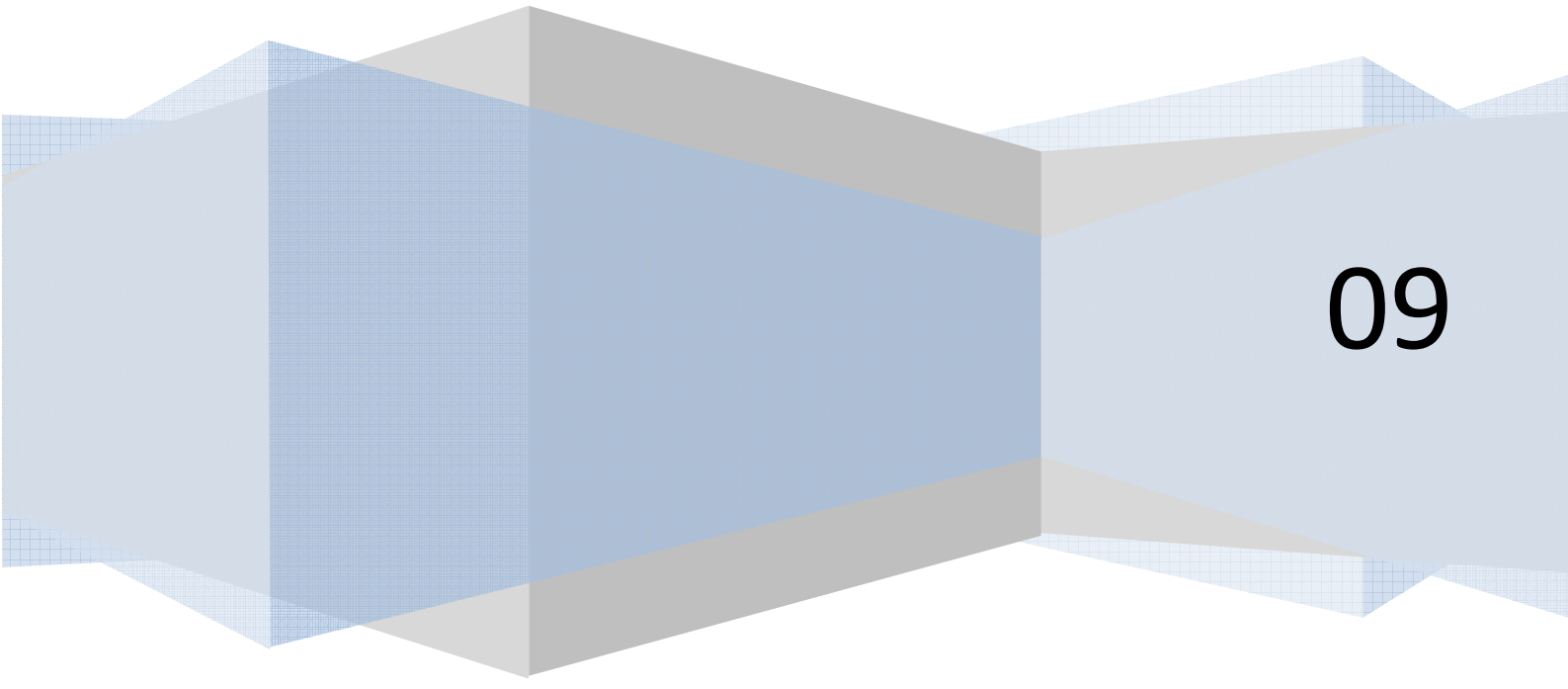
Research In Motion

C10 Audio And Video Playback

For BlackBerry SmartPhones

Andre Fabris



09

Contents

C10 Audio And Video Playback 3
Audio Playback 4
Manager Class 4
Player Class 5
Playing Audio 6
PlayerListener Interface 7
Audio Player Sample 8
Video Playback 13
Conclusion 17
Links 18

C10 Audio And Video Playback

This tutorial will show you how to play audio and video files in your application. I will show you how to use the Mobile Media API (MMAPI), also known as JSR 135 on BlackBerry SmartPhones.

I will guide you through the MMAPI, then show you on example on how to play simple tunes, how to play music from file bundled with the application, and how to play the file from your SD card.

I will also tell you which files are supported and how to implement PlayerListener to track events like when the song has finished.

Then we will cover the video playback. I will show you how to play video in full screen and non-full screen modes.

Audio Playback

To play audio on BlackBerry SmartPhones, we will use classes and interfaces from the javax.microedition.media package. This package is fully compatible with the MMAPI (JSR 135) and if you already have knowledge on how to use it, you can start using it straight away.

Let's start with an overview of the Manager class.

Manager Class

If you did not notice already there are two classes called Manager in the BlackBerry APIs. The first class is called javax.microedition.media.Manager and the other one is net.rim.device.api.ui.Manager. The first one is the one used for media handling and the latter one is used for managing the UI components of the screen. It is worth paying attention when importing packages.

The Manager class was introduced in version 4.0.0 of BlackBerry JDE. We need the Manager to get access to system resources such as the Player for multimedia processing.

So first thing the Manager does is create a Player which we use to play our media files.

But it can also be used to query a Player, what controls it supports, i.e. volume controls as well as supported content types and protocols. It can be also used to play single tones.

Content Types and Protocols

Content types indentify the type of media. They are also registered as MIME types.

Common audio content types are:

Wave audio files: audio/x-wav, usually files with extension .wav

MP3 audio files: audio/mpeg, usually files with extension .mp3

MIDI audio files: audio/midi, usually files with extension .midi

To find our which protocols specific content types support on the device we can use:

```
public static String[] getSupportedProtocols(String content_type).
```

You can also use:

```
public static String[] getSupportedContentTypes(String protocol),
```

to find out which content types specific protocols support.

Playing Single Tones

To play a single tone you can use:

```
public static void playTone(int note, int duration, int volume)
throws MediaException
```

where:

- SEMITONE_CONST = 17.31234049066755 = 1/(ln(2^(1/12)))
- note = ln(freq/8.176)*SEMITONE_CONST
- The musical note A = MIDI note 69 (0x45) = 440 Hz
- duration is note duration in milliseconds, and
- volume is in range from 0 to 100.

We can use ToneControl constants to play a single note to make our life a little bit easier:

```
try {
    Manager.playTone(ToneControl.C4, 1000, 100);
} catch (MediaException e) { }
```

This will play middle C note for 1 second at full volume.

Creating a Player

To create a player which will play our audio files we use the following:

```
Player Manager.createPlayer(String url);
```

Where url is the location of the audio file. It can be bundled with the application or located on the device's memory or SD card.

Player Class

Player has a life cycle. It has 5 states: unrealized, realized, prefetched, started and closed.

Once you create a player it is in the unrealized state. Once it is created it tries to connect to the source of the audio. When the system finds the source (which can be time consuming – especially when using the network) it moves the player in realized state.

The Prefetched state is when the player is ready to play audio. From the prefetched state we can start the player and it enters the started state. If we stop the playback or if the player reaches the end it will move back to the prefetched state. If we do not need it anymore we can close it.

It is important to know you cannot call some methods in some states.

It makes no sense for example to call method start(); to start playback when the player is in unrealized, realized or closed state. Figure 1 shows all the states a player can be in, and all the methods which we can call to get from one state into another.

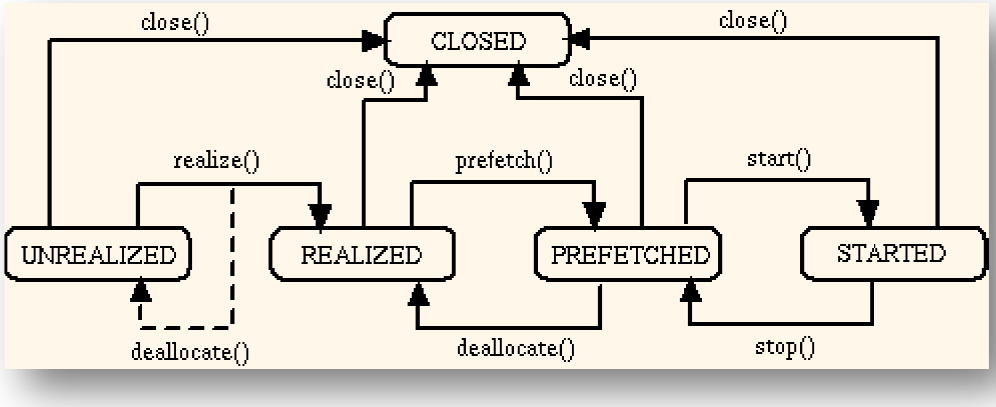


Figure 1

Playing Audio

To play audio in your application you might first want to choose which format to use. For background instrumental music, MIDI is the best format as the file sizes are very small, typically 10 – 20 kb, and the sound quality is very good. However if you want to have vocals, or just voice, or any other sound effects, such as thunder or explosions, you will probably choose mp3.

Mp3 files are bigger, but you can reduce sample rates, to create smaller files, at cost of quality.

Here is a code sample which opens a midi file called test.mid:

```
import java.io.InputStream;
import javax.microedition.media.*;
import javax.microedition.media.control.*;
.
.
.
public Player myPlayer;
public VolumeControl vc;
.
.
.
```

```
try {
     InputStream is = getClass().getResourceAsStream("/music/test.mid");
     myPlayer = Manager.createPlayer(is, "audio/midi");
     myPlayer.addPlayerListener(this);
     myPlayer.realize();
     myPlayer.prefetch();
     vc = (VolumeControl) myPlayer.getControl("VolumeControl");
     vc.setLevel(50);
} catch (Exception e) {    }
```

In this typical sample, we have a file test.mid bundled with the application. The application gets access to that file as a stream.

We pass that stream and content type (audio/midi) to the createPlayer method of the Manager class.

The next line is not needed but it is quite useful. It creates a listener which will notify our application of player event – such as if player reached the end of media.

After creating the player we need to move it into the realised and/or prefetched state, so it will be ready for playback.

If the device supports it we can get a volumeControl for the player and set a level.

It is a good practice to monitor the states of volume keys on the BlackBerry SmartPhone and increase and decrease the volume in your application.

To start or stop the playback we will use:

`myPlayer.start();` and `myPlayer.stop();` methods.

To close the player you might want to use a code like this:

```
try {
     myPlayer.removePlayerListener(this);
     myPlayer.stop();
     myPlayer.deallocate();
     myPlayer.close();
     myPlayer = null;
} catch (Exception e) {      }
```

Removing PlayerListener is needed if you used one. We move through the states of the player backwards - we stop it, deallocate it, and then we close it.

PlayerListener Interface

If your class implements this interface, it needs to overwrite the following method:

```
public void playerUpdate(Player player, String event, Object eventData)
```

This interface will notify your application as to which player caused the event, which event occurred, and will pass any associated data to the event.

Interesting events include:

- END_OF_MEDIA
- ERROR
- CLOSED

For the full list you can check the relevant API documents. The way you handle (or ignore) these events is up to you.

For example if you want to play one song after another, event END_OF_MEDIA will trigger a call to this method and then you might want to close the player and create and start another one for the next song.

Audio Player Sample

To add midi file to your application right click on your src folder and select Import (Figure 2). This is the way you add any type of resource files to your projects.

On the (Figure 3) you need to choose which type of resource you want to add. Select File System, and then browse to the location of the MIDI. You will need to check the box (Figure 4) next to the file name and then click Finish.

The full listing of the Audio Player sample is very simple and uses all the methods I described above.

It lists all media types and plays a MIDI file bundled within the application.

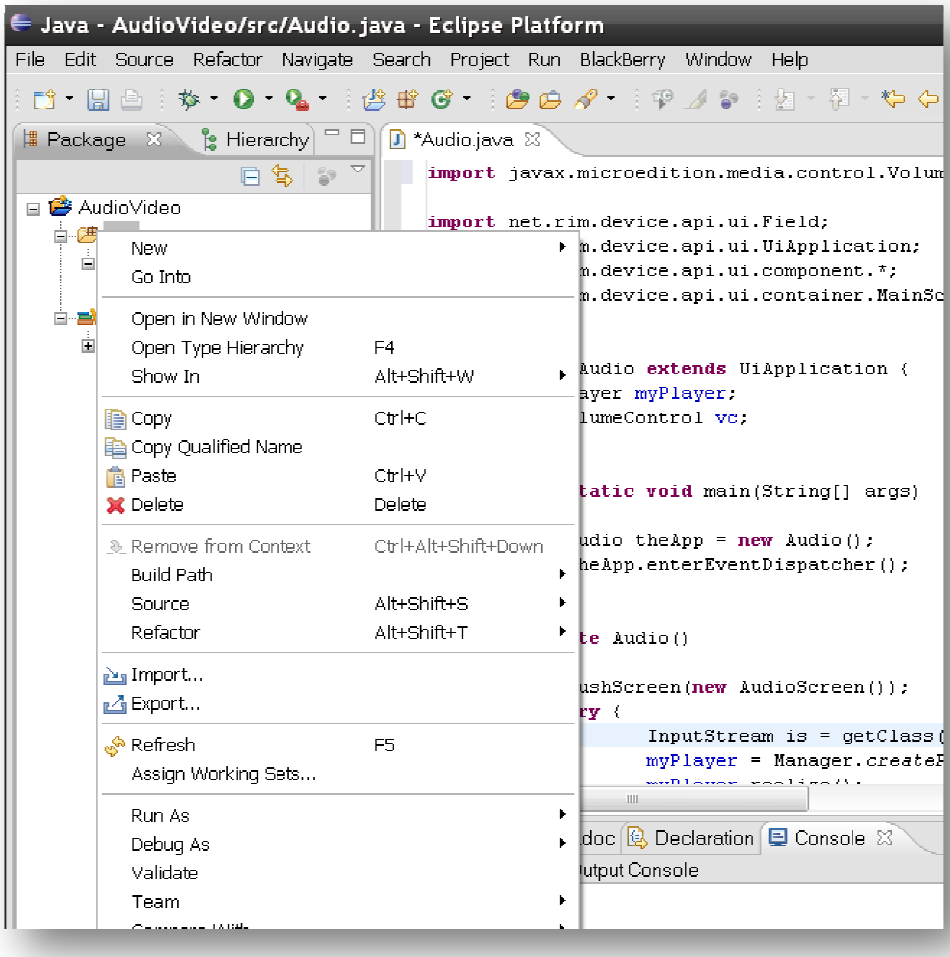

Java - AudioVideo/src/Audio.java - Eclipse Platform
File Edit Source Refactor Navigate Search Project Run BlackBerry Window Help
Package
Hierarchy
*Audio.java
AudioVideo
New
Go Into
Open in New Window
Open Type Hierarchy F4
Show In Alt+Shift+W
Copy Ctrl+C
Copy Qualified Name
Paste Ctrl+V
Delete Delete
Remove from Context Ctrl+Alt+Shift+Down
Build Path
Source Alt+Shift+S
Refactor Alt+Shift+T
Import...
Export...
Refresh F5
Assign Working Sets...
Run As
Debug As
Validate
Team
import javax.microedition.media.control.Volum
import net.rim.device.api.ui.Field;
m.device.api.ui.UiApplication;
m.device.api.ui.component.*;
m.device.api.ui.container.MainSc
Audio extends UiApplication {
ayer myPlayer;
lumeControl vc;
tatic void main(String[] args)
udio theApp = new Audio();
heApp.enterEventDispatcher();
te Audio()
ushScreen(new AudioScreen());
ry {
InputStream is = getClass(
myPlayer = Manager.createP
.doc
Declaration
Console
utput Console

Figure 2

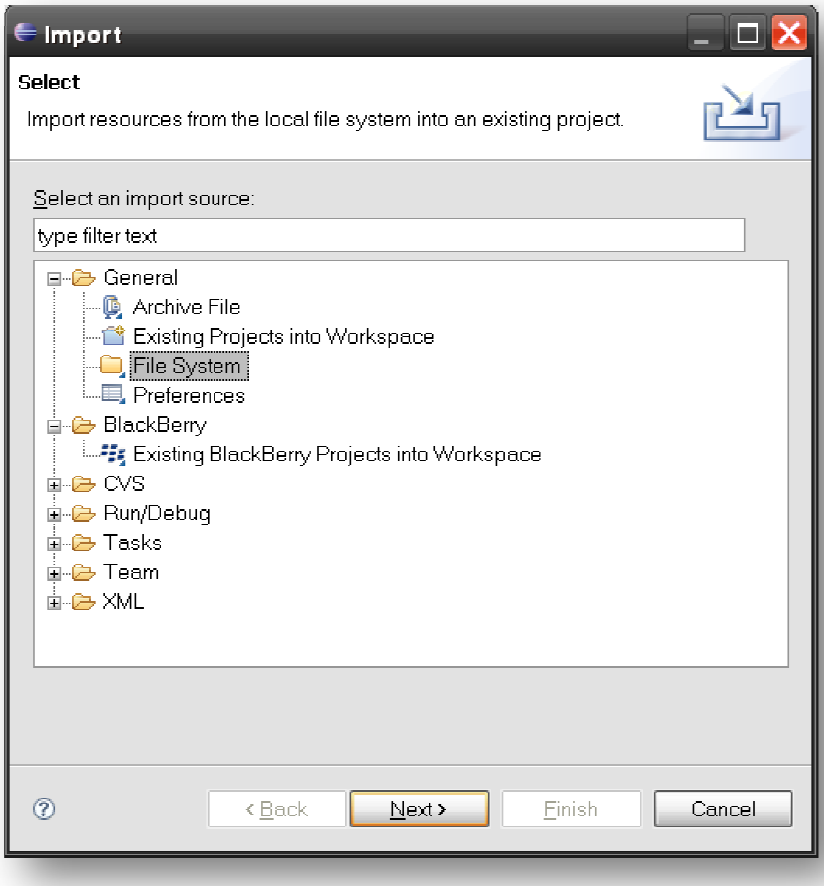

Import
Select
Import resources from the local file system into an existing project.
Select an import source:
type filter text
General
Archive File
Existing Projects into Workspace
File System
Preferences
BlackBerry
Existing BlackBerry Projects into Workspace
CVS
Run/Debug
Tasks
Team
XML
< Back
Next >
Finish
Cancel

Figure 3

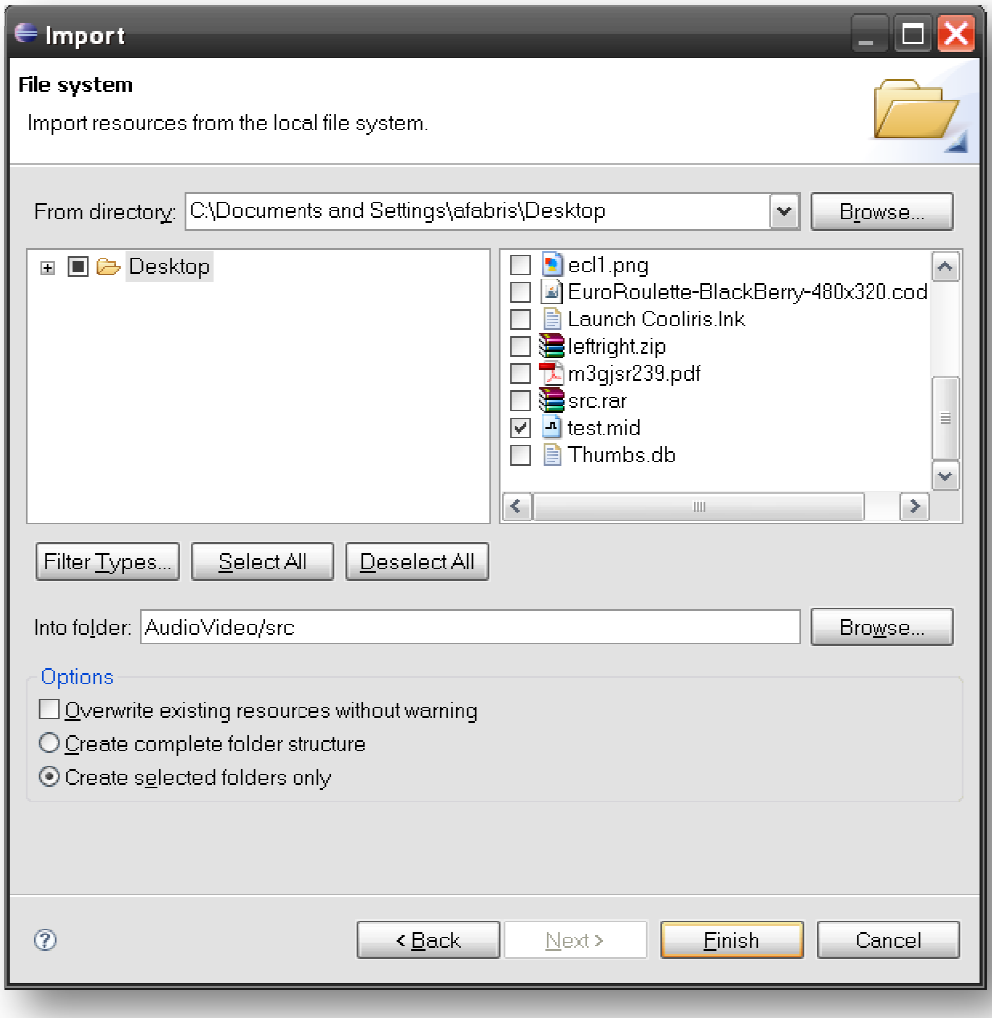

Import
File system
Import resources from the local file system.
From directory: C:\Documents and Settings\afabris\Desktop
Browse...
Desktop
ecl1.png
EuroRoulette-BlackBerry-480x320.cod
Launch Cooliris.lnk
leftright.zip
m3gjsr239.pdf
src.rar
test.mid
Thumbs.db
Filter Types...
Select All
Deselect All
Into folder: AudioVideo/src
Browse...
Options
Overwrite existing resources without warning
Create complete folder structure
Create selected folders only
< Back
Next >
Finish
Cancel

Figure 4

```
import java.io.InputStream;
import javax.microedition.media.*;
import javax.microedition.media.Manager;
import javax.microedition.media.control.VolumeControl;
import net.rim.device.api.ui.*;
import net.rim.device.api.ui.component.*;
import net.rim.device.api.ui.container.MainScreen;

public class Audio extends UiApplication {
      public Player myPlayer;
      public VolumeControl vc;

       public static void main(String[] args)
          {
              Audio theApp = new Audio();
              theApp.enterEventDispatcher();
          }

          private Audio()
          {
              pushScreen(new AudioScreen());
              try {
                      InputStream is =
getClass().getResourceAsStream("test.mid");
                      myPlayer = Manager.createPlayer(is, "audio/midi");
                      myPlayer.realize();
                      myPlayer.prefetch();
                      vc = (VolumeControl)
myPlayer.getControl("VolumeControl");
                      vc.setLevel(50);
                      myPlayer.start();
                  } catch (Exception e) {
                      System.out.println(e.toString());
                  }
          }
}

final class AudioScreen extends MainScreen
{
    AudioScreen()
    {
        LabelField title = new LabelField("Audio Playback Demo" ,
LabelField.ELLIPSIS | LabelField.USE_ALL_WIDTH);
        setTitle(title);
        add(new RichTextField("Supported types on this device"
,Field.NON_FOCUSABLE));
        String types[] = Manager.getSupportedContentTypes(null);
        for (int i = 0; i < types.length; i++) {
            add(new LabelField(types[i] ,Field.FOCUSABLE));
        }
    }
}
```

Video Playback

To play video in your application we will use the same Manager and Player classes.

Usually the videos are much larger so you will not include them with your application, so in this sample I will show you how to play video from the SD Card.

The previous sample listed all the supported audio and video types. Figure 5 shows the supported video types on 4.6.1 device.

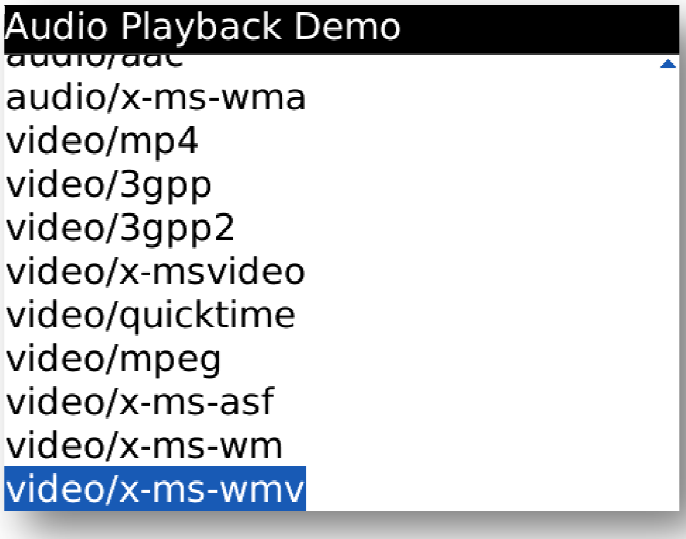


Figure 5

Mobile devices have small screens and limited memory. There is no need to have huge video files in high resolution, with a frame rate of 100 frames per second, and surround sound attached to it. If you have such a file I suggest you optimize it for the mobile devices. There are many convertors available on the internet, and some of them are even free.

```
try
{
   player = Manager.createPlayer("file:///SDCard/Trailers/Trailer.avi");
   player.realize();
   videoControl = (VideoControl)player.getControl("VideoControl");
   videoControl.initDisplayMode(VideoControl.USE_GUI_PRIMITIVE,
             "net.rim.device.api.ui.Field");
   videoControl.setVisible(true);
   player.start();
}
catch (Exception e){   }
```

In this sample we create a player, and then from that player we get VideoControl, which we use to create a field. At the end we make that field visible.

This sample will create a full screen video, which will be drawn over all the other components on the screen.

```
try {
       player = Manager.createPlayer("file:///SDCard/vid.3GP");
       player.realize();
       vc = (VideoControl) player.getControl("VideoControl");
       vc.initDisplayMode(VideoControl.USE_DIRECT_VIDEO, this);
       GUIControl gc;
       if ((gc = (GUIControl)player.getControl("GUIControl")) != null) {
            add((Field)gc.initDisplayMode(GUIControl.USE_GUI_PRIMITIVE, null));
        }
       vc.setDisplayLocation(15, 200);
       vc.setDisplaySize(200, 200);
       vc.setVisible(true);
        player.start();
} catch (Exception e) { }
```

To run this sample in the simulator you will need to "load" an SD Card. In the Simulate menu, select change SD Card. Select Add Directory and then browse to the folder on your PC which contains the video file you want to play (Figure 6).

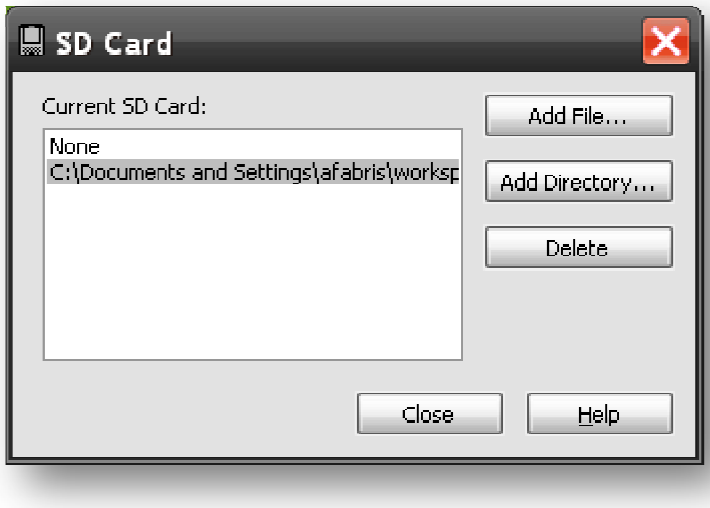


Figure 6

Figure 7 shows how the application looks in the simulator. Following this is the entire sample code used to create it.

Figure 7

```
import javax.microedition.media.Manager;
import javax.microedition.media.Player;
import javax.microedition.media.control.*;
import net.rim.device.api.ui.*;
import net.rim.device.api.ui.container.MainScreen;
import net.rim.device.api.ui.component.*;

class Video extends UiApplication {
	public static void main(String[] args) {
		Video theApp = new Video();
		theApp.enterEventDispatcher();
	}
	private Video() {
		pushScreen(new VideoScreen());
	}
}

final class VideoScreen extends MainScreen {
	private Player player;
	private VideoControl vc;

	VideoScreen() {
	try {
		player = Manager.createPlayer("file:///SDCard/vid.3GP");
		player.realize();
		vc = (VideoControl) player.getControl("VideoControl");
		vc.initDisplayMode(VideoControl.USE_DIRECT_VIDEO, this);
		GUIControl gc;
		if ((gc = (GUIControl) player.getControl("GUIControl")) != null)
			add((Field) gc.initDisplayMode(GUIControl.USE_GUI_PRIMITIVE,
			null));
		vc.setDisplayLocation(15, 100);
		vc.setDisplaySize(200, 200);
		vc.setVisible(true);
		player.start();
```

```
        } catch (Exception e) {    }
        LabelField title = new LabelField("Video Playback Sample",
               LabelField.ELLIPSIS | LabelField.USE_ALL_WIDTH);
        setTitle(title);
        add(new RichTextField("Playing video in window", Field.NON_FOCUSABLE));
        }
}
```

There are a couple of things we need to explain. The method setDisplayLocation sets where you want the video to appear (those are the coordinates of the top left corner of the video).

We also set the desired size of the video with the sedDisplaySize method. All the values are in pixels. The video size will be automatically scaled to the desired size.

This field does not act like a normal Field in a Manager, as the Manager has no control over its placement.

Conclusion

In this tutorial we have seen how easy it is to play various audio and video files within your application.

We did not touch the topic of streaming audio or video from the Internet as it is the topic of another tutorial.

We have described the basic functions of the Manager and Player classes as well as the PlayerListener and Video Control Interfaces.

There are more topics on multimedia that you may want to investigate. We suggest you follow the links below to learn more.

Links

BlackBerry Developers Web Site:

http://na.blackberry.com/eng/developers/

Developer Video Library:

- Playing Audio in Your Application

  http://www.blackberry.com/DevMediaLibrary/view.do?name=audioplayback

- Playing Video in Your Application

  http://www.blackberry.com/DevMediaLibrary/view.do?name=videoplaybacka

Developer Labs:

- Multimedia Labs
  http://na.blackberry.com/eng/developers/resources/developer_labs.jsp#tab_tab_jde

Documentation:

- Documentation for developers can be found here:

  http://na.blackberry.com/eng/support/docs/developers/?userType=21

Knowledge Base Articles:

- How To – Play audio in application
  http://www.blackberry.com/knowledgecenterpublic/livelink.exe/fetch/2000/348583/800332/1089414/How_To_-_Play_audio_in_an_application.html?nodeid=1168553&vernum=0
- How to – Play video within a BlackBerry SmartPhone application
  http://www.blackberry.com/knowledgecenterpublic/livelink.exe/fetch/2000/348583/800332/1089414/How_To_-_Play_video_within_a_BlackBerry_smartphone_application.html?nodeid=1383173&vernum=0

Forums:

- The link to BlackBerry Development Forums:

  http://supportforums.blackberry.com/rim/?category.id=BlackBerryDevelopment

Sample Code:

- There is a sample application located in your JDE folder i.e.:
  C:\Program Files\Research In Motion\BlackBerry JDE 4.6.0\samples\com\rim\samples\device\embeddedmediademo