Multimedia

Índice

Hemos visto que MIDP 1.0 carece de soporte para la reproducción de sonido. Esta carencia se soluciona en MIDP 2.0 que proporciona soporte para la reproducción de distintos tipos de sonidos y músicas. Esta API que incorpora MIDP 2.0 es un subconjunto de la API multimedia MMAPI, que se ofrece como API opcional para los dispositivos MIDP. Esta API multimedia, a parte de sonido, nos permite reproducir video, y además con ella podremos capturar sonido, imágenes o video utilizando el micrófono y la cámara del móvil.

Vamos a comenzar viendo cómo reproducir sonido, cosa que podremos hacer tanto en los dispositivos MIDP 2.0 como en los dispositivos MIDP 1.0 que incorporen la API opcional MMAPI. Después de esto veremos como reproducir y capturar otros medios utilizando las características que sólo están disponibles en MMAPI, y que podrán implementar los modelos de teléfonos móviles multimedia.

Esta API multimedia se encuentra en el paquete `javax.microedition.media` y subpaquetes. En MIDP 2.0 tendremos sólo el subconjunto de clases de la API MMAPI necesarias para la reproducción de sonido.

1. Reproductor de medios

Para poder reproducir los distintos tipos de medios (sonidos, músicas, videos) deberemos utilizar un objeto reproductor de medios, al que referenciaremos mediante la interfaz `Player`. Para obtener el reproductor adecuado para un determinado medio utilizaremos la clase `Manager`. Esta clase ofrece una serie de métodos estáticos que nos permiten crear objetos reproductores para los medios que queramos reproducir.

Para crear un reproductor utilizaremos el método estático `createPlayer` de la clase `Manager`. Podemos crear el reproductor a partir de un flujo de datos abierto de donde leer el medio, o a partir de una URL que nos sirva para localizar este medio:

```
// A partir de un flujo de datos y un tipo
InputStream in = abrirFlujo();
Player player = Manager.createPlayer(in, tipo);

// A partir de una URL
Player player = Manager.createPlayer(url);
```

Cuando creamos el reproductor a partir del flujo de datos deberemos indicar explícitamente el tipo de medio del que se trata, ya que del flujo podrá leer el contenido pero no conocerá qué tipo de medio está leyendo. En el caso de la URL no será necesario, ya que podrá obtener información sobre el tipo de contenido conectando a dicha URL.

Si queremos abrir un medio almacenado en un fichero dentro del JAR de la aplicación podremos obtener un flujo para leer de dicho recurso con `getResourceAsStream`, y crear un reproductor para reproducir el contenido leído a través de este flujo. En este caso deberemos especificar el tipo MIME del medio que estamos abriendo. Cada implementación de MMAPI puede soportar tipos de datos distintos. Por ejemplo, posibles

tipos que podemos encontrar son:

`audio/x-wav`	Ficheros WAV
`audio/basic`	Ficheros AU
`audio/mpeg`	Ficheros MP3
`audio/midi`	Ficheros MIDI
`audio/x-tone-seq`	Secuencias de tonos
`video/mpeg`	Videos MPEG
`video/3gpp`	Videos 3GPP

Utilizando URLs podremos, además de abrir estos tipos de ficheros de medios proporcionando la URL donde se localizan, acceder a dispositivos de captura y reproducir *streams* de audio y video en tiempo real. Estas URLs tendrán la siguiente forma:

```
protocolo:dirección
```

Por ejemplo, podemos reproducir un sonido que haya disponible en Internet proporcionando su URL:

```
Player player =
    Manager.createPlayer("http://jtech.ua.es/pdm/sonido.wav");
```

Normalmente en este caso descargará el medio desde Internet y lo almacenará en un buffer temporal para reproducirlo. Podemos utilizar otras URLs, como por ejemplo `capture://dispositivo` para acceder a un dispositivo de captura (micrófono o cámara), o `rtp://direccion` para conectarnos a un servidor de *streaming* de audio o video. El soporte para la captura, *streaming*, o diferentes tipos de medios dependerá de cada modelo concreto de móvil.

1.1. Estados

La reproducción de muchos de estos tipos de medios consume gran cantidad de recursos. Deberemos almacenar el medio a reproducir en memoria, y cuando vayamos a reproducirlo necesitamos reservar una serie de recursos de forma exclusiva, como puede ser por ejemplo el altavoz del dispositivo. Dado que estos recursos son escasos, los reproductores nos permitirán decidir cuando los reservamos y cuando los liberamos.

Para controlar esta asignación de recursos, los reproductores pasarán por una serie de estados. En cada uno de ellos se habrán reservado una serie de recursos. Cambiando el estado actual podremos controlar este proceso de reserva de recursos por parte del reproductor. Hemos de tener en cuenta que si hemos reservado todos los recursos necesarios podremos empezar a reproducir el medio de forma casi inmediata, mientras que si todavía no se han reservado y queremos reproducir el sonido, el comienzo de la reproducción puede tener un cierto retardo debido a que previamente deberá adquirir los

recursos necesarios, lo cuál llevará un tiempo.

Los estados por los que pasará el reproductor son los siguientes:

- **Unrealized**: Nada más crear el reproductor se encontrará en este estado. En este estado todavía no se ha reservado ningún recurso. En este estado no podremos obtener ninguna información sobre los medios a reproducir. Podremos pasar al siguiente estado invocando el método `realize`.
- **Realized**: Normalmente en este estado el reproductor habrá adquirido todos los recursos que necesita para la reproducción del medio excepto aquellos que sean de acceso exclusivo, como puede ser el altavoz del móvil. Se habrá leído ya el medio a reproducir, por lo que a partir de este momento podremos obtener información sobre él. Para pasar el siguiente estado invocaremos el método `prefetch`.
- **Prefetched**: En el estado *realized* todavía no tenemos todos los recursos necesarios para empezar a reproducir el medio. Será al pasar a estado *prefetched* cuando obtengamos estos recursos. Esto implica por ejemplo reservar los recursos de uso exclusivo necesario, crear los *bufferes* de datos necesarios, etc. Cuando estemos en estado *prefetched* podremos empezar a reproducir el medio en cualquier momento, ya que tenemos todos los recursos necesarios. Si queremos empezar a reproducir el medio utilizaremos el método `start`. Si por el contrario, queremos liberar los recursos de uso exclusivo y volver a estado *realized*, utilizaremos `deallocate`.
- **Reproduciendo**: Nos encontraremos en este estado mientras se esté reproduciendo el medio. Cuando termine la reproducción se volverá automáticamente al estado *prefetched*. También podemos detener manualmente la reproducción invocando el método `stop`.
- **Cerrado**: Desde cualquiera de los estados anteriores podemos cerrar el reproductor invocando el método `close`. Con esto liberaremos todos los recursos y ya no podremos utilizar más el reproductor.

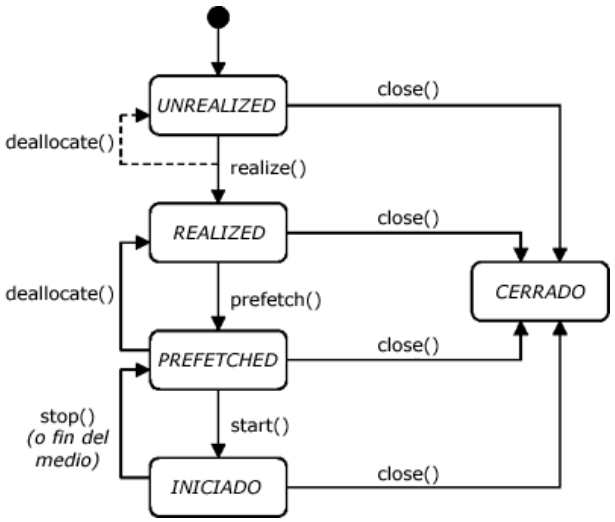


Diagrama de estados del reproductor

No tendremos que pasar por todos estos estados manualmente. Podremos llamar a `start` sin haber llamado antes a `realize` y `prefetch`. De esta forma en la misma llamada a `start` se pasará implícitamente por los estados *realized* y *prefetched*.

Podremos obtener el estado en el que se encuentra el reproductor en cada momento con el método:

```
int estado = player.getState();
```

1.2. Información y control del medio

Una vez en estado *realized*, podremos obtener información sobre el medio y controlar la forma en la que se va a reproducir. El objeto `Player` nos ofrecerá una serie de métodos con los que podremos acceder a esta información. Podemos obtener el tipo MIME del contenido del medio con:

```
String tipo = player.getContentType();
```

Podemos obtener la duración del medio en milisegundos cuando se reproduce a velocidad normal con:

```
long tiempo = player.getDuration();
```

Si no es posible obtener el tiempo del medio nos devolverá `Player.TIME_UNKNOWN`. Podemos modificar el tiempo de reproducción, de forma que el medio se reproduzca más lentamente o más rápidamente para ajustarse al nuevo tiempo. Podemos establecer u obtener este tiempo de reproducción con:

```
long tiempo = player.getMediaTime();
player.setMediaTime(tiempo);
```

Podemos también hacer que el medio se reproduzca de forma cíclica. Para ello deberemos indicar el número de ciclos que queremos que se reproduzca con:

```
player.setLoopCount(int numero);
```

Por defecto los medios se reproducirán una sola vez. Si queremos que se reproduzca en un ciclo infinito como número debemos especificar `-1`. Este método lo deberemos ejecutar siempre antes de que el medio haya empezado a reproducirse, nunca deberemos ejecutarlo en estado de reproducción ya que esto producirá una excepción.

Todas estas características son genéricas para todos los medios y por lo tanto podemos acceder a ellas a través del objeto `Player`. Si queremos acceder a otras características propias sólo de un determinado tipo de medios deberemos utilizar controles.

1.3. Controles de medios

Podemos encontrar controles para controlar determinadas características de los medios.

Según el tipo de medio del que se trate tendremos disponibles unos controles u otros. Por ejemplo, para los clips de audio tendremos disponibles controles para cambiar el volumen o los tonos reproducidos, mientras que para secuencias de video tendremos controles que nos permitan mostrar el video en la pantalla.

Podremos obtener estos controles a través del reproductor proporcionando el nombre del control al que queremos acceder, o bien obtener la lista completa de controles disponibles:

```
Control [] controles = player.getControls();
Control control = player.getControl(nombre);
```

Para obtener estos controles deberemos estar por lo menos en estado *realized*. Deberemos hacer una conversión *cast* al tipo de control concreto que estemos obteniendo en cada momento, como puede ser `VolumeControl`, `ToneControl` o `VideoControl`, para poder acceder a las características que controla cada uno.

1.4. Listener de medios

Podemos utilizar un *listener* sobre el reproductor de medios para conocer cuándo suceden determinados eventos, como por ejemplo cuándo terminan de reproducirse los medios. Este *listener* lo crearemos en una clase que herede de `PlayerListener`:

```
public class MiListener implements PlayerListener {
    public void playerUpdate(Player player,
                             String evento, Object datos) {

        if(evento == PlayerListener.STARTED) {
            // Ha comenzado la reproduccion
        } else if(evento == PlayerListener.STOPPED) {
            // Se ha detenido la reproduccion
        } else
            ...
    }
```

Cuando suceda un evento en el reproductor, se invocará el método `playerUpdate` que hayamos definido proporcionándonos como parámetro el reproductor sobre el que se ha producido el evento, el tipo de evento y una serie de datos que podemos tener asociados al evento. Los tipos de eventos están definidos como constantes en la clase `PlayerListener`. Entre ellos podemos encontrar los siguientes:

`PlayerListener.STARTED`	Ha comenzado la reproducción
`PlayerListener.STOPPED`	Se ha detenido la reproducción
`PlayerListener.END_OF_MEDIA`	Se ha llegado al final del medio
`PlayerListener.VOLUME_CHANGED`	Se ha cambiado el volumen del dispositivo
`PlayerListener.CLOSED`	Se ha cerrado el reproductor
`PlayerListener.ERROR`	Se ha producido un error en el reproductor

Una vez creado el *listener*, para que empiece a funcionar deberemos registrarlo en el reproductor con:

```
player.addPlayerListener(new MiListener());
```

2. Reproducción de tonos

Vamos a comenzar viendo la reproducción básica de tonos en el altavoz del móvil. Una forma sencilla de reproducir un tono individual es invocando el método:

```
Manager.playTone(nota, duracion, volumen);
```

Aquí deberemos especificar la nota que queremos tocar, su duración en milisegundos y el volumen que será un valor entero de 0 a 100.

Para reproducir una secuencia de tonos ya deberemos crear un reproductor. Este reproductor se creará con una URL especifica que representa el dispositivo de reproducción de tonos del móvil. Lo haremos de la siguiente forma:

```
Player player = Manager.createPlayer(Manager.TONE_DEVICE_LOCATOR);
```

Para poder establecer la secuencia de tonos deberemos crear un control de tonos asociado al reproductor de la siguiente forma:

```
player.realize();
ToneControl tc = (ToneControl)player.getControl("ToneControl");
tc.setSequence(new byte[] {
    ToneControl.VERSION,1,
    ToneControl.TEMPO,30,
    ToneControl.C4,16,
    ToneControl.C4+2,16,     //D4
    ToneControl.C4+4,16,     //E4
    ToneControl.C4+5,16,     //F4
    ToneControl.C4+7,16,     //G4
    ToneControl.C4+9,16,     //A4
    ToneControl.C4+11,16,    //B4
    ToneControl.C4+9,8,      //A4
    ToneControl.C4+7,8,      //G4
    ToneControl.C4+5,8,      //F4
    ToneControl.C4+4,8,      //E4
    ToneControl.C4+2,8,      //D4
    ToneControl.C4,8
});
```

Como secuencia de tonos debemos crear un *array* de *bytes* en el que indicaremos la secuencia de notas y una serie de parámetros como el *tempo*. En esta secuencia se especifican las notas y su duración en posiciones consecutivas.

Hemos de destacar que es necesario estar en estado *realized* para poder obtener los controles asociados al reproductor. Por esta razón hemos realizado previamente la transición a este estado.

Una vez establecida esta secuencia podremos reproducirla con:

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

3. Música y sonidos

Vamos a ver ahora cómo reproducir música y efectos de sonido utilizando el altavoz del móvil. Estos medios podemos leerlos por ejemplo de ficheros WAV o MIDI. Los ficheros MIDI serán adecuados para reproducir una música de fondo en nuestras aplicaciones, mientras que los ficheros WAV son más apropiados para efectos de sonido cortos.

De esta forma podemos poner música y efectos de sonido por ejemplo a los juegos que hagamos. Debemos tener en cuenta que algunos modelos de móviles no permiten realizar mezclado de medios, es decir, no podemos reproducir más de un medio al mismo tiempo. En este caso no podríamos hacer sonar efectos de sonido mientras oímos una música de fondo.

Para crear un reproductor para reproducir música o efectos de sonido deberemos crearlo a partir de un fichero de este tipo (WAV, AU, MID, MP3, etc). Podremos especificarlo o bien como una URL o como un flujo de entrada:

```
// A partir de una URL (fichero remoto en internet)
Player player =
    Manager.createPlayer("http://jtech.ua.es/pdm/musica.mid");
player.start();

// A partir de un fichero local
InputStream in = getClass().getResourceAsStream("/musica.mid");
Player player = Manager.createPlayer(in, "audio/midi");
player.start();
```

Para todos los medios que reproduzcan audio podremos obtener un control de volumen que nos permitirá cambiar el volumen del altavoz, o bien silenciarlo por completo. Para obtener este control haremos lo siguiente:

```
VolumeControl vol =
(VolumeControl)player.getControl("VolumeControl");
```

Una vez tengamos este control de volumen podremos cambiar el volumen o silenciarlo con los siguientes métodos:

```
vol.setLevel(volumen);
vol.setMute(true);
```

El nivel de volumen será un número entero de 0 a 100. Tendremos también otros dos métodos que nos permitirán consultar el nivel del volumen y si está silenciado o no.

4. Reproducción de video

La API multimedia incorporada en MIDP 2.0 sólo soporta audio, como hemos visto anteriormente. Sin embargo los teléfonos que incorporen la API completa MMAPI podrán reproducir también video. Podremos crear un reproductor de video de la misma forma que los de audio, utilizando en este caso un fichero de video (como por ejemplo MPEG, o 3GPP). Podremos crearlo de la siguiente forma:

```
InputStream in = getClass().getResourceAsStream("/video.3gp");
Player player = Manager.createPlayer(in, "video/3gpp");
```

Sin embargo, con esto todavía no podremos mostrar el video en pantalla. Para poder hacer esto necesitaremos obtener un control de video que nos permita mostrar el video en algún área de la pantalla del móvil. Para ello pasaremos a estado *realized* y obtendremos en control necesario:

```
player.realize();
VideoControl vc = (VideoControl)player.getControl("VideoControl");
```

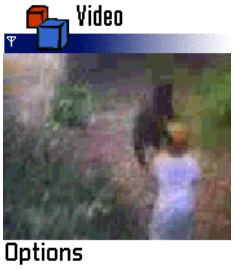
Ahora podremos añadir este video de distintas formas. Podemos o bien añadirlo a un *canvas*, o bien como elemento de un formulario. La primera forma nos permitirá tener un mayor control a bajo nivel sobre cómo se muestra el video en la pantalla.

Para añadirlo como primitiva de alto nivel a un formulario haremos lo siguiente:

```
Item item =
    (Item)vc.initDisplayMode(VideoControl.USE_GUI_PRIMITIVE, null);
formulario.append(item);
player.start();
```

Mientras que para visualizarlo en un *canvas* lo haremos de la siguiente forma:

```
vc.initDisplayMode(VideoControl.USE_DIRECT_VIDEO, canvas);
vc.setVisible(true);
player.start();
```



Reproductor de video

5. Captura

La API multimedia también nos permitirá capturar audio o video a través del micrófono o de la cámara que incorpore el teléfono móvil. Para acceder a estos dispositivos de captura utilizaremos una URL como la siguiente:

```
capture://dispositivo
```

Por ejemplo, con las siguientes URLs:

```
capture://audio
capture://video
capture://audio_video
```

Crearemos un reproductor para capturar audio, video o audio y video respectivamente. Con estas URLs se capturará con el formato que tengan estos medios por defecto. Además, en esta URL podremos añadir parámetros para establecer el formato y codificación de la captura. Por ejemplo, las siguientes URLs:

```
capture://audio?encoding=pcm
capture://video?width=160&height=120
```

Nos servirán para capturar audio con formato PCM y video con resolución de 160x120 *pixels* respectivamente.

Podremos crear un reproductor utilizando estas URLs igual que para los anteriores tipos de medios:

```
Player player = Manager.createPlayer("capture://video");
```

Ahora podremos mostrar el video capturado en pantalla igual que hemos visto anteriormente para la reproducción de ficheros de video:

```
player.realize();
VideoControl vc = (VideoControl)player.getControl("VideoControl");
vc.initDisplayMode(VideoControl.USE_DIRECT_VIDEO, canvas);
vc.setVisible(true);
player.start();
```

5.1. Grabación de medios

Normalmente nos interesará poder grabar el medio que estemos capturando. Para grabar un medio deberemos obtener un control de grabación `RecordControl`:

```
RecordControl rc = (RecordControl)
    player.getControl("RecordControl");
```

Ahora deberemos decidir donde grabar el medio. Deberemos proporcionar un flujo de salida para que almacene el video a través de él. Por ejemplo, podemos hacer que lo almacene en memoria asignando el siguiente flujo de salida:

```
ByteArrayOutputStream out = new ByteArrayOutputStream();
rc.setRecordStream(out);
```

Suponiendo que el medio ya se está reproduciendo, podemos comenzar a grabarlo en el flujo proporcionado con:

```
rc.startRecord();
```

Podemos detener la grabación con:

```
rc.stopRecord();
```

Después de detenerse puede reanudarse volviendo a llamar a `startRecord`. Si queremos finalizar la grabación, deberemos ejecutar:

```
rc.commit();
```

De esta forma se finalizará la escritura del video y en este caso ya no podremos reanudarlo.

5.2. Captura de imágenes

Mientras se reproduce el video capturado por la cámara o cualquier otro video podemos también capturar imágenes, de forma que se comporte como una cámara de fotos. Para capturar fotografías con la cámara utilizaremos el siguiente método del control de video:

```
byte [] img_png = vc.getSnapshot(null);
```

Con esto obtendremos la imagen en el formato por defecto, que es el formato PNG que soportan todos los móviles. Si queremos podemos especificar como parámetro de este método el formato en el que queremos que se capture la imagen.

Una vez tenemos los *bytes* de la imagen PNG, podremos crear un objeto `Image` a partir de ellos con:

```
Image img = Image.createImage(img_png, 0, img_png.length);
```