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/LTP

HE-AACv1 (AAC+)

HE-AACv2 (Enhanced ACC+)

AMR-NB

AMR-WB

FLAC

MP3

MIDI

Ogg Vorbis

PCM/Wave

Con respecto al vídeo, los formatos soportados son:

H.263

H.264 AVC

MPEG-4 SP

VP8

En esta sesión echaremos un vistazo a las herramientas necesarias para poder reproducir contenido multimedia (audio o vídeo) en una actividad.

También veremos cómo añadir la capacidad a nuestra aplicación para la toma de fotografías, una característica perfectamente emulada por el emulador en las últimas versiones del Android SDK.

1. Reproducción de audio

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).

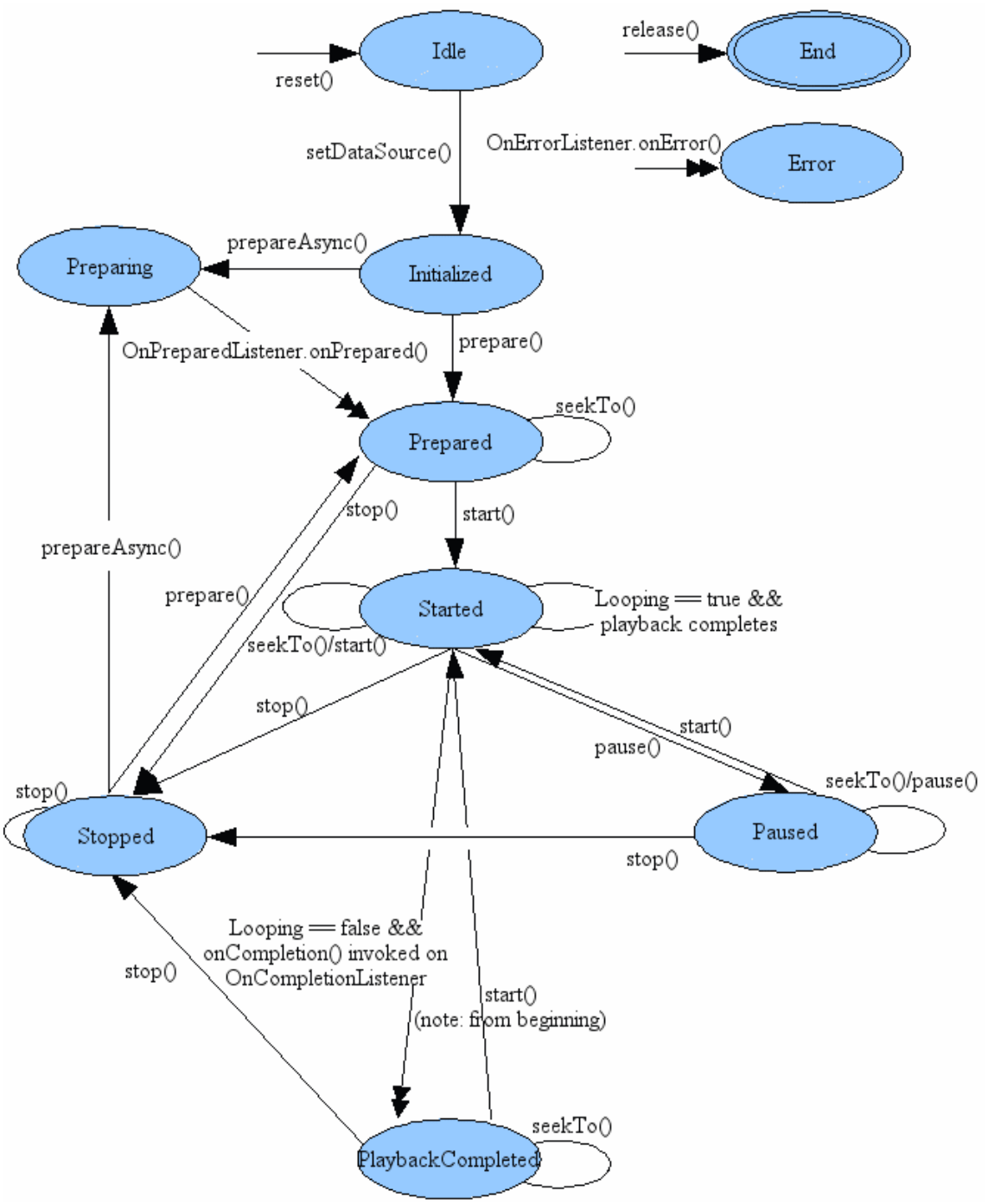
Para reproducir un fichero de audio tendremos que seguir una secuencia de pasos. En primer lugar deberemos crear una instancia de la clase MediaPlayer. El siguiente paso será indicar qué fichero será el que se reproducirá. Por último ya podremos llevar a cabo la reproducción en sí misma del contenido 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, tal como se puede ver en el siguiente ejemplo:

```
//Desde fichero raw en la aplicacion

MediaPlayer mediaPlayer = MediaPlayer.create(context, R.raw.sound_file_1);
mediaPlayer.start(); // no need to call prepare(); create() does that for you

//Desde datos del sistema (Por ejemplo un content provider)

Uri myUri = ....; // initialize Uri here
MediaPlayer mediaPlayer = new MediaPlayer();
mediaPlayer.setAudioStreamType(AudioManager.STREAM_MUSIC);
mediaPlayer.setDataSource(getApplicationContext(), myUri);
mediaPlayer.prepare();
mediaPlayer.start();

//Desde una URL externa

String url = "http://........"; // your URL here
MediaPlayer mediaPlayer = new MediaPlayer();
mediaPlayer.setAudioStreamType(AudioManager.STREAM_MUSIC);
mediaPlayer.setDataSource(url);
mediaPlayer.prepare(); // might take long! (for buffering, etc)
mediaPlayer.start();
```

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()).

La clase `MediaPlayer` funciona como una máquina de estados. Se deben seguir los diferentes pasos para la preparación y reproducción del contenido multimedia en el orden correcto. En caso contrario se lanzará una excepción. El diagrama de estados de dicha clase se puede consultar en http://developer.android.com/reference/android/media/MediaPlayer.html.


Idle
reset()
release()
End
setDataSource()
OnErrorListener.onError()
Error
Preparing
prepareAsync()
Initialized
OnPreparedListener.onPrepared()
prepare()
seekTo()
Prepared
stop()
start()
prepareAsync()
prepare()
Started
Looping == true &&
playback completes
seekTo()/start()
stop()
start()
pause()
seekTo()/pause()
stop()
Stopped
Paused
stop()
Looping == false &&
onCompletion() invoked on
OnCompletionListener
stop()
start()
(note: from beginning)
seekTo()
PlaybackCompleted

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 play().

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 desde el principio una vez que éste llegue al final.

```
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 la actividad, por ejemplo) es destruir la instancia de la clase MediaPlayer y liberar su memoria. Para ello deberemos hacer uso del método release().

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

Preparación asíncrona

Hay que tener en cuenta que el método *prepare*() encargado de preparar la reproducción puede tomar demasiado tiempo, ya que lleva a cabo tareas como la decodificación del archivo. Esto, puede provocar que, al ser ejecutado en el hilo principal, este quede bloqueado y de lugar a un error de ANR.

Para evitar este caso, tenemos el método prepareAsync(), cuando el sistema termina de preparar el método onPrepare de onPreparedListener es llamado, para esto debe establecerse previamente a través de setOnPreparedListener().

Manejo de Errores.

Si se producen errores en tareas síncronas esto se traduce en algún error en la reproducción o la aparición de alguna excepción. Por el contrario, no ocurre lo mismo en tareas asíncronas, sino que sucede una llamada al método onError.

```
MediaPlayer mMediaPlayer;

    public void initMediaPlayer() {
        // ...initialize the MediaPlayer here...

        mMediaPlayer.setOnErrorListener(this);
    }

    @Override
    public boolean onError(MediaPlayer mp, int what, int extra) {
        // ... react appropriately ...
        // The MediaPlayer has moved to the Error state, must be
reset!
    }
```

Es importante recordar que cuando se ejecuta este método, el objeto MediaPlayer pasa al estado Error, por lo que es obligatorio llamar a reset() para volver a usarlo

Reproducción en un Servicio

Sería interesante pensar que un reproductor de música no requiere de interfaz para poder reproducirse. Una opción sería el uso de servicios para poder reproducir audios en segundo plano. Hay que tener en cuenta que la clase Service se ejecuta en el hilo principal de la aplicación asociada, por lo que aquellas tareas costosas deberían ejecutarse en hilos secundarios.

Un ejemplo sería el siguiente:

```
public class MyService extends Service implements
MediaPlayer.OnPreparedListener {
    private static final String ACTION_PLAY =
"com.example.action.PLAY";
    MediaPlayer mMediaPlayer = null;

    public int onStartCommand(Intent intent, int flags, int
startId) {
        ...
        if (intent.getAction().equals(ACTION_PLAY)) {
            mMediaPlayer = ... // initialize it here
            mMediaPlayer.setOnPreparedListener(this);
            mMediaPlayer.prepareAsync(); // prepare async to not
block main thread
        }
    }

    /** Called when MediaPlayer is ready */
    public void onPrepared(MediaPlayer player) {
        player.start();
    }
}
```

Liberar recursos

Un objeto MediaPlayer puede llegar a consumer bastantes recursos del Sistema, por lo que tendríamos que tomar la precaución de no mantener las instancias más de lo necesario. Para esto, cuando hayamos acabado con ellas, deberíamos llamar al método release() para asegurarnos que los recursos que hayan sido reservados sean liberados. Por ejemplo, si una actividad que usa un objeto MediaPlayer recibe una llamada a onStop(), debes liberar el objeto mediaPlayer, siempre y cuando no estés reproduciendo en segundo plano. Obviamente, en este caso, cuando una actividad vuelva a ser relanzada o activa de nuevo, el objeto debe crear otra instancia del objeto y prepararlo(prepare()) para la reproducción.

```
mediaPlayer.release();
mediaPlayer = null;
```