

# Multimedia

## Índice

1 Reproducción de audio.....	2
2 Reproducción de vídeo usando el control Video View.....	4
3 Reproducción de vídeo basada en Media Player.....	6
4 Toma de fotografías.....	7
5 Agregar ficheros multimedia en el Media Store.....	9
6 Sintetizador de voz de Android.....	10

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 AAC+)
- 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. Por desgracia la versión actual del emulador de Android no es capaz de llevar a cabo determinadas acciones relacionadas este tipo de contenido, como la grabación de audio o vídeo, por lo que tratamos de ceñirnos a aquellas características que sí puedan ser probadas con el emulador. La única excepción se da en la sección de toma de fotografías, cuyos ejemplos sólo podrán ser probados en un dispositivo real.

## 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();

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

El otro modo de inicializar la reproducción multimedia es por medio del método `setDataSource()` para asignar 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. En este último caso la reproducción continuará tras hacer una llamada al método `resume()`.

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 el diálogo 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. Reproducción de vídeo usando el control Video View

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 un control de tipo *Video View*, 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 el control *Video View* a la interfaz gráfica de la aplicación. En el fichero XML correspondiente deberemos añadir un elemento de tipo

VideoView:

```
<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 control *Video View* se encargará de la inicialización del *Media Player*. Para asignar un video a reproducir podemos utilizar cualquiera de estos dos métodos:

```
videoView1.setVideoUri("http://www.mysite.com/videos/myvideo.3gp");
lvideoView2.setVideoPath("/sdcard/test2.3gp");
```

Una vez inicializado el control 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 un control *Video View* 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 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 vídeo 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.



### 3. Reproducción de vídeo basada en Media Player

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 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`.

Para añadir un elemento *Surface Holder* a la interfaz gráfica de la actividad debemos incluir un elemento `SurfaceView` en el archivo XML correspondiente

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

El siguiente paso será la inicialización del 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);
            SurfaceHolder holder = superficie.getHolder();
            holder.addCallback(this);
            holder.setType(SurfaceHolder.SURFACE_TYPE_PUSH_BUFFERS);
        }

        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());
            }
        }

        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 identificador del 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("/sdcard/prueba.3gp");
        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 Android. Aunque no vamos a poder probarlo en el emulador, consideramos que el tema es lo suficientemente importante para ser tratado aquí, de tal forma que lo que se va a mostrar a continuación pueda ser probado por alguien que disponga de un dispositivo real.

De todas las alternativas existentes para la toma de fotografías desde Android veremos la más sencilla. Dicha alternativa consiste en hacer uso de un `Intent` cuyo parámetro sea la constante `ACTION_IMAGE_CAPTURE` definida en la clase `MediaStore` (de la que hablaremos más adelante). Al hacer uso de la siguiente línea de código

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

se ejecutará la aplicación nativa para la toma de fotografías, que también le permitirá al usuario modificar las opciones pertinentes. Esto le evitará al desarrollador tener que crear una nueva aplicación desde cero.

Usando este método podremos optar por dos modos de funcionamiento:

- **Modo thumbnail:** este es el modo de funcionamiento por defecto. Se devolverá un thumbnail de tipo `Bitmap` en el parámetro extra de nombre `data` devuelto por el `Intent` en el método `onActivityResult()`.
- **Modo de imagen completa:** si se especifica una URI en el parámetro extra `MediaStore.EXTRA_OUTPUT` del `Intent`, se guardará la imagen tomada por la cámara, en su resolución real, en el destino indicado. En este caso no se devolverá un thumbnail como resultado del `Intent` y el campo `data` valdrá `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` devuelto por el `Intent`. 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 getThumbnailPicture() {
    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.

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 automáticamente 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 escaneo 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 `onMediaScannerConnected()` 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("/sdcard/test1.jpg", null);
    }

    public void onScanCompleted(String path, Uri uri) {
        // Realizar otras acciones adicionales
        msc.disconnect();
    }
};
```

## 6. Sintetizador de voz de Android

Android incorpora desde la versión 1.6 un motor de síntesis de voz conocido como *Text to Speech*. Mediante su API podremos hacer que nuestros programas "lean" un texto al usuario. Es necesario tener en cuenta que por motivos de espacio en disco los paquetes de lenguaje pueden no estar instalados en el dispositivo. Por lo tanto, antes de que nuestra aplicación utilice *Text to Speech* se podría considerar una buena práctica de programación el comprobar si dichos paquetes están instalados. Para ello podemos hacer uso de un `Intent` como el que se muestra a continuación:

```
Intent intent = new
Intent(TextToSpeech.Engine.ACTION_CHECK_TTS_DATA);
startActivityForResult(intent, TTS_DATA_CHECK);
```

El método `onActivityResult()` recibirá un `CHECK_VOICE_DATA_PASS` si todo está correctamente instalado. En caso contrario deberemos iniciar una nueva actividad por medio de un nuevo `Intent` que haga uso de la acción `ACTION_INSTALL_TTS_DATA` del motor *Text to Speech*.

Una vez comprobemos que todo está instalado deberemos crear e inicializar una instancia

de la clase `TextToSpeech`. Como no podemos utilizar dicha instancia hasta que esté inicializada, la mejor opción es pasar como parámetro al constructor un método `onInit()` de tal forma que en dicho método se especifiquen las tareas a llevar a cabo por el sintetizador de voz una vez esté inicializado.

```
boolean ttsIsInit = false;
TextToSpeech tts = null;

tts = new TextToSpeech(this, new OnInitListener() {
    public void onInit(int status) {
        if (status == TextToSpeech.SUCCESS) {
            ttsIsInit = true;
            // Hablar
        }
    }
});
```

Una vez que la instancia esté inicializada se puede utilizar el método `speak()` para sintetizar voz por medio del dispositivo de salida por defecto. El primer parámetro será el texto a sintetizar y el segundo podrá ser o bien `QUEUE_ADD`, que añade una nueva salida de voz a la cola, o bien `QUEUE_FLUSH`, que elimina todo lo que hubiera en la cola y lo sustituye por el nuevo texto.

```
tts.speak("Hello, Android", TextToSpeech.QUEUE_ADD, null);
```

Otros métodos de interés de la clase `TextToSpeech` son:

- `setPitch()` y `setSpeechRate()` permiten modificar el tono de voz y la velocidad. Ambos métodos aceptan un parámetro real.
- `setLanguage()` permite modificar la pronunciación. Se le debe pasar como parámetro una instancia de la clase `Locale` para indicar el país y la lengua a utilizar.
- El método `stop()` se debe utilizar al terminar de hablar; este método detiene la síntesis de voz.
- El método `shutdown()` permite liberar los recursos reservados por el motor de *Text to Speech*.

El siguiente código muestra un ejemplo en el que se comprueba si todo está correctamente instalado, se inicializa una nueva instancia de la clase `TextToSpeech`, y se utiliza dicha clase para decir una frase en español. Al llamar al método `initTextToSpeech()` se desencadenará todo el proceso.

```
private static int TTS_DATA_CHECK = 1;
private TextToSpeech tts = null;
private boolean ttsIsInit = false;

private void initTextToSpeech() {
    Intent intent = new Intent(Engine.ACTION_CHECK_TTS_DATA);
    startActivityForResult(intent, TTS_DATA_CHECK);
}
```

```

protected void onActivityResult(int requestCode, int resultCode,
Intent data) {
    if (requestCode == TTS_DATA_CHECK) {
        if (resultCode == Engine.CHECK_VOICE_DATA_PASS) {
            tts = new TextToSpeech(this, new
OnInitListener() {
                public void onInit(int status) {
                    if (status ==
TextToSpeech.SUCCESS) {
                        ttsIsInit = true;
                        Locale loc = new
Locale("es", "", "");
                        if
(tts.isLanguageAvailable(loc) >= TextToSpeech.LANG_AVAILABLE)
tts.setLanguage(loc);
                        tts.setPitch(0.8f);
                        tts.setSpeechRate(1.1f);
                        speak();
                    }
                }
            });
        } else {
            Intent installVoice = new
Intent(Engine.ACTION_INSTALL_TTS_DATA);
            startActivity(installIntent);
        }
    }
}

private void speak() {
    if (tts != null && ttsIsInit) {
        tts.speak("Hola Android", TextToSpeech.QUEUE_ADD,
null);
    }
}

@Override
public void onDestroy() {
    if (tts != null) {
        tts.stop();
        tts.shutdown();
    }
    super.onDestroy();
}

```

