

MMAPI

Introduction

Mobile Media API (MMAPI), JSR 135, extends the functionality of the Java ME platform by providing audio, video and other time-based multimedia support to resource-constrained devices.

Before MMAPI was available, each vendor provided proprietary APIs to solve this problem, like [Nokia UI API](#).

MMAPI 1.0 was the first specification that worked with [MIDP 1.0](#). When devices included [MIDP 2.0](#), MMAPI 1.1 was released including the new security mechanisms in MIDP.

The MMAPI reference implementations include support for simple tone generation, tone sequencing, audio/video file playback and streaming, interactive MIDI, and audio/video capture.

An extension to this API is offered as [Advanced Multimedia Supplements](#).

Main Package

Main Package in this API is **javax.microedition.media**. This package includes the class **Manager**, that is the access point for obtaining system dependent resources such as Players for multimedia processing.

This package also includes these interfaces:

- Control: used to control some media processing functions.
- Controllable: provides an interface for obtaining the Controls from an object like a Player
- Player: controls the rendering of time based media data.
- PlayerListener: is the interface for receiving asynchronous events generated by Players
- TimeBase: is a constantly ticking source of time.

The Manager Class The Manager class is the static factory class in the MMAPI. The createPlayer() method is the factory method used to create Player instances.

Create Player from URI Locators The createPlayer() method takes in a URI locator string to specify the network location of the media file, or the data capturing mode, or the in-memory empty device type.

```
static Player createPlayer (String locator)
```

In the MMAPI, three types of URI locator strings are supported.

For media playback, the URI could point to a media file available on a remote HTTP server. The server must return the correct MIME type in the HTTP header for the createPlayer() method to determine which player instance to instantiate. For example, the URI locator string http://host/sample.mid is typically associated with the audio/midi MIME type on HTTP servers, and it would result in an audio MIDI file player instance.

MIME Types Description

audio/x-tone-seq tone sequence

audio/wav wav audio format, but player cannot be created from InputStream using this MIME type

audio/x-wav wav audio format

audio/au au audio format

audio/x-au au audio format, but player cannot be created from InputStream using this MIME type

audio/basic raw audi format

audio/amr amr audio format

audio/amr-wb amr wb audio format

audio/midi midi audio format

audio/sp-midi extended midi format

video/mp4 Mpeg4 video format

video/mpeg4 mpeg4 video format, but player cannot be created from InputStream using this MIME type

video/3gpp 3gpp video format

application/vnd.rn-realmedia real media video format

For media capture, the URL string takes the special format

capture://audio for audio capture or **capture://video** for still-image capture on a camera phone. The video mode displays video from the camera's viewfinder until you instruct the program to take a snapshot.

For MIDI and tone sequence players, we can instantiate empty players in memory and then use MIDIControl and ToneControl objects to set content dynamically.

The URI locator strings for such empty players are **device://midi**, which corresponds to the static value Manager.MIDI_DEVICE_LOCATOR, and **device://tone**, which corresponds to Manager.TONE_DEVICE_LOCATOR.

'Create Player'

The URI locator based approach is simple and powerful.

```
static Player createPlayer (InputStream is, String type)
```

Each individual device supports only a subset of the above mentioned MIME

types

Methods in Manager Class

```
// Returns the supported media types for a given protocol static String [] getSupportedContentTypes (String protocol) // Returns the supported protocols for a given media type static String [] getSupportedProtocols (String type)
```

System Properties in MMAPI

(1) supports.mixing ,

Query for whether audio mixing is supported.

(2) supports.audio.capture, Query for whether audio capture is supported.

(3) supports.video.capture, Query for whether video capture is supported.

(4) supports.recording, Query for whether recording is supported.

(5) audio.encodings, The string returned specifies the supported capture audio formats.

(6) video.encodings, The string returned specifies the supported capture video formats (video recording).

(7) video.snapshot.encodings, Supported video snapshot formats for the VideoControl.getSnapshot() method.

(8) microedition.media.version, Returns version for an implementation

(9) streamable.contents Returns formats that can be streamed. No streaming format is supported at this time.

Player Interface

The Player interface specifies the common behavior of all Player implementation classes provided by MMAPI implementers. The most important attribute of a Player is its life cycle states.

Player Life Cycle

A Player object can have the following states.

CLOSED: The player has released most of its resources, and it can never be used again. We can change the player from any other state to the closed state by calling the Player.close() method.

UNREALIZED: The player object has just been instantiated in the heap memory. It has not allocated any resources.

REALIZED: If the Player.realize() method is called in a unrealized state, the player acquires required media resources and moves itself to the realized state.

For example, if the player plays a remote media file over the HTTP network, the entire file is downloaded during the realizing process.

PREFETCHED: If the `Player.prefetch()` method is called, the player performs a number of potentially time-consuming startup tasks and moves itself to the prefetched state.

STARTED: By calling the `Player.start()` method, we can start the player, which starts the media playback or starts the capture player. Once the player is started, we can also call the `Player.stop()` method to stop it and return it to the prefetched state. A stopped player can be started again, and it will resume playing from the point at which it was stopped

,

,

,

,

,

,

PlayerListener Interface

The `PlayerListener` interface declares only one method, **`playerUpdate()`**, which is invoked every time the registered player receives an event. The caller `Player` object passes the event and any application-specific data.

```
void playerUpdate (Player player,
```

```
String event, Object data)
```

Player state changes have their corresponding events, such as `CLOSED`, `STARTED`, and `STOPPED`. The player life cycle method always returns immediately, and we can process state changes asynchronously.

A player could be stopped under several conditions. The `END_OF_MEDIA` event occurs when the entire media content is played back. The `STOPPED_AT_TIME` event occurs when the player is stopped at a preset time in a `StopTimeControl` (discussed later). The `STOPPED` event occurs only when the player's `stop()` method is invoked.

The `DEVICE_UNAVAILABLE` event occurs when there is an incoming call. The `DEVICE_AVAILABLE` event occurs when the call is ended.

The `Player` class provides methods to register and remove the `PlayerListener` objects.

```
void addPlayerListener (PlayerListener listener)  
void removePlayerListener (PlayerListener listener)
```

The `Player` class supports methods to query the status of the current media file.

```
String getContentType ()  
long getDuration ()  
long getMediaTime ()  
int getState ()  
TimeBase getTimeBase ()
```

The following methods set how many times the player will loop and play the content, the media time of the current play position, and a new `TimeBase` to synchronize this player with another one.

```
void setLoopCount (int count)  
long setMediaTime (long now)  
void setTimeBase (TimeBase master)
```

Control Interface

The `Control` interfaces in `MMAPI` allow developers to control aspects of media-specific players programmatically.

```
Control getControl (String type)  
Control [] getControls ()
```

Control objects are identified by the type strings. For example, the following code retrieves a VolumeControl object from an audio player and then adjusts the volume level.

```
VolumeControl vc = player.getControl ("VolumeControl");  
vc.setLevel (50);  
player.start ();
```

Links

[JCP specification](#)

[Nokia Developer resources for mobile application developers](#)

[Imprints>Addison-Wesley Professional](#)

[Advanced Multimedia Supplements](#)

