracted. Since the CLASSPATH is setup, it is not required to navigate to the directory where the actual class file is present. From this directory, execute the following command.

java PushAudioServer

Now the server is running and ready to accept audio push requests. To verify the functionality, run the client application as follows:

Open another command prompt and navigate to the directory where the .zip file is extracted and execute the following command.

java MediaClientImpl <type> <server_ip> <ip_telephone_ip>

Following is the description for the various parameters.

- **`<type>:`** This parameter corresponds to the type of push operation . Give its value as - `push`.
- **`<server_ip>:`** This parameter corresponds to the IP address of the audio push server. Example: `192.168.11.9`
- **`<ip_telephone_ip>:`** This parameter corresponds to the IP address of the phone to which the audio file will be streamed. Example: `192.168.12.10`

Verify that the required audio file is played successfully on the IP phone. The client will terminate on its own. Explicitly stop the server either by closing the command prompt from which the server is running or by pressing CTRL+C within the command prompt from which the server is running.

5.2 Transmit Audio Push

File Name: **TransmitAudioServer**

At the command prompt, navigate to the directory where the .zip file is extracted. Since the CLASSPATH is setup, it is not required to navigate to the directory where the actual class file is present. From this directory, execute the following command.

java TransmitAudioServer

Now the server is running and ready to accept transmit requests. To verify, run the client application provided along with as follows:

Open another command prompt and move to the directory where the .zip file is extracted and execute the following command.

java MediaClientImpl <type> <server_ip> <ip_telephone_ip>

Following is the description for the various parameters.

- **`<type>:`** This parameter corresponds to the type of push operation . Example: `transmit`.
- **`<server_ip>:`** This parameter corresponds to the IP address of the transmit push server. Example: `192.168.11.9`
- **`<ip_telephone_ip>:`** This parameter corresponds to the IP address of the phone that will receive the Transmit Audio Push request. Example: `192.168.12.10`

Verify that the audio file gets stored at the required location on the server. The client will terminate on its own. Explicitly stop the server either by closing the command prompt where the server is running or by pressing CTRL+C at the command prompt where the server is running.

6. References

[1] 4600 Series IP Telephones Application Programmer Interface (API) Guide, Issue 1, April 2005.

[2] Avaya one-X™ Deskphone Edition for 9600 Series IP Telephones Application Programmer Interface (API) Guide, Issue 2, January 2007.