Mobile Media API

Version 0.9, Draft

API GUIDE

COPYRIGHT

Trademarks and Service Marks

About This Document

This document will cover in brief about MMAPI (Mobile Media API) for Samsung specific devices. It contains overview and Architecture of MMAPI, followed by a Sample code on how to play audio and video.

Scope:

This document is intended for MIDP developers who wish to develop Java ME applications. It assumes good knowledge of java programming language.

Document History:

Date	Version	Comment
05/02/09	0.9	Draft

References:

1. MMAPI Specification:

   http://jcp.org/en/jsr/detail?id=135

2. MMAPI Overview:

   http://developers.sun.com/mobility/midp/articles/mmapioverview/

Abbreviations:

Java ME	Java Micro Edition
CLDC	Connection Limited Device Configuration
MIDP	Mobile Information Device Profile
JSR	Java Specification Request
RMS	Record Management System
JAR	Java Application Archive

Table of Contents

Table of Figures

Introduction

The Mobile Media API (MMAPI) supports multimedia applications for Java ME enabled devices. This is an optional package. MMAPI enables MIDlet to play video, take pictures, play and record audio.

MMAPI Overview

There are two parts of multimedia processing in MMAPI. They are:

- Protocol Handling
- Content Handling

Protocol Handling

This refers to reading data from a source such as file or streaming server into a media-processing system.

Content Handling

This refers to rendering media data on a device by parsing or decoding/encoding. There are two high level objects in MMAPI, which encapsulate both the handling data using the protocol and the content. See Figure 1.

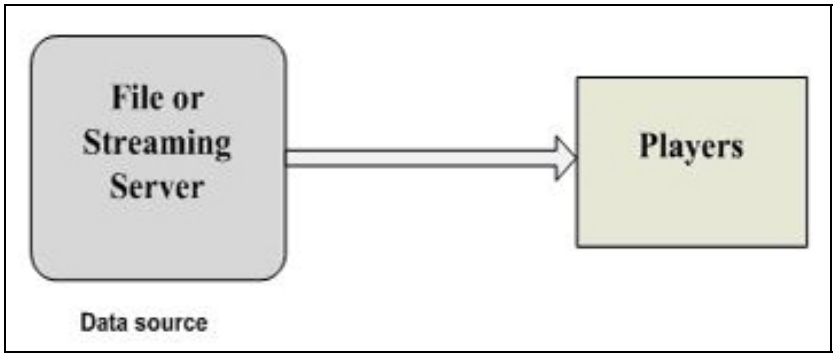

Figure 1: Data Handling

Data Source for protocol handling

javax.microedition.media.protocol.DataSource for protocol handling allows the user to access the data for processing. Data source hides the details of how the data is being read from the source whether it may be from file, streaming server or proprietary delivery mechanism.

Player for content handling

javax.microedition.media.Player provides methods to manage the player's life cycle. It reads the data from the *DataSource*, processes it and renders it to the output device. It controls the playback progress, obtains the presentation component, controls and provides the means to synchronize with other players. *Player* also provides some type-specific controls to access features for specific media types.
Player can be created from *javax.microedition.media.Manager*, factory mechanism class, which provides access to set methods for creating *Player*.

The overall architecture of MMAPI is shown in Figure 2.

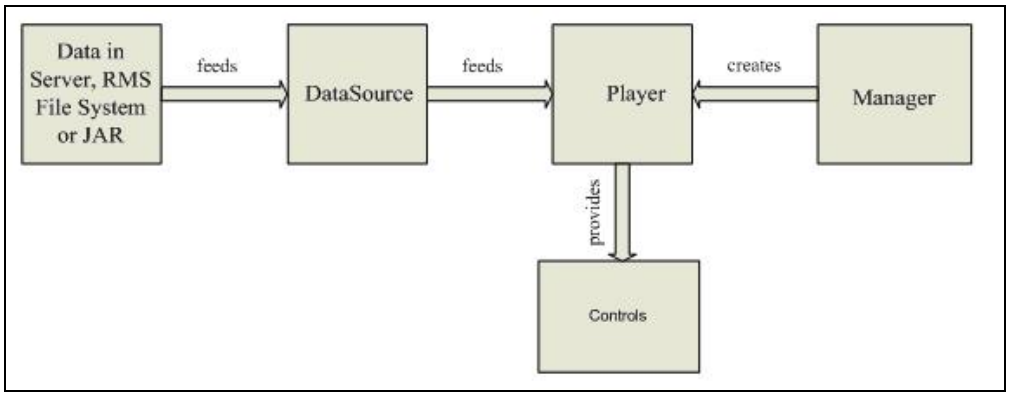

Figure 2: MMAPI architecture

Data source gets the media data from source such as File or Server to a *Player*.

Usually Using factory mechanism class *Manager* creates *Player* from *DataSource*, and also from *InputStream*. *Player* interprets media data and can use controls (such as *VolumeContol*, *VideoControl*, *ToneContol*) to modify the behavior of a player.

Player's life cycle

Player's life cycle consists of five states:

UNREALIZED
REALIZED,
PREFETCHED
STARTED
CLOSED

Player class contains six methods for controlling transitions:

realize()
prefetch()
start()
stop()
deallocate()
close()

Figure 3 shows the transitions and life cycle states.

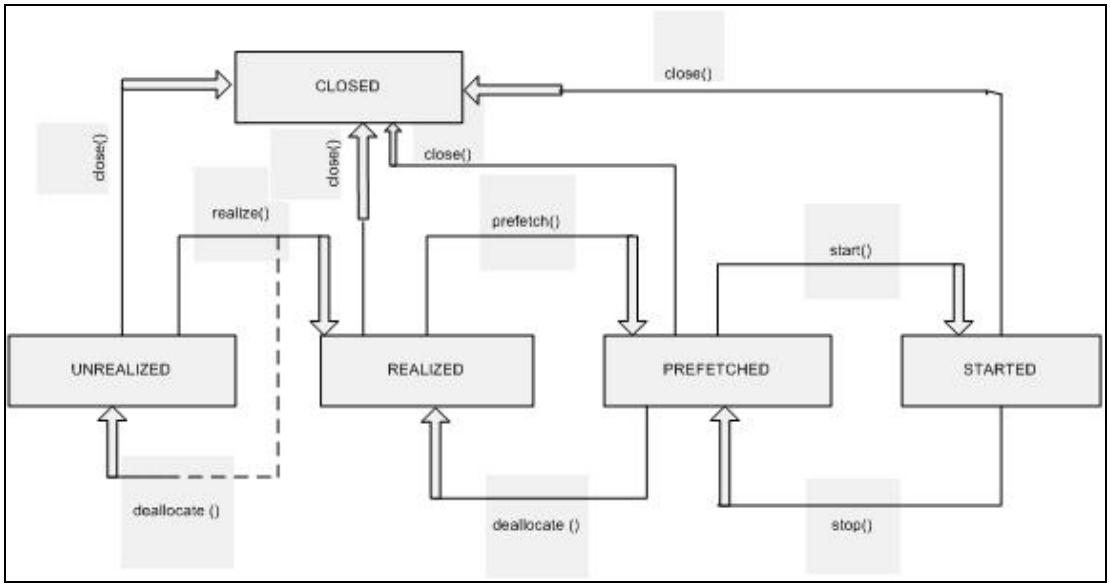

Figure 3: Player Life Cycle

UNREALIZED State

When a player is created, it is in the UNREALIZED state. An unrealized Player does not have enough information to acquire all the resources it needs to function.

REALIZED State

Calling realize() moves it from UNREALIZED to the REALIZED state and initializes the information that player needs to acquire media resources. The only way to reach back UNREALIZED state is if deallocate() is invoked before REALIZED state is completed.

A Player is in the REALIZED state when it has obtained the information required to acquire the media resources.

Once a Player reaches the REALIZED state, it remains in one of the four states: REALIZED, PREFETCHED, STARTED or CLOSED. Player never returns to the UNREALIZED state.

PREFETCHED State

Even after reaching REALIZED state, player still needs to perform a number of potentially time-consuming tasks before it is ready to be started.

For example, it may need to acquire exclusive resources, fill buffers with media data, perform other start-up processing, establishes network connections for streaming data, or performs other initialization tasks.

Calling prefetch() moves it to PREFETCHED state from REALIZED state. Once a Player is in the PREFETCHED state, it may be started by calling start() method. Player returns to the PREFETCHED state when player stops playing or it has reached end of media. stop() causes the player to stop playing and return to PREFETCHED state.

STARTED State

STARTED state indicates, Player is running and processing data. Calling start() causes player transition to the STARTED state from PREFETCHED state, where the player can start processing data.

When the Player moves from the PREFETCHED to the STARTED state, it posts a STARTED event.

When it moves from the STARTED state to the PREFETCHED state, it posts a STOPPED, END_OF_MEDIA or STOPPED_AT_TIME event depending on the reason it stopped.

CLOSED state

In the CLOSED state, Player has released most of its resources and same Player object must not be used again.
Calling close() method from UNREALIZED, REALIZED, PREFETCHED, and STARTED states moves the Player to CLOSED state.

Mobile Media API Description

There are three packages in MMAPI

1. javax.microedition.media:

This package contains some Interfaces, *MediaException* and *Manager* class.

2. javax.microedition.media.control:

This package contains all Interfaces specific to control types such as *VolumeContol*, *VideoControl* and *ToneControl*.

3. javax.microedition.media.protocol:

This package contains one interface and two classes for protocol handling like *DataSource* class, which is abstraction for media-control handler.

Playing audio and video using MMAPI

Playing audio

Content can be read from Server, JAR, File System and/or RMS to play it on the device. The following example shows reading of an Audio file from JAR. Sample code snippets for playing audio and video are given at the end of this document.

Read the Audio file from the resource by creating InputStream as follows:

```
/*Read audio file from resource */
InputStream inputStream = getClass().getResourceAsStream("/music.mid");
```

After reading data successfully from JAR, create a player with input as InputStream and content type of the Audio file.

```
Player player = Manager.createPlayer(inputStream, "audio/midi");
```

setLoopCount(int count) method sets the number of times the Player will loop and play the content. By passing parameter as -1 to setLoopCount() leads to infinite playback.

```
player.setLoopCount(-1);
```

Then initialize the player by calling following method as follows.

```
player.realize();
```

Move the player to the PREFETCHED state.

```
player.prefetch();
```

Set the player for listening events.

```
player.addPlayerListener(this);
```

Start playing audio by using following method.

```
player.start();
```

Playing Video

Follow the same procedure of audio till setting a listener for a player and get the video control object through which video can be displayed in the Form as an Item.

```
VideoControl videoControl = (VideoControl) Player.getControl("VideoControl");
if (videoControl != null) {
    Item videoItem = (Item)
      videoControl.initDisplayMode(VideoControl.USE_GUI_PRIMITIVE, null);
    append(videoItem);
}
```

Start playing Video by using following method.

```
player.start();
```

Sample Code snippet for audio

The example of MIDlet classes below shows how to play audio and video using MMAPI.

This AudioMIDlet is a sample class to play audio as follows:

Class: AudioMidlet.java

```
import java.io.InputStream;

import javax.microedition.lcdui.Command;
import javax.microedition.lcdui.CommandListener;
import javax.microedition.lcdui.Display;
import javax.microedition.lcdui.Displayable;
import javax.microedition.lcdui.Form;
import javax.microedition.lcdui.StringItem;
import javax.microedition.media.Manager;
import javax.microedition.media.MediaException;
import javax.microedition.media.Player;
import javax.microedition.media.PlayerListener;
import javax.microedition.midlet.MIDlet;

public class AudioMidlet extends MIDlet implements CommandListener,
      PlayerListener, Runnable {

   private static final String EXIT = "Exit";
   private static final String PAUSE = "Pause";
   private static final String PLAY = "Play";
   private Display display;
   private Form playerForm;
   private Thread thread;
   private Player player;
   private StringItem curStatus;

   /*this command is used to exit from the current application*/
   private final Command cmdExit = new Command(EXIT, Command.EXIT, 2);
   /*this command is used to bring the player in pause state*/
   private final Command cmdPause = new Command(PAUSE, Command.ITEM, 1);
   /*this command is used to play the player*/
   private final Command cmdPlay = new Command(PLAY, Command.ITEM, 1);

   public AudioMidlet() {
      display = Display.getDisplay(this);
      playerForm = new Form("MMAPI Audio");
      initAudio();
      curStatus = new StringItem("Player Status:", "Reading Content");
      playerForm.append(curStatus);
      playerForm.addCommand(cmdExit);
      playerForm.addCommand(cmdPause);
      playerForm.setCommandListener(this);

   }
```

```
public void startApp() {
   display.setCurrent(playerForm);
}

public void pauseApp() {
   /*all interupts can be handled here*/
}

public void destroyApp(boolean unconditional) {
   playerClose();
}

public void exitMidlet() {
   destroyApp(true);
   notifyDestroyed();
}

public void initAudio() {
   /*recomeneded to start the players in a separate thread*/
   thread = new Thread(this);
   thread.start();
}

public void run() {
   try {
      /*gets the content from the JAR as a resource stream data*/
      InputStream inputStream =
            getClass().getResourceAsStream("/piano.mid");
      /*create a player object*/
      player = Manager.createPlayer(inputStream, "audio/midi");
      /*this is used to set the loop count, here -1 is indefinite*/
      player.setLoopCount(-1);
      player.realize();
      player.prefetch();
      player.addPlayerListener(this);
   } catch (Exception ex) {
      if (player != null) {
         player.close();
         player = null;
      }
      System.err.println("Problem creating player" + ex);
   }

   startPlayer();

}
```

```
/*this method will responds to the player events*/
public void playerUpdate(Player plyr, String evt, Object evtData) {
   if (evt == STARTED) {
      curStatus.setText("Started Playing Audio");
   } else if (evt == STOPPED) {
      curStatus.setText("Stopped Playing Audio");
   } else if (evt == CLOSED) {
      curStatus.setText("Closed Playing Audio");
   }
}

public void commandAction(Command cmd, Displayable disp) {
   if (cmd == cmdPause && disp == playerForm) {
      playerPause();
      playerForm.removeCommand(cmdPause);
      playerForm.addCommand(cmdPlay);
   } else if (cmd == cmdExit && disp == playerForm) {
      exitMidlet();
   } else if (cmd == cmdPlay && disp == playerForm) {
      playerForm.removeCommand(cmdPlay);
      playerForm.addCommand(cmdPause);
      startPlayer();
   }
}

public void startPlayer() {
   if (player != null) {
      try {
         player.start();
      } catch (MediaException me) {
         System.err.println(me);
      } catch (Exception ex) {
         System.err.println(ex);
      }
   }
}

public void playerPause() {
   if (player != null) {
      try {
         player.stop();
      } catch (MediaException me) {
         System.err.println(me);
      }
   }
}
```

```
    public void playerClose() {
        synchronized (this) {
            playerPause();
            if (player != null) {
                player.close();
                player = null;
            }
        }
    }
}
```

Sample Code snippet for video

The sample code given below shows how to play video using MMAPI.

Class: VideoMidlet.java

```
import java.io.InputStream;

import javax.microedition.lcdui.Command;
import javax.microedition.lcdui.CommandListener;
import javax.microedition.lcdui.Display;
import javax.microedition.lcdui.Displayable;
import javax.microedition.lcdui.Form;
import javax.microedition.lcdui.Item;
import javax.microedition.lcdui.StringItem;
import javax.microedition.media.Manager;
import javax.microedition.media.MediaException;
import javax.microedition.media.Player;
import javax.microedition.media.PlayerListener;
import javax.microedition.media.control.VideoControl;
import javax.microedition.midlet.MIDlet;

public class VideoMidlet extends MIDlet implements CommandListener,
        PlayerListener, Runnable {

    private static final String EXIT = "Exit";
    private static final String PAUSE = "Pause";
    private static final String PLAY = "Play";
    private Display display;
    private Form playerForm;
    private Thread thread;
    private Player player;
    private StringItem curStatus;
```

```
/*this command is used to exit from the current application*/
private final Command cmdExit = new Command(EXIT, Command.EXIT, 2);
/*this command is used to bring the player in pause state*/
private final Command cmdPause = new Command(PAUSE, Command.ITEM, 1);
/*this command is used to play the player*/
private final Command cmdPlay = new Command(PLAY, Command.ITEM, 1);

public VideoMidlet() {
    display = Display.getDisplay(this);
    playerForm = new Form("MMAPI Video");
    initAudio();
    curStatus = new StringItem("Player Status:", "Reading Content");
    playerForm.append(curStatus);
    playerForm.addCommand(cmdExit);
    playerForm.addCommand(cmdPause);
    playerForm.setCommandListener(this);
}

public void startApp() {
    display.setCurrent(playerForm);
}

public void pauseApp() {
    /*all interupts can be handled here*/
}

public void destroyApp(boolean unconditional) {
}

public void exitMidlet() {
    destroyApp(true);
    notifyDestroyed();
}

public void initAudio() {
    /*recomeneded to start the players in a separate thread*/
    thread = new Thread(this);
    thread.start();
}

public void run() {
    try {
        /*gets the content from the JAR as a resource stream data*/
        InputStream inputStream = getClass().getResourceAsStream("/video.mpg");
        /*creates a player object from Manager by using factory mechanism*/
        player = Manager.createPlayer(inputStream, "video/mpeg");
        player.realize();
```

```
        player.prefetch();
        player.addPlayerListener(this);
        VideoControl videoControl =
              (VideoControl) player.getControl("VideoControl");
        if (videoControl != null) {
          Item videoItem = (Item)
             videoControl.initDisplayMode(VideoControl.USE_GUI_PRIMITIVE, null);
          playerForm.append(videoItem);
        }
    } catch (Exception ex) {
        if (player != null) {
          player.close();
          player = null;
        }
        System.err.println("Problem creating player" + ex);
    }
    startPlayer();
}

public void commandAction(Command cmd, Displayable disp) {
    if (cmd == cmdPause && disp == playerForm) {
        playerPause();
        playerForm.removeCommand(cmdPause);
        playerForm.addCommand(cmdPlay);
    } else if (cmd == cmdExit && disp == playerForm) {
        exitMidlet();
    } else if (cmd == cmdPlay && disp == playerForm) {
        playerForm.removeCommand(cmdPlay);
        playerForm.addCommand(cmdPause);
        startPlayer();
    }
}

/*this method will responds to the player events*/
public void playerUpdate(Player plyr, String evt, Object evtData) {
    if (evt == STARTED) {
        curStatus.setText("Started Playing Video");
    } else if (evt == STOPPED) {
        curStatus.setText("Stopped Playing Video");
    } else if (evt == CLOSED) {
        curStatus.setText("Closed Playing Video");
    }
    if (evt == END_OF_MEDIA) {
        try {
          player.setMediaTime(0);
          player.start();
        } catch (MediaException me) {
```

```
            System.err.println(me);
        }
    }
  }

  public void startPlayer() {
    if (player != null) {
      try {
        player.start();
      } catch (Exception ex) {
        System.out.println("Error in starting a player=" + ex);
      }
    }
  }

  public void playerPause() {
    if (player != null) {
      try {
        player.stop();
      } catch (MediaException me) {
        System.err.println(me);
      }
    }
  }

  public void playerClose() {
    synchronized (this) {
      playerPause();
      if (player != null) {
        player.close();
        player = null;
      }
    }
  }
}
```