Tutorial : Getting started with the Java™ Media Framework

With the growth of Internet there has been a radical change in the method of software design and deployment. Software applications are becoming more and more distributed. Java has emerged as an ideal programming language for developing Internet applications.

The increase in the processing speed of the personnel computers coupled with the increase in the bandwidth of the Internet has resulted in new and exiting multimedia applications. There has been a growth in the multimedia traffic over the Internet in recent years. Some of the promising multimedia applications that are deployable over the Internet include

(i) video conferencing,
(ii) media-on-demand,
(iii) access to multimedia database over Internet,
(iv) interactive distance education,
(v) collaborative computing,
(vi) virtual reality based scene navigation,
(vii) interactive games rich in 3D graphics, etc.

To provide a fully integrated web-based multimedia solution the JavaSoft along with its industry partners have developed a family of APIs called the Java Media APIs. The Java Media APIs is a set of APIs that enables Java programmers to add multimedia to their applications The Java Media API's include the

(1) Java™ 2D API
(2) Java™ 3D API
(3) Java™ Advanced Imaging API
(4) Java™ Image I/O
(5) Java™ Media Framework (JMF) API
(6) Java™ Image I/O
(7) Java™ Shared Data Toolkit (JSDT)
(8) Java™ Sound API
(9) Java™ Speech API

In this tutorial we will concentrate on the Java™ media framework.

What is JMF?

JMF is a framework for handling streaming media in Java programs. JMF is an optional package of Java 2 standard platform. JMF provides a unified architecture and messaging protocol for managing the acquisition, processing and delivery of time-based media. JMF enables Java programs to

(i) Present (playback) multimedia contents,
(ii) capture audio through microphone and video through Camera,
(iii) do real-time streaming of media over the Internet,
(iv) process media (such as changing media format, adding special effects),
(v) store media into a file.

Features of JMF

JMF supports many popular media formats such as JPEG, MPEG-1, MPEG-2, QuickTime, AVI, WAV, MP3, GSM, G723, H263, and MIDI. JMF supports popular media access protocols such as file, HTTP, HTTPS, FTP, RTP, and RTSP.

JMF uses a well-defined event reporting mechanism that follows the “Observer” design pattern. JMF uses the “Factory” design pattern that simplifies the creation of JMF objects. The JMF support the reception and transmission of media streams using Real-time Transport Protocol (RTP) and JMF supports management of RTP sessions.

JMF scales across different media data types, protocols and delivery mechanisms. JMF provides a plug-in architecture that allows JMF to be customized and extended. Technology providers can extend JMF to support additional media formats. High performance custom implementation of media players, or codecs possibly using hardware accelerators can be defined and integrated with the JMF.

Criticisms on JMF

Multimedia processing and presentation is compute-intensive. Therefore most of the existing media players and processors for desktop computers are implemented using native code for performing computationally intensive tasks like media encoding, decoding, and rendering.

The general criticism on Java-based applications and therefore on JMF is that they lack performance as compared to native codes. The answer to this criticism is as follows.

Why JMF?

The main drawback of native implementations of media players is that they are platform dependent. Hence they are not portable across platforms. This directly means applications using platform-dependent media players and processors are unsuitable for web-deployment. JMF provides a platform-neutral framework for handling multimedia.

The JMF API provides an abstraction that hides these implementation details from the developer. For example, a particular JMF Player implementation might choose to leverage an operating system's capabilities by using native methods. Indeed Sun's implementation of JMF has different versions each one tailored for one platform.

The JMF model

JMF adopts the same model that is used by the consumer electronics industry in handling the media. According to the JMF model, the life cycle of the media starts from a media source, and ends in a media sink. In between the media is handled by media handlers.

The media source can be a (i) a capture device, or (ii) a media file stored locally or remotely on the network or (iii) a real-time media stream available on the network. The media handlers process the media which may involve demultiplexing or multiplexing or encoding or decoding. The media processing can be implemented partly in hardware but mostly it is done by software. The media sink or destination can be rendering devices, or storage files or media streams.

For audio the capture device is a microphone along with a sound card. For images and video the capture device is a PC add-on digital camera. We typically use CRT monitor for rendering images or video and speakers for rendering audio.

The JMF Time model

Media is any form of data that changes meaningfully with respect to time. Therefore programs that handle multimedia contents should have a sense for time and its progression. To meet this requirement JMF

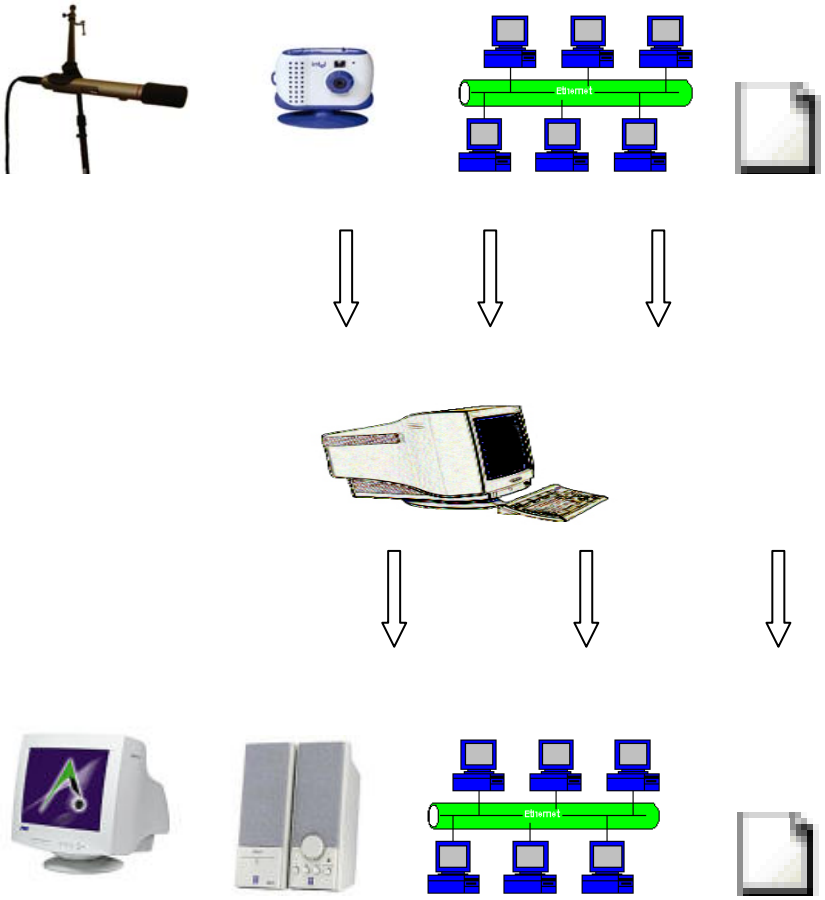

Fig : The JMF model

defines the class Time, and interfaces Timebase, Clock and Duration.

The time is represented in JMF using the class Time. A Time object represents a particular instant of time in the time axis. The interface TimeBase is an uncontrolled source for time. The interface clock represents a controlled source of time. A clock can be stopped, started and its rate can be adjusted. The interface Duration represents time duration.

Representing media

All multimedia contents are invariably stored in a compressed form using one of the various standard formats. Each format basically defines the method used to encode the media. Therefore we need a class to define the format of the multimedia contents we are handling.

To this end JMF defines the class Format that specifies the common attributes of the media Format. The class Format is further specialized into the classes AudioFormat and VideoFormat.

Specifying the source of media

The next most important support an API should offer is the ability to specify the media data source, and to control the access of data from it. We all know that Java provides us with the class java.net.URL to identify any resource in the Internet. Using an URL object we can indeed specify the media source.

JMF provides another class called MediaLocator to locate a media source. You can construct a MediaLocator using a “Locator string” which identifies the source or using an URL.

The source of the media can be of varying nature. The JMF class “DataSource” abstracts a source of media and offers a simple connect-protocol to access the media data.

Specifying the media destination

A DataSink abstracts the location of the media destination and provides a simple protocol for rendering media into destination. A DataSink can read the media from a DataSource and render the media to a file or a stream.

Media Playback

JMF supports playback of media by defining the Player interface The Player interface extends the interfaces “Controller” and MediaHandler.
Player lifecycle.

The Player (Controller) defines various states that represent different stages that are involved in the construction of a media player. When a Player is first constructed it has no knowledge of the media it has to handle. Therefore a Controller cannot identify the resources it needs to handle the media. This state of the Player is called as the Unrealized state. The Controller then enters into the Realizing state.

In the Realizing state the Player identifies and locates all the resources it needs to handle the media. Once the process of realization is over the Player enters the Realized state. Next the Player gets into the Prefetching state wherein it prefetches the media needed for the presentation. A Player that has prefetched the media is in the Prefetched state. After that the Player can be started using the start() method. A started Player can be stopped and its resources can be deallocated.

The Controller posts a number of ControllerEvents to advertise its state transitions and the error conditions. To listen to these events you can define a class implementing the ControllerListener interface. Then instantiate an object belonging to that class implementing the ControllerListener. The instantiated object should be registered with the Controller whose events we want to listen.

The interface Player defines additional methods that are required for the playback of media. Player provides a GUI for the user to control the media playback. Player may also provide a visual component associated with the media presentation. Player provides a way to manage a group of Controllers. Managing multiple Controllers is important in the case of synchronized media presentation.

Media Processing

Typical stages that are involved in the media processing are (I) Demultiplexing, (ii) transcoding, (iii) multiplexing, and (iv) rendering. In the demultiplexing stage different streams (say audio and video) of the media are demultiplexed or extracted from a composite stream. Transcoding involves a change in the encoding format of the media. During multiplexing different streams of the media are multiplexed into a single stream. Rendering involves the presentation of the media.

PlugIn

JMF defines the interface PlugIn to represent a media-processing stage. PlugIns are of following types. They are (i) MULTIPLEXER, (ii) DEMULTIPLXER, (iii) CODEC, (iv) EFFECT and (v) RENDERER.

Processor

The interface Processor abstracts a media processing class. It extends Player and allows programmatic control over the media processing stages. Processor introduces a state called Configuring wherein you can set the Codec and renderer PlugIns. The output of a Processor can be taken and can be handed over to (i) another Processor for further processing or (ii) to a Player for playback or (iii) to a DataSink for media transmission or storage.

Manager

The class Manager acts as a factory for creating the JMF objects such as DataSource, DataSink, Player, Processor, cloneableDataSource, and MergedDataSource.

Media capture

Capture devices include cameras and sound cards. All capture devices should be registered with a registry called CaptureDeviceManager in order to use them with the JMF. You can query the CaptureDeviceManager for a list of all capture devices that support a given output format.

JMF controls.

JMF API has many interfaces that allows us to control the attributes of JMF objects such as Players, Processors, DataSources etc. For example the GainControl object can be used to control the audio volume of a Player.

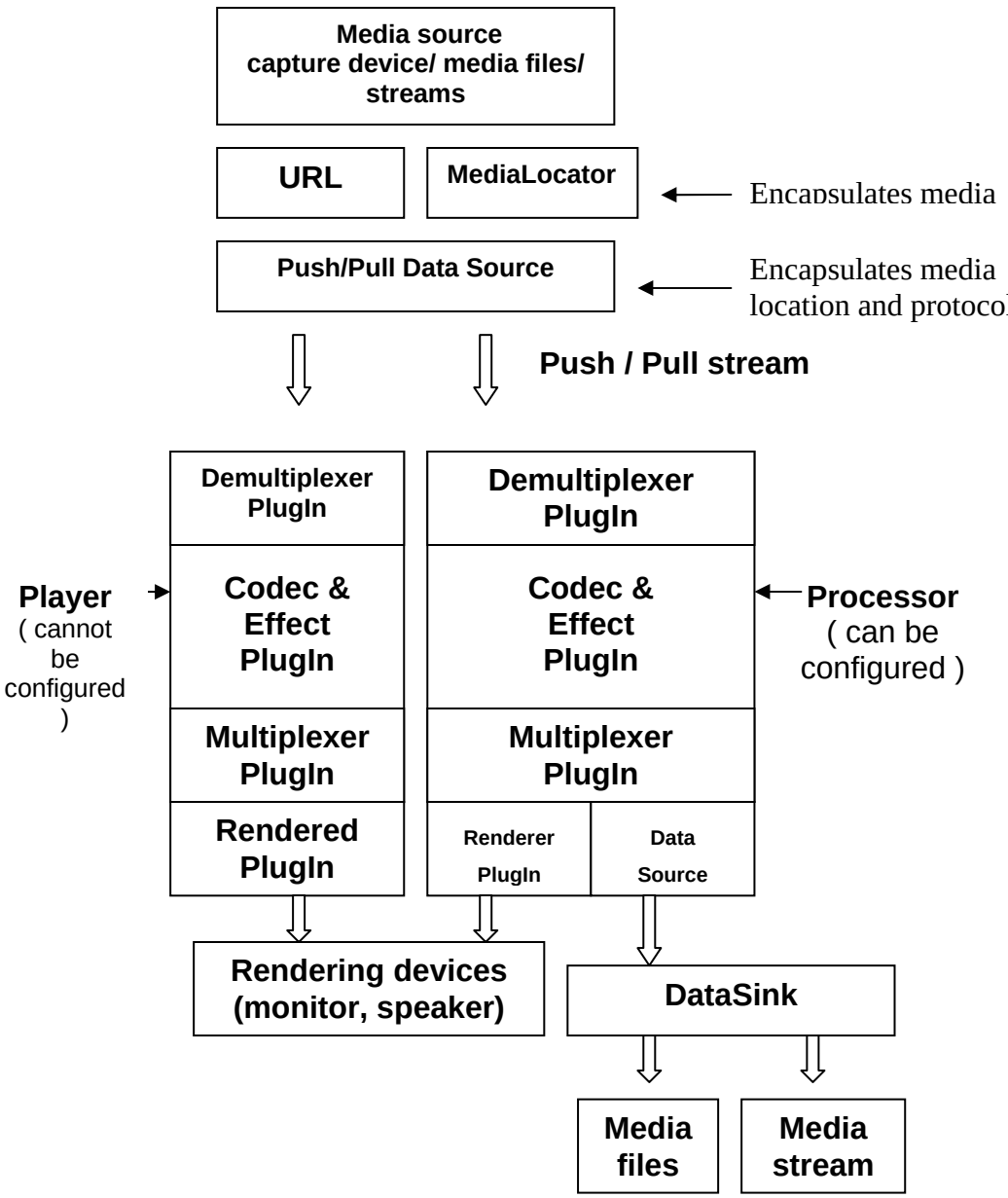

Figure: The JMF architecture

JMF RTP API

JMF supports a protocol called Real-time Transport Protocol (RTP) for media transmission and reception [rtp]. RTP is a transport layer protocol and is typically used above the UDP layer. RTP packets has (i) time stamping to indicate the time instant at which the media carried by the packet has to be played, (ii) sequence number that can be used for the ordered delivery of the packets, (iii) identification of the media source and (iv) payload media format identification.

A RTP session consists of a set of applications exchanging media using the RTP. Each of these applications is called a participant. Every participant uses an object called RTPManager to co-ordinate the RTP session on its behalf. Media streams exchanged in a RTP session are called RTPStreams. The RTPStreams can be of two types, sendStream and ReceiveStream.

The JMF RTP API allows us to construct RTPManagers using which we can send and receive RTPStreams among the participating JMF applications.

Installing JMF

The following are the different versions of JMF 2.1.1 that are currently available[jmf]

1) JMF 2.1.1 FCS Cross-Platform
This is an implementation of JMF with only Java byte codes and without any native codes.

2) JMF 2.1.1 FCS with the Windows Performance Pack which is optimized for Window's platform.

3) JMF 2.1.1 FCS with the Solaris Sparc Performance pack

4) JMF2.1.1 FCS for Linux from Blackdown

Decide the version of the JMF you wish to install. The version depends primarily on whether you want to work on Windows, or Solaris or Linux platform etc. You can download the corresponding version of JMF from the following JMF web site and install it.

http://java.sun.com/products/java-media/jmf/index.jsp

After successful installation you can see that JMF has created its directory. The JMF home directory for Windows performance pack and see what it has is typically c:\Program Files \JMF 2.1.1\. It will have three subdirectories. The subdirectories and their contents are as follows.

1) doc sub directory consisting of readme.html file.

2) The bin sub directory consists of a JMF Icon file and files to run the following applications.
   JMF customizer
   JMFinit
   JMFRegistry
   JMStudio.
3) The lib subdirectory has the following Jar files,
   jmf.jar
   sound.jar
   customizer.jar
   mediaplayer.jar and
   multiplayer.jar.

It also has the jmf.properties file and the soundbank.gm file. The Jar files contain the compiled class files of the JMF API along with the native libraries that are specific to the performance pack.

JMStudio

JMStudio is an important tool that allows you to playback, capture, store, transmit or receive media. When JMF is installed a JMStudio icon is automatically created in the desktop. In Windows environment you can invoke it though the menus: programs → Java Media FrameWork 2.1.1 → JMStudio.

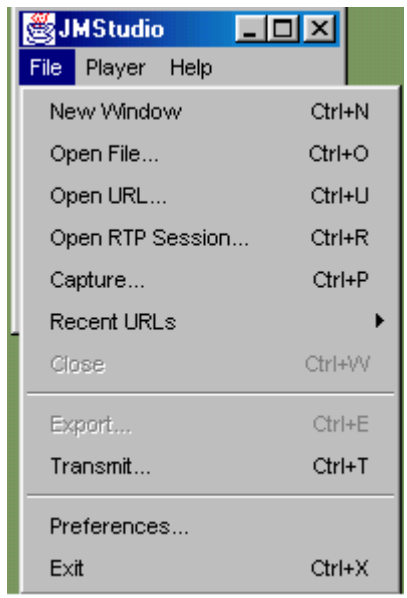

Fig The JMStudio open File options

We will introduce you JMStudio by going through some typical usage scenarios of JMStudio. Go to the File menu and its submenu “Open File”. This lets you to select a media file and play it. Note that the file chooser lists all files. Browse to select a media file and play it. While playing you will see what is called as the control panel in the bottom of the JMStudio window.

Now let us capture and save media. If you have registered atleast one audio (video) capture device with the JMF then you can capture audio (and/or video) using the option Capture. To look at the capture devices that are registered with your JMF go to the JMStudio ->file -> preferences option. It opens the JMFRegistryEditor.

The JMFRegistry is a registry for the Codecs, capture devices that are used by the JMF. Open the capture device tabpane of JMFRegistryEditor to find the capture devices that you can use with JMF. Normally you would have audio capture devices. In case you have a JMF-compatible camera in your system that would have also been listed.

First let us capture only audio. Go to the Capture option and check only the option “Use audio device”. As of now accept the default media formats details during the capture. The captured media will be played back by default.

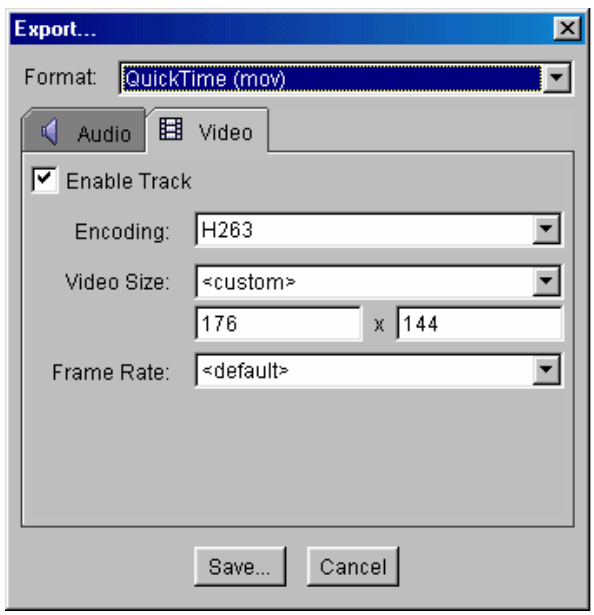

Figure : The media capture options of JMStudio

You may now wish to store the captured media in a file. While the media capture is still under progress use the option Export to store media into a file. Choose a suitable format for the audio (wav for example). See to that the Enable track is checked. Now press the save button.

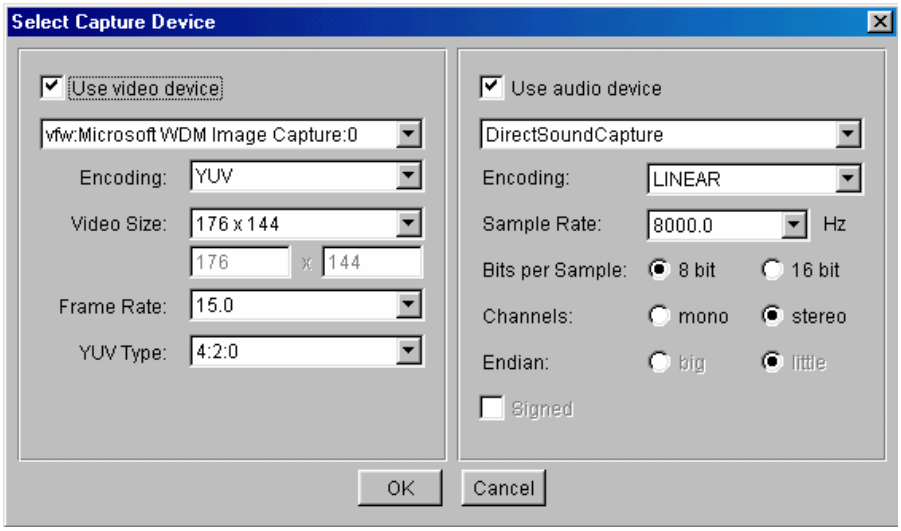

Figure : JMStudio Export option for storing the media.

Now you will be given a file dialog to choose the file in which you want to save the media. You can give the name of an existing file or a new name. You can monitor the audio and possibly adjust the volume while the audio is getting stored. The time lapsed since the beginning of file storage is displayed. You can pause and restart or stop the storage.

Media transmission using JMStudio

To transmit media using JMStudio go to the menu file of JMStudio and use the transmit option. Now you will be given two options. You can either transmit a file or transmit a captured media. If you click the button “file”, a file dialog will let you choose the file you want to transmit. Similarly the button “capture” will let you choose a capture device(s) and the format of the captured media.

After choosing the file/ capture device(s) press the “Next” button. You will be prompted to specify the content type and parameters for the output. There will be one tab pane for each track. For example in case you want to transmit a .mpg file there will be one track for audio and another track for video. You can enable/disable each of the tracks.

For the video track you can set the following parameters: (I) Encoding format (ii) video size and (iii) frame rate. For the audio track you can set the following parameters (I) encoding format (ii) sample rate (iii) bits per sample (iv) number of channels (iv) big/little endian and (v) signed /unsigned. If you are not aware of what values of the parameters are desirable we advise you to accept the default values.

Press the “Next” button after specifying the content type and parameters for the output. Now you will be prompted to enter the session address and the port number for each track. The session address should be the IP address (in dotted decimal notation) of the machine which is supposed to receive the media.

The port number should be an unused port number supported in the destination machine. You may try an integer in the range 7000 to 12,000. You may give the value of the TTL field as "1". Finally click the “Finish” button. The media will get transmitted. Figure below shows the GUI of the JMStudio for the media transmission.

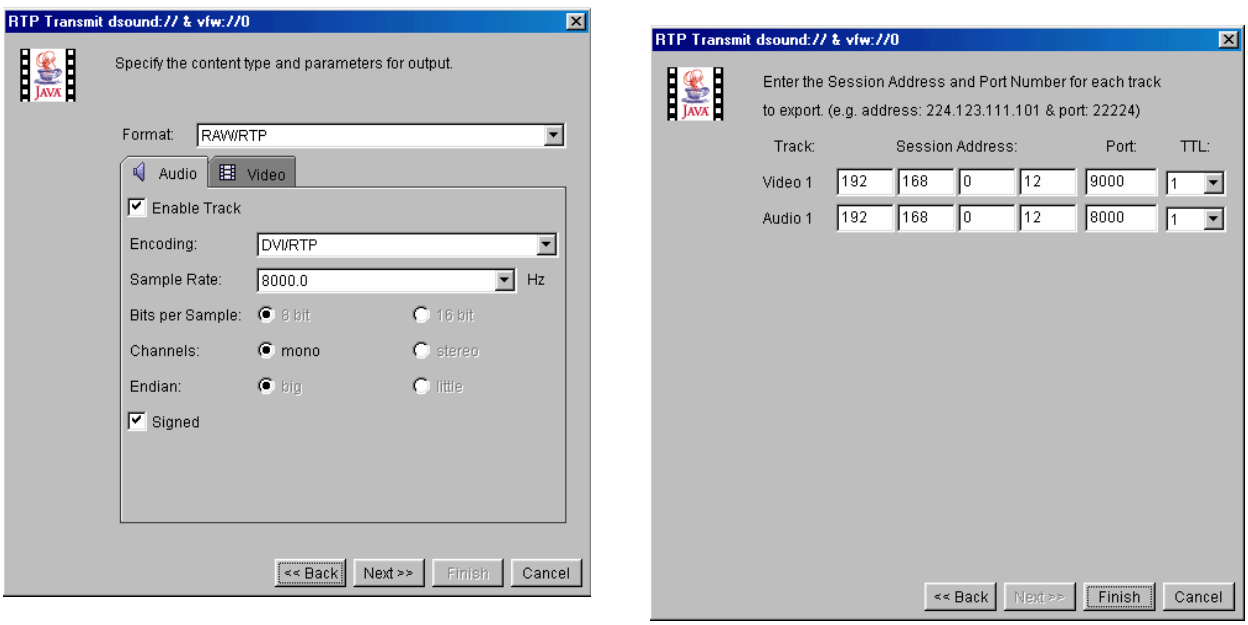

Figure : Transmission of audio and video using JMStudio

If you transmit a file you would see a window entitled “Transcoding Processor”. If you transmit captured video and /or audio you might see two more windows entitled “Capture Monitor”, “Capture Device Controls”. The Capture Monitor allows you to monitor the video being captured. The “Capture Device Controls” is a control panel that lets you control the capture device.

To test whether the media is really getting transmitted receive the media at the destination computer as follows.

Media reception using JMStudio

To receive media using JMStudio use the option “open RTP session” under the file menu. You will be prompted to specify the session address, port number and TTL. Give the IP address of the machine whose transmission you wish to receive as the session address. The port number of the “open RTP session” should be the port number to which the transmitter is transmitting.

The value of the TTL can be set to 1. Click the “open”button. If the media were already getting transmitted the media will be received. A Player will be created to present the media. The visualComponent (if there is one) and controlPanelComponent of the Player will be displayed.

Supposing the media is is not yet available when we open the RTP session, a GUI will indicate that JMStudio is waiting for data. After waiting for 60 seconds GUI will give an option to the user to either continue waiting or close the RTP session.

Now you can play with the JMStudio to your heart’s content. JMStudio is nothing but a (fairly big) JMF program. You can write JMF program and handle multimedia programmatically in any manner you want. You can do things like design your own media player with control panel and introduce components like graphic equalizer, merge two media streams, extract audio from a .mpg file, give background music to a slide presentation etc.

HelloJMF.java

Let us try our first JMF program. Our first program is the JMF version of the indispensable “Hello, World” program. We call this as the HelloJMF.java. The program is given below.

```
/* Program HelloJMF.java
    Plays an audio file that is present in the same directory wherein the
    HelloJMF.class file resides.
    */
    import javax.swing.*;
    import java.awt.*;
    import javax.media.*;
    import java.awt.event.*;
    import java.net.*;

    public class HelloJMF {

    JFrame frame = new JFrame(" Hello JMF Player");
    static Player helloJMFPlayer = null;

    public HelloJMF(){

    try{                                                    // method using URL
                URL url=new URL("file",null,"hello.wav");
                helloJMFPlayer = Manager.createRealizedPlayer( url);
     } catch( Exception e) {
                System.out.println(" Unable to create the audioPlayer :" + e );
    }

    Component control = helloJMFPlayer.getControlPanelComponent();

    frame.getContentPane().add( control, BorderLayout.CENTER);
    frame.addWindowListener( new WindowAdapter() {
                                        public void windowClosing(WindowEvent we) {
                                                HelloJMF.stop();
```

```
                                        System.exit(0);
                                    }
    });
    frame.pack();
    frame.setSize( new Dimension(200,50) );
    frame.setVisible(true);
    helloJMFPlayer.start();

    }

    public static void stop(){
                helloJMFPlayer.stop();
                helloJMFPlayer.close();
    }

    public static void main( String args[]){
                HelloJMF helloJMF = new HelloJMF();
    }
    }
```

Here is a brief explanation of the HelloJMF.java. The constructor of HelloJMF.java constructs an URL (Uniform Resource Locator) object representing the file called Hello.wav. The file Hello.wav is expected to be present in the same directory as that of HelloJMF.class. We use the following constructor of the java.net.URL class to construct the URL object.

```
public URL(String protocol,          // the protocol to use
           String host,              //  the host name to connect to
           String file)              //  the specified file name on that host
   throws MalformedURLException
```

The protocol used to access the hello.wav resource is the “file” protocol. The host name can be null since for the case of file protocol the host is the local host. With this URL we then construct a media Player object using the utility class Manager.

The control panel of the media player is attached to the application frame. The application frame is made visible and the player is started. When the user closes the application frame the anonymous WindowAdaptor Object uses the WindowClosing event to stop the media player and exit.

Type this program in a file called HelloJMF.java or you may cut and copy this program listing into a file. We hope you are familiar with compiling and executing Java programs. If you use any IDE for program development you should consult its manual for how to edit, compile, execute and debug programs. In case you use the JDK for program development use the following command

javac HelloJMF.java

to compile HelloJMF.java .

In order to test this program create a wav file named hello.wav and store it in same directory where the helloJMF.class is present.

To execute the program use the command

java HelloJMF

from the directory which holds both the HelloJMF.class and the Hello.wav file. On execution of HelloJMF.class a window entitled “Hello JMF Player” is created and you should hear the hello.wav. If you encounter any problem it may be because JMF is not installed properly. Check whether you can invoke and use JMStudio to play Hello.wav.

Conclusions

Let us recall the main points of this tutorial.

JMF enables Java programs to playback, capture, transmit, receive and store real-time streaming media, JMF supports many popular media formats and access protocols. The JMF API provides an abstraction that hides these implementation details from the developer. JMF offers portability and at the same time does not loose performance.

According to the JMF model, the life cycle of the media starts from a media source, and ends in a media sink. In between the media is handled by media handlers. To this end JMF provides a number of classes and interfaces such as the TimeBase, clock, MediaLocator, DataSource, Controller, Player, Processor, DataSink, Manager RTPManager, etc,

JMF uses a Plugin architecture that allows JMF to be extended to support adition media formats and access protocols. Tools such as the JMStudio, JMF

RegistryEditor, and JMF Customizer helps us in developing JMF-based multimedia applications.

The mobile media API (MMAPI) is a framework that is similar to JMF. MMAPI offers multimedia capabilities to the Java MIDlets that run on the J2ME enabled devices.

References

1) http://java.sun.com/products/java-media/jmf2.1.1/apidocs/

2) http://java.sun.com/products/java-media/jmf2.1.1/guide/

3) Developing multimedia applications with the Java media Framework (includes JMF 2.1.1 and MMAPI), T.G.Venkatesh, cosmos software, First Edition 2004.