How to take pictures, record videos and play music using MMAPI

The Mobile Media API (MMAPI) is a flexible and powerful API that allows the rendering and capture of audio and video data. As a simple and lightweight optional package, it allows Java developers to gain access to native multimedia services available on a given device. This article describes how to use MMAPI to take snapshots, record videos and play music. For more information you can access here.

Warning: The examples below may not work on emulators.

Useful classes

In this example we need to use Manager and Player classes from audio package javax.microedition.media and concepts of thread.

The Manager class is the access point for obtaining system dependent resources such as Players for multimedia processing. A Player is an object used to control and render media that is specific to the content type of the data. Content types identify the type of media data. For example, here are a few common content types examples extracted from class Manager

```
     * Wave audio files: audio/x-wav
     * AU audio files: audio/basic
     * MP3 audio files: audio/mpeg
     * MIDI files: audio/midi
     * Tone sequences: audio/x-tone-seq
```

If you want to know more about content types click here.

Once a Player is created an application can request Control objects that encapsulate behaviors to control the media playback, capture, and so forth. The Control objects available vary depending on the media supported by the player. Figure 1 shows how the Manager, Player, and Control classes relate.

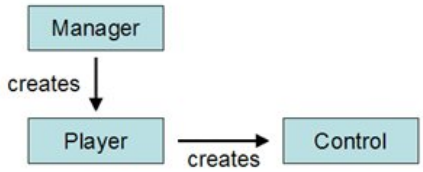
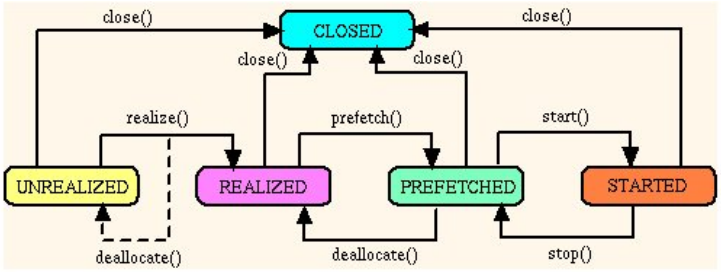


Figure 1. Manager, Player, Control: Relationship of the framework classes and interfaces used to manage multimedia using the Mobile Media APIs.

A Player has five states : UNREALIZED, REALIZED, PREFETCHED, STARTED, and CLOSED. The following figure shows the five states and the state transition methods. For more information about Player's states and transition methods click here

The **Figure 2.** shows the Player lifecycle.



Taking Pictures

In this example we take a picture and later we can see it on a Form. Before taking the picture we can see on display what the camera is capturing and choose the best position to take the picture.

Taking pictures requires one instance of Player, one instance of VideoControl and one instance of Display. VideoControl is used to control the video coming from the camera and display it in a Canvas/Form. First we create a instance of Player and set it to realize state.

```
// "capture://image" is used to indicate that we want a image.
// If we put another string here, we cannot use the method getSnapshot()
//in VideoControl .
Player mPlayer = Manager.createPlayer("capture://image");
mPlayer.realize();
```

Then we initialize one instance of VideoControl and append the video in the Canvas/Form, doing this the video become visible.

(C) Copyright Nokia 2013. All rights reserved.

```
//Creates a new VideoControl in order to reproduce videos.
VideoControl mVideoControl=(VideoControl) mPlayer.getControl("VideoControl");
Form form = new Form("Camera form");
//Creates a view of the video
Item item = (Item) mVideoControl.initDisplayMode(
GUIControl.USE_GUI_PRIMITIVE, null);
form.append(item);
```

We use the method getSnapshot() from VideoControl to capture pictures. This method returns a byte array corresponding to our picture. Displaying this image requires an instance of Image for append it on the screen. Later we set our player's state to "close" and set to null our instance of VideoControl.

```
// Get the image.
byte[] raw = mVideoControl.getSnapshot(null);
Image image = Image.createImage(raw, 0, raw.length);

// Place it in the main form.
if (this.size() > 0 && this.get(0) instanceof StringItem)
     this.delete(0);

append(image);

// Flip back to the main form.
mDisplay.setCurrent(this);
// Shut down the player.
mPlayer.close();
mPlayer = null;
mVideoControl = null;
```

Recording Video

In this example we can record a video and after recording we can play it. As in the previous section recording videos requires one Player instance and one VideoControl instance. But in this case we change the string inside the method createPlayer to "capture://video".

```
//Creates a new Video player
Player myPlayer = Manager.createPlayer("capture://video");
//Sets to the realize state
myPlayer.realize();
//Gets the video control
videoControl = (VideoControl) myPlayer.getControl("VideoControl");
```

Showing on the screen what the camera is capturing requires create an instance of another class called "VideoCanvas". Our MIDlet is not recording anything yet it is just showing what the camera is capturing.

In VideoCanvas we added one command called "cmd_capture". When we action this command RecordCammera is called. RecordCamera is a thread responsible for recording a video and put the content in an byte array.

```
class RecordCamera extends Thread { // RECORDING!!!.....
 RecordControl rc;
 public void run() {
  try {
   rc = (RecordControl) myPlayer.getControl("RecordControl");
   if (rc == null) {
   return;
  }
                //The video stream will be stored in a ByteArrayOutputStream object
  output = new ByteArrayOutputStream();
  rc.setRecordStream(output);
  rc.startRecord();

  } catch (Exception e) {
   e.printStackTrace();
  }
 }
//method called when the user stops the recording
 public void StopRecord() {
  try {
   if (rc != null) {
                                //Stop recording
    rc.stopRecord();
                                //Finalizes recording
    rc.commit();
   }
  } catch (Exception e) {
   e.printStackTrace();
  }
 }
}
```

Displaying what was recorded involves getting the byte array created when we recorded the video, setting our player's type, setting our player's status to realize, setting the VideoControl again because kind of the data changed and finally display the byte array as a video using the class VideoCanvas.

```
ByteArrayInputStream bis = new ByteArrayInputStream(output.toByteArray());
//Creates a new video player with "3gpp" as its content type
myPlayer = Manager.createPlayer(bis, "video/3gpp");
//Sets to the realize state
myPlayer.realize();
videoControl = (VideoControl) myPlayer.getControl("VideoControl");
if (videoCanvas != null) {
     videoCanvas.initControls(videoControl, myPlayer);
     display.setCurrent(videoCanvas);
     //starts playing
     myPlayer.start();
}
```

 (C) Copyright Nokia 2013. All rights reserved.

Playing Music

In our example we will play a mp3 music that is stored into a folder called “res”. Playing music requires a InputStream to store the music and one instance of Player. Our class has to implement Runnable because the music is played by another thread.

The string “audio/mpeg” inside the method createPlayer indicates that the music is in mp3 format.

```
//Loads the audio from the /res folder
InputStream in = getClass().getResourceAsStream("/123.mp3");
//Creates a new player for audio rendering
player = Manager.createPlayer(in, "audio/mpeg");
//sets to the prefetch state
player.prefetch();
//starts rendering
player.start();
```

 (C) Copyright Nokia 2013. All rights reserved.

 (C) Copyright Nokia 2013. All rights reserved.