JMF player

Sumit Yadav; Usha Verma & Chhavi Bhardwaj

(sumityadav1918@gmail.com),
(usha.verma1991@gmail.com),
(chhavi985@gmail.com)

Abstract:

This research paper help to create your own media player. Only need for creating a media player through JMF player is the basic knowledge of java programming. This research paper also provides a brief description of the various classes of JMF player. This research paper also provides information to install the JMF player.

1. INTRODUCTION

JMF Player:

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.

2. INSTALLATION

JAVA package:

JMF player need java package to run.
Install jdk1.7.0 for windows.

1) Download jdk1.7.0 from oracle website. It is an open source package.
2) Start installation by double clicking on it.
3) A setup window will open on your computer screen. Click on next option.
4) Select development tools and click on next. Don't change the path of package, it will create problem at programming time.
5) The installation will start. When finish click on the finish button.
6) You can see a folder named java in c: drive.

<u>Netbeans IDE 7.0.1:</u>

If you are not a skilled developer I would prefer that you use this tool for developing program. It provides error detection and error correction in the program which make programming easy. As for notepad and command prompt one can't find out the reason for error. This is freely available software.

Steps for installation:-

1. Download NetBeans IDE7.0.1 from the internet.
2. Start installation by double clicking on NetBeans IDE7.0.1.
3. Netbeans IDE installer windows open. Click on next button.
4. Accept the terms and conditions and click on next button.
5. A path is shown where your software will be installed. Don't change the path and click on next button.
6. Click on install button and installation will start.
7. After completion of installation you can see a folder named NetBeans 7.0.1 in c:\program files(x86) mostly.

JMF Player:

JMF Player provides features connected to media player. It provide accessibility to media files in java programming. This package makes the programming easier than core java programming.

Steps for installation:

1. Download JMF2.1.1e for windows from oracle website under java API's category.
2. Start the start the setup by double clicking on th jmf-2_1_1e-windows package.
3. It asks for license agreement. Click on yes.
4. At next step path is shown. Don't change the path and click on next.
5. Again click on next.
6. Click on finish.
7. You can see JMF2.1.1e named folder in c:\program files(x86) mostly.

3. JMF Player directory

In JMF player the first one is bin directory. It contain following application.

- **JMStudio** - A simple player GUI.
- **JMFRegistry** - A GUI for managing the JMF "registry," which manages preferences, plug-ins, etc.
- **JMFCustomizer** - Used for creating a JAR file that contains only the classes needed by a specific JMF application, which allows developers to ship a smaller application.
- **JMFInit**

4. 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.

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.

5. Component Architecture

JMF is built around a component architecture. The compenents are organized into a number of main categories:

- Media handlers
- Data sources
- Codecs/Effects
- Renderers
- Mux/Demuxes

Media Handlers

MediaHandlers are registered for each type of file that JMF must be able to handle. To support new fileformats, a new MediaHandler can be created.

Data Sources

A DataSource handler manages source streams from various inputs. These can be for network protocols, such as http or ftp, or for simple input from disk.

Codecs/Effects

Codecs and Effects are components that take an input stream, apply a transformation to it and output it. Codecs may have different input and output formats, while Effects are simple transformations of a single input format to an output stream of the same format.

Renderers

A renderer is similar to a Codec, but the final output is somewhere other than another stream. A VideoRenderer outputs the final data to the screen, but another kind of renderer could output to different hardware, such as a TV out card.

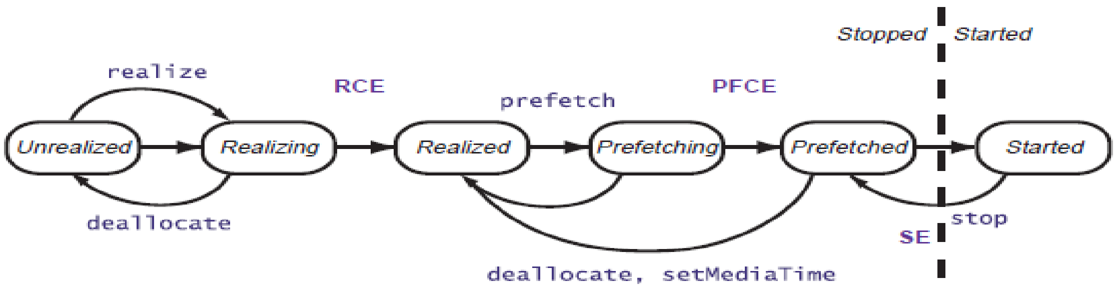
Mux/Demuxes

Multiplexers and Demultiplexers are used to combine multiple streams into a single stream or vice-versa, respectively. They are useful for creating and reading a package of audio and video for saving to disk as a single file, or transmitting over a network.

6. Player states

A Player can be in one of six states. The Clock interface defines the two primary states: *Stopped* and *Started*. To facilitate resource management, Controller breaks the *Stopped* state down into five standby states:

· Prefetched
· Prefetching
· Realized
· Realizing
· Unrealized



Transition Events:

RCE	RealizeCompleteEvent
PFCE	PrefetchCompleteEvent
SE	StopEvent

In normal operation, a Player steps through each state until it reaches the *Started* state:

- A Player in the ***Unrealized* state** has been instantiated, but does not yet know anything about its media. When a media Player is first created, it is *Unrealized*.
- When realize is called, a Player moves from the *Unrealized* state into the ***Realizing* state.** A *Realizing* Player is in the process of determining its resource requirements. During realization, a Player acquires the resources that it only needs to acquire once. These might include rendering resources other than exclusive-use resources. (Exclusive use resources are limited resources such as particular hardware devices that can only be used by one Player at a time; such resources are acquired during Prefetching.) A *Realizing* Player often downloads assets over the network.
- When a Player finishes *Realizing*, it moves into the ***Realized* state**. A *Realized* Player knows what resources it needs and information about the type of media it is to present. Because a *Realized* Player knows how to render its data, it can provide visual components and controls. Its connections to other objects in the system are in place, but it does not own any resources that would prevent another Player from starting.
- When prefetch is called, a Player moves from the *Realized* state into the ***Prefetching state***. A *Prefetching* Player is preparing to present its media. During this phase, the Player preloads its media data, obtains

exclusive-use resources, and does whatever else it needs to do to prepare itself to play. *Prefetching* might have to recur if a Player objects media presentation is repositioned, or if a change in the Player objects rate requires that additional buffers be acquired or alternate processing take place.

- When a Player finishes *Prefetching*, it moves into the ***Prefetched state***. A *Prefetched* Player is ready to be started.
- Calling start puts a Player into the ***Started state.*** A *Started* Player objects time-base time and media time are mapped and its clock is running, though the Player might be waiting for a particular time to begin presenting its media data.

7. Presenting Data

The Java Media Framework provides a number of pre-built classes that handle the reading, processing and display of data. Using the Player, media can easily be incorporated into any graphical application (AWT or Swing). The Processor allows you to control the encoding or decoding process at a finer level than the Player, such as adding a custom codec or effect between the input and output stages.

Using the Player

The Player class is an easy way to embed multimedia in an application. It handles the setup of the file handler, video and audio decoders, and media renderers automatically. It is possibly to embed the Player in a Swing application, but care must be taken as it is a heavy-weight component (it won't clip if another component is placed in front of it).

```
import java.applet.*;
import java.awt.*;
import java.net.*;
import javax.media.*;
public class PlayerApplet extends Applet {
Player player = null;
public void init() {
setLayout( new BorderLayout() );
String mediaFile = getParameter( "FILE" );
try {
URL mediaURL = new URL(
getDocumentBase(), mediaFile );
player = Manager.createRealizedPlayer(
mediaURL );
if (player.getVisualComponent() != null)
add("Center",
player.getVisualComponent());
if (player.getControlPanelComponent() !=
null)
add("South",
player.getControlPanelComponent());
} catch (
Exception e) {
System.err.println( "Got exception " + e );
}
} public void start() {
player.start();
}

public void stop() {
player.stop();
player.deallocate();
} public void destroy() {
player.close();
}
}
```

Audio player using swing component

The Player can be easily used in a Swing application as well. The following code creates a Swing-based TV capture program with the video output displayed in the entire window:

```
import javax.media.*;
import javax.swing.*;
import java.awt.*;
import java.net.*;
import java.awt.event.*;
import javax.swing.event.*;
public class JMFTest extends JFrame {
Player _player;
JMFTest() {
addWindowListener( new WindowAdapter()
{
public void windowClosing( WindowEvent e
) {
_player.stop();
_player.deallocate();
_player.close();
System.exit( 0 );
}
});
setExtent( 0, 0, 320, 260 );
JPanel panel = (JPanel)getContentPane();
panel.setLayout( new BorderLayout() );
String mediaFile = "vfw://1";
try {
MediaLocator mlr = new MediaLocator(
mediaFile );
_player = Manager.createRealizedPlayer(
mlr );
if (_player.getVisualComponent() != null)
panel.add("Center",
_player.getVisualComponent());
if (_player.getControlPanelComponent() !=
null)
panel.add("South",
_player.getControlPanelComponent());
}
catch (Exception e) {
System.err.println( "Got exception " + e );
}
}
public static void main(String[] args) {
JMFTest jmfTest = new JMFTest();
jmfTest.show();

}}
```

8. References

[1] Gordon, R., & Talley, S. (1999). *Essential JMF: Developer's Java Media Players*. Prentice Hall PTR.

[2] DeCarmo, L. (1999). *Core Java media framework*. Prentice Hall PTR.

[3] Abdel-Wahab, H., Kim, O., Kabore, P., & Favreau, J. P. (1999, April). Java-based multimedia collaboration and application sharing environment. In *Colloque Francophone sur L'Ingenierie des Protocoles (CFIP'99), Nancy, France* (pp. 451-463).

[4] Sullivan, S., Winzeler, L., Brown, D., & Deagen, J. (1998). *Programming with the Java media framework*. John Wiley & Sons, Inc..

[5] Wong, D. C., Rivas, J. D., & Yamasani, A. (2001). *U.S. Patent No. 6,216,152*. Washington, DC: U.S. Patent and Trademark Office.