Reproducción de medios en Android

Índice

La capacidad de reproducir contenido multimedia es una característica presente en la práctica totalidad de las terminales telefónicas existentes en el mercado hoy en día. Muchos usuarios prefieren utilizar las capacidades multimedia de su teléfono, en lugar de tener que depender de otro dispositivo adicional para ello. Android incorpora la posibilidad de reproducir no sólo audio en diversos formatos, sino que también vídeo. Los formatos de audio soportados son los siguientes:

- AAC LC/LT
- HE-AACv1 (AAC+)
- HE-AACv2 (Enhanced ACC+)
- AMR-NB
- AMR-WB
- MP3
- MIDI
- Ogg Vorbis
- PCM/Wave

Con respecto al vídeo, los formatos soportados son:

- H.263
- H.264 AVC
- MPEG-4 SP

En esta sesión vamos a aprender a añadir contenido multimedia en nuestras aplicaciones. En concreto, veremos cómo reproducir audio o video en una actividad. También hablaremos brevemente de la toma de fotografías y de cómo incluir esta funcionalidad en nuestras aplicaciones. También describiremos brevemente el elemento *Media Store*.

1. Reproducción de audio

La reproducción de contenido multimedia se lleva a cabo por medio de la clase `MediaPlayer`. Dicha clase nos permite la reproducción de archivos multimedia almacenados como recursos de la aplicación, en ficheros locales, en proveedores de contenido, o servidos por medio de streaming a partir de una URL. En todos los casos, como desarrolladores, la clase `MediaPlayer` nos permitirá abstraernos del formato así como del origen del fichero a reproducir.

Incluir un fichero de audio en los recursos de la aplicación para poder ser reproducido durante su ejecución es muy sencillo. Simplemente creamos una carpeta *raw* dentro de la carpeta *res*, y almacenamos en ella sin comprimir el fichero o ficheros que deseamos reproducir. A partir de ese momento el fichero se identificará dentro del código como `R.raw.nombre_fichero` (obsérvese que no es necesario especificar la extensión del fichero).

Para reproducir un fichero de audio tendremos que seguir una secuencia de pasos. En primer lugar deberemos crear una instancia de la clase `MediaPlayer`. El siguiente paso

será indicar qué fichero será el que se reproducirá. Por último ya podremos llevar a cabo la reproducción en sí misma del contenido multimedia.

Veamos primero cómo inicializar la reproducción. Tenemos dos opciones. La primera de ellas consiste en crear una instancia de la clase `MediaPlayer` por medio del método `create`. En este caso se deberá pasar como parámetro, además del contexto de la aplicación, el identificador del recurso, como se puede ver en el siguiente ejemplo:

```
Context appContext = getApplicationContext();

// Recurso de la aplicación
MediaPlayer resourcePlayer =
        MediaPlayer.create(appContext, R.raw.my_audio);
// Fichero local (en la tarjeta de memoria)
MediaPlayer filePlayer =
        MediaPlayer.create(appContext,
Uri.parse("file:///sdcard/localfile.mp3"));
// URL
MediaPlayer urlPlayer =
        MediaPlayer.create(appContext,
Uri.parse("http://site.com/audio/audio.mp3"));
// Proveedor de contenido
MediaPlayer contentPlayer =
        MediaPlayer.create(appContext,
Settings.System.DEFAULT_RINGTONE_URI);
```

El otro modo de inicializar la reproducción multimedia es por medio del método `setDataSource`, el cual asigna una fuente multimedia a una instancia ya existente de la clase `MediaPlayer`. En este caso es muy importante recordar que se deberá llamar al método `prepare` antes de poder reproducir el fichero de audio (recuerda que esto último no es necesario si la instancia de `MediaPlayer` se ha creado con el método `create`).

```
MediaPlayer mediaPlayer = new MediaPlayer();
mediaPlayer.setDataSource("/sdcard/test.mp3");
mediaPlayer.prepare();
```

Una vez que la instancia de la clase `MediaPlayer` ha sido inicializada, podemos comenzar la reproducción mediante el método `start`. También es posible utilizar los métodos `stop` y `pause` para detener y pausar la reproducción. Si se detuvo la reproducción de audio mediante el método `stop` será imprescindible invocar el método `prepare` antes de poder reproducirlo de nuevo mediante una llamada a `start`. Por otra parte, si se detuvo la reproducción por medio de `pause`, tan sólo será necesario hacer una llamada a `start` para continuar en el punto donde ésta se dejó.

Otros métodos de la clase `MediaPlayer` que podríamos considerar interesante utilizar son los siguientes:

- `setLooping` nos permite especificar si el clip de audio deberá volver a reproducirse cada vez que finalice.

```
if (!mediaPlayer.isLooping())
        mediaPlayer.setLooping(true);
```

- `setScreenOnWhilePlaying` nos permitirá conseguir que la pantalla se encuentre activada siempre durante la reproducción. Tiene más sentido en el caso de la

reproducción de video, que será tratada en la siguiente sección.

```
mediaPlayer.setScreenOnWhilePlaying(true);
```

- `setVolume` modifica el volumen. Recibe dos parámetros que deberán ser dos números reales entre 0 y 1, indicando el volumen del canal izquierdo y del canal derecho, respectivamente. El valor 0 indica silencio total mientras que el valor 1 indica máximo volumen.

```
mediaPlayer.setVolume(1f, 0.5f);
```

- `seekTo` permite avanzar o retroceder a un determinado punto del archivo de audio. Podemos obtener la duración total del clip de audio con el método `getDuration`, mientras que `getCurrentPosition` nos dará la posición actual. En el siguiente código se puede ver un ejemplo de uso de estos tres últimos métodos.

```
mediaPlayer.start();

int pos = mediaPlayer.getCurrentPosition();
int duration = mediaPlayer.getDuration();

mediaPlayer.seekTo(pos + (duration-pos)/10);
```

Una acción muy importante que deberemos llevar a cabo una vez haya finalizado definitivamente la reproducción (porque se vaya a salir de la aplicación o porque se vaya a cerrar la actividad donde se reproduce el audio) es destruir la instancia de la clase `MediaPlayer` y liberar su memoria. Para ello deberemos hacer uso del método `release`.

```
mediaPlayer.release();
```

2. Reproducir vídeo mediante VideoView

La reproducción de vídeo es muy similar a la reproducción de audio, salvo dos particularidades. En primer lugar, no es posible reproducir un clip de vídeo almacenado como parte de los recursos de la aplicación. En este caso deberemos utilizar cualquiera de los otros tres medios (ficheros locales, streaming o proveedores de contenidos). Un poco más adelante veremos cómo añadir un clip de vídeo a la tarjeta de memoria de nuestro terminal emulado desde la propia interfaz de Eclipse. En segundo lugar, el vídeo necesitará de una superficie para poder reproducirse. Esta superficie se corresponderá con una vista dentro del layout de la actividad.

Existen varias alternativas para la reproducción de vídeo, teniendo en cuenta lo que acabamos de comentar. La más sencilla es hacer uso de una vista de tipo `VideoView`, que encapsula tanto la creación de una superficie en la que reproducir el vídeo como el control del mismo mediante una instancia de la clase `MediaPlayer`. Este método será el que veamos en primer lugar.

El primer paso consistirá en añadir la vista `VideoView` a la interfaz gráfica de la aplicación. Para ello añadimos el elemento en el archivo de layout correspondiente:

```
<VideoView android:id="@+id/superficie"
              android:layout_height="fill_parent"
              android:layout_width="fill_parent">
</VideoView>
```

Dentro del código Java podremos acceder a dicho elemento de la manera habitual, es decir, mediante el método `findViewById`. Una vez hecho esto, asignaremos una fuente que se corresponderá con el contenido multimedia a reproducir. El `VideoView` se encargará de la inicialización del objeto `MediaPlayer`. Para asignar un video a reproducir podemos utilizar cualquiera de estos dos métodos:

```
videoView1.setVideoUri("http://www.mysite.com/videos/myvideo.3gp");
videoView2.setVideoPath("/sdcard/test2.3gp");
```

Una vez inicializada la vista se puede controlar la reproducción con los métodos `start`, `stopPlayback`, `pause` y `seekTo`. La clase `VideoView` también incorpora el método `setKeepScreenOn(boolean)`con la que se podrá controlar el comportamiento de la iluminación de la pantalla durante la reproducción del clip de vídeo. Si se pasa como parámetro el valor `true` ésta permanecerá constantemente iluminada.

El siguiente código muestra un ejemplo de asignación de un vídeo a una vista `VideoView` y de su posterior reproducción. Dicho código puede ser utilizado a modo de esqueleto en nuestra propia aplicación. También podemos ver un ejemplo de uso de `seekTo`, en este caso para avanzar hasta la posición intermedia del video.

```
VideoView videoView = (VideoView)findViewById(R.id.superficie);
videoView.setKeepScreenOn(true);
videoView.setVideoPath("/sdcard/ejemplo.3gp");

if (videoView.canSeekForward())
        videoView.seekTo(videoView.getDuration()/2);

videoView.start();

// Hacer algo durante la reproducción

videoView.stopPlayback();
```

En esta sección veremos en último lugar, tal como se ha indicado anteriormente, la manera de añadir archivos a la tarjeta de memoria de nuestro dispositivo virtual, de tal forma que podamos almacenar clips de vídeo y resolver los ejercicios propuestos para la sesión. Se deben seguir los siguientes pasos:

- En primer lugar el emulador debe encontrarse en funcionamiento, y por supuesto, el dispositivo emulado debe hacer uso de una tarjeta SD.
- En Eclipse debemos cambiar a la perspectiva *DDMS*. Para ello hacemos uso de la opción *Window->Open Perspective...*
- A continuación seleccionamos la pestaña *File Explorer*. El contenido de la tarjeta de memoria se halla (normalmente) en la carpeta */mnt/sdcard*.
- Dentro de dicha carpeta deberemos introducir nuestros archivos de vídeo, dentro del directorio *DCIM*. Al hacer esto ya podrán reproducirse desde la aplicación nativa de reproducción de vídeo y también desde nuestras propias aplicaciones. Podemos

introducir un archivo de video con el ratón, arrastrando un fichero desde otra carpeta al interior de la carpeta *DCIM*, aunque también podemos hacer uso de los controles que aparecen en la parte superior derecha de la perspectiva *DDMS*, cuando la pestaña *File Explorer* está seleccionada. La función de estos botones es, respectivamente: guardar en nuestra máquina real algún archivo de la tarjeta de memoria virtual, guardar en la tarjeta de memoria virtual un archivo, y eliminar el archivo seleccionado.

Intercambio de ficheros con la tarjeta de memoria virtual

Aviso:

A veces es necesario volver a arrancar el terminal emulado para poder acceder a los vídeos insertados en la tarjeta de memoria desde la aplicación *Galería* de Android.

3. Reproducir vídeo con MediaPlayer

La segunda alternativa para la reproducción de vídeo consiste en la creación de una superficie en la que dicho vídeo se reproducirá y en el uso directo de la clase `MediaPlayer`. La superficie deberá ser asignada manualmente a la instancia de la clase `MediaPlayer`. En caso contrario el vídeo no se mostrará. Además, la clase `MediaPlayer` requiere que la superficie sea un objeto de tipo `SurfaceHolder`.

Un ejemplo de objeto `SurfaceHolder` podría ser la vista `SurfaceView`, que podremos añadir al XML del layout correspondiente:

```
<SurfaceView
        android:id="@+id/superficie"
        android:layout_width="200dip"
        android:layout_height="200dip"
        android:layout_gravity="center">
</SurfaceView>
```

El siguiente paso será la inicialización el objeto `SurfaceView` y la asignación del mismo a la instancia de la clase `MediaPlayer` encargada de reproducir el vídeo. El siguiente código muestra cómo hacer esto. Obsérvese que es necesario que la actividad implemente la interfaz `SurfaceHolder.Callback`. Esto es así porque los objetos de la clase `SurfaceHolder` se crean de manera asíncrona, por lo que debemos añadir un mecanismo que permita esperar a que dicho objeto haya sido creado antes de poder reproducir el vídeo.

```
public class MiActividad extends Activity implements
SurfaceHolder.Callback
{
        private MediaPlayer mediaPlayer;

        @Override
```

```
        public void onCreate(Bundle savedInstanceState) {
                super.onCreate(savedInstanceState);
                setContentView(R.layout.main);
                mediaPlayer = new MediaPlayer();
                SurfaceView superficie =
(SurfaceView)findViewById(R.id.superficie);
                // Obteniendo el objeto SurfaceHolder a partir del
SurfaceView
                SurfaceHolder holder = superficie.getHolder();
                holder.addCallback(this);
                holder.setType(SurfaceHolder.SURFACE_TYPE_PUSH_BUFFERS);
        }

        // Este manejador se invoca tras crearse la superficie, momento
        // en el que podremos trabajar con ella
        public void surfaceCreated(SurfaceHolder holder) {
                try {
                        mediaPlayer.setDisplay(holder);
                } catch (IllegalArgumentException e) {
                        Log.d("MEDIA_PLAYER", e.getMessage());
                } catch (IllegalStateException e) {
                        Log.d("MEDIA_PLAYER", e.getMessage());
                }
        }

        // Y este manejador se invoca cuando se destruye la superficie,
        // momento que podemos aprovechar para liberar los recursos
asociados
        // al objeto MediaPlayer
        public void surfaceDestroyed(SurfaceHolder holder) {
                mediaPlayer.release();
        }

        public void surfaceChanged(SurfaceHolder holder, int format, int
width, int height) { }
        }
}
```

Una vez que hemos asociado la superficie al objeto de la clase `MediaPlayer` debemos asignar a dicho objeto el clip de vídeo a reproducir. Ya que habremos creado el objeto `MediaPlayer` previamente, la única posibilidad que tendremos será utilizar el método `setDataSource`, como se muestra en el siguiente ejemplo. Recuerda que cuando se utiliza dicho método es necesario llamar también al método `prepare`.

```
public void surfaceCreated(SurfaceHolder holder) {
        try {
                mediaPlayer.setDisplay(holder);
                mediaPlayer.setDataSource("/mnt/sdcard/DCIM/video.mp4");
                mediaPlayer.prepare();
                mediaPlayer.start();
        } catch (IllegalArgumentException e) {
                Log.d("MEDIA_PLAYER", e.getMessage());
        } catch (IllegalStateException e) {
                Log.d("MEDIA_PLAYER", e.getMessage());
        } catch (IOException e) {
                Log.d("MEDIA_PLAYER", e.getMessage());
        }
}
```

4. Toma de fotografías

En esta sección veremos cómo tomar fotografías desde nuestra aplicación y utilizar la imagen obtenida para realizar alguna tarea. Como veremos se tratará ni más ni menos que un ejemplo clarísimo de `Intent` implícito, en el que pediremos al sistema que se lance una actividad que pueda tomar fotografías. Por medio de este mecanismo de comunicación obtendremos la imagen capturada (o una dirección a la localización de la misma en el dispositivo) para trabajar con ellas.

Nota:

En versiones anteriores del SDK de Android la emulación de la cámara no estaba soportada. Hoy en día es posible simular la cámara del dispositivo virtual por medio de una webcam, así que ya no es necesario utilizar un dispositivo real para poder probar estos ejemplos.

La acción a solicitar mediante el `Intent` implícito será `MediaStore.ACTION_IMAGE_CAPTURE` (más adelante hablaremos de la clase `MediaStore`). Lanzaremos el `Intent` por medio del método `startActivityForResult`, con lo que en realidad estaremos haciendo uso de una subactividad. Recuerda que esto tenía como consecuencia que al terminar la subactividad se invoca el método `onActivityResult` de la actividad padre. En este caso el identificador que se le ha dado a la subactividad es `TAKE_PICTURE`, que se habrá definido como una constante en cualquier otro lugar de la clase:

```
startActivityForResult(new Intent(MediaStore.ACTION_IMAGE_CAPTURE),
TAKE_PICTURE);
```

Si no hemos hecho ningún cambio al respecto en nuestro sistema, esta llamada lanzará la actividad nativa para la toma de fotografías. No podemos evitar recordar una vez más la ventaja que esto supone para el desarrollador Android, ya que en lugar de tener que desarrollar una nueva actividad para la captura de imágenes desde cero, es posible hacer uso de los recursos del sistema.

Según los parámetros del `Intent` anterior, podemos hablar de dos modos de funcionamiento en cuanto a la toma de fotografías:

- **Modo thumbnail**: este es el modo de funcionamiento por defecto. El `Intent` devuelto como respuesta por la subactividad, al que podremos acceder desde `onActivityResult`, contendrá un parámetro extra de nombre `data`, que consistirá en un thumbnail de tipo `Bitmap`.
- **Modo de imagen completa**: la captura de imágenes se realizará de esta forma si se especifica una URI como valor del parámetro extra `MediaStore.EXTRA_OUTPUT` del `Intent` usado para lanzar la actividad de toma de fotografías. En este caso se guardará la imagen obtenida por la cámara, en su resolución completa, en el destino indicado en dicho parámetro extra. En este caso el `Intent` de respuesta no se usará para devolver un thumbnail, y por lo tanto el parámetro extra `data` tendrá como valor `null`.

En el siguiente ejemplo tenemos el esqueleto de una aplicación en el que se utiliza un

`Intent` para tomar una fotografía, ya sea en modo thumbnail o en modo de imagen completa. Según queramos una cosa o la otra deberemos llamar a los métodos `getThumbnailPicture` o `saveFullImage`, respectivamente. En `onActivityResult` se determina el modo empleado examinando el valor del campo extra `data` del `Intent` de respuesta. Por último, una vez tomada la fotografía, se puede almacenar en el *Media Store* (hablamos de esto un poco más adelante) o procesarla dentro de nuestra aplicación antes de descartarla.

```
private static int TAKE_PICTURE = 1;
private Uri ficheroSalidaUri;

private void getThumbailPicture() {
        Intent intent = new Intent(MediaStore.ACTION_IMAGE_CAPTURE);
        startActivityForResult(intent, TAKE_PICTURE);
}

private void saveFullImage() {
        Intent intent = new Intent(MediaStore.ACTION_IMAGE_CAPTURE);
        File file = new File(Environment.getExternalStorageDirectory(),
"prueba.jpg");
        ficheroSalidaUri = Uri.fromFile(file);
        intent.putExtra(MediaStore.EXTRA_OUTPUT, ficheroSalidaUri);
        startActivityForResult(intent, TAKE_PICTURE);
}

@Override
protected void onActivityResult(int requestCode, int resultCode, Intent
data) {
        if (requestCode == TAKE_PICTURE) {
                Uri imagenUri = null;
                // Comprobamos si el Intent ha devuelto un thumbnail
                if (data != null) {
                        if (data.hasExtra("data")) {
                                Bitmap thumbnail =
data.getParcelableExtra("data");
                                // HACER algo con el thumbnail
                        }
                }
                else {
                         // HACER algo con la imagen almacenada en
ficheroSalidaUri
                }
        }
}
```

5. Agregar ficheros multimedia en el Media Store

El comportamiento por defecto en Android con respecto al acceso de contenido multimedia es que los ficheros multimedia generados u obtenidos por una aplicación no podrán ser accedidos por el resto. En el caso de que deseemos que un nuevo fichero multimedia sí pueda ser accedido desde el exterior de nuestra aplicación deberemos almacenarlo en el *Media Store*, que mantiene una base de datos de la metainformación de todos los ficheros almacenados tanto en dispositivos externos como internos del terminal telefónico.

Nota:

El *Media Store* es un proveedor de contenidos, y por lo tanto utilizaremos los mecanismos ya estudiados en sesiones anteriores (consultar el módulo de persistencia) para acceder a la información que contiene.

Existen varias formas de incluir un fichero multimedia en el *Media Store*. La más sencilla es hacer uso de la clase `MediaScannerConnection`, que permitirá determinar automáticamente de qué tipo de fichero se trata, de tal forma que se pueda añadir sin necesidad de proporcionar ninguna información adicional.

La clase `MediaScannerConnection` proporciona un método `scanFile` para realizar esta tarea. Sin embargo, antes de escanear un fichero se deberá llamar al método `connect` y esperar una conexión al *Media Store*. La llamada a `connect` es asíncrona, lo cual quiere decir que deberemos crear un objeto `MediaScannerConnectionClient` que nos notifique en el momento en el que se complete la conexión. Esta misma clase también puede ser utilizada para que se lleve a cabo una notificación en el momento en el que el escaneado se haya completado, de tal forma que ya podremos desconectarnos del *Media Store*.

En el siguiente ejemplo de código podemos ver un posible esqueleto para un objeto `MediaScannerConnectionClient`. En este código se hace uso de una instancia de la clase `MediaScannerConnection` para manejar la conexión y escanear el fichero. El método `onMediaScannerConected` será llamado cuando la conexión ya se haya establecido, con lo que ya será posible escanear el fichero. Una vez se complete el escaneado se llamará al método `onScanCompleted`, en el que lo más aconsejable es llevar a cabo la desconexión del *Media Store*.

```
MediaScannerConnectionClient mediaScannerClient = new
MediaScannerConnectionClient() {
        private MediaScannerConnection msc = null;
        {
                msc = new MediaScannerConnection(getApplicationContext(),
this);
        msc.connect();
        }

        public void onMediaScannerConnected() {
                msc.scanFile("/mnt/sdcard/DCIM/prueba.mp4", null);
        }

        public void onScanCompleted(String path, Uri uri) {

                // Realizar otras acciones adicionales

                msc.disconnect();
        }
};
```