



Introducción a Android

Manuel Báez, Álvaro Borrego, Jorge Cordero,
Luis Cruz, Miguel González, Francisco Hernández,
David Palomero, José Rodríguez de Llera, Daniel Sanz,
Mariam Saucedo, Pilar Torralbo, Álvaro Zapata



G-TeC

www.tecnologiaUCM.es

Introducción a Android

ISBN: 978-84-96285-39-5

E.M.E. Editorial ©

Editores: Victoria López (vlopez@fdi.ucm.es) y Grupo Tecnología UCM (www.tecnologiaUCM.es)

Copyright de los autores, los editores y la Universidad Complutense de Madrid

Se permite la reproducción con fines académicos no lucrativos.

1. INTRODUCCIÓN

Daniel Sanz – Mariam Saucedo – Pilar Torralbo

¿QUÉ ES ANDROID?

En los últimos años los teléfonos móviles han experimentado una gran evolución, desde los primeros terminales, grandes y pesados, pensados sólo para hablar por teléfono en cualquier parte, a los últimos modelos, con los que el término “medio de comunicación” se queda bastante pequeño.

Es así como nace Android. Android es un sistema operativo y una plataforma software, basado en Linux para teléfonos móviles. Además, también usan este sistema operativo (aunque no es muy habitual), tablets, netbooks, reproductores de música e incluso PC's. Android permite programar en un entorno de trabajo (framework) de Java, aplicaciones sobre una máquina virtual Dalvik (una variación de la máquina de Java con compilación en tiempo de ejecución). Además, lo que le diferencia de otros sistemas operativos, es que cualquier persona que sepa programar puede crear nuevas aplicaciones, *widgets*¹, o incluso, modificar el propio sistema operativo, dado que Android es de código libre, por lo que sabiendo programar en lenguaje Java, va a ser muy fácil comenzar a programar en esta plataforma.

HISTORIA DE ANDROID

Fue desarrollado por Android Inc., empresa que en 2005 fue comprada por Google, aunque no fue hasta 2008 cuando se popularizó, gracias a la unión al proyecto de Open Handset Alliance, un consorcio formado por 48 empresas de desarrollo hardware, software y telecomunicaciones, que decidieron promocionar el software libre. Pero ha sido Google quien ha publicado la mayor parte del código fuente del sistema operativo, gracias al software Apache, que es una fundación que da soporte a proyectos software de código abierto.

Dado que Android está basado en el núcleo de Linux, tiene acceso a sus recursos, pudiendo gestionarlo, gracias a que se encuentra en una capa por encima del Kernel, accediendo así a recursos como los controladores de pantalla, cámara, memoria flash...

En la Figura 1, abajo, se muestran las capas que conforman el sistema operativo Android:

¹ Un *widget* es una pequeña aplicación que facilita el acceso a funciones frecuentes. Más información en el Capítulo 10

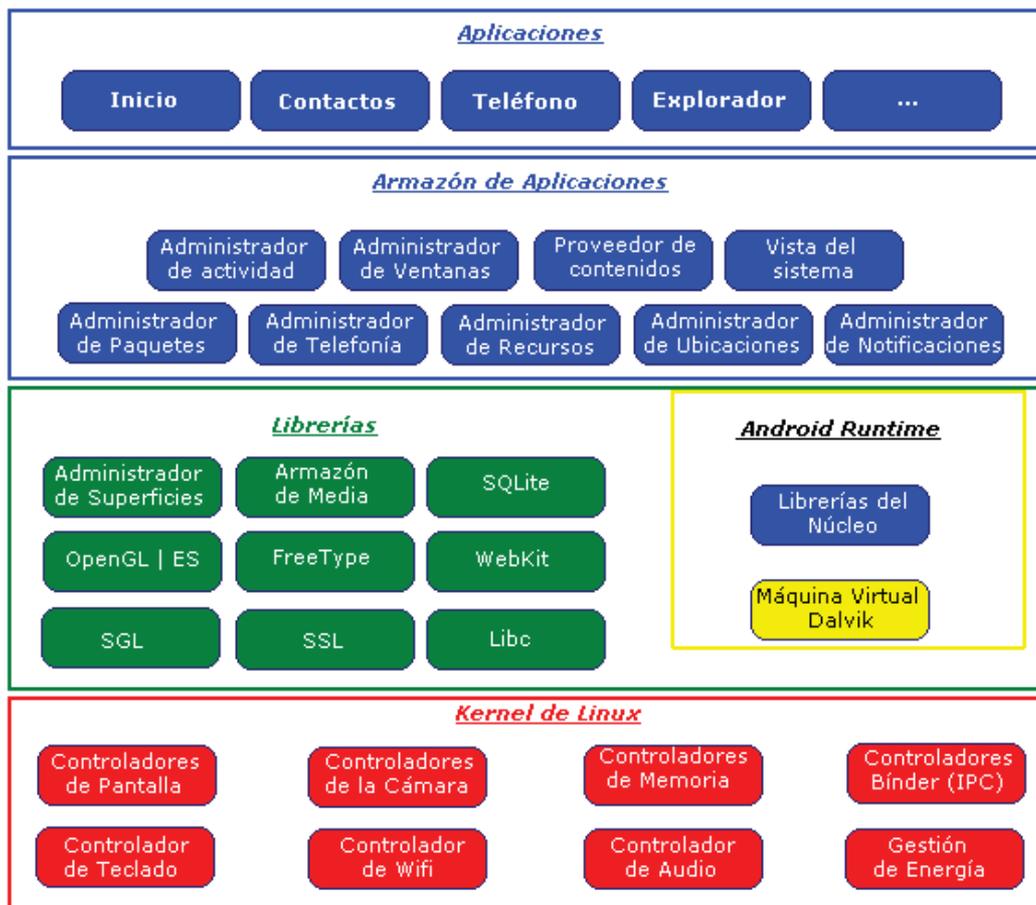


Figura 1. Sistema de capas de Android

En la imagen se distinguen claramente cada una de las capas: la que forma parte del propio Kernel de Linux, donde Android puede acceder a diferentes controladores, las librerías creadas para el desarrollo de aplicaciones Android, la siguiente capa que organiza los diferentes administradores de recursos, y por último, la capa de las aplicaciones a las que tiene acceso.

VERSIONES DISPONIBLES

El sistema operativo Android, al igual que los propios teléfonos móviles, ha evolucionado rápidamente, acumulando una gran cantidad de versiones, desde la 1.0 para el QWERTY HTC G1, hasta la 4.0 que acaba de salir al mercado.

- **Cupcake: Android Version 1.5**

Características: Widgets, teclado QWERTY virtual, copy & paste, captura de vídeos y poder subirlos a Youtube directamente.

- **Donut: Android Version 1.6**

Características: Añade a la anterior la mejoría de la interfaz de la cámara, búsqueda por voz, y navegación en Google Maps.

- **Eclair: Android Version 2.0/2.1**

Características: Mejoras en Google Maps, salvapantallas animado, incluye zoom digital para la cámara, y un nuevo navegador de internet.

- **Froyo: Android Version 2.2**

Características: Incluye hotspot Wifi, mejora de la memoria, más veloz, Microsoft Exchange y video-llamada.

- **Ginger Bread: Android Version 2.3**

Características: Mejoras del consumo de batería, el soporte de vídeo online y el teclado virtual, e incluye soporte para pagos mediante NFC².

- **Honey Comb: Android Version 3.0/3.4**

Características: Mejoras para tablets, soporte Flash y Divx, integra Dolphin, multitarea pudiendo cambiar de aplicación dejando las demás en espera en una columna, widgets y homepage personalizable.

- **Ice Cream Sandwich: Android Version 4.0**

Características: Multiplataforma (tablets, teléfonos móviles y netbooks), barras de estado, pantalla principal con soporte para 3D, widgets redimensionables, soporte usb para teclados, reconocimiento facial y controles para PS3.

ECLIPSE COMO ENTORNO DE TRABAJO

En este curso de Android, se da por supuesto que el alumno está familiarizado con el entorno Eclipse y que además tiene nociones básicas de programación en el lenguaje Java. Lo primero que necesitaremos para poder programar en Android, es preparar el entorno de trabajo. Es necesario disponer de una versión de Eclipse Galileo 3.5 o superior para poder desarrollar nuestros proyectos. Lo segundo que necesitamos es el kit de desarrollo software para Android o Android SDK, del que se pueden encontrar varias versiones para diferentes plataformas en la página web:

<http://developer.android.com/sdk/index.html>

Si el sistema operativo es Windows, lo más recomendable, es descargar el instalador automático **installer_rXX-windows.exe**, y simplemente seguir las instrucciones. Una vez se inicia la instalación, el instalador comprueba si el equipo dispone del Java SE Development Kit (JDK). Si no es así, muestra un mensaje como el siguiente:

² NFC (Near-Field Communication) es una plataforma abierta para la comunicación instantánea.

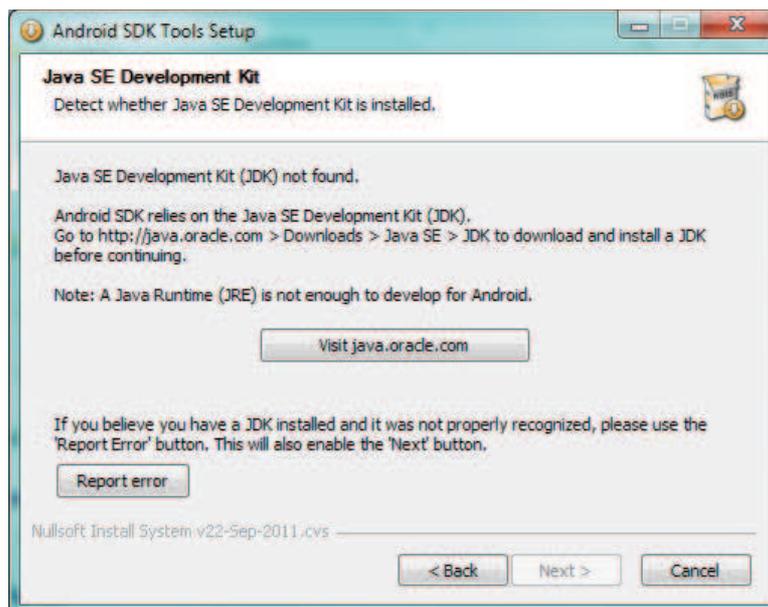


Figura 2. Android Setup

Simplemente pincha sobre el botón “Visit java.oracle.com” (Figura 1.1) que redireccionará a la página mencionada para descargar el paquete necesario. Una vez instalado el JDK, se continúa con la instalación del SDK. Cuando finalice el instalador, se ejecutará el SDK Manager, en el que se deberán seleccionar todas las casillas deshabilitadas, para instalar todas las versiones de Android así como sus herramientas (Tools).

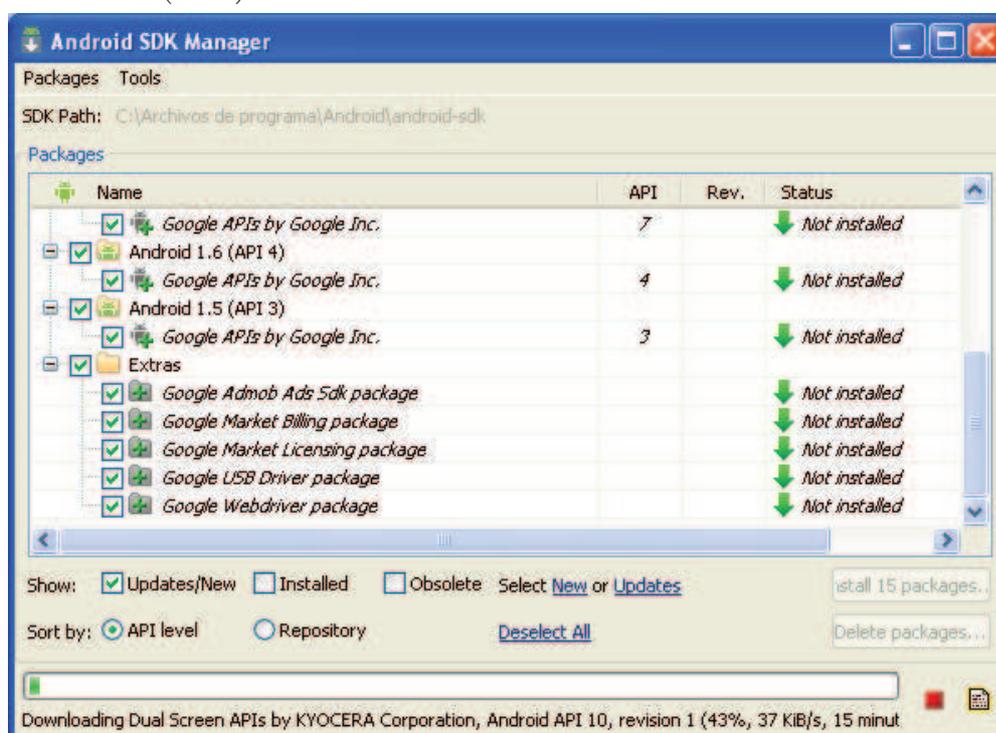


Figura 3. SDK Manager

Una vez todo esté descargado e instalado, abrir Eclipse para descargar el **ADT Plugin** e instalarlo en el entorno de desarrollo. Se deben seguir los siguientes pasos:

- 1.- En la pestaña “Help”, seleccionar “Install New Software”.
- 2.- Presionar el botón “Add” en la esquina superior derecha.
- 3.- En el cuadro de dialogo que aparece, escribir “ADT Plugin” en el campo “Name”, y la siguiente URL

en el campo “Location” y pulsar “OK” (Si existe algún problema para enlazar el entorno con éste link,

probar a poner http: eliminando la ‘s’):

<https://dl-ssl.google.com/android/eclipse/>

- 4.- En “Avalaible Software”, seleccionar la casilla correspondiente a “Developer Tools” y pulsar “Next”.

- 5.- Leer y aceptar el Acuerdo de licencia y presionar “Finish”(si salta una advertencia de seguridad

informando de que la autenticidad o validez del software no se puede establecer, simplemente pulsar

“OK”), y reiniciar Eclipse.

Lo único que queda es configurar el ADT Plugin. En Eclipse, en la pestaña “Window”, seleccionar “Preferences”, y elegir “Android” en el panel de la izquierda. Aparecerá un cuadro de dialogo preguntando si se quiere enviar estadísticas a Google, seleccionar la elección y pulsar “Proceed”. Ahora presionar el botón “Browse” y seleccionar la ruta del directorio dónde se haya ubicado el SDK (normalmente C:\Archivos de programa\Android\Android-sdk\)) y pulsar “Apply” y “OK”.

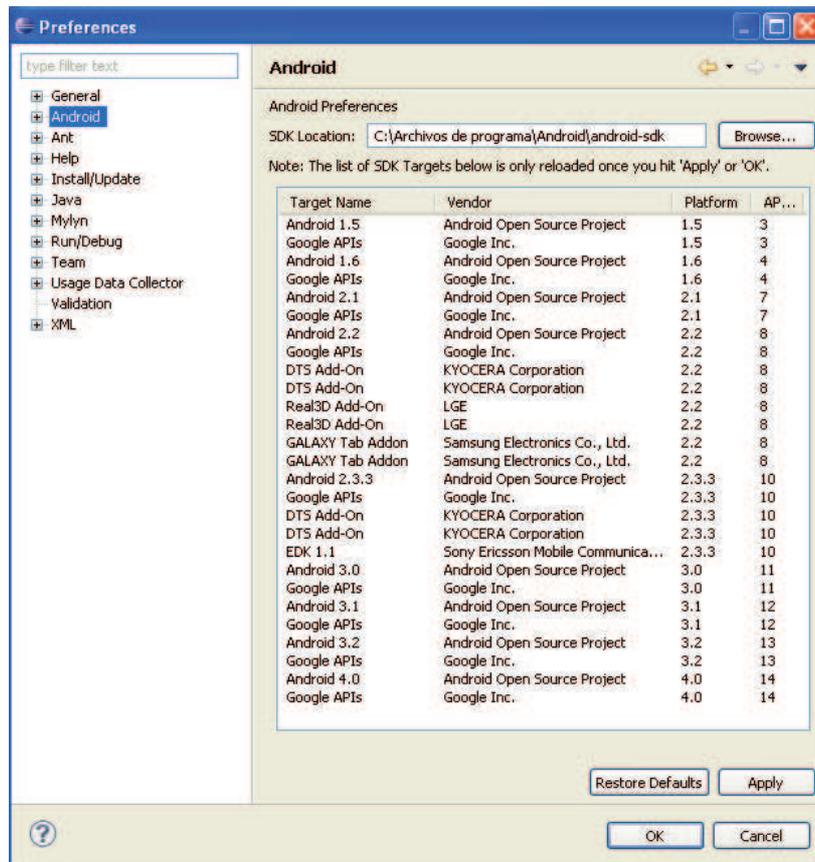


Figura 4. Preferences

Por último, hay que comprobar que el SDK está completamente actualizado. Para ello, en la pestaña “Window”, seleccionar “Android SDK and AVD Manager”. En la sección “Available

Packages”, seleccionar todas aquellas casillas a instalar. Presionar “Install Selected” para comenzar con la descarga e instalación.

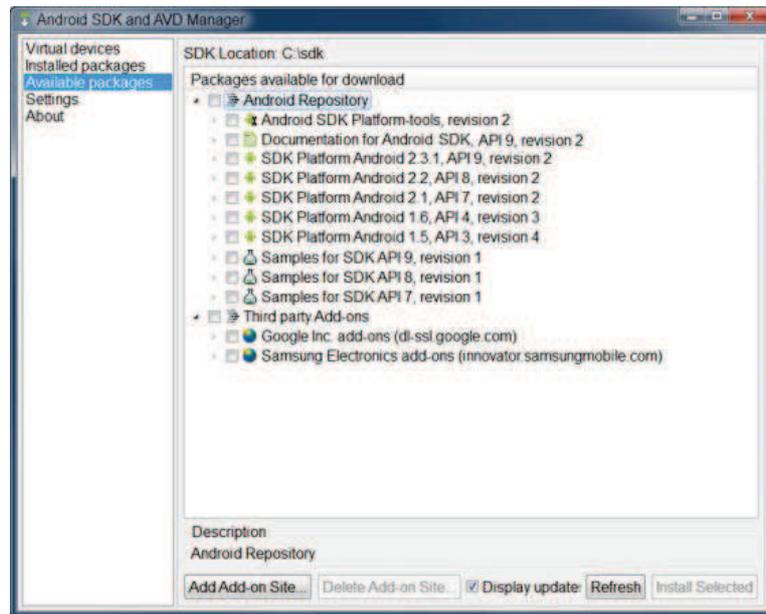


Figura 5. Repository

¡Y ya está! Ya tenemos preparado el entorno para poder programar en Android.

PERSPECTIVAS Y EMULADOR

1. PERSPECTIVA JAVA

Dados por sabidos los conocimientos básicos sobre Eclipse y la programación en Java, ésta perspectiva debe ser conocida por todos.

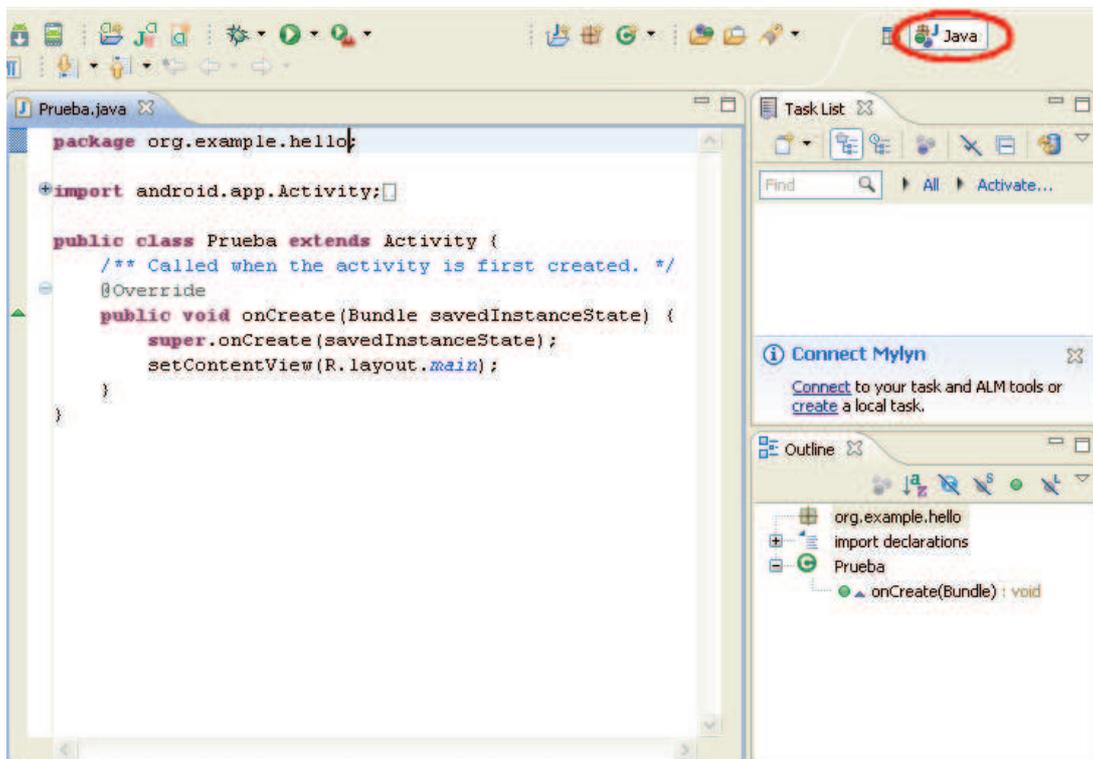


Figura 6. Perspectiva Java

Es la interfaz de usuario (o conjunto de vistas) que provee el JDT Plugin para poder programar en lenguaje Java. Esta interfaz, proporciona una serie de herramientas (se puede considerar como una determinada organización de las vistas), para el correcto desarrollo de programas y aplicaciones, y será la que utilizaremos para programar en este curso de Android.

2. PERSPECTIVA DDMS

En este caso, es el ADT Plugin el que nos proporciona la nueva perspectiva, por lo que lo primero que hay que hacer es habilitarla.

En la pestaña “Window”, seleccionar “Open Perspective” -> “Other”-> “DDMS”.

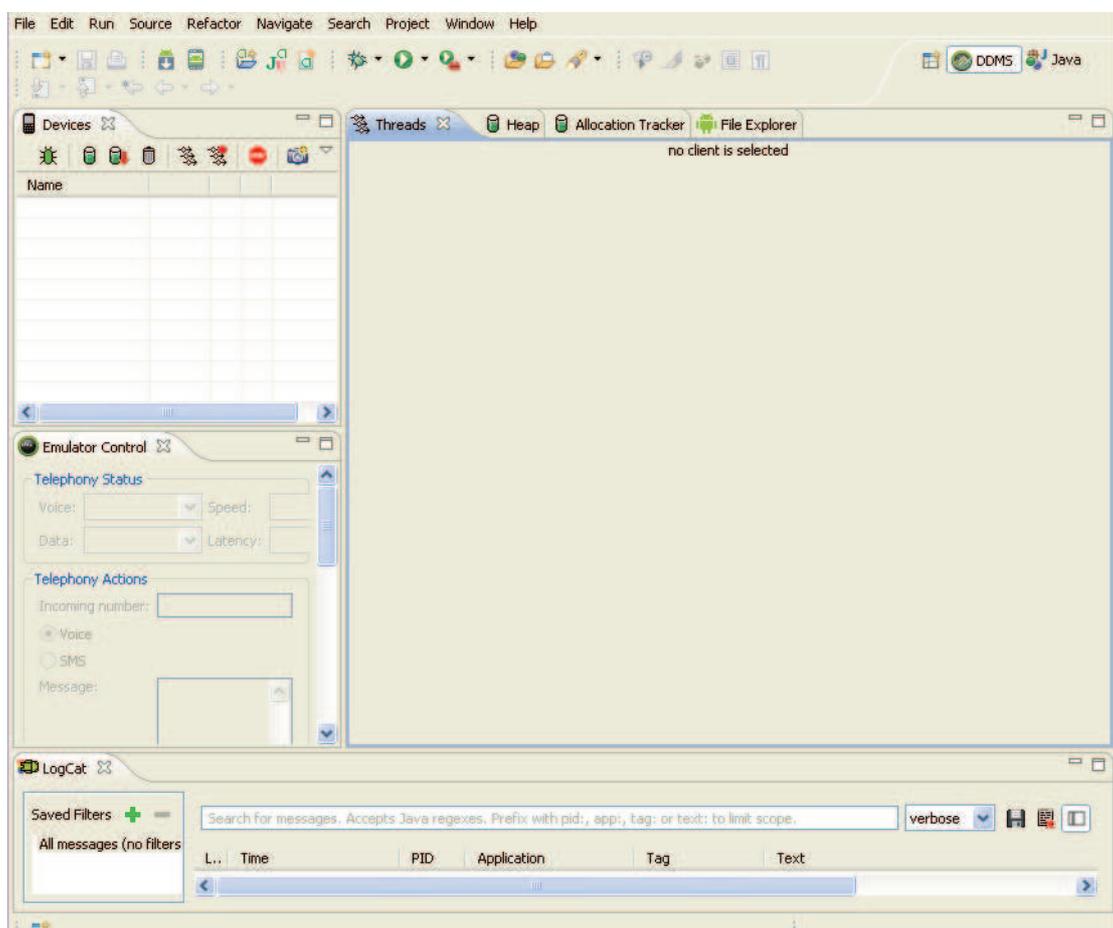


Figura 7. Perspectiva DDMS

Esta perspectiva, sirve para poder programar y realizar debugging al mismo tiempo, lo que es una forma muy efectiva de programar.

Aunque se programará con la perspectiva Java, a la hora de corregir errores se puede pasar a la perspectiva DDMS.

3. EMULADOR

Una vez tengamos el proyecto listo para ejecutar, entra en escena el emulador de Android. Éste proporciona una vista especial para comprobar si la aplicación hace lo que se desea. A continuación se muestra la vista del emulador para la versión 2.2 de Android:

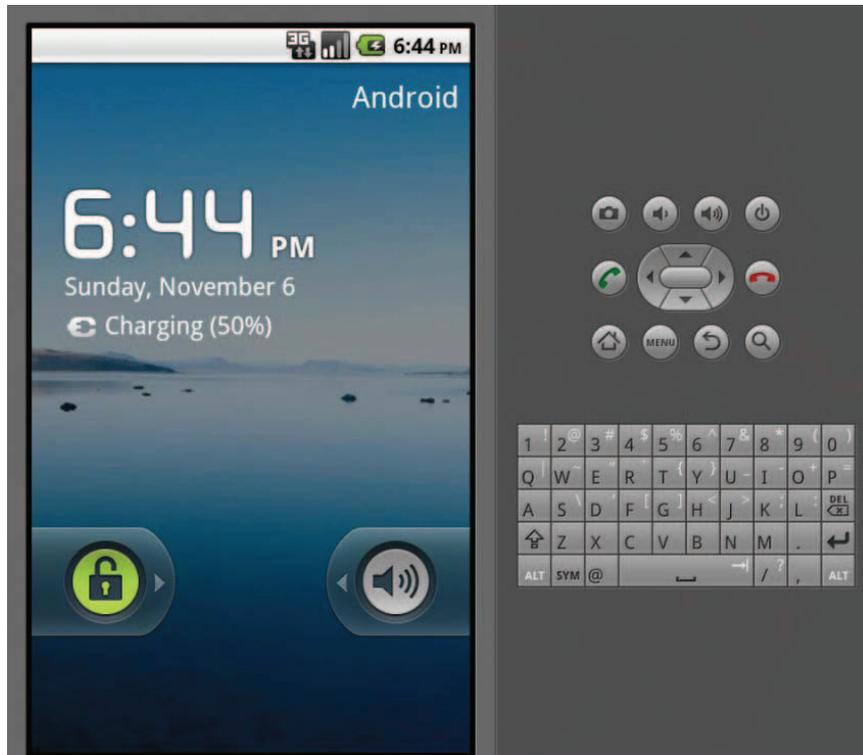


Figura 8. Emulador para Android 2.2

Lo primero que hay que hacer cuando se quiere ejecutar una aplicación, es pinchar sobre el proyecto con el botón derecho, y en “Run as” seleccionar “Android Application”, entonces se lanzará el emulador más apropiado siempre que esté creado (más adelante, se explicará cómo generar los emuladores).

No se debe parar la ejecución del emulador, dado que cada vez que se ejecuta el mismo, necesita de muchos recursos del computador, por lo que tarda bastante en lanzarse, y realmente no es necesario cerrarlo, puesto que cada vez que se lleva a cabo una ejecución del proyecto, la aplicación se reinstala en el emulador.

UN EJEMPLO: “HOLA ANDROID”

Vamos a crear nuestro primer proyecto en Android, pero antes veamos de qué se compone cada uno. Al generar un nuevo proyecto de Android, dado que estamos utilizando el entorno Eclipse, éste va a generar automáticamente la distribución de carpetas que contendrá la aplicación, la cuál será común a todos los proyectos Android.

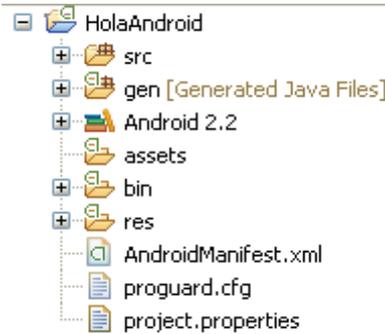


Figura 9. Sistema de carpetas de un proyecto

Veamos el significado de cada carpeta por separado:

- **Carpeta src:**

Recoge la totalidad del código fuente (Java) de la aplicación. En el ejemplo que vamos a llevar a cabo, Eclipse generará automáticamente el código base de la ventana principal (Activity).

- **Carpeta res:**

Contiene los recursos necesarios para generar una aplicación Android:

- res/drawable/: Guarda las imágenes y se divide en: drawable-ldpi, drawable-mdpi y drawable-hdpi, que

 - dependarán de la resolución del dispositivo.

- res/raw/: Contiene archivos de propósito general, en otro formato que no es XML.

- res/layout/: Incluye los archivos que definen el diseño de la interfaz gráfica, siempre en XML.

- res/values/: Guarda los datos y tipos que utiliza la aplicación, tales como colores, cadenas de texto, estilos, dimensiones...

- **Carpeta gen:**

Ésta carpeta guarda un conjunto de archivos (de código Java) creados automáticamente cuando se compila el proyecto, para poder dirigir los recursos de la aplicación. El archivo R ajusta automáticamente todas las referencias a archivos y valores de la aplicación (guardados en la carpeta res).

- **Carpeta assets:**

Guarda el resto de archivos necesarios para el correcto funcionamiento de la aplicación, como los archivos de datos o de configuración. La principal diferencia entre los recursos que almacena ésta carpeta y los que guarda la carpeta “res”, es que los recursos de ésta última generan un identificador por recurso, identificador que se encargará de gestionar el fichero R y sólo se podrá acceder a ellos a través de determinados métodos de acceso, mientras que los recursos almacenados en la carpeta “assets” no generan identificador alguno y se accederá a ellos a través de su ruta, como se hace con cualquier otro fichero.

- **Archivo AndroidManifest.xml:**

Éste archivo es uno de los más importantes de cualquier aplicación Android. Se genera automáticamente al crear el proyecto, y en él se encuentra definida la configuración del proyecto en XML (Actividades, Intents, los permisos de la aplicación, bibliotecas, etc.). Por ejemplo, el proyecto que vamos a generar (“Hola Android”), contiene un AndroidManifest.xml como el siguiente:

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="org.example.hello"
    android:versionCode="1"
    android:versionName="1.0" >

    <uses-sdk android:minSdkVersion="8" />

    <application
        android:icon="@drawable/ic_launcher"
        android:label="@string/app_name" >
        <activity
            android:label="@string/app_name"
            android:name=".Hola" >
            <intent-filter >
                <action android:name="android.intent.action.MAIN" />

                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
    </application>

</manifest>
```

Otra vista diferente del manifiesto es la siguiente:

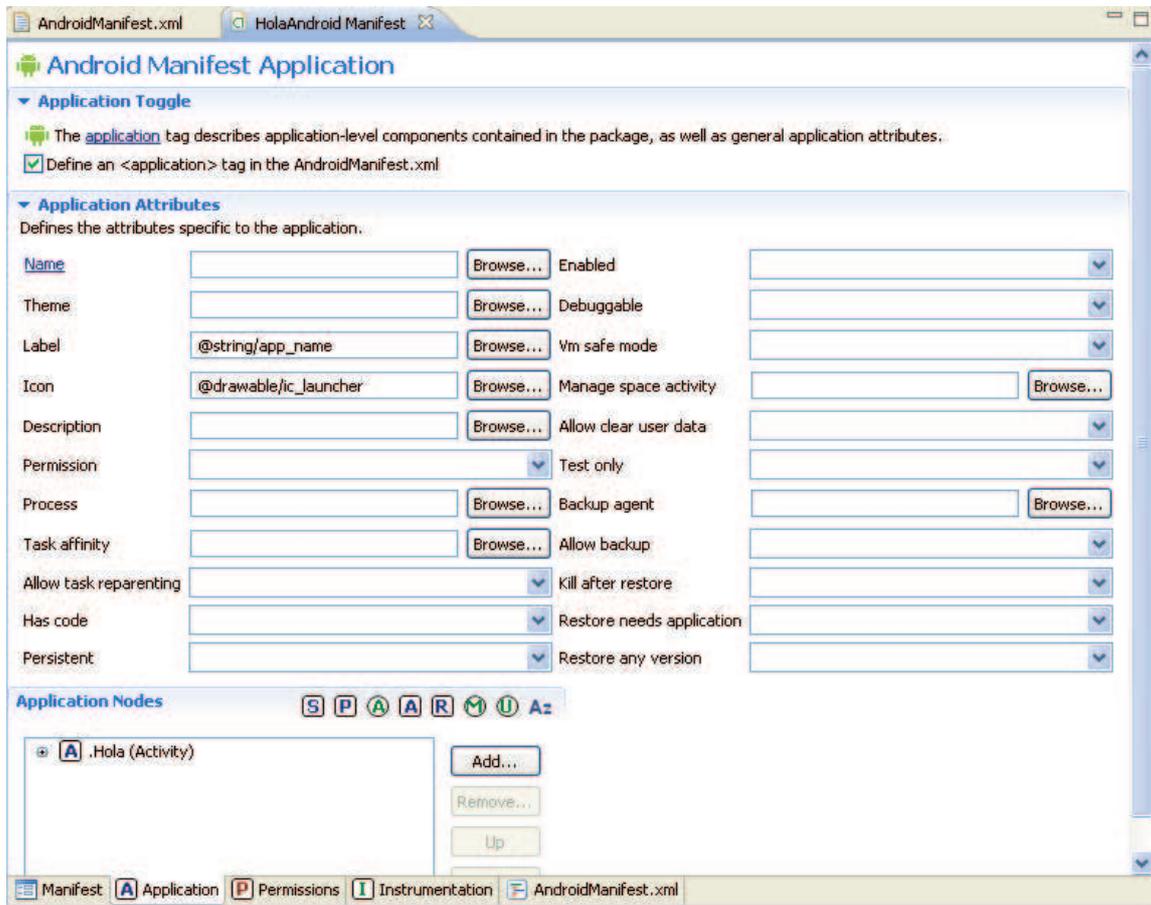


Figura 10. Editor Android Manifest

Y por fin, una vez explicadas cada una de las partes que componen el proyecto, vamos a crear nuestro primer proyecto con Android. Primero pinchar en “File”->“New”->“Other”->“Android Project”, saldrá la pantalla que se muestra a continuación (“Create Android Project”). Simplemente en “Project Name” poner: HelloAndroid y pulsar “Next”. En la siguiente pantalla, “Select Build Target”, seleccionar la versión de Android sobre la que construir el proyecto. Vamos a seleccionar Android 2.2, para que nuestra aplicación pueda correr en cualquier terminal que tenga ésta versión o una posterior.

En la última pantalla antes de dar por concluida la configuración del nuevo proyecto, “Application Info”, completar los siguientes campos:

- “Application Name”: Hello, Android
- “Package Name”: org.example.hello
- “Create Activity”: Hello

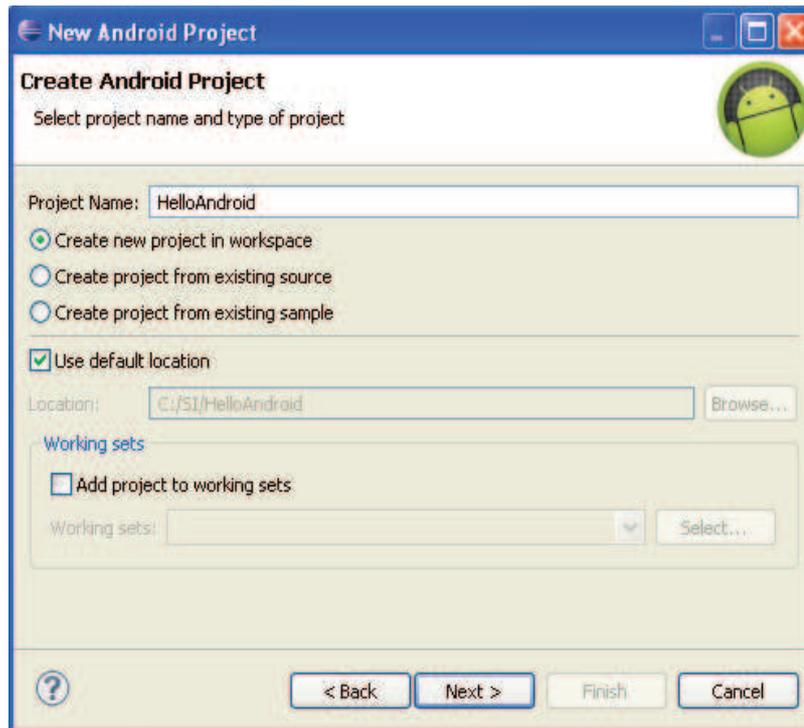


Figura 11. Create Android Project

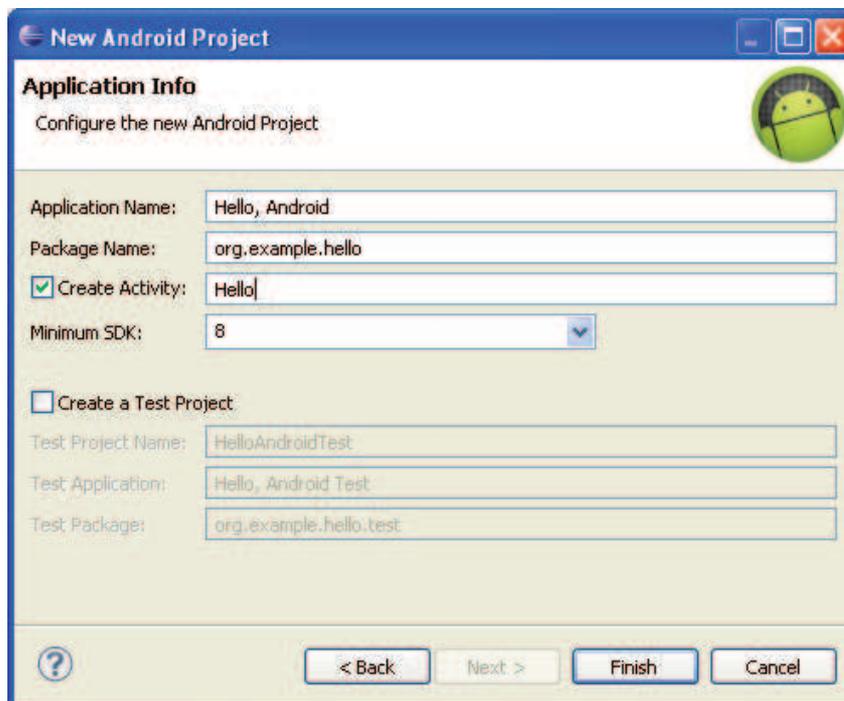


Figura 12. Application Info

Para ejecutar nuestra aplicación, primero debemos tener creado un emulador de nuestra versión de Android. Para ello, pinchar en el símbolo que abre el “Android Virtual Device Manager”, y pulsar en “New”:

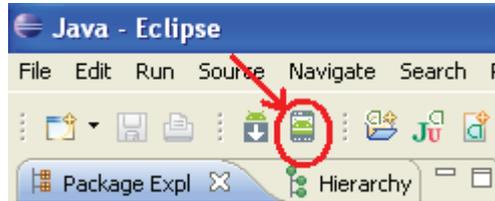


Figura 13. Símbolo Android Virtual Device Manager

En la siguiente pantalla rellenar los siguientes campos:

- .- “Name”: em2.2
- .- “Target”: Android 2.2 – API Level 8
- .- “Size”: 128 MiB
- .- “Built-in”: Default(WVGA800)
- .- Si se quiere añadir funcionalidades Hardware, en “Hardware” pulsar “New”, y seleccionar la opción/es deseada/s. En el ejemplo se ha añadido la funcionalidad “Camera support”
- .- Por último, pulsar “Create AVD”.

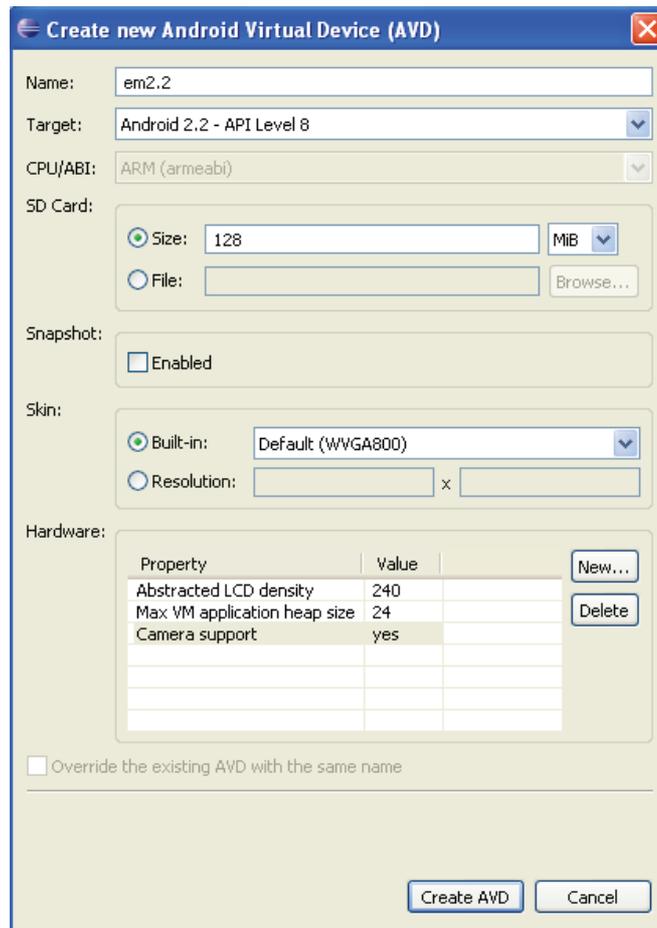


Figura 14. Create AVD

Ahora ya podemos ejecutar nuestra aplicación. Pinchar con el botón derecho del ratón sobre el proyecto, y en “Run As”, seleccionar “Android Application”. Se lanzará el emulador (hay que tener paciencia, pues debido a que consume muchos recursos, tardará un rato), y pasado un tiempo, se mostrará nuestro primer programa en Android:

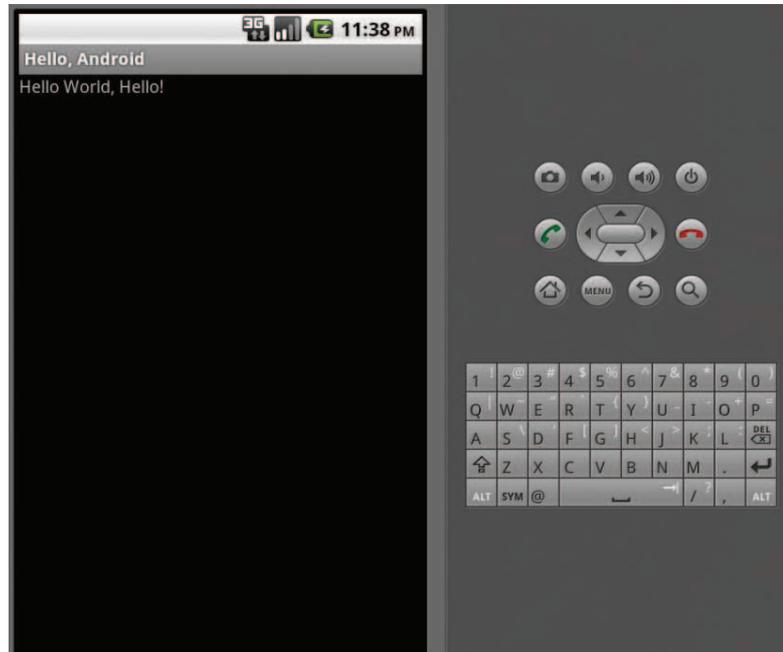


Figura 15. Simulación Hello Android

2. CONCEPTOS BÁSICOS

Daniel Sanz – Mariam Saucedo – Pilar Torralbo

COMPONENTES DE UNA APLICACION

Para diseñar una aplicación en Android, es necesario tener claros los elementos que la componen y la funcionalidad de cada uno de ellos. Ya hemos visto el ejemplo del “Hola Android”, por lo que podemos intuir algunos de ellos. Uno de los aspectos más importantes a tener en cuenta es su funcionamiento. Android trabaja en Linux, y cada aplicación utiliza un proceso propio. Se distinguen por el ID, un identificador para que solo ella tenga acceso a sus archivos. Los dispositivos tienen un único foco, la ejecución principal, que es la aplicación que está visible en la pantalla, pero puede tener varias aplicaciones en un segundo plano, cada una con su propia pila de tareas. La pila de tareas es la secuencia de ejecución de procesos en Android. Se componen de actividades que se van apilando según son invocadas, y solo pueden terminarse cuando las tareas que tiene encima están terminadas, o cuando el sistema las destruye porque necesita memoria, por lo que tienen que estar preparadas para terminar en cualquier momento. El sistema siempre eliminará la actividad que lleve más tiempo parada. En caso de que el sistema necesite mucha memoria, si la aplicación no está en el foco, puede ser eliminada por completo a excepción de su actividad principal.



Figura 1. Pila de actividades Android

Una de las características principales del diseño en Android es la reutilización de componentes entre las aplicaciones, es decir, dos aplicaciones diferentes pueden utilizar una misma componente, aunque esté en otra aplicación para así, evitar la repetición innecesaria de código, y la consiguiente ocupación de espacio. Los componentes son los elementos básicos con los que se construyen el proyecto. Hay cuatro tipos, pero las aplicaciones se componen principalmente de actividades. Habrá tantas actividades como ventanas distintas tenga la aplicación. Sin embargo, por si solos, los componentes no pueden hacer funcionar una aplicación. Para ello están los *intents*.

Todos ellos deben declararse en el `AndroidManifest.xml` (junto con otros elementos que se mostrarán después) con el mismo nombre que lleve la clase asociada. Por ejemplo, la clase `MainActivity`, será definida en el `AndroidManifest` con el mismo nombre.

ACTIVIDADES

Una actividad (o `Activity`) es la componente principal encargada de mostrar al usuario la interfaz gráfica, es decir, una actividad sería el equivalente a una ventana, y es el medio de comunicación entre la aplicación y el usuario. Se define una actividad por cada interfaz del proyecto. Los elementos que se muestran en ella deben ser definidos en el fichero xml que llevan asociado (que se guarda en `./res/layout`) para poder ser tratados en la clase `NameActivity.class`, que hereda de la clase `Activity`.

Dentro del fichero xml asociado a la actividad, se definen los elementos tales como ubicación de los elementos en la pantalla (layouts), botones, textos, checkbox, etc., como se verá en capítulos posteriores. Las actividades tienen un ciclo de vida, es decir, pasan por diferentes estados desde que se inician hasta que se destruyen. Sus 3 posibles estados son:

- **Activo:** ocurre cuando la actividad está en ejecución, es decir, es la tarea principal
- **Pausado:** la actividad se encuentra semi-suspendida, es decir, aun se está ejecutando y es visible, pero no es la tarea principal. Se debe guardar la información en este estado para prevenir una posible pérdida de datos en caso de que el sistema decida prescindir de ella para liberar memoria.
- **Parado:** la actividad está detenida, no es visible al usuario y el sistema puede liberar memoria. En caso de necesitarla de nuevo, será reiniciada desde el principio.

Una vez definido el ciclo de vida, hay que tener en cuenta qué métodos son importantes en cada uno de ellos. Aquí están los métodos más importantes de una actividad:

- **OnCreate (Bundle savedInstanceState):** es el método que crea la actividad. Recibe un parámetro de tipo `Bundle`, que contiene el estado anterior de la actividad, para preservar la información que hubiera, en caso de que hubiera sido suspendida, aunque también puede iniciarse con un `null` si la información anterior no es necesaria o no existe.
- **OnRestart():** reinicia una actividad tras haber sido parada (si continúa en la pila de tareas). Se inicia desde cero.
- **Onstart():** inmediatamente después de `onCreate(Bundle savedInstanceState)`, o de `onRestart()` según corresponda. Muestra al usuario la actividad. Si ésta va a estar en un primer plano, el siguiente método debe ser `onResume()`. Si por el contrario se desarrolla por debajo, el método siguiente será `onStop()`. Es recomendable llamar al método `onRestoreInstanceState()` para asegurar la información
- **OnResume():** establece el inicio de la interactividad entre el usuario y la aplicación. Solo se ejecuta cuando la actividad está en primer plano. Si necesita información previa, el método `onRestoreInstanceState()` aportará la situación en que estaba la

actividad al llamar al `onResume()`. También puede guardar el estado con `onSaveInstanceState()`.

- `OnPause()`: se ejecuta cuando una actividad va a dejar de estar en primer plano, para dar paso a otra. Guarda la información, para poder restaurar cuando vuelva a estar activa en el método `onSaveInstanceState()`. Si la actividad vuelve a primer plano, el siguiente método será `onResume()`. En caso contrario, será `onStop()`.
- `OnStop()`: la actividad pasa a un segundo plano por un largo período. Como ya se ha dicho, el sistema puede liberar el espacio que ocupa, en caso de necesidad, o si la actividad lleva parada mucho tiempo.
- `OnDestroy()`: es el método final de la vida de una actividad. Se llama cuando ésta ya no es necesaria, o cuando se ha llamado al método `finish()`.

Además de estos métodos, cabe destacar dos más, que son de vital importancia:

- `OnSaveInstanceState()`: guarda el estado de una actividad. Es muy útil cuando se va a pausar una actividad para abrir otra.
- `OnRestoreInstanceState()`: restaura los datos guardados en `onSaveInstanceState()` al reiniciar una actividad.

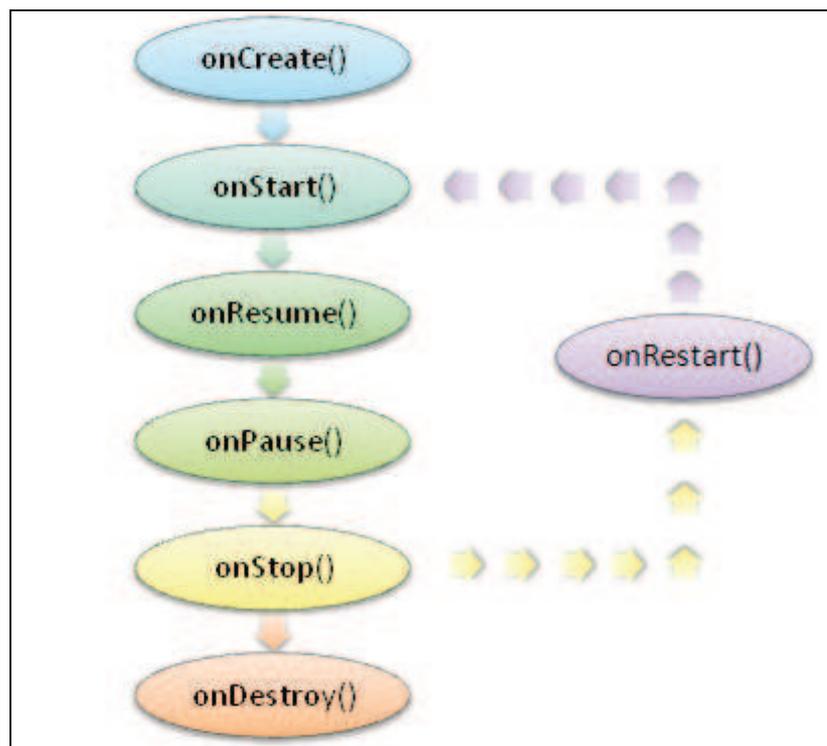


Figura 2. Ciclo de vida de una actividad

SERVICIOS

Los servicios (o service) son tareas no visibles que se ejecutan siempre por debajo, incluso cuando la actividad asociada no se encuentra en primer plano. Tiene un hilo propio (aunque no se pueden ejecutar solo), lo que permite llevar a cabo cualquier tarea, por pesada que sea. No necesita interfaz, a no ser que se pida explícitamente, en cuyo caso la clase Service la exportaría.

El ciclo de vida de un servicio se inicia con el método `onCreate(Bundle)`, y se libera con el método `onDestroy()`. Sin embargo, el desarrollo puede llevarse a cabo de dos maneras, dependiendo de cómo se lance:

- Si se llama al método `startService()`, esto implicará que el servicio ejecutará todo su ciclo vital. El siguiente método tras `onCreate(Bundle)` será `onStartComand(Intent, int, int)`. Para terminar el servicio externamente, se usa `stopService()`, e internamente, `stopSelf()` ó `stopSelfResult()`, ambos de la clase Service.
- En otro caso, si el servicio se llama con `bindService()`, el usuario podrá interactuar mediante la interfaz que exporta el servicio, y tras `onCreate(Bundle)` se ejecutará el método `onBind(Intent)`. En este caso, el servicio se termina llamando al método `onUnbind(Intent)`. También es posible reiniciarlo con el método `onRebind(Intent)`.

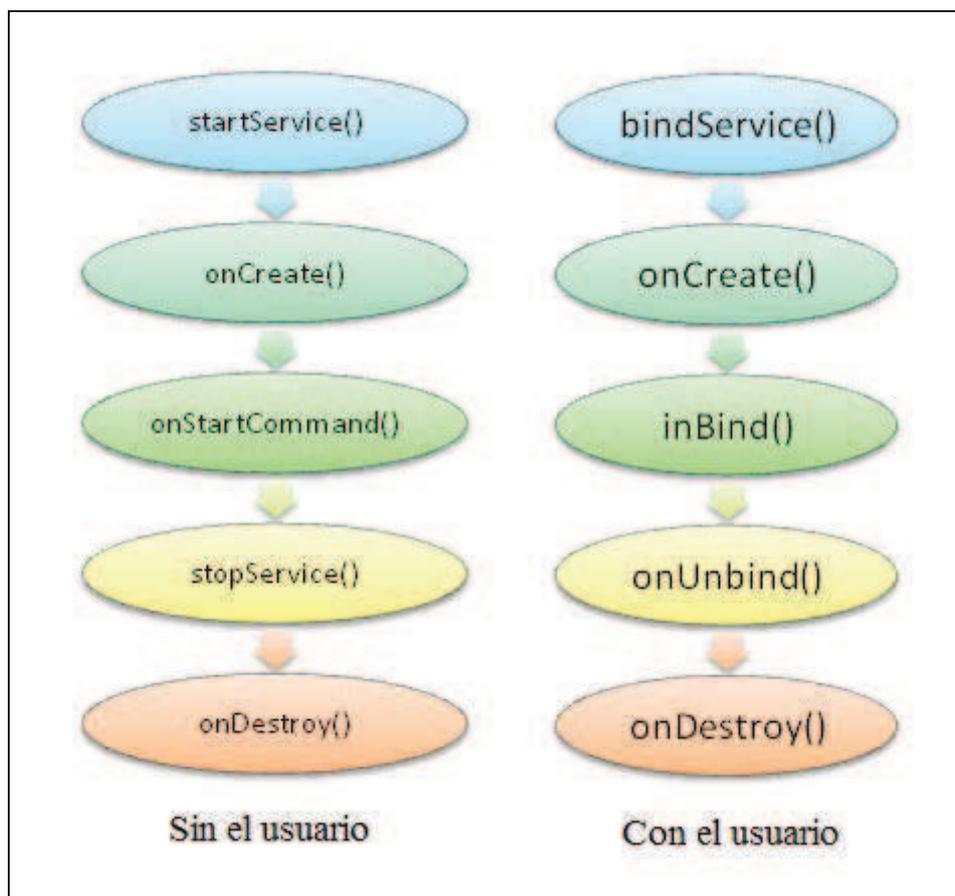


Figura 3. Ciclo de vida de un servicio

Receptores de Mensajes de Distribución

También llamados broadcast receiver o notificaciones, son los encargados de reaccionar ante los eventos ocurridos en el dispositivo, ya sean generados por el sistema o por una aplicación externa. No tienen interfaz, pero pueden lanzar una actividad por medio de un evento. La clase que defina estos componentes heredará de la clase `BroadcastReceiver`. Su ciclo de vida es muy corto, ya que solo están activos mientras se ejecuta el método `onReceive (Context, Intent)`, que es equivalente al `onCreate(Bundle)` de otros componentes. El objeto `Context` nos pasa el estado actual, y el `Intent`, nos permitirá lanzar el evento.

Proveedores de contenidos

Estos proveedores en inglés llamados content provider, se encargan de que la aplicación pueda acceder a la información que necesita, siempre que se haya declarado el correspondiente provider en el `AndroidManifest`, compartiendo información sin revelar estructura u orden interno. Implementan una interfaz, pero se comunica con ella a través de la clase `ContentResolver`. Cada vez que se usa un `ContentResolver`, se activa un `ContentProvider`. Para obtener los datos necesarios, es necesario conocer la URI (identificador) del dato, los campos que tiene, y los tipos de esos campos. Con esto ya podemos llamar al método `ContentResolver.query()`.

Intents

Los intents son el medio de activación de los componentes (excepto los content provider, que se activan usando `ContentResolver`). Contiene los datos que describen la operación que desarrollará el componente a quien va dirigido. Se declaran en el `AndroidManifest` con la etiqueta `<Intent>`. Pueden ser explícitos o implícitos. Los implícitos no especifican el componente al que va destinado, mientras que el explícito, sí. Según el componente, los intents se tratan de diferentes maneras:

- **Activity:** los intents se lanzan desde el método `startActivity(Intent)` ó `startActivityForResult(Intent)`. La información se extrae con el método `getIntent()`.

Los intents tienen definidas algunas acciones para las activity, es decir, informan de la acción a realizar. Entre ellas, por ejemplo se encuentra `ACTION_CALL` que inicia una llamada.

- **Service:** para este tipo de componentes, los intents se pasan a los métodos `startService(Intent)` o `bindService(Intent)` dependiendo del tipo de ciclo que escojamos. La información será extraída por el método `getIntent()` en el primer caso y `onBind()` en el segundo.

Otra posibilidad es que el servicio sea lanzado por un intent, si aun no esta en funcionamiento.

- **Broadcast Receiver:** en este caso, el intent será enviado a todos los métodos que pueden recibir el intent : `sendBroadcast()`, `sendOrderedBroadcast(Intent, String, BroadcastReceiver, android.os.Handler, int, String, Bundle)`,

sendStickyBroadcast()..., que lo analizarán en su método onReceive(Context, Intent).

También tienen acciones definidas para este componente, aunque en este caso lo que hacen es informar de que ha ocurrido el evento. Por ejemplo tenemos ACTION_BATTERY_LOW, que informa de que la batería esta baja, o ACTION_SCREEN_ON, para cuando la pantalla se ilumina.

Intent-filters

Utilizados únicamente por los intents implícitos, los intent-filters definen (y delimitan) qué tipos de intent puede lanzar la actividad, o qué tipos de intent puede recibir un broadcast. Por ejemplo, para un intent que no especifica a que actividad va dirigido, se consulta el intent filter de una de ellas, y si lo satisface, el intent usará lanzará esa actividad. Se definen en el AndroidManifest con la etiqueta <intent-filter>. La información que pasan los intents debe estar contenida en la definición del intent filter para que la componente pueda ser activada (o pueda recibirlo en el caso del broadcast). Esta información se compone de tres campos:

- Action: string que informa del tipo de acción llevada a cabo. Las acciones pueden ser dadas por la clase Intent, por una API de Android o definidas por el diseñador.
- Data: informa del identificador (URI) del dato que se asocia a la acción y del tipo de ese dato. Es importante la coherencia ya que si la acción requiere un dato de tipo texto, un intent con un dato de tipo imagen no podría ser lanzado.
- Category: string que contiene información adicional sobre el tipo de componente al que va dirigido el intent. La lista de categorías esta incluida en la clase Intent

AndroidManifest

Como ya se introdujo en el tema anterior, este fichero es un documento xml en el que se declaran los elementos de la aplicación, así como sus restricciones, permisos, procesos, acceso a datos e interacciones con elementos de otras aplicaciones. Cada elemento se declara con una etiqueta única. No debe confundirse este documento con el xml asociado a cada actividad. Los elementos gráficos y distribución de la pantalla serán definidos para cada actividad dentro de su xml, pero no en el AndroidManifest. Al implementar el AndroidManifest se deben seguir unas pautas para hacer más comprensible el documento:



```
1<?xml version="1.0" encoding="utf-8"?>
2<manifest xmlns:android="http://schemas.android.com/apk/res/android"
3    package="com.rec.app"
4    android:versionCode="1"
5    android:versionName="1.0" >
6
7    <uses-sdk android:minSdkVersion="8" />
8
```

Figura 4. Código generado automáticamente al crear el AndroidManifest

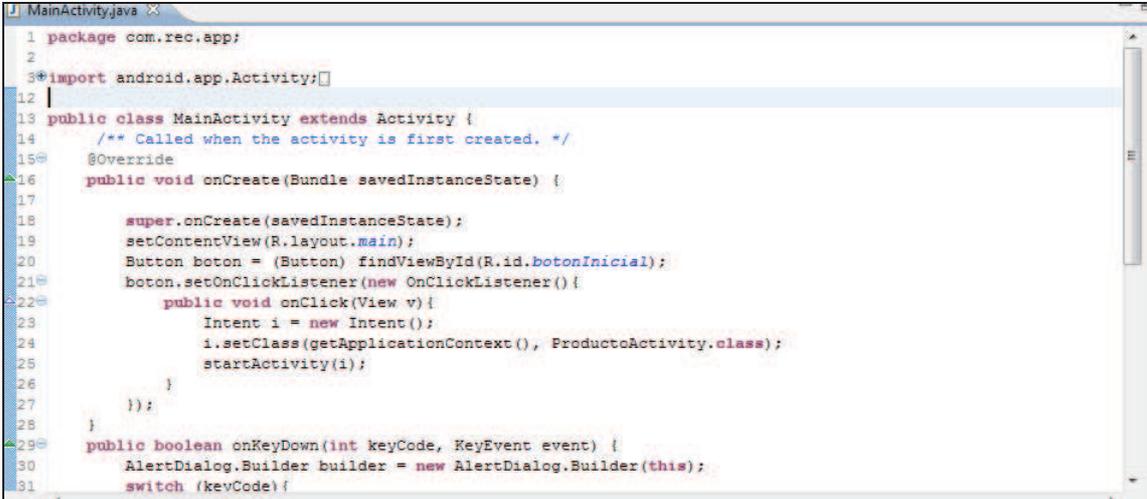
Este código es generado por el SDK a partir de la información que se ha proporcionado al crear el proyecto. Se declara el manifiesto con la etiqueta <manifest > y dentro se incluye el paquete en que se encuentra la aplicación y la versión del código. También incluye la versión del sdk que usa

(con la etiqueta <uses-sdk>). A continuación, el usuario definirá la aplicación, incluyendo todos sus componentes en la etiqueta <application>. La declaración de componentes puede ser desordenada, pero para un mejor manejo de este fichero, se recomienda seguir algún tipo de orden. Las actividades se declaran con la etiqueta <activity>. En ellas, lo primero es añadir el nombre de la actividad (android:name), que coincidirá con el de la clase en que se define el comportamiento. Además se pueden añadir imágenes, así como cambiar los atributos de que se dispone. A continuación, se declararían los intent filters asociados a la actividad, en caso de que los haya.

Los service se declaran con la etiqueta <Service> y aunque tienen menos atributos que las actividades, lo principal es darles un nombre y especificar si el sistema puede o no utilizarlo mediante el atributo enabled (android:enabled). Después irían los intent filters. Los broadcast receiver utilizan <receiver> y al igual que service, necesita los atributos name y enabled, así como intent filter en caso de necesitarlos. Todos los componentes anteriores declaran del mismo modo sus intent filters. Los content provider utilizan la etiqueta <provider> y son los únicos componentes en los que no se declaran intent filters, ya que no son necesarios. De nuevo el único atributo necesario es el nombre.

Un ejemplo práctico

A continuación se describe un ejemplo en Android aplicando los recursos aprendidos anteriormente. Al comienzo de un proyecto, tras crearlo en eclipse, lo único que tenemos es la MainActivity, o clase principal. En la siguiente imagen se muestra un ejemplo en el que se puede ver la estructura básica de una actividad. Se sobrescribe el método onCreate(Bundle) y se muestra con el método setContentView(). En la actividad se definirán las acciones propias de los elementos del xml. En este caso, tenemos un botón que, al presionar, abre una nueva actividad llamada producto.



```
1 package com.rec.app;
2
3 import android.app.Activity;
4
5
6
7
8
9
10
11
12
13 public class MainActivity extends Activity {
14     /** Called when the activity is first created. */
15     @Override
16     public void onCreate(Bundle savedInstanceState) {
17
18         super.onCreate(savedInstanceState);
19         setContentView(R.layout.main);
20         Button boton = (Button) findViewById(R.id.botonInicial);
21         boton.setOnClickListener(new OnClickListener() {
22             public void onClick(View v) {
23                 Intent i = new Intent();
24                 i.setClass(getApplicationContext(), ProductoActivity.class);
25                 startActivity(i);
26             }
27         });
28     }
29     public boolean onKeyDown(int keyCode, KeyEvent event) {
30         AlertDialog.Builder builder = new AlertDialog.Builder(this);
31         switch (keyCode) {
```

Figura 5. Ejemplo de Actividad

Es conveniente implementar la acción del botón back, pausando la actividad, sin destruirla. Además, también se puede incluir la típica pregunta “¿Está seguro de que desea salir?”:

```
MainActivity.java
27     });
28 }
29 public boolean onKeyDown(int keyCode, KeyEvent event) {
30     AlertDialog.Builder builder = new AlertDialog.Builder(this);
31     switch (keyCode){
32         case KeyEvent.KEYCODE_BACK:
33             builder.setMessage("¿Seguro que deseas salir?");
34             builder.setCancelable(false);
35             builder.setPositiveButton("Si", new DialogInterface.OnClickListener() {
36                 public void onClick(DialogInterface dialog, int id) {
37                     onPause();
38                     System.exit(RESULT_OK);
39                 }
40             });
41             builder.setNegativeButton("No", new DialogInterface.OnClickListener() {
42                 public void onClick(DialogInterface dialog, int id) {
43                     //Acciones que se ejecutan al presionar el botón No
44                 }
45             });
46             break;
47     }
```

Figura 5. Implementación del botón back

El archivo xml asociado a la MainActivity contendría el botón con el id botonInicial.

```
MainActivity.java main.xml
1 <?xml version="1.0" encoding="utf-8"?>
2 <LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
3     android:layout_width="fill_parent"
4     android:layout_height="fill_parent"
5     android:background="@drawable/prod_background"
6     android:gravity="center"
7     android:orientation="vertical" >
8     <Button
9         android:id="@+id/botonInicial"
10        android:layout_width="112dp"
11        android:layout_height="146dp"
12        android:layout_gravity="center"
13        android:background="@drawable/logo"
14        android:gravity="center" />
15
16 </LinearLayout>
```

Figura 6. Ejemplo xml de Main activity (a)

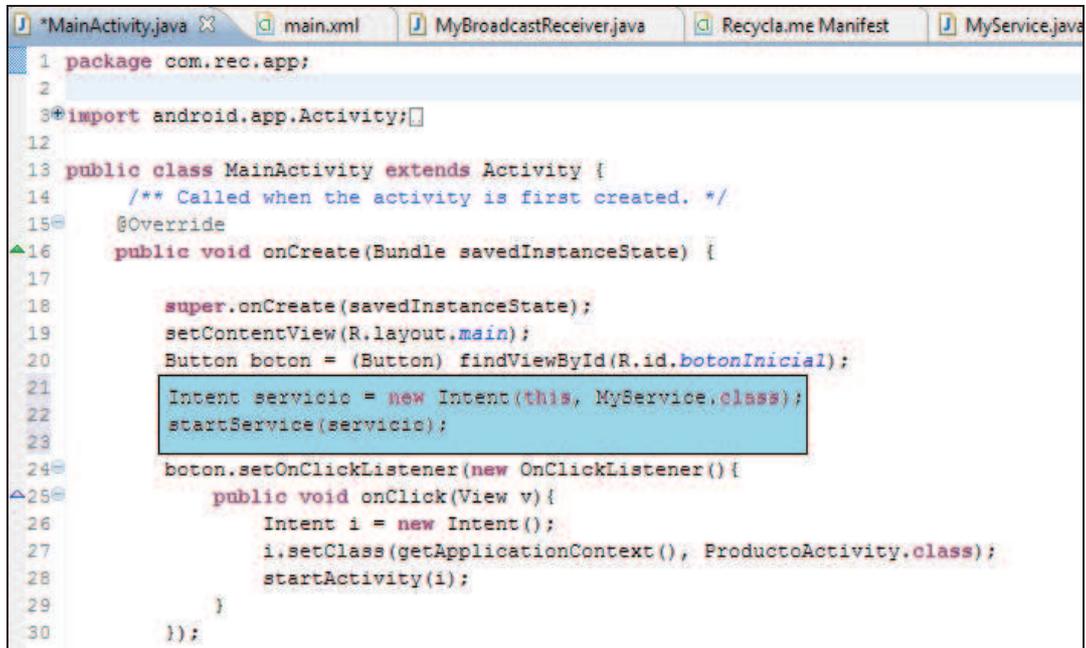
```
6
7 public class MyService extends Service {
8
9     private static Reproductor sonido;
10
11     @Override
12     public void onCreate() {
13         sonido = Reproductor.create(this, R.raw.sonido_inicio);
14     }
15
16     public int onStartCommand(Intent intent, int flags, int startId) {
17
18         sonido.start();
19         onDestroy();
20         return Service.START_NOT_STICKY;
21     }
22
23     @Override
24     public IBinder onBind(Intent arg0) {
25         // TODO Auto-generated method stub
26         return null;
27     }
28 }
```

Figura 7 Ejemplo xml de Main activity (b)

Una vez la actividad está definida, hay que tener claro que recursos le serán necesarios. En este caso, y para que sirva de ejemplo, la actividad activará un Service (debidamente añadida en el androidManifest) que no tendrá que interactuar con el usuario, por lo que no habrá que sobrescribir el método `onBind(Intent)`. Los métodos que se necesitan son `onCreate()`, `startService()`, `onStartCommand` y `onDestroy()`. En este ejemplo, el servicio reproducirá un sonido al inicio de la aplicación. De la gestión del sonido se ocupa la clase `Reproductor`.

Ejemplo de Service

Cuando se ha creado el servicio, se añade en la actividad en la que será llamado.



```
1 package com.rec.app;
2
3 import android.app.Activity;
12
13 public class MainActivity extends Activity {
14     /** Called when the activity is first created. */
15     @Override
16     public void onCreate(Bundle savedInstanceState) {
17
18         super.onCreate(savedInstanceState);
19         setContentView(R.layout.main);
20         Button boton = (Button) findViewById(R.id.botonInicial);
21         Intent servicio = new Intent(this, MyService.class);
22         startService(servicio);
23
24         boton.setOnClickListener(new OnClickListener() {
25             public void onClick(View v) {
26                 Intent i = new Intent();
27                 i.setClass(getApplicationContext(), ProductoActivity.class);
28                 startActivity(i);
29             }
30         });
31     }
32 }
```

Figura 8. Uso de Service en la clase MainActivity

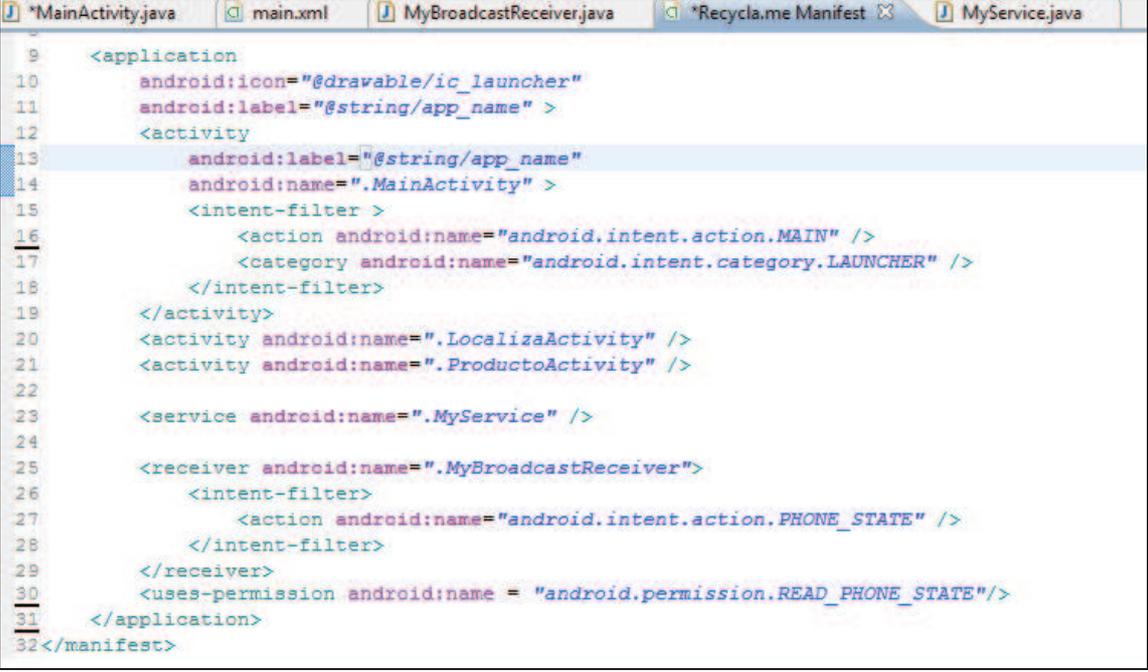
Otro componente que se puede incluir en la aplicación es el broadcastReceiver, por ejemplo para que al registrar una llamada entrante, la aplicación actúe de un modo determinado: se cierre, se suspenda o guarde la información existente.



```
1 package com.rec.app;
2 import android.content.BroadcastReceiver;
6
7
8 public class MyBroadcastReceiver extends BroadcastReceiver {
9
10
11     @Override
12     public void onReceive(Context context, Intent intent) {
13
14         String llamando = intent.getStringExtra(TelephonyManager.EXTRA_STATE_RINGING);
15         if (!llamando.equals(TelephonyManager.EXTRA_STATE_RINGING)) {
16             //Acciones llevadas a cabo por la aplicación cuando se recibe una llamada
17         }
18     }
19 }
20 }
```

Figura 9. Ejemplo de Broadcast Receiver

Se puede añadir tantos componentes como sean necesarios. También es conveniente incluir algún proveedor de contenidos, pero para ello es necesario estudiar previamente el manejo de bases de datos. No se debe olvidar incluir todos estos componentes en el AndroidManifest.



```
9 <application
10     android:icon="@drawable/ic_launcher"
11     android:label="@string/app_name" >
12     <activity
13         android:label="@string/app_name"
14         android:name=".MainActivity" >
15         <intent-filter >
16             <action android:name="android.intent.action.MAIN" />
17             <category android:name="android.intent.category.LAUNCHER" />
18         </intent-filter>
19     </activity>
20     <activity android:name=".LocalizaActivity" />
21     <activity android:name=".ProductoActivity" />
22
23     <service android:name=".MyService" />
24
25     <receiver android:name=".MyBroadcastReceiver">
26         <intent-filter>
27             <action android:name="android.intent.action.PHONE_STATE" />
28         </intent-filter>
29     </receiver>
30     <uses-permission android:name = "android.permission.READ_PHONE_STATE"/>
31 </application>
32</manifest>
```

Figura 10. Ejemplo AndroidManifest

En resumen, las aplicaciones en Android tienen diverso grado de dificultad, dependiendo de su funcionalidad, pero la estructura siempre es la misma. Uniendo estos conocimientos al uso de recursos gráficos, bases de datos, mapas, y otros elementos, las posibilidades de estas aplicaciones son bastante amplias.

3. INTERFAZ DEL USUARIO

Luis Cruz – José Rodríguez de Llera – Alvaro Zapata

BREVE INTRODUCCIÓN

La interfaz de usuario es la principal sección de interacción entre persona y dispositivo. A todas las funcionalidades disponibles se accede a través de la pantalla, que es por donde se muestra. Es muy importante conseguir que el manejo sea intuitivo y sencillo, y que el aspecto visual sea atractivo.

Para construirla, se emplean diferentes objetos que veremos a continuación, todos ellos descendientes de la clase View. Fundamentalmente hay 2: los propios objetos de tipo View, como por ejemplo botones o etiquetas, y que son la base de una subclase llamada widgets; y los de tipo ViewGroup, que es una clase que extiende a View, y que son la base de una subclase llamada layouts. La estructura de la interfaz puede resumirse en el cuadro que se muestra a continuación.

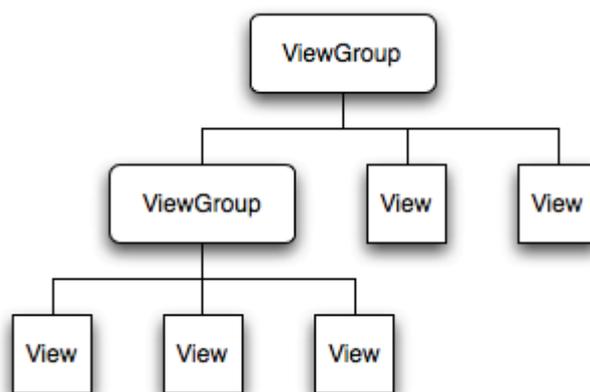


Figura 1. Estructura de la interfaz

Como podemos comprobar, los nodos ViewGroup son contenedores de elementos de tipo View e incluso de otros elementos de su mismo tipo. Los controles de tipo View (obsérvese que los nodos de este tipo son siempre hojas del árbol) son con los que el usuario interactúa.

LAYOUTS EN XML

Un layout es un recurso con el que puedes describir lo que quieres mostrar por pantalla y cómo lo quieres mostrar. La manera más común de crearlo es a través de un archivo XML (en el directorio `res/layout` del proyecto), con un formato muy similar a HTML, que sigue este patrón: `<Nombre_del_layout atributo1="valor1"... atributoN="valorN"> elementos/componentes </Nombre_del_layout>`. Una vez se ha creado el archivo XML que definirá el layout, hay que cargarlo desde el código de la aplicación, en el `onCreate()` de la actividad. Debe ser similar a esto:

```
public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.nombre_del_layout);
}
```

Todos los layouts tienen unos parámetros específicos. Los únicos que son comunes a todos son `layout_height` y `layout_width`, que sirven para especificar el valor de la altura y anchura del entorno, respectivamente. Aunque se pueden emplear valores numéricos, con “`wrap_content`”, el tamaño se ajusta a las dimensiones del contenido. Con “`fill_parent`”, será tan grande como lo permita su padre o contenedor. Estas 2 opciones son más recomendables que el ajuste manual. También se pueden establecer y consultar los márgenes, bordes y la posición del layout, entre otras opciones, mediante métodos y funciones a los que, en esta ocasión, se llaman desde el código de la aplicación.

Cada componente tiene su propia variedad de atributos en XML. El atributo “ID” se encarga de distinguirlos del resto, otorgándole un nombre único. Una vez establecido (mediante, por ejemplo, `android:id="@+id/nombre"`), para referenciarlo desde el código de la aplicación es preciso hacerlo de esta manera:

```
Tipo myTipo = (Tipo) findViewById(R.id.nombre);
```

Existen otros muchos atributos, que varían dependiendo del componente con el que estemos tratando, y con los que podemos establecer el color de fondo, tamaño, gravedad y un sinnúmero de opciones más.

FRAME LAYOUT

Es el más simple de todos los existentes. Todos los objetos que se introduzcan se situarán en la esquina superior izquierda, por lo que si hay más de uno, se ocultarán total o parcialmente entre ellos, salvo que los declaremos como transparentes. Por este motivo, su uso ideal es el de mostrar una sola imagen que complete toda la pantalla.

LINEAR LAYOUT

Se trata del Layout que viene por defecto, y uno de los más sencillos. Los objetos son estructurados horizontal o verticalmente, dependiendo del atributo “`orientation`” de su archivo

XML correspondiente, y siempre en una única fila o columna, con un comportamiento similar al de una pila. Aquí podemos ver un código de ejemplo:

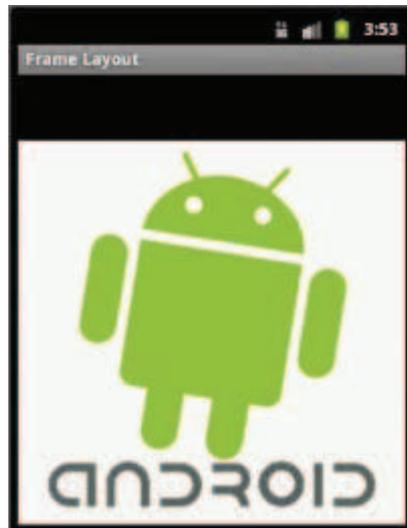


Figura 2. Frame Layout

```
<LinearLayout
  xmlns:android="http://schemas.android.com/apk/res/android"
  android:layout_width="fill_parent"
  android:layout_height="fill_parent"
  android:orientation="vertical">

  <EditText android:id="@+id/NombreEditText"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent" />

  <Button android:id="@+id/NombreButton"
    android:layout_width="wrap_content"
    android:layout_height="fill_parent" />
</LinearLayout>
```

TABLELAYOUT

Utilizando esta opción, se consigue una distribución tabular de los elementos de nuestra interfaz. El comportamiento es similar al empleado en HTML: se definen las filas, y dentro de ellas, las columnas. La tabla tendrá tantas columnas como la fila con un mayor número de celdas. En cada casilla, se podrá introducir el objeto deseado (e incluso dejarla vacía). También existe la posibilidad de combinar celdas.

Por lo general, la dimensión de cada casilla la determina el elemento contenido en ella. No obstante, este comportamiento puede variarse utilizando diversos atributos.

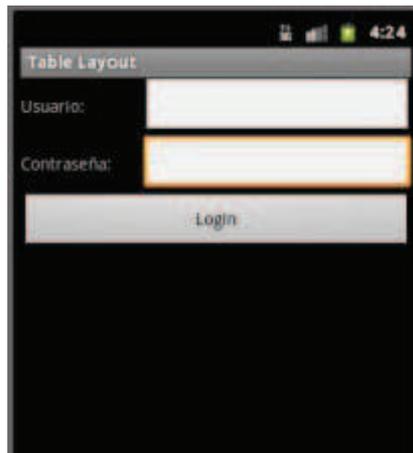


Figura 3. Table Layout

RELATIVE LAYOUT

Es la opción que ofrece más posibilidades, y por tanto, la más compleja. Básicamente, empleando este Layout podremos colocar cada elemento en el lugar que deseemos, basándonos en su posición relativa al padre (contenedor) o a otros elementos existentes, pudiendo modificar las distancias entre objetos al antojo del programador.



Figura 3. Relative Layout

OTROS

Existen, además, otros elementos con comportamiento similar a los Layouts que hemos visto hasta ahora. Algunos de ellos proporcionan estructuras invisibles para el usuario (como los aquí descritos hasta ahora) y otros sí que las muestran. Algunos ejemplos son Gallery, GridView, Spinner o ViewPager.

EVENTOS DE USUARIO

Los eventos de usuario sirven para capturar la interacción del usuario con una determinada aplicación. Existen varias maneras de realizarlo.

EVENT LISTENERS

Un Event Listener es una interfaz de la clase View a través de la cual se pueden detectar diferentes tipos de pulsación del usuario sobre el dispositivo. Los métodos más comunes que incluye esta interfaz son:

-onClick(): este método es llamado cuando el usuario hace una pulsación simple, ya sea por contacto, con teclas de navegación o de cualquier manera posible, sobre un determinado elemento de la interfaz. Una posible implementación, sobre un botón:

```
private OnClickListener nombreEvento = new OnClickListener() {
    public void onClick(View v) {
        // hacer lo que se desee
    }
};
```

```
Button boton = (Button)findViewById(R.id.corky);
boton.setOnClickListener(nombreEvento);
```

-onKey(): este método es llamado si el usuario presiona determinada tecla o botón de su dispositivo mientras el cursor está situado sobre el elemento deseado.

-onTouch(): este método es llamado cuando el usuario toca la pantalla y realiza una presión, se despega o se mueve por ella.

Para implementar los métodos aquí vistos y otros existentes, es preciso implementarlos en la Activity correspondiente o definirlos como una nueva clase anónima. Tras esto, deberemos pasar una instancia de la implementación a su respectivo método, con, por ejemplo, `setOnClickListener()` de `OnClickListener`.

EVENT HANDLERS

Si se está creando un componente de la clase View sobre cualquier tipo de Layout, se pueden definir varios métodos por defecto, denominados Event Handlers, que son llamados cuando se produce un evento de cualquier tipo, como puede ser una pulsación sobre la pantalla.

TOUCH MODE

Determinados dispositivos pueden ser controlados a través de un cursor con sus respectivas teclas de movimiento, tocándoles la pantalla, o ambas. En el primer y último caso, es preciso contar con un selector de componente cuya función sea advertir sobre el elemento con el que se está tratando en cada momento, dependiendo del movimiento y situación del cursor. Sin embargo, en el momento que el usuario “toca” la pantalla, no es necesario resaltar el elemento con el que se va a trabajar. Por ello, en caso de tratar con un dispositivo que permita ambos tipos de navegación a través de su interfaz, existirá un modo llamado “touch mode”, que determinará si es necesario resaltar el componente con el que se va a trabajar o no.

HANDLING FOCUS

Por lo comentado en el punto anterior, el sistema debe encargarse del selector de componente, manejando una rutina, en respuesta a la entrada dada por el usuario. Esto incluye el dónde situar el foco si un elemento es suprimido u ocultado, basándose en un algoritmo que selecciona al componente vecino más cercano.

MENÚS Y BARRAS DE ACCIONES

Los menús son la forma más habitual de proporcionar al usuario una serie de acciones a realizar, ya sea sobre una aplicación o sobre las opciones del propio dispositivo. Para crearlo, la mejor opción es definirlo en un archivo XML que irá contenido en el directorio `res/menú` del proyecto. Los elementos que lo componen son `<menú>`, que es el contenedor principal del resto de elementos; `<ítem>`, que representa cada una de las opciones que ofrece el menú, y `<group>`, que posibilita agrupar los “ítems” del menú a gusto del usuario, y que puede ser visible o no. Por ejemplo:

```
<?xml version="1.0" encoding="utf-8"?>
<menu xmlns:android="http://schemas.android.com/apk/res/android">
  <item android:id="@+id/Opcion1" android:title="Opción1"
        android:icon="@drawable/miIcono1"></item>
  <item android:id="@+id/Opcion2" android:title="Opción2"
        android:icon="@drawable/miIcono2"></item>
</menu>
```

Hay 3 tipos de menús de aplicación:

-**Menú principal:** es la colección primaria de una actividad, que aparece cuando el usuario acciona el botón “Menú” del dispositivo. Una forma de crearlo es esta:

```
public boolean onCreateOptionsMenu(Menu menu) {  
    MenuInflater inflater = getMenuInflater();  
    inflater.inflate(R.menu.miMenu, menu);  
    return true;  
}
```

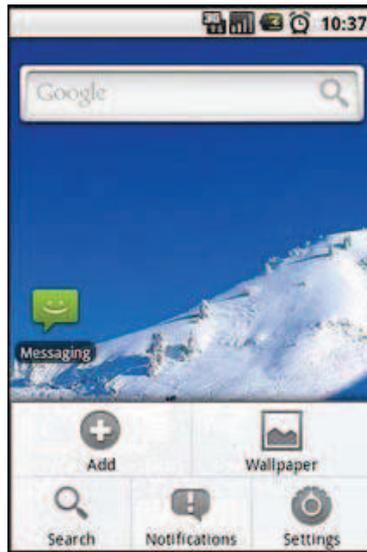


Figura 4. Menú principal

-**Menú contextual:** es una “lista” que surge en la pantalla que aparece cuando el usuario mantiene presionado un elemento determinado. Se implementa como el caso anterior, pero añadiendo un nuevo elemento de tipo <menu> al mismo nivel que los elementos <ítem>.

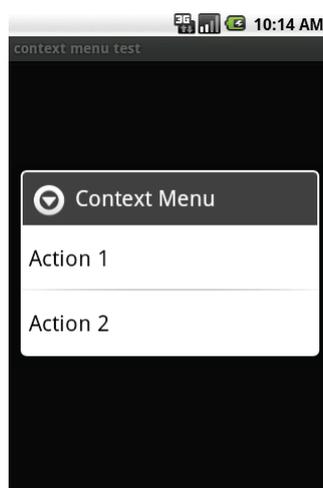


Figura 5. Menú contextual

-Submenús: es una lista de opciones que surge cuando el usuario acciona un elemento del menú, que contiene otro menú anidado.

Las barras de acciones son barras de título que, además de mostrar algún tipo de información, pueden ejecutar acciones. El aspecto de ésta es acorde con el de la aplicación a la que pertenece. Es importante saber que sólo están disponibles para versiones de Android a partir de la 3.0 (versión 11 del sdk).

DIÁLOGOS Y NOTIFICACIONES

Los diálogos son avisos o comprobaciones que surgen de una determinada aplicación, en forma de ventana. La principal diferencia entre ellos y las notificaciones es que las segundas son meramente informativas, no se ejecutan en primer plano, y no requieren interacción directa con el usuario, mientras que los primeros sí se realizan en primer plano y pueden necesitar esa interacción para llevar a cabo una tarea, aunque se puede decir que un diálogo es un tipo de notificación. Un diálogo se crea con la instrucción “onCreateDialog(numero)” como parte de una actividad, y también se muestra como tal, con “showDialog(numero)”, donde “numero”, en ambos casos, representa un identificador único de cada dialogo. Para rechazarlos, se puede conseguir con dismiss() o dismissDialog(int). Si se desea modificar el texto a mostrar o cualquier otro atributo, existe una colección de métodos, variable en función del tipo concreto del diálogo, que facilitan la labor. Podemos enumerar 3 tipos de diálogos:

-AlertDialog: Se recomienda usarlo para mostrar un título, un mensaje de texto, uno o varios botones, o una lista de elementos para seleccionar.

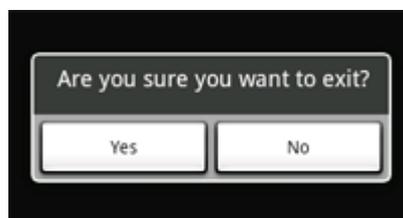


Figura 6. Diálogo de Alerta

En un diálogo con botones, se añaden también a través de métodos específicos del tipo concreto, y es preciso incorporar un evento que detecte su pulsación. Como ejemplo, el código de la última imagen mostrada en este manual:

```
AlertDialog.Builder builder = new AlertDialog.Builder(this);
builder.setMessage("Are you sure you want to exit?")
    .setCancelable(false)
    .setPositiveButton("Yes", new DialogInterface.OnClickListener() {
        public void onClick(DialogInterface dialog, int id) {
            MyActivity.this.finish();
        }
    })
    .setNegativeButton("No", new DialogInterface.OnClickListener() {
        public void onClick(DialogInterface dialog, int id) {
            dialog.cancel();
        }
    });
AlertDialog alert = builder.create();
```

En caso de que se desee incorporar una lista de elementos, es preciso crear un array de strings, donde cada elemento corresponderá a cada opción que se mostrará por pantalla, y posteriormente añadirla al diálogo de manera similar a como se ha visto para los botones.

-ProgressDialog: Su función es indicar que una tarea está siendo llevada a cabo. Si se conoce el esfuerzo que va a requerir, se mostrará una barra de progreso que suministre la información sobre lo que se lleva completado y lo que resta; y si no se conoce, se mostrará un haz que describirá un movimiento circular. Este tipo de diálogo puede incorporar también botones, cuya función puede ser cancelar el progreso. Aquí, un ejemplo de cada (el código y la imagen no están relacionados):

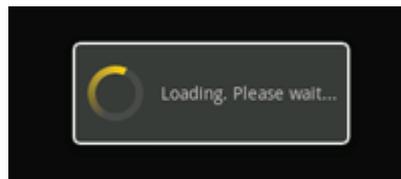


Figura 7. Diálogo de Progreso

```
ProgressDialog dialogo = new ProgressDialog(contexto);  
dialogo.setProgressStyle(ProgressDialog.STYLE_HORIZONTAL);  
dialogo.setMessage("Cargando...");
```

-Diálogo personalizado: Android da la opción de crear un diálogo con los elementos/aspecto que el usuario desee. Para ello, es preciso crearse un layout en XML, guardarlo como “custom_dialog.xml”, y añadirlo como vista al crear el diálogo. Además, se pueden modificar otros atributos de la misma manera que hemos visto para otros casos.

Hay 2 tipos de notificaciones:

-Toast Notification: Esta notificación aparece sobre la ventana en la que estemos trabajando, y solamente ocupa el espacio que necesite el mensaje que muestra. No cierra ni modifica la actividad que estemos llevando a cabo, ni acepta interacción con el usuario, por lo que se desvanecerá en un periodo corto de tiempo.

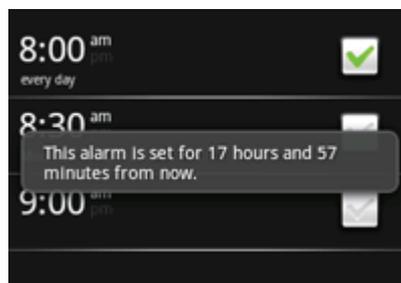


Figura 8. Toast Notification

Para crearla, mostrarla y situarla, es suficiente con incorporar unas instrucciones similares a las mostradas a continuación:

```
Toast toast = Toast.makeText(context, text, duration);  
toast.show();  
toast.setGravity(Gravity.TOP | Gravity.LEFT, 0, 0);
```

También se puede crear un layout (en este manual ya se ha visto cómo) y utilizarlo como una notificación de este tipo.

-Status Bar Notification: Esta notificación añade un mensaje en la ventana de notificaciones del sistema, y un icono en la barra de estado que, al seleccionarlo, lanzará la aplicación que lo ha activado. Además, se pueden configurar de tal modo que al producirse, el dispositivo suene, vibre o alerte al usuario de alguna manera.

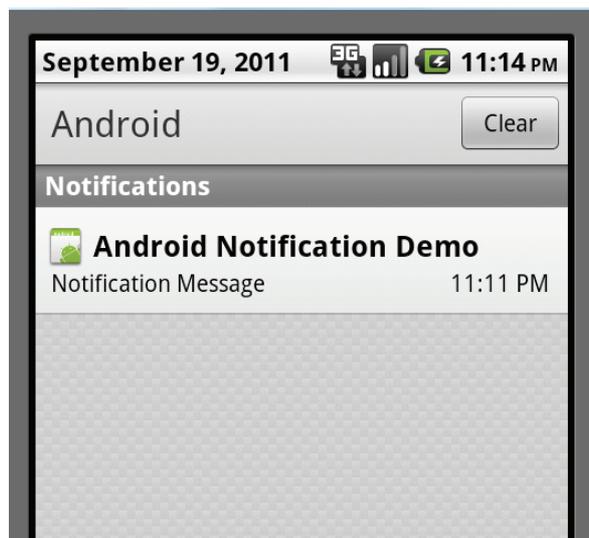


Figura 9. Status Bar Notification

Este tipo de notificaciones sólo deben referirse a actividades que se estén ejecutando en “background”, y no podrán pasar a “foreground” sin la acción del usuario.

ESTILOS Y TEMAS

Un estilo es una colección de propiedades que permite especificar el aspecto y formato de una vista o una ventana. Un tema es lo mismo que un estilo, pero aplicado a una actividad al completo, no sólo a una vista.

Para definir uno o varios estilos, se crea un archivo XML en el directorio `res/values/` del proyecto. Por cada uno nuevo, se debe incorporar un par de etiquetas `<style></style>`, que debe ir, a su vez, entre unas globales dentro de ese documento (`<resource> </resource >`). Por cada propiedad del estilo que se esté creando, debemos añadir un elemento `<item>`, otorgándole un nombre con “name”. Por ejemplo:

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
  <style name="NombreEstilo1"
    parent="@android:style/TextAppearance.Medium">
    <item name="android:layout_width">fill_parent</item>
    <item name="android:layout_height">wrap_content</item>
    <item name="android:textColor">#00FFFF</item>
    <item name="android:typeface">monospace</item>
  </style>

  <style name="NombreEstilo2" parent="@android:style/TextAppearance.Medium">
    <item name="android:layout_width">fill_parent</item>
    <item name="android:layout_height">wrap_content</item>
    <item name="android:textColor">#FF0000</item>
  </style>
</resources>
```

Para aplicar un estilo a una vista individual, es suficiente con añadir el atributo “style” al elemento deseado dentro del layout, en el archivo XML correspondiente. Por ejemplo:

```
<TextView style="@style/NombreEstilo1" android:text="@string/hola" />
```

Para aplicar el estilo a una actividad (aplicación) al completo, es preciso incorporar el atributo “android:theme” en el `<activity>` (`<application>`) del Android Manifest. Por ejemplo:

```
<activity android:theme="@style/ NombreEstilo1">
```

Además, Android contiene una gran cantidad de estilos y temas predefinidos a completa disposición del usuario.

4. RECURSOS DE APLICACIÓN

Luis Cruz – José Rodríguez de Llera – Alvaro Zapata

DEFINIENDO RECURSOS

Para la programación en Android, resulta conveniente separar los recursos que vaya a necesitar la aplicación (como imágenes u otro tipo de variables), de su código, de tal modo que se puedan mantener independientemente.

Conviene agruparlos en carpetas. El directorio `res/`, contiene todos los grupos de recursos, y permite contener dentro otros tales como: `animator/`, `anim/`, `color/`, `drawable/`, `layout/`, `menú/`, `raw/`, `values/`, `xml/`. Excepto la 5ª y la 7ª opción de la lista anterior, que soportan recursos de tipo Bitmap o archivos arbitrarios, respectivamente, el resto sólo pueden contener documentos en XML.

También se debería especificar un recurso para diferentes configuraciones posibles de los dispositivos, puesto que mientras se ejecuta una aplicación, éste sistema operativo utiliza los recursos apropiados para la configuración activa. Es decir, que conviene tener recursos alternativos (`drawable`, por ejemplo), por si el que tenemos por defecto no se ajusta apropiadamente en caso de utilizar un dispositivo no habitual.

Para especificar esta configuración alternativa, se ha de crear una nueva carpeta dentro de `res/`, del estilo `<resources_name>-<config_qualifier>`, que representan el directorio por defecto de un recurso determinado, y una configuración individual sobre en qué caso se puede usar ese recurso, respectivamente. Se permite añadir más de un `<config_qualifier>`, pero siempre separados con “-“. Posteriormente, sólo queda incorporar los recursos alternativos a la nueva carpeta, que deben tener el mismo nombre que los que son “por defecto”. Por ejemplo, donde `hdpi` se refiere a dispositivos con densidad alta de pantalla:

```
res/  
  drawable/  
    icono.png  
    fondo.png  
  drawable-hdpi/  
    icono.png  
    fondo.png
```

Respecto a las configuraciones disponibles que pueden ajustarse dentro del ya mencionado `<config_qualifier>`, las posibilidades son amplísimas. La anchura, el idioma, el tamaño de la pantalla, la orientación o la densidad en píxeles, son sólo alguno de los campos existentes, y dentro de cada uno de ellos, hay diferentes opciones.

Es muy importante conocer las restricciones, en caso de utilizar más de una configuración. El orden de los elementos en el nombre debe ser el correcto, puesto que de no ser así no serían

tenidos en cuenta (siguiendo una tabla/lista que se puede consultar en la web de Android). Tampoco se permiten anidar las carpetas. Todas deben ir en res/. Las mayúsculas no son tenidas en cuenta, puesto que se transforma todo a minúsculas al compilar. Por supuesto, no se puede incorporar más de un valor por cada tipo.

Ejemplos no válidos:

```
drawable -hdpi-port / (orden)
res/ drawable /drawable-en/ (anidar)
drawable-rES-rFR/ (sólo un elemento por tipo)
```

Sus respectivos ejemplos válidos:

```
drawable-port-hdpi/
res/drawable-en/
drawable-rES /
```

Cuando existe un recurso que se quiere utilizar en más de una configuración (sin ser el “por defecto”), Android permite no tener que duplicarlo. Para ello, es preciso asignarle un nombre que lo diferencie del resto, e introducirlo en la carpeta por defecto. Esto se puede hacer con los tipos “layout”, “drawable” y “strings y otros valores simples”.

¿CÓMO ENCUENTRA ANDROID EL RECURSO MÁS APROPIADO PARA CADA DISPOSITIVO?

Android selecciona qué recurso utilizar durante la ejecución de una aplicación, obteniendo la configuración del dispositivo en que se realiza, y comparándola con los recursos disponibles en el proyecto, siguiendo este algoritmo:

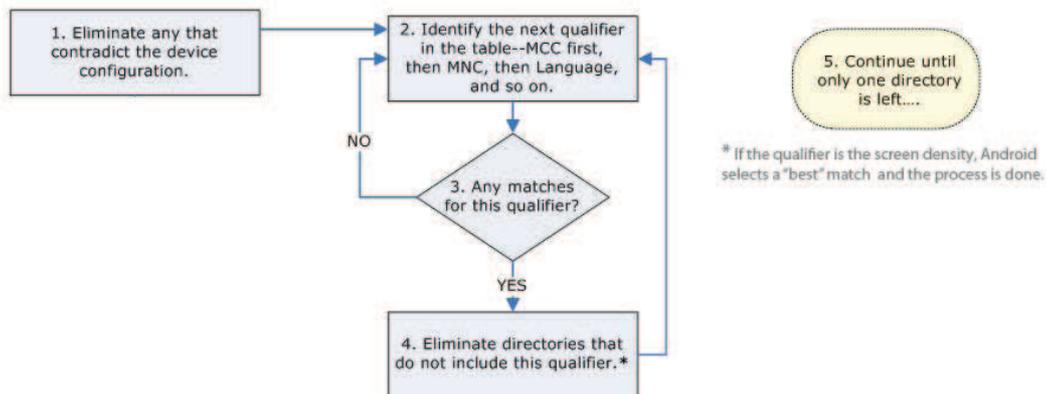


Figura 1. Algoritmo de búsqueda de recursos

1. Eliminar las carpetas con recursos contradictorios a la configuración del dispositivo.
2. Coger el siguiente calificador con precedencia más alta, en la tabla que podemos encontrar en la Web de Android, mencionada anteriormente
3. ¿Está ese recurso en algún directorio?
 - Si no es así, volver al paso 2.
 - Si está, seguir con el paso 4.
4. Eliminar los directorios que no contengan ese calificador.
5. Repetir los pasos 2, 3 y 4 hasta que sólo quede 1 directorio.

Además de este algoritmo, el sistema, más adelante, optimiza algunos aspectos, siempre en función de la configuración del dispositivo.

UTILIZANDO LOS RECURSOS

Una vez que se han creado todos los recursos, se puede referir a ellos a través de su identificador. Todos ellos están definidos en la clase R del proyecto. Ésta, se genera automáticamente al compilar, y contiene identificadores únicos, compuestos siempre por un tipo y un nombre, para cada recurso existente en el directorio res/. Hay 2 formas de acceder a ellos:

- A través del código, como una subclase de R. Se puede usar un recurso pasando su identificador como parámetro de una función, siguiendo el patrón

```
[<package_name>].R.<resource_type>.<resource_name>
```

que representan el paquete donde se encuentra el recurso (no es necesario poner nada si se referencia desde el propio paquete), la subclase de R con el tipo y el nombre del recurso sin su extensión, respectivamente. Este método se puede utilizar para establecer una imagen de fondo, un layout para una pantalla o establecer el título de una actividad. Por ejemplo:

```
getWindow().setBackgroundDrawableResource(R.drawable.fondo) ;
```

- Desde XML, usando la sintaxis correspondiente a la configuración del identificador del recurso. Se pueden definir elementos o atributos de un fichero XML como referencia a ellos. La sintaxis sigue el patrón

```
@[<package_name>:]<resource_type>/<resource_name>
```

que representan el paquete donde se encuentra el recurso (no es necesario poner nada si se referencia desde el propio paquete), la subclase de R con el tipo y el nombre del recurso sin su extensión, respectivamente. Este método se puede utilizar para establecer el color de un texto o qué debe mostrar, o incluso para crear alias. Por ejemplo:

```
<?xml version="1.0" encoding="utf-8"?>
<EditText
xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    android:textColor="@color/azul"
    android:text="@string/hola" />
```

Con esta última opción, se puede establecer un atributo como referencia a un estilo perteneciente al tema que está en uso, incluyendo, por ejemplo,

```
android:textColor="?android:colorSecundario".
```

LOCALIZACIÓN

Android está a disposición del usuario en muchas regiones y dispositivos diferentes. Lo ideal, es que las aplicaciones utilicen los textos, la moneda, los números... de acuerdo a donde se estén ejecutando. Todo eso se controla incorporando la mayor parte del contenido de la interfaz de usuario a los directorios de recursos (como hemos visto anteriormente), y manejando el comportamiento de la interfaz desde el código Java. Siempre que una aplicación se ejecute de forma “local”, y no haya definidos un texto específico para ella, Android cargará el “string.xml” por defecto. Sin embargo, si no existe ese archivo, se mostrará un error y la aplicación no se ejecutará.

La prioridad de los recursos en función de la localización es absoluta. Por ejemplo, si nos encontramos en Francia, y tenemos un dispositivo que soporta alta calidad de gráficos, dará prioridad al directorio `res/drawable-fr/`, independientemente de si en su interior los recursos definidos son para dispositivos que soportan exclusivamente gráficos de baja calidad, frente a un directorio denominado `res/drawable-hdpi/`, que inicialmente parecería más adecuado.

Se debe saber que los calificadores MCC y MCN son una excepción y son siempre prioritarios, por lo que los recursos contenidos en semejantes carpetas serán cargados por delante de cualquier otro. En ocasiones es preciso crear un layout flexible, de tal modo que se adecúe a un contexto, pero que no se limite a él. Por ejemplo, si el entorno local es España, la agenda de contactos es preciso que tenga campos para nombre y 2 apellidos, sin embargo, si el entorno es Irlanda, sobraría uno de los huecos para los apellidos. Pues bien, la única solución para llevar a cabo esta situación no es duplicar los recursos, puesto que también se permite crear un único layout en el que un campo se active o desactive en función del idioma. Es decir, evaluando una condición sobre la configuración del dispositivo, y se recomienda esta opción por encima de crear más recursos. Se puede utilizar Android para buscar recursos locales, a través de la siguiente instrucción:

```
String locale =  
context.getResources().getConfiguration().locale.getDisplayName();
```

PROBAR APLICACIONES LOCALIZADAS

Para probar cómo funcionarían aplicaciones en otra localización, se puede cambiar la configuración del emulador con adb shell, siguiendo éstos pasos:

- 1- Elegir qué región se quiere simular y determinar su lenguaje y códigos de región, por ejemplo fr de francés y CA de Canadá.
- 2- Lanza el emulador.
- 3- Ejecuta mediante comandos, desde el ordenador que lanza el emulador, “abd Shell”, o “adb -e Shell”, si existe un dispositivo adjunto.

4- Una vez dentro, se debe ejecutar este comando:

```
setprop persist.sys.language [language code];setprop  
persist.sys.country [country code];stop;sleep 5;start
```

reemplazando los corchetes por los códigos apropiados (fr y Ca respectivamente, para este ejemplo).

Estos pasos reiniciarán el emulador, y cuando se vuelva a ejecutar la aplicación, se hará sobre la nueva configuración. De hecho, cuando esté lista, se pueden publicar en el Market para diferentes localizaciones, ya sea con diferentes apk's, o incluyendo todos los recursos en la misma.

¿ESTÁN TODOS LOS RECURSOS BÁSICOS NECESARIOS EN LA APLICACIÓN?

Una forma sencilla de comprobar si la aplicación tiene los recursos necesarios para ejecutarse, independientemente de la configuración consiste en establecer el emulador en una localización para la que la aplicación no tenga recursos definidos. Si al ejecutarla nos sale un mensaje de error, es que faltan los recursos por defecto, o no están bien definidos.

TIPOS DE RECURSOS

En Android existen varios tipos de recursos. A lo largo de este manual ya se han ido viendo, más o menos detalladamente, algunos de ellos.

MENU

Este recurso define al Menú Principal, Menú Contextual y Submenú.

- **Localización:** res/menú/nombre_del_archivo.xml
- **Referencia:** R.menu.nombre_del_archivo (en JAVA) y [package:]menú.nombre_del_fichero (en XML).
- **Sintaxis** (con todas las opciones):

```
<?xml version="1.0" encoding="utf-8"?>
<menu xmlns:android="http://schemas.android.com/apk/res/android">
  <item android:id="@+[package:]id/resource_name"
        android:title="string"
        android:titleCondensed="string"
        android:icon="@[package:]drawable/drawable_resource_name"
        android:onClick="method name"
        android:showAsAction=["ifRoom" | "never" | "withText" |
"always" | "collapseActionView"]

        android:actionLayout="@[package:]layout/layout_resource_name"
        android:actionViewClass="class name"
        android:actionProviderClass="class name"
        android:alphabeticShortcut="string"
        android:numericShortcut="string"
        android:checkable=["true" | "false"]
        android:visible=["true" | "false"]
        android:enabled=["true" | "false"]
        android:menuCategory=["container" | "system" | "secondary" |
"alternative"]
        android:orderInCategory="integer" />
  <group android:id="@+[package:]id/resource name"
        android:checkableBehavior=["none" | "all" | "single"]
        android:visible=["true" | "false"]
        android:enabled=["true" | "false"]
        android:menuCategory=["container" | "system" | "secondary"
| "alternative"]
        android:orderInCategory="integer" >
    <item />
  </group>
  <item >
    <menu>
      <item />
    </menu>
  </item>
</menu>
```

- **Elementos:**

- 1- <menu>: Requerido. Tiene que ser el nodo raíz e incluir el atributo xmlns:android obligatoriamente.
- 2- <ítem>: Tiene que ser el hijo de un elemento de tipo <menu> o <grupo>. Si además contiene un elemento del 1er tipo, existirá un Submenú.
- 3- <group>: Para crear una colección de elementos (de tipo <item>. Los contiene), que pueden compartir unas características. Es necesario que sea hijo de un elemento <menú>.

RESTO DE RECURSOS

El recurso “Menu” es un ejemplo, escogido en este manual por su importancia y por haberse visto parcialmente con anterioridad, pero todos los demás tienen un comportamiento análogo. Aquí se detallan más brevemente:

- **Animation:** Define animaciones predeterminadas. Las denominadas “Tween” se guardan en res/anim/ y son accedidas mediante la clase R.anim. Las “Frame” se almacenan en res/drawable/ y se accede a ellas desde R.drawable.
- **Color State List:** Define unos colores que cambian basándose en la “View”. Se guardan en res/color/ y se accede a ellos desde la clase R.color.
- **Drawable:** Define diferentes tipos de gráficos con Bitmaps o con XML. Son accedidos a través de R.drawable y almacenados en res/drawable/.
- **Layout:** Definen la estructura general de la interfaz de usuario. Se guardan en res/layout/ y se accede a ellos desde la clase R.layout.
- **String:** Define strings, arrays de strings y plurales, incluyendo formato y estilo de ellos. Se almacenan en res/values/ y se pueden acceder desde las clases R.string, R.array y R.plurals respectivamente.
- **Style:** Define el aspecto y formato de los elementos de la interfaz de usuario. Se guardan en res/values/ y se accede a ellos desde la clase R.values.
- **Otros:** Definen valores tales como booleanos, enteros, dimensiones, colores u otros arrays. Todos ellos se almacenan en res/values/, pero cada uno se accede desde subclases de R únicas, tales como R.bool, R.integer, R.dimen, etcétera.

Android permite definir recursos que se ajusten a diferentes configuraciones en diferentes dispositivos. A la hora de ejecutar una aplicación, sólo se utilizarán los adecuados y necesarios en función de cada situación. El resultado de ello es una optimización en la relación esfuerzo-calidad, para cada dispositivo.

5. DATOS EN ANDROID.

IDEAS PRINCIPALES

Manuel Báez – Jorge Cordero – Miguel González

INTRODUCCIÓN

Hay cuatro maneras de almacenar los datos en Android, usaremos una u otra dependiendo de la función que tengan dichos datos. Es importante destacar que los datos de cada aplicación son privados a dicha aplicación, pero tenemos la posibilidad de compartirlos si así lo deseamos, como veremos más adelante.

Tenemos cuatro tipos de datos distintos, en función de lo que queramos almacenar. Dichos tipos son:

- *Preferencias*: Conjunto de datos, clave/valor
- *Archivos locales*: Almacenamiento interno y externo (SD)
- *Base de datos*: SQLite
- *Proveedores de contenidos*: Único mecanismo para compartir datos, no se trata con ellos directamente.

PREFERENCIAS COMPARTIDAS

Tenemos a nuestra disposición un tipo de datos llamado *preferencias*. Estas *preferencias*, son una forma ágil para poder guardar datos simples de la aplicación. Estos datos son un conjunto clave/valor que perdura después de que la aplicación se cierre, por lo que es principalmente útil para guardar, como su nombre indica, las preferencias de una aplicación.

Por ejemplo guardar el idioma, configuraciones de sonido de la aplicación, fuentes, colores, etc. Los datos que permite guardar son primitivos y se invocan con los siguientes métodos `putInt()`, `putBoolean()`, `putFloat()`, `putLong()`, `putString()`, `putStringSet()` de la forma (String key, tipo value).

SharedPreferences

Ahora vamos a ver cómo guardar los datos de las preferencias y cómo acceder a ellos al inicializar la aplicación.

Para ello utilizaremos una clase que hereda de `PreferenceActivity`, la cual guarda en un xml las preferencias de la aplicación que, en principio, serán cargadas cuando la iniciemos y guardadas cuando la cerremos. Dicho xml se guarda en `SharedPreferences`. Es importante destacar que podemos tener una colección de preferencias, para ello tendremos que asociarles un identificador. Almacenándose los datos en:

```
/data/data/nombre_paquete/shared_prefs/preferencias_por_defecto.xml
```

Vamos a ver primero cómo guardar los datos. Tenemos a nuestra disposición para ello los métodos `onSaveInstanceState()` o `onStop()`, éste último será el que usemos para el ejemplo. Ambos se invocan antes de cerrar la aplicación, siempre y cuando se cierre de manera correcta.

Para guardar los datos seguiremos el siguiente patrón:

1. Llamamos a `edit(void)` para editar las preferencias con `SharedPreferences.Editor`.
2. Añadimos valores con los métodos `put` anteriormente nombrados.
3. Actualizamos los nuevos valores usando `commit(void)`

Código Android

```
protected void onStop() {
    super.onStop();
    // Necesitamos un objeto Editor para poder modificar las preferencias
    SharedPreferences preferencias = getSharedPreferences(PREFS_NAME, 0);
    SharedPreferences.Editor editor = preferencias.edit();
    editor.putString("NumeroAlAzar", "58472");
    // Por ejemplo un número al azar
    // Actualizamos las nuevas preferencias
    editor.commit();
}
```

Ahora que tenemos las preferencias guardadas vamos a ver cómo podemos cargarlas.

Código Android

```
public class PreferenciasActivity extends PreferenceActivity implements
    OnSharedPreferenceChangeListener {
    //... Tu código

    protected void onCreate(Bundle icle) {
        super.onCreate(icle);
        // Carga las preferencias del XML.
        addPreferencesFromResource(R.xml.preferencias);
        getPreferenceScreen().getSharedPreferences().registerOnSharedPreferenceChangeListener(this);
    }
    //... Tu código
}
```

Preference Activity

Por ahora que sabemos modificar las preferencias en código, vamos a ver cómo hacer nuestras preferencias en xml. Dichos xml se almacenan en /res/xml, y pueden constar de los siguientes tipos de opciones:

1. CheckBoxPreference: Check, valor booleano.
2. EditTextPreference: Cadena de texto.
3. ListPreference: Lista de selección única.
4. MultiSelectListPreference: Lista de selección múltiple.

Código xml

```
<PreferenceScreen xmlns:android="http://schemas.android.com/apk/res/android">
    <PreferenceCategory>
        TU CÓDIGO DE ESTA CATEGORÍA
    </PreferenceCategory>
    <PreferenceCategory>
        TU CÓDIGO DE ESTA CATEGORÍA
    </PreferenceCategory>
</PreferenceScreen>
```

Cuando hayamos definido nuestras preferencias, tendremos que implementar una nueva actividad, para ello tendremos que crear una clase que extienda de PreferenceActivity de la siguiente forma

Código Android

```
public class Opciones extends PreferenceActivity {
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        addPreferencesFromResource(R.xml.opciones);
    }
}
```

También tendremos que añadirla al Manifest como actividad, para ello bastará con añadir el siguiente código en el Manifest.

Código Android

```
<activity android:name=".PantallaOpciones"
    android:label="@string/app_name">
</activity>
```

ARCHIVOS LOCALES

A la hora de guardar distintos tipos de datos, que no se ajusten al patrón clave/valor de las preferencias, tendremos que guardarlos como *archivos locales*, dichos archivos, por defecto, no son accesibles desde otras aplicaciones, regulado por permisos Unix. El acceso es parecido al de Java estándar, debemos crear tanto input stream como output stream. Puede soportar solamente archivos creados en la carpeta de la aplicación. Por lo que necesitamos saber dónde están almacenadas las aplicaciones. Están almacenadas en:

/data/data/nombre_paquete/files/nombre_fichero,

pudiendo en las últimas versiones almacenarlo en la SDCard. Es importante ser conscientes de que algunos dispositivos tienen una memoria interna limitada, por lo que no deberíamos de abusar de este espacio guardando archivos de gran tamaño.

Memoria interna.

Para empezar vamos a ver cómo crear los output stream y los input stream, hay que destacar que en el output stream tenemos que seleccionar el modo de acceso, que puede ser:

1. MODE_PRIVATE (por defecto) acceso privado desde nuestra aplicación.
2. MODE_APPEND para poder añadir datos a un fichero existente.
3. MODE_WORLD_READABLE para permitir lectura a otras aplicaciones.
4. MODE_WORLD_WRITABLE para permitir escritura a otras aplicaciones.

A continuación veremos cómo crear los output:

Código Android

```
try
{
    OutputStreamWriter fOut=
        new OutputStreamWriter(
            openFileOutput("texto_ejemplo.txt", Context.MODE_PRIVATE));

    fOut.write("Cualquier cosa"); // Escribimos "Cualquier cosa" en fOut
```

```
fOut.close();           // Cerramos la escritura en el fichero fOut
}
catch (Exception e)
{
    Log.e("Error de fichero", "Error al escribir el fichero.");
}
```

Ahora para los input hay que tener en cuenta que solamente podremos leer aquellos ficheros para los que tengamos permisos, el código viene detallado a continuación.

Código Android

```
try
{
    BufferedReader fIn =
        new BufferedReader(
            new InputStreamReader(
                openFileInput("texto_ejemplo2.txt")));

    String texto = fIn.readLine(); // Cogemos la línea de fIn
    fIn.close(); // Salimos para guardar la línea
}
catch (Exception e)
{
    Log.e("Error de fichero", "Error al leer el fichero");
}
```

RECURSOS

Otra forma de almacenar datos en la memoria interna es almacenarlo como recurso en la carpeta “/res/raw” de nuestro proyecto, en eclipse dicha carpeta no se crea por defecto, tendremos que crearla. Es importante destacar que este método de almacenamiento es útil para ficheros que no vamos a modificar, ya que posteriormente serán de solo lectura. La forma de acceder a dichos archivos viene detallada a continuación.

Código Android

```
try
{
    InputStream fRaw =
        getResources().openRawResource(R.raw.ejemplo_raw);
        //ejemplo_raw es el nombre del fichero
    BufferedReader bRIn =
        new BufferedReader(new InputStreamReader(fRaw));
    String linea = bRIn.readLine();
    fRaw.close();
}
catch (Exception e)
{
    Log.e("Error de fichero", "Error al leer fichero desde recurso");
}
```

MEMORIA EXTERNA

La última alternativa para almacenar archivos locales será la de guardarlos en la memoria externa, que generalmente será una tarjeta de memoria SD. Es importante destacar que en el caso de la memoria externa, y al contrario que con la memoria interna, puede no estar disponible ya sea porque no está presente o porque el dispositivo no la reconoce. Para ello tendremos que cercionarnos de que está disponible antes de utilizarla como sistema de almacenamiento, con ese fin usaremos el método `getExternalStorageStatus()` que nos devuelve el estado de la memoria externa. Según queramos leer o escribir en ella los estados que nos interesarán son `MEDIA_MOUNTED` que nos dice que podemos escribir y leer en ella o `MEDIA_MOUNTED_READ_ONLY` que nos dice que está disponible con permisos de solo lectura.

El siguiente fragmento de código nos muestra una forma sencilla de verificar el estado de la memoria externa y almacenarlo en dos booleanos.

Código Android

```
boolean tarjetaDisponible = false;
boolean tarjetaEscritura = false;

//Comprobamos el estado de la memoria externa (tarjeta SD)
String estadoTarjeta = Environment.getExternalStorageState();

if (estadoTarjeta.equals(Environment.MEDIA_MOUNTED))
{
    // Tarjeta disponible y habilitada para escritura
    tarjetaDisponible = true;
    tarjetaEscritura = true;
}
else if (estadoTarjeta.equals(Environment.MEDIA_MOUNTED_READ_ONLY))
{
    // Tarjeta disponible y deshabilitada para escritura
    tarjetaDisponible = true;
    tarjetaEscritura = false;
}
else
{
```

```
// Tarjeta NO disponible
tarjetaDisponible = false;
tarjetaEscritura = false;
}
```

También tendremos que indicar en el manifest que nuestra aplicación necesita permisos para almacenar datos en memoria externa, para ello pondremos:

Código xml

```
<uses-permission android.permission.WRITE_EXTERNAL_STORAGE>
</uses-permission>
```

Ahora que sabemos el estado de la memoria externa y tenemos permisos vamos a ver la ruta absoluta de la tarjeta de memoria. Para ello usaremos el método `getExternalStorageDirectory()` que nos devolverá un `File`, ahora ya podremos crear un fichero en la dirección de dicho `File` como mostramos a continuación.

Código Android

```
try
{
    File ruta_tarjeta = Environment.getExternalStorageDirectory();
    File f = new File(ruta_tarjeta.getAbsolutePath(), "ejemplo_tarjeta.txt");
    OutputStreamWriter fOut =
        new OutputStreamWriter(
            new FileOutputStream(f));
    fOut.write("Smile! :D.");
    fOut.close();
}
catch (Exception e)
{
    Log.e("Error de ficheros", "Error al escribir fichero en la tarjeta.");
}
```

Para leer de un fichero el código es muy similar a la escritura. Mostramos el código a continuación:

Código Android

```
try
{
    File ruta_tarjeta = Environment.getExternalStorageDirectory();

    File f = new File(ruta_tarjeta.getAbsolutePath(), "ejemplo_tarjeta.txt");
    BufferedReader fIn =
        new BufferedReader(
            new InputStreamReader(
                new FileInputStream(f)));
    String linea = fIn.readLine();
    fIn.close();
}
catch (Exception e)
{
    Log.e("Error de ficheros", "Error al leer fichero de la tarjeta.");
}
```

BASES DE DATOS: SQLITE

Android tiene integrado en el propio sistema una API completa que nos permite manejar BBDD en SQLite. SQLite es un motor de bases de datos que se ha ido popularizando en los últimos años dado que maneja archivos de poco tamaño, no necesita ejecutarse en un servidor, cumple el estándar SQL-92 y, además, es de código libre.

El conjunto de la base de datos (definiciones, tablas, índices, y los propios datos), son guardados como un único fichero en la máquina host. Este diseño simple se logra bloqueando todo el fichero de base de datos al principio de cada transacción. Esta es, al mismo tiempo, su gran virtud y su mayor inconveniente, ya que gracias a ello dispone de unas latencias muy bajas, pero también impide el acceso múltiple a la base de datos.

Centrándonos en el lenguaje que abordamos en este tutorial, la manera más fácil de acceder a la información de una base de datos local es creando una clase que herede de SQLiteOpenHelper sobre la cual tendremos que adaptar/sobreescribir los métodos proporcionados para obtener la

funcionalidad con la base de datos deseada. Básicamente en esta clase definimos los atributos de nuestra base de datos y el comportamiento ante creación y actualización de la misma. Los métodos que deberemos sobrescribir serán `onCreate()` y `onUpgrade()`, además de la constructora, donde podremos incluir todo lo que creamos necesario.

A modo de ejemplo, vamos a crear una base de datos de una entidad Domicilio que incluirá como atributos: calle, ciudad, CP y número. Así pues crearíamos la clase `DomicilioSQLiteHelper`:

Código Android

```
public class DomicilioSQLiteHelper extends SQLiteOpenHelper {

    //Aquí creamos el String que creará la base de datos en el onCreate

    String creaBD= "CREATE TABLE Domicilio (calle TEXT, ciudad TEXT, CP
    INTEGER, numero INTEGER)";

    public DomicilioSQLiteHelper(Context context, String nombre,
        CursorFactory factory, int version) {
        super(context, nombre, factory, version);
    }

    @Override
    public void onCreate(SQLiteDatabase db) {
        //Aquí se crea la BD, se llamaría solo cuando no exista
        db.execSQL(sqlCreate);
    }

    @Override
    public void onUpgrade(SQLiteDatabase db, int a, int b) {
        /*Utilizamos como opción de actualización la de borrado de la tabla
        anterior, para luego crearla vacía con la nueva versión*/
        db.execSQL("DROP TABLE IF EXISTS Domicilio"); //Borrado anterior
        db.execSQL(sqlCreate); //Creación nueva
    }
}
```

Cabe mencionar cuando se usarán los dos métodos anteriores, el método onCreate() se ejecutará de manera automática cuando se necesite crear la base de datos, lo que se reduce a cualquier llamada o referencia a la base de datos mientras esta todavía no exista. Y el método onUpgrade() que también será llamado automáticamente cuando se produzca un cambio en la estructura de la base de datos, como podría ser la inclusión/eliminación de un campo en una tabla, en el código de ejemplo anterior hemos simplemente borrado la tabla anterior y creado una tabla nueva vacía, sin embargo, en la mayoría de los casos es necesario migrar los datos del modelo anterior al nuevo.

Teniendo nuestra clase heredada SQLiteOpenHelper, estamos en disposición de utilizar los métodos getReadableDatabase() o getWritableDatabase() para acceder a la base de datos en modo lectura o lectura/escritura. Veamos un ejemplo simple en el que abramos la base de datos para escritura y creamos un par de registros:

Código Android

```
Int version = 1;
String nombreDB = "DomiciliosDB";
/***** Nuestro código anterior aquí *****/
    DomicilioSQLiteHelper domicilioHelper =
        new UsuariosSQLiteHelper(this, nombreDB, null, version);

// Abrimos la Base de datos, en caso de que no existiera, se crearía ahora
SQLiteDatabase db = domicilioHelper.getWritableDatabase();

//Insertamos 2 domicilios para inicializar la tabla
db.execSQL("INSERT INTO Domicilio(calle, ciudad, CP, numero)
           VALUES ('C/Mayor', 'Madrid', 28001, 13)");
db.execSQL("INSERT INTO Domicilio(calle, ciudad, CP, numero)
           VALUES ('Avda. Grande', 'Ibiza', 26050, 45)");

//Cerramos la base de datos ya actualizada
db.close();

/***** Resto de código *****/
```

Pese a haber aparecido antes, no se ha comentado el uso del método `execSQL(String query)`. Como el lector habrá podido imaginar, este método recibe Strings de los query que se pueden ejecutar sobre la base de datos y los ejecuta. La intención de este tutorial, no es ni mucho menos enseñar la utilización de bases de datos con el estándar de SQLite, para obtener información sobre él por favor, visite la página oficial de SQLite y encontrará información sobre su sintaxis:

<http://www.sqlite.org/lang.html>

A pesar de lo citado anteriormente, para no quedarnos sólo con estas breves pinceladas sobre la sintaxis de SQLite, estas son las funcionalidades más comunes de una base de datos en modo escritura:

- Insertar un registro

```
db.execSQL("INSERT INTO Usuarios (usuario,email) VALUES ('usu1','usu1@email.com' ");
```

- Eliminar un registro

```
db.execSQL("DELETE FROM Usuarios WHERE usuario='usu1' ");
```

- Actualizar un registro

```
db.execSQL("UPDATE Usuarios SET email='nuevo@email.com' WHERE usuario='usu1' ");
```

Ahora vamos a crear otro código en el que mostremos las funcionalidades de los cursores. Los cursores simplemente nos servirán para recorrer el resultado de una base de datos como si un iterador se tratase. Tratamos de ejecutar algunas queries sobre la clase domicilio, para ello, los permisos con los que abriremos la base de datos de los domicilios serán sólo de lectura.

Código Android

```
Int version = 1;

String nombreDB = "DomiciliosDB";

/***** Nuestro código anterior aquí *****/

DomicilioSQLiteHelper domicilioHelper = new UsuariosSQLiteHelper(this, nombreDB,
null, version);

// Abrimos la Base de datos, en casa de que no existiera, se crearía ahora
SQLiteDatabase db = domicilioHelper.getReadableDatabase();

Cursor c = db.rawQuery("SELECT ciudad, CP FROM Domicilio WHERE calle=
"C/Mayor");

//Obtenemos los índices de las columnas

int ciudadIndex = mCursor.getColumnIndexOrThrow("ciudad");

int CPIndex = mCursor.getColumnIndexOrThrow("CP");
```

```

// Posicionamos el cursor al principio de la lista
if (c.moveToFirst()) {
    do {
        //Obtenemos en nuestras variables los datos del registro que está leyendo.
        String ciudad= c.getString(ciudadIndex);
        int CP = c.getString(CPIndex);
    } while(c.moveToNext());
    /*Lo seguimos adelantando mientras tengamos registros que leer,
    Aunque parezca extraño, por la falta de uso, es muy común utilizar aqui la
    estrucutra DO- WHILE*/
    /***** Resto de código *****/

```

CONTENT PROVIDERS

Los Content Providers son el mecanismo que tiene Android para comunicar datos entre distintas aplicaciones. Funcionalmente suelen ser muy parecidos a las bases de datos en SQLite que hemos comentado en el punto anterior. El propio Android trae (desde sus versiones 2.0+) incluidos la agenda ,los SMS y el listado de llamadas mediante el método de los Content Providers, simplemente una aplicación debe acceder a la URI donde tenemos el Content Provider y una vez tengamos acceso pedirle los datos que necesitamos.

Al igual que para las bases de SQLite para los Content Providers deberemos crear una clase que herede de ContentProvider y declararla en el manifest.

AndroidManifest.xml

```

<application>...
<provider
    android:name="DomiciliosProvider"
    android:authorities="com.pruebasandroid.ejemplo" />
</application/>

```

La facilidad de su uso reside en tener una base de datos que realice las mismas funciones que los métodos que tenemos que sobrescribir, para llamarlos basta con hacer un buen uso de las URIs. A continuación se muestra el código, a modo de ejemplo, de DomiciliosProvider:

```
package com.pruebasandroid.ejemplo;

...

public class DomiciliosProvider extends ContentProvider {

    //Establecemos como una constante la URI para aeder a estos datos
    private static final String uri = "content://com.pruebasandroid.ejemplo/domicilios";

    //Uri del String anterior

    public static final Uri CONTENT_URI = Uri.parse(uri);

    //Necesario para UriMatcher

    private static final int DOMICILIOS = 1;

    private static final int DOMICILIOS_ID = 2;

    private static final UriMatcher uriMatcher;

    //Clase interna para declarar las constantes de columna
    public static final class Clientes implements BaseColumns
    {
        private Clientes() {}

        // Inicializamos las columnas, para poder acceder a ellas con un mnemotécnico
        public static final String COL_CALLE = "calle";
        public static final String COL_CP = "cp";
        public static final String COL_NUMERO = "numero";
        public static final String COL_CIUDAD = "ciudad";
    }

    //Los datos referentes a la base de datos, que creamos en el ejemplo anterior
    private UsuariosSQLiteHelper helper;

    private static final String NOMBRE_BD = "DomicilioDB";

    private static final int VERSION = 1;

    private static final String TABLA_DOMICILIOS = "Domicilios";

    //Inicializamos el UriMatcher
    static {

        uriMatcher = new UriMatcher(UriMatcher.NO_MATCH);

        uriMatcher.addURI("com.pruebasandroid.ejemplo", "domicilios", DOMICILIOS);
    }
}
```

```

        uriMatcher.addURI("com.pruebasandroid.ejemplo", "domicilios/#",
DOMICILIOS_ID);
    }

    /*Sobreescribimos los métodos de ContentProvider*/
    @Override
    public boolean onCreate() {
        helper = new UsuariosSQLiteHelper(
            this.getContext(), NOMBRE_DB, null, VERSION);

        return true;
    }

    /*Aqui devolvemos el cursor de nuestra BD de domicilios*/
    @Override
    public Cursor query(Uri uri, String[] projection,
        String selection, String[] selectionArgs, String sortOrder) {

        /*Construimos un where si se refiere a un id concreto */
        String where = selection;
        if(uriMatcher.match(uri) == DOMICILIOS_ID){
            where = "_id=" + uri.getLastPathSegment();
        }

        /*Escritura y lectura y devolvemos el cursor de nuestra base de datos*/
        SQLiteDatabase db = helper.getWritableDatabase();

        return (Cursor)db.query(TABLA_DOMICILIOS, projection, where,
            selectionArgs, null, null, sortOrder);
    }

    @Override
    public Uri insert(Uri uri, ContentValues values) {

        /*para escribir, insertamos en la base de datos que nos proporciona*/
        long id = 1;

        SQLiteDatabase db = helper.getWritableDatabase()

        id = db.insert(TABLA_DOMICILIOS, null, values);
    }

```

```

    /*Devolvemos su correspondiente Uri*/
    Uri newUri = ContentUris.withAppendedId(CONTENT_URI, id);
    return newUri;
}

@Override
public int update(Uri uri, ContentValues values,
                 String selection, String[] selectionArgs) {
    /* Aqui el código para el update*/
}

@Override
public int delete(Uri uri, String selection, String[] selectionArgs) {
    /* Aqui el código para el delete*/
}

@Override
public String getType(Uri uri) {
    int tipo = uriMatcher.match(uri);
    switch (tipo)
    {
        case DOMICLIOS:
            return "vnd.android.cursor.dir/vnd.pruebasandroid.domicilio";
        case DOMICILIOS_ID:
            return "vnd.android.cursor.item/vnd.pruebasandroid.domicilio";
        default:
            return null;
    }
}
}
}

```

DATOS EN RED

A pesar de lo que hemos comentado sobre SQLite anteriormente, muchas son las aplicaciones que necesitan la integración con una base de datos existente en red como un MySQL. La diferencia principal respecto a los datos que podríamos manejar con SQLite reside en que estos datos se guardaban en un archivo de forma local mientras que con las BBDD que podemos manejar a través de la red tenemos cambios de otros usuarios de manera inmediata.

La mejor forma de conectar nuestro terminal con una base de datos de manera remota es mediante el uso de un servicio web. Para nuestro tutorial, explicaremos como crear de manera sencilla un servicio web en php que nos proporcione datos desde una base de datos MySQL y como interactuar desde Android con ese servicio. Este tipo de servicio es simplemente un ejemplo, si el lector conoce o prefiere utilizar cualquier otro tipo de lenguaje en la creación del servicio web es una opción totalmente válida, pero a modo de seguir acorde con el nivel del tutorial lo crearemos en php ya que es un lenguaje muy sencillo para el manejo de bases de datos que esperamos sea de gusto del lector.

A modo de ejemplo vamos a crear una base de datos que tenga una única tabla que contenga los datos de una persona. A cada persona le vamos a dar los atributos DNI, nombre, sexo y fecha de nacimiento. Entramos en la consola de SQL de nuestro servidor y escribimos:

Código SQL.

```
CREATE TABLE `nuestraBD`.`persona` (  
  `DNI` VARCHAR( 10 ) NOT NULL ,  
  `nombre` VARCHAR( 30 ) NOT NULL ,  
  `sexo` VARCHAR( 10 ) NOT NULL ,  
  `fechaNacimiento` DATE NOT NULL  
)
```

Hasta aquí tendríamos creada ya en la base de datos la entidad persona, vamos ahora a introducir 3 registros:

Código SQL.

```
INSERT INTO `nuestraBD`.`persona` (  
  `DNI` ,  
  `nombre` ,  
  `sexo` ,  
  `fechaNacimiento`  
)
```

```
VALUES (
'000000000X', 'Jorge', 'Hombre', '1980-11-20'
), (
'111111111Y', 'Francisco', 'Hombre', '1977-09-10'
), (
'222222222Z', '', 'Mujer', '1987-01-15'
);
```

Ahora con nuestra base de datos terminada, vamos a crear un servicio web que obtenga todos los datos de la tabla. Obviamente el servicio web aceptará cualquier sintaxis y restricción que le pidamos como al propio SQL (Where, Groupby, limit ,sort....)

Código PHP.

```
<?php
mysql_connect("host","username","password");
mysql_select_db("nuestraBD");

$pregunta = "select * from persona"

$sql=mysql_query($pregunta);
while($row=mysql_fetch_assoc($sql))
    $output[]=$row;
print(json_encode($output));
mysql_close();
?>
```

Publicamos en la web el código anterior y lo renombramos a “personas.php”. Recordar que los datos de la conexión especificados por "host","username","password", deberán corresponder con los de la base de datos que el usuario requiera para su aplicación. En el código anterior, simplemente hemos lanzado una query que nos devuelva todos los datos de la tabla persona y después hemos ido creando un array que contenga cada entrada a la propia tabla. Como se puede observar, al final siempre cerramos la conexión, pero antes de ello hemos escrito print(json_encode(\$output)); para poder obtener los datos como salida en el estándar JSON.

JSON (JavaScript Object Notation) es un lenguaje de etiquetas como XML que se utiliza como sustituto para intercambio de contenidos dada su sencillez permite crear parsers mucho más sencillos que en XML. La estructura que nos devolvería JSON para el ejemplo anterior sería:

Respuesta JSON

```
[{"DNI":"000000000X","nombre":"Jorge","sexo":"Hombre","fechaNacimiento":"1980-11-20"}, {"DNI":"111111111Y","nombre":"Francisco","sexo":"Hombre","fechaNacimiento":"1977-09-10"}, {"DNI":"222222222Z","nombre":"Patricia","sexo":"Mujer","fechaNacimiento":"1987-01-15"}]
```

Después de todo esto, ahora sí volvemos a Android. Lo primero que vamos a hacer es introducir en el *Manifest* permisos para que la aplicación pueda acceder a Internet, de esta forma podremos concertar con nuestro servicio Web:

```
<uses-permission android:name="android.permission.INTERNET" />
```

Ahora vamos a crear una función que consiga conectar con el servicio web y pueda enviar/recibir datos:

Código Android

```
public class Persona extends Activity {
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
        try{
            HttpClient httpclient = new DefaultHttpClient();
            HttpPost httppost = new HttpPost("URL/persona.php");
            HttpResponse response = httpclient.execute(httppost);
            HttpEntity entity = response.getEntity();
            InputStream is = entity.getContent();
        }catch(Exception e){
            System.out.println("Error en la conexión");
            // En esta caputra de excepción podemos ver que hay un problema con la
            // conexión e intentarlo más adelante.
        }
        try{
            BufferedReader reader = new BufferedReader(new InputStreamReader
                (is,"iso-8859-1"),8);
```

```

StringBuilder sb = new StringBuilder();

sb.append(reader.readLine() + "\n");
String line="0";
while ((line = reader.readLine()) != null) {
    sb.append(line + "\n");
}
is.close();
String cadena =sb.toString();
} catch(Exception e){
    System.out.println("Error al obtener la cadena desde el buffer");
}

//Creamos los atributos de nuestra clase persona
String DNI; String nombre; String sexo; String fechaNac;
try{
    JSONArray jsonArray = new JSONArray(cadena);
    JSONObject jsonObject=null;
    for(int i=0;i<jsonArray.length();i++){
        jsonObject= jsonArray .getJSONObject(i);
        DNI=jsonObject.getString("DNI");
        nombre=jsonObject.getString("nombre");
        sexo=jsonObject.getString("sexo");
        fechaNac=jsonObject.getString("fechaNacimiento");
    }
}
catch(JSONException e1){
    Toast.makeText(getApplicationContext(), "No se encuentran los datos"
,Toast.LENGTH_LONG).show();
} catch (ParseException e1) {
    e1.printStackTrace();
}
}
}
}

```

Así obtendremos los valores en las variables DNI, nombre, sexo y fechaNac del contenido de la tabla que nos ha proporcionado el servicio Web. Con ello ya podremos usarlas en nuestra aplicación, mostrarlas en un ListView o todo aquello que veamos necesario.

Ahora bien, hasta lo que hemos visto, solo podemos pedirle al servicio Web datos pero ¿qué pasa si queremos introducir algún tipo de información? La respuesta es muy sencilla en php una de las cosas más comunes es pedir el usuario y la contraseña para acceder a la base de datos, o simplemente los datos a introducir en un *insert* o cualquier otro dato que necesitemos desde php. Para obtener datos externos desde PHP utilizamos \$_REQUEST:

Código PHP

```
<?php
    $host = $_REQUEST[host];
    $user = $_REQUEST[user];
    $password = $_REQUEST[password];
    mysql_connect($host , $user, $password);
// .....Nuestro código intermedio aquí.....
    mysql_close();
?>
```

Para introducir estos datos desde Android también es muy sencillo, sirviéndonos como código el ejemplo anterior, sólo anotamos aquí lo que deberemos añadir al HttpPost:

Código Android

```
//Creamos un ArrayList de Nombre Valor
// e introducimos los datos que deseamos pasar al web service
ArrayList<NameValuePair> nameValuePair = new ArrayList<NameValuePair>();
nameValuePair.add(new BasicNameValuePair("host","url de nuestro host"));
nameValuePair.add(new BasicNameValuePair("user","usuario de la BD"));
nameValuePair.add(new BasicNameValuePair("password","contaseña"));
// Añadimos los valores al HttpPost
HttpClient httpclient = new DefaultHttpClient();
HttpPost httpPost = HttpPost("URL/persona.php");
httpPost.setEntity(new UrlEncodedFormEntity(nameValuePair));
```


6. MAPAS Y GPS

Álvaro Borrego – Francisco Hernández – David Palomero

USO DE MAPAS

Para poder usar los mapas, Android provee de una librería externa que se encuentra en el paquete *com.google.maps*

¿COMO OBTENER LA API KEY PARA USAR GOOGLE MAPS?

Para poder acceder a los datos desde el MapView es necesario registrarse en el servicio de Google Maps y aceptar los términos de uso. Obtendremos una clave alfanumérica que nos dará acceso.

El registro para obtener la clave consta de dos partes:

1. Registrar la huella digital MD5 de la aplicación para que pueda acceder a los datos de Google Maps.
2. Agregar una referencia a la clave en cada MapView (en el XML o en código).

INFORMACIÓN GENERAL

Para asegurar que las aplicaciones utilizan los datos de manera adecuada, el MapView necesita una clave para poder usar la API. Esta clave es una secuencia alfanumérica que identifica la aplicación y el desarrollador. Sin esta clave, el MapView no podrá descargar los datos de los mapas.

Cada ApiKey de Google Maps es el único asociado a un certificado en concreto. Y cada MapView debe hacer referencia a un clave de la API (ApiKey).

Varias vistas Map puede referirse al mismo o distintos si se han registrado varios certificados a la misma aplicación.

¿CÓMO OBTENER LA HUELLA DIGITAL MD5 DEL CERTIFICADO?

Para generarla necesitamos usar la herramienta *keytool* del SDK de java. Los parámetros para el *keytool* se muestran en la siguiente tabla:

Tabla 1. Parámetros de Keytool

Opciones Keytool	Descripción
<code>-list</code>	Muestra la huella MD5 del certificado
<code>-keystore <keystore-name>.keystore</code>	El nombre del keystore que contiene la clave
<code>-storepass <password></code>	Una clave para el keystore.
<code>-alias <alias_name></code>	The alias for the key for which to generate the MD5 certificate fingerprint.
<code>-keypass <password></code>	The password for the key. As a security precaution, do not include this option in your command line unless you are working at a secure computer. If not supplied, Keytool prompts you to enter the password. In this way, your password is not stored in your shell history.

La siguiente figura muestra un ejemplo de la obtención de la clave.

```

C:\Windows\system32\cmd.exe
Microsoft Windows [Versión 6.1.7601]
Copyright (c) 2009 Microsoft Corporation. Reservados todos los derechos.

C:\Users\Fran>cd C:\Program Files (x86)\Java\jre6\bin
C:\Program Files (x86)\Java\jre6\bin>keytool -list -keystore "C:/Documents and Settings/Fran/.android/debug.keystore"
Escriba la contraseña del almacén de claves:

Tipo de almacén de claves: JKS
Proveedor de almacén de claves: SUN

Su almacén de claves contiene entrada 1

androiddebugkey, 05-dic-2011, PrivateKeyEntry,
Huella digital de certificado (MD5): 65:C8:3B:A4:E3:83:A0:70:07:70:C6:0D:CD

C:\Program Files (x86)\Java\jre6\bin>
    
```

Figura 1. Ejemplo de la obtención de la clave

REGISTRAR LA HUELLA MD5 EN EL SERVICIO DE GOOGLE MAPS

Cuando la tengamos, accedemos a la siguiente dirección Web:

<http://code.google.com/android/maps-api-signup.html>

Una vez en la página, leemos las condiciones, pegamos la huella digital, y generamos la clave.

Sign Up for the Android Maps API

The Android Maps API lets you embed [Google Maps](#) in your own Android applications. A single Maps API key for more information about application signing. To get a Maps API key for your certificate, you will need to provide the fingerprint of the certificate. If you are using Linux or Mac OSX, you would examine your debug keystore like this:

```
$ keytool -list -keystore ~/.android/debug.keystore
...
Certificate fingerprint (MD5): 94:1E:43:49:87:73:BB:E6:A6:88:D7:20:F1:8E:B5:98
```

If you use different keys for signing development builds and release builds, you will need to obtain a separate certificate for each, and provide the corresponding fingerprint.

You also need a [Google Account](#) to get a Maps API key, and your API key will be connected to your Google

```
companies of which Google is the parent will be third party
beneficiaries to the Terms and that such other companies will be
entitled to directly enforce, and rely upon, any provision of the
Terms that confers a benefit on (or rights in favor of) them. Other
than this, no other person or company will be a third party
beneficiary to the Terms.

18.6. The Terms, and your relationship with Google under the
Terms, will be governed by the laws of the State of California, USA,
without regard to its conflict of laws provisions. You and Google
agree to submit to the exclusive jurisdiction of the courts located in
the County of Santa Clara, California, USA, to resolve any legal
matter arising from the Terms. Notwithstanding this, you agree that
Google will be allowed to apply for injunctive remedies (or an
equivalent type of urgent legal relief) in any jurisdiction.
```

I have read and agree with the terms and conditions ([printable version](#))

My certificate's MD5 fingerprint:

Figura 2. Registro huella MD5 en Google Maps



Figura 3. Api de Google Maps

AÑADIR LA CLAVE DEL API DE ANDROID MAPS A NUESTRA APLICACIÓN Y AÑADIRLA AL LAYOUT

Al generar nuestra clave, ya hemos visto como añadirla desde el XML del layout:

```
<com.google.android.maps.MapView
  android:layout_width="fill_parent"
  android:layout_height="wrap_content"
  android:enabled="true"
  android:clickable="true"
  android:apiKey="example_Maps_ApiKey_String"
 />
```

Pero si quisiéramos añadirla desde una actividad, el código sería el siguiente:

```
MapView mMapView = new MapView(this, "example_Maps_ApiKey_String");
```

PASOS FINALES PARA HABILITAR EL USO DEL MAPVIEW

Hay que añadir la referencia a la librería externa com.google.android.maps en el Manifest. Será un hijo del elemento **<application>**:

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
  package="es.ucm.fdi.localyza"
  android:versionCode="1"
  android:versionName="1.0">
  ...
  <application android:icon="@drawable/icon"
    android:label="@string/app_name"
    android:theme="@style/OrangeTheme">
    <uses-library android:name="com.google.android.maps" />
```

Cuando la aplicación esté lista para ser distribuida en el Market, debermos modificar la clave de los mapas por la generada con el keystore con el que hemos firmado nuestra aplicación.

MOSTRAR EL ZOOM VIEW

Para añadir los controles de Zoom, solamente tenemos que llamar a la función `displayZoomControls`:

```
private MapView mapa;  
mapa = (MapView)findViewById(R.id.mapa);  
mapa.displayZoomControls(true);
```

Y con la siguiente función, se hace zoom o se aleja el mapa:

```
public boolean onKeyDown(int keyCode, KeyEvent event)  
{  
    MapController mc = mapa.getController();  
    switch (keyCode){  
        case KeyEvent.KEYCODE_3:  
            mc.zoomIn();  
            break;  
        case KeyEvent.KEYCODE_1:  
            mc.zoomOut();  
            break;  
    }  
    return super.onKeyDown(keyCode, event);  
}
```

CAMBIAR LAS VISTAS DEL MAPA

Activación de la vista satélite:

```
mapa.setSatellite(true);
```



Activación de la vista de calles:

```
mapa.setStreetView(true);
```



SITUAR LA POSICIÓN DEL MAPA EN UN PUNTO CONCRETO

Se define la longitud y latitud, creamos un GeoPoint, y mediante el MapController posicionamos la vista en las coordenadas.

```
mapa.setBuiltInZoomControls(true);  
Double latitud = 40.452723*1E6;  
Double longitud = -3.733253*1E6;  
GeoPoint loc =  
    new GeoPoint(latitud.intValue(), longitud.intValue());  
MapController controlMapa = mapa.getController();  
controlMapa.animateTo(loc);
```

AÑADIR CAPAS A LOS MAPAS

Primero se define la capa Overlay que pintará el icono.

```
class MapOverlay extends com.google.android.maps.Overlay  
{  
    ...  
}
```

Y se redefine el método draw:

```
public boolean draw(Canvas canvas, MapView mapView, boolean shadow,  
long when)
```

```

{
    super.draw(canvas, mapView, shadow);
    //---transformamos el geopoint a pixeles---
    Point screenPts = new Point();
    mapView.getProjection().toPixels(loc, screenPts);

    //--- añadimos el marcador---
    Bitmap bmp = BitmapFactory.decodeResource(
        getResources(), R.drawable.point_mor);

    canvas.drawBitmap(bmp, screenPts.x, screenPts.y-50, null);
    return true;
}

```

Ahora, solo falta crear una lista de MapsOverlays e indicar sobre que mapView ha de pintarse:

```

//---Añadimos el marcador de la localización---

MapOverlay mapOverlay = new MapOverlay();
List<Overlay> listOfOverlays = mapa.getOverlays();
listOfOverlays.clear();
listOfOverlays.add(mapOverlay);

mapa.invalidate();

```



GPS

Actualmente, existe una gran cantidad de aplicaciones basadas en localización, y día a día siguen aumentando. Actualmente, existen treinta y un satélites sin nada mejor que hacer que proveernos de estos servicios.

SOLICITUDES DE PERMISOS

```
<manifest ... >
  <uses-permission
    android:name="android.permission.ACCESS_FINE_LOCATION" />
  ...
</manifest>
```

OBTENIENDO LAS ACTUALIZACIONES DE LA LOCALIZACIÓN

Para obtener la localización en Android funciona por medio de la devolución de la llamada. Lo recibimos por medio de las actualizaciones del LocationManager, llamando al método requestLocationUpdates, y pasándole un LocationListener.

```
// Obtención de una referencia al LocationManager
    LocationManager locationManager = (LocationManager)
    this.getSystemService(Context.LOCATION_SERVICE);

    //Definición del listener que responde a las actualizaciones
    de la localización.

    LocationListener locationListener = new LocationListener() {

        public void onStatusChanged(String provider, int status,
        Bundle extras) {}

        public void onProviderEnabled(String provider) {}

        public void onProviderDisabled(String provider) {}

        public void onLocationChanged(Location location) {

            //Que hacer con la nueva localización

        }

    };

    // Registrar el listener con el LocationManager para recibir
    actualizaciones

    locationManager.requestLocationUpdates(LocationManager.NETWORK_PROVIDE
    R, 0, 0, locationListener);
```

En esta última línea, tenemos que indicar si queremos usar la red o los GPS como proveedor sustituyendo

```
LocationManager.NETWORK_PROVIDER
```

por

```
LocationManager.GPS_PROVIDER.
```

Los otros dos parámetros son:

- `minTime`: es el mínimo intervalo de tiempo para las notificaciones en milisegundos. Se usa para evitar hacer demasiadas peticiones al servicio de localización y ahorrar batería.
- `minDistance`: el mínimo intervalo de distancia para las notificaciones, en metros.

Para detener las actualizaciones, tenemos que eliminar el listener que previamente habíamos indicado al `LocationManager`:

```
locationManager.removeUpdates(locationListener);
```

OBTENER LA ÚLTIMA LOCALIZACIÓN CONOCIDA

En ocasiones, podemos querer una rápida localización y en un primer instante, puede ser útil utilizar las últimas coordenadas válidas. Podemos obtenerlas de la siguiente forma:

```
locationProvider = LocationManager.NETWORK_PROVIDER;  
Location lastKnownLocation =  
locationManager.getLastKnownLocation(locationProvider);
```

GEOCODER

Es una clase que permite transformar una dirección o descripción de un lugar en las coordenadas (longitud y latitud). La geo codificación inversa es el proceso de transformar las coordenadas en una dirección parcial. La cantidad de detalles obtenidos puede variar según el lugar.

¿CÓMO OBTENER EL LUGAR A PARTIR DE UNAS COORDENADAS?

```
Geocoder gc=new Geocoder(getApplicationContext());  
List<Address> address= gc.getFromLocation (latitude, longitude,  
maxResults);
```

¿CÓMO OBTENER LUGARES A PARTIR DE UNA DESCRIPCIÓN?

```
Geocoder gc=new Geocoder(getApplicationContext());  
List<Address> address=gc.getFromLocationName(locationName,  
maxResults);
```

7. TELEFONÍA

Manuel Báez – Jorge Cordero – Miguel González

MENSAJES DE TEXTO

La posibilidad de enviar y recibir mensajes de texto provocó una revolución en los teléfonos móviles, siendo una de las principales formas de comunicación durante la primera década del SXXI hasta la llegada del internet móvil. Como es de esperar Android nos da la posibilidad de enviar SMS. El encargado de ello es el `Android.telephony.SmsManager` que soporta tanto GSM(*Groupe spéciale mobile*) como CDMA(*Code Division Multiple Access*).

Dicha clase tiene únicamente cinco métodos públicos que son los siguientes.

- `ArrayList<String> divideMessage(String text)`

Su función es la de recibir como parámetro un *String text* y convertirlo en un *ArrayList* de Strings. El tamaño de estos Strings será menor o igual al máximo permitido por un SMS

- `static SmsManager getDefault()`

Devuelve la instancia por defecto de `SmsManager`

- `void sendDataMessage(String destinationAddress, String scAddress, short destinationPort, byte[] data, PendingIntent sentIntent, PendingIntent deliveryIntent)`

Es el método que se encarga de enviar el SMS. Le pasamos como parámetro la dirección de destino, el puerto de destino, el SMS en sí y dos *PendingIntent* que sirven para el envío y el acuse de recibo.

- `void sendMultipartTextMessage(String destinationAddress, String scAddress, ArrayList<String> parts, ArrayList<PendingIntent> sentIntents, ArrayList<PendingIntent> deliveryIntents)`

En el caso de tener un SMS de longitud mayor que lo permitido para uno tendremos que usar este método. Su funcionamiento es el mismo que el de *sendDataMessage*, la diferencia es que hay un array list para cada *PendingIntent* así como para cada parte(parts).

- `void sendTextMessage(String destinationAddress, String scAddress, String text, PendingIntent sentIntent, PendingIntent deliveryIntent)`

Vamos a ver ahora como enviar un SMS sirviéndonos de dichos métodos.

Empezamos suponiendo que tenemos ya el texto que queremos enviar en un `String text` y el número al que lo queremos enviar en un `String phoneNumber`, también necesitaremos un `PendingIntent` que lo obtendremos del siguiente modo:

```
PendingIntent pi = PendingIntent.getActivity(this, 0, new Intent(this, SMS.class), 0);
```

Ahora vamos a utilizar `SmsManager.getDefault()` para tener la instancia por defecto de `SmsManager` que llamaremos `sms`.

Tenemos dos opciones ante nosotros, que el texto quepa en un SMS o que no, en el caso de que quepa, la solución es más sencilla, solamente tendremos que llamar a `sms.sendTextMessage(phoneNumber, null, text, pi, null)` como podemos apreciar hemos dejado el campo de `scAddress` como `null`, para que tome un valor por defecto y el de `deliveryIntent` también.

En el caso de que `text` no quepa en un único SMS tendremos que dividir el texto del SMS en varios mensajes, para ello utilizaremos el método `sms.divideMessage(text)` y guardaremos el resultado obtenido en `arrayText`. Ahora procederemos a enviar el mensaje al número deseado, en este caso usaremos el método `sms.sendMultipartTextMessage(phoneNumber, null, arrayText, pi, null)` y ya habremos enviado el mensaje múltiple.

Es importante destacar que en el caso de los sms y mms, al ser información que ha de ser accesible desde cualquier aplicación, es necesario que se almacenen como `Content Providers`, y desde la versión 2.0+ es así. Están almacenados en la ruta `/data/data/com.android.providers.telephony/databases/mmsms.db`, y podemos ver cómo acceder a ellos en el punto del manual de los [Content Providers](#).

Es importante destacar que el URI de los sms es el siguiente

Código Android

```
String url = "content://sms/";  
Uri uri = Uri.parse(url);  
getContentResolver().registerContentObserver(uri, true, new  
MyContentObserver(handler));
```

LLAMADAS A TELÉFONO

La razón por la que se crearon los teléfonos móviles era la de poder tener una línea no fija desde la que pudiéramos hacer llamadas desde cualquier punto, siempre y cuando haya cobertura por supuesto, sin necesidad de cables ni nada por el estilo. Por ello es lógico suponer e incluso obvio que Android nos de la posibilidad de llamar desde una aplicación propia. A continuación vamos a ver cómo realizar dichas llamadas. Para poder explicarlo primero tendremos que explicar qué es un Intent y cómo usarlos con un startActivity.

Un Intent (intento) es un paquete, dicho paquete contiene información del componente que reciba el intento. La constructora de Intent que vamos a utilizar en este caso es de la forma Intent(String action, Uri uri) siendo el primer argumento, action, la acción a realizar y el segundo, uri, el dato sobre el que realizar dicha acción. Por su parte startActivity(Intent), hay otras formas de llamar a startActivity pero la que nos interesa es ésta, lo que hace es intentar ejecutar la actividad determinada por Intent, poniéndola en la cima de la pila de actividad.

A continuación tenemos un fragmento de código que lo que muestra un AlertDialog cuando pulsamos sobre un contacto, el cual recibimos por parámetro en este método. Dicho AlertDialog tiene varias opciones de contacto entre ellas está la de “Llamar” cuyo código está enmarcado en rojo.

Código Android

```
private void launchContactOptions(final ContactListItem contacto) {  
    // Elementos del Dialog  
    final String items[] = {"Llamar","Otro"};  
    AlertDialog.Builder dialog = new AlertDialog.Builder(ContactosActivity.this);  
    dialog.setTitle("Opciones de contacto");  
    dialog.setItems(items, new DialogInterface.OnClickListener() {  
        public void onClick(DialogInterface d, int choice) {  
            switch (choice) {  
                case 0: // Éste es el código que nos interesa  
                    //Elegida opción “Llamar” del AlertDialog  
                    String url = "tel:" + contacto.getNum();  
                    Intent intent = new Intent(Intent.ACTION_CALL, Uri.parse(url));  
                    ContactosActivity.this.startActivity(intent);  
                    break;  
                case 1:  
                    //Elegida opción “Otro” del AlertDialog
```

```

        break;
        default:
        break;
    }
}
});

dialog.show();
}

```

Registro de llamadas (CallLog) e información de las llamadas.

Para obtener la lista de llamadas tendremos que llamar al `android.provider.CallLog` el cual se encarga de guardar un registro de las llamadas tanto recibidas como realizadas. Algunos campos importantes del registro CallLog son:

Campo	Descripción
<code>android.provider.CallLog.Calls.NUMBER</code>	El número involucrado
<code>android.provider.CallLog.Calls.TYPE</code>	Tipo de llamada. Entrante (1) o saliente (0).
<code>android.provider.CallLog.Calls.CACHED_NAME</code>	Nombre asociado con el número en caso de existir en la agenda.
<code>android.provider.CallLog.Calls.DATE</code>	Fecha de la llamada
<code>android.provider.CallLog.Calls.DURATION</code>	Duración de la llamada en segundos

Código ejemplo: Consulta de llamadas salientes de duración superior a 0 segundos.

Código Android

```
ArrayList<String[]> result = new ArrayList<String[]>();

// Consulta SQL
    String[] strFields = {
        android.provider.CallLog.Calls.NUMBER,
        android.provider.CallLog.Calls.TYPE,
        android.provider.CallLog.Calls.CACHED_NAME,
        android.provider.CallLog.Calls.DATE,
        android.provider.CallLog.Calls.DURATION
    };

//Condición llamadas salientes:
    String where = new String("1=" + android.provider.CallLog.Calls.TYPE + "");
    where += " AND " + android.provider.CallLog.Calls.DURATION + ">0";
    String strOrder = android.provider.CallLog.Calls.DATE + " DESC";

    Cursor mCursor = getContentResolver().query(
        android.provider.CallLog.Calls.CONTENT_URI,
        strFields,
        where,
        null,
        strOrder
    );

// Iteramos sobre la lista de resultados a través del cursor.
    int nameIndex =
mCursor.getColumnIndexOrThrow(android.provider.CallLog.Calls.CACHED_NAME);
    int numberIndex =
mCursor.getColumnIndexOrThrow(android.provider.CallLog.Calls.NUMBER);
    int typeIndex =
mCursor.getColumnIndexOrThrow(android.provider.CallLog.Calls.TYPE);
```

```

        int dateIndex =
mCursor.getColumnIndexOrThrow(android.provider.CallLog.Calls.DATE);

        int durIndex =
mCursor.getColumnIndexOrThrow(android.provider.CallLog.Calls.DURATION);

        if (mCursor.moveToFirst()) {
            do {
                String name = mCursor.getString(nameIndex);
                String number = mCursor.getString(numberIndex);
                String type = mCursor.getString(typeIndex);
                String date = mCursor.getString(dateIndex);
                String dur = mCursor.getString(durIndex);
                result.add(new String{type, name, number, date, dur});

            } while (mCursor.moveToNext());
        }
    }
}

```

ACCEDER A LA AGENDA

Accedemos a la agenda mediante la clase `android.provider.ContactsContract`, Disponible desde la versión 2.0+ (API Level 5). Para poder acceder a los contactos debemos añadir permisos a la aplicación en el archivo `AndroidManifest.xml`.

Código Android Manifest.xml

```
<uses-permission android:name="android.permission.READ_CONTACTS" />
```

Para obtener datos de la agenda de contactos almacenamos una instancia del `ContentResolver` y sobre este ejecutamos `query()`. La llamada al método `query()` tiene como primer argumento la URI que identifica el conjunto de datos sobre el que queremos ejecutar la query. Como segundo argumento se introduce un array con las columnas que queremos obtener. Adicionalmente se pueden añadir condiciones a la consulta y el orden en el queremos que devuelva los resultados. A continuación un método que obtiene información de los contactos y devuelve en un array con el nombre y número de los contactos que tienen número de móvil.

Código Android

```
public ArrayList<String[]> dameContactos() {
    ArrayList<String[]> result = new ArrayList<ContactListItem>();
    ContentResolver cr = getContentResolver();
    // Consulta SQL
    Cursor mCursor = cr.query(
        Data.CONTENT_URI,
        //Columnas a seleccionar
        new String[] { Data.CONTACT_ID, Data.DISPLAY_NAME,
            Phone.NUMBER, Phone.TYPE,
            ContactsContract.CommonDataKinds.Photo.CONTACT_ID},
        //Condiciones
        Phone.NUMBER + " IS NOT NULL",
        null,
        //Orden
        Data.DISPLAY_NAME + " ASC");
    startManagingCursor(mCursor);

    // Iteramos sobre la lista de resultados a través del cursor.
    int nameIndex = mCursor.getColumnIndexOrThrow(Data.DISPLAY_NAME);
    int numberIndex = mCursor.getColumnIndexOrThrow(Phone.NUMBER);
    int contactoIndex = mCursor.getColumnIndex(Data.CONTACT_ID);

    if (mCursor.moveToFirst()) {
        do {
            String name = mCursor.getString(nameIndex);
            String number = mCursor.getString(numberIndex);
            result.add(new String[] {name, number});
        } while (mCursor.moveToNext());
    }
    return result;
}
```


8. SENSORES

Manuel Báez – Jorge Cordero – Miguel González

Android dispone del paquete *Android.Hardware* para dar soporte a la cámara del dispositivo y a otros sensores.

<uses-feature>

Los sensores utilizados por una aplicación deberán de estar declarados en el AndroidManifest.xml haciendo uso de la etiqueta <uses-feature> para que la aplicación pueda hacer uso de ellos. Esta información es utilizada por el Android Market para saber si una aplicación es compatible con un dispositivo concreto.

Por ejemplo si la aplicación que queremos desarrollar necesita usar el acelerómetro, incluiremos una línea en el AndroidManifest.xml e indicaremos que es requerido (atributo android:required) como se muestra en el siguiente código.

Código de AndroidManifest.xml

```
...  
<uses-feature android:name="android.hardware.sensor.accelerometer" android:required="true" />  
<uses-feature android:name="android.hardware.bluetooth" android:required="false"/>  
<uses-feature android:name="android.hardware.camera"/>  
...
```

El atributo android:required está por defecto a *true*, por lo que si no lo incluimos Android entenderá que el hardware indicado será necesario para la aplicación como sucede con la cámara en el código anterior.

Las clases e interfaces disponibles en android.hardware para el control de sensores, son:

Nombre	Tipo	Descripción
SensorManager	Clase	Gestiona los sensores del dispositivo.
Sensor	Clase	Representa a un sensor
SensorEvent	Clase	Representa el evento de un gestor
SensorEventListener	Interfaz	Recibe los cambios de un sensor.

Para hacer uso de un sensor hay que implementar la clase SensorEventListener y obtener una instancia del SensorManager. Esta instancia la utilizaremos para indicar que queremos registrar

los cambios del sensor mediante el metodo `registerListener(SensorEventListener listener, Sensor sensor, int rate, Handler handler)` del `SensorManager`. El primer argumento de este método es la instancia de la clase que implementa la interfaz `SensorEventListener`. El segundo parámetro es el sensor que queremos registrar. El último parámetro es para indicar la frecuencia ideal con la que nos gustaría registrar el sensor, la cual puede no corresponder con la real.

Una vez indicado al `SensorManager` que sensor vamos a registrar disponemos de dos métodos para manejar los eventos registrados:

- `public void onAccuracyChanged(Sensor sensor, int accuracy)`
- `public void onSensorChanged(SensorEvent event)`

Una vez queramos dejar de registrar los cambios en los sensores debemos llamar a la función `unregisterListener(SensorEventListener listener)` de la instancia del `SensorManager`.

El `SensorEvent` contiene todos los datos en relación al evento.

Tipo	Nombre	Descripción
public Sensor	sensor	Sensor implicado en el evento
public int	accuracy	Precisión del sensor
public long	timestamp	Tiempo (ns) en el que ocurrió el evento
public final float[]	values	Contiene la información del sensor dependiendo del sensor

Método	Valor devuelto
<code>public String getName ()</code>	Nombre del sensor
<code>public float getMaximumRange ()</code>	Valor máximo que alcanza el sensor
<code>public float getPower ()</code>	Potencia (mA) consumida por el sensor cuando está en funcionamiento
<code>public int getMinDelay ()</code>	Tiempo (ns) mínimo que transcurre entre dos eventos de un Sensor
<code>public int getType()</code>	Tipo del sensor
<code>public int getVersion()</code>	Versión del sensor

public float getResolution()	Resolución del sensor
------------------------------	-----------------------

Ejemplo de código:

```
public class miActividad extends Activity implements SensorListener {
    SensorManager miSensorManager = null;
    //... resto de código
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        //... resto de código

        //Obtenemos la instancia del SensorManager
        miSensorManager = (SensorManager)
            getSystemService(SENSOR_SERVICE);
    }

    @Override
    public void onSensorChanged(SensorEvent event) {
        if (event.getType() == Sensor.TYPE_ACCELEROMETER) {
            //Tratamiento del evento si es
        }
        //Tratamiento de eventos de otros sensores
    }

    @Override
    public void onAccuracyChanged(Sensor sensor, int accuracy) {
    }

    @Override
    protected void onResume() {
        super.onResume();
        //Indicamos al sensor manager que registre los cambios del acelerómetro
        miSensorManager.registerListener(this,
            SensorManager.SENSOR_ACCELEROMETER,
            SensorManager.SENSOR_DELAY_NORMAL);
    }

    @Override
    protected void onStop() {
        //D
        miSensorManager.unregisterListener(thejamos de registrar los cambios del sensor);
        super.onStop();
    }
}
```

GESTOS

Android es un sistema operativo para dispositivos móviles, donde el puntero clásico de los sistemas operativos convencionales pasa a ser el contacto táctil del usuario en la pantalla. Podemos ver que algunas aplicaciones tienen funcionalidades distintas cuando un usuario toca la pantalla, incluso variando en función de la duración o forma de dicho toque. Ésto es lo que se llama un gesto en Android y vamos a ver cómo implementarlos.

Estos gestos dan a Android un gran potencial a la hora de poder realizar un gran abanico de acciones con un escaso número de gestos, hay aplicaciones que su única funcionalidad consiste en ahorrarnos tiempo haciendo que ciertos gestos se conviertan en una especie de accesos directos. Incluso hay un teclado llamado Swype que es un teclado mediante gestos haciendo que sea mucho más ágil que un teclado tradicional.

Hay varios tipos de gestos que podemos capturar, para ello tendremos que utilizar `GestureDetector.OnGestureListener` que nos notificará cuando salta un evento dentro del listener. Tendremos a nuestra disposición los siguientes subtipos del listener:

Métodos:

abstract boolean	<code>onDown(MotionEvent e)</code>	Notifica cuando se produce un down devolviéndolo en el <code>MotionEvent e</code> .
abstract boolean	<code>onFling(MotionEvent e1, MotionEvent e2, float velocityX, float velocityY)</code>	Notifica una pulsación mantenida devolviendo el primer punto en <code>MotionEvent e1</code> y el final en <code>MotionEvent e2</code> .
abstract void	<code>onLongPress(MotionEvent e)</code>	Notifica cuando se produce una pulsación larga devolviendo los datos en <code>MotionEvent e</code> .
abstract boolean	<code>onScroll(MotionEvent e1, MotionEvent e2, float distanceX, float distanceY)</code>	Notifica cuando se está moviendo un scroll devolviendo los valores de la posición inicial en <code>MotionEvent e1</code> y la final en <code>MotionEvent e2</code>
abstract void	<code>onShowPress(MotionEvent e)</code>	Indica que se ha seleccionado un elemento pero no se ha dejado de pulsar todavía, para mostrar al usuario se ha reconocido su interacción.
abstract boolean	<code>onSingleTapUp(MotionEvent e)</code>	Notifica cuando se produce un up devolviéndolo en <code>MotionEvent e</code>

A continuación vamos a ver un ejemplo con el `onFling` haciendo el gesto de izquierda a derecha:

Código Android

```
public class ContactosActivity extends Activity {
    private static final int SWIPE_MIN_DISTANCE = 120;
    private static final int SWIPE_MAX_OFF_PATH = 250;
    private static final int SWIPE_THRESHOLD_VELOCITY = 200;
    private ContactArrayAdapter listAdapter = null;
    private GestureDetector gestureDetector = null;
    private ContactosActivity actividad = null;
    //private ProgressDialog progressDialog = null;
    class MyGestureDetector extends SimpleOnGestureListener {
        @Override
        public boolean onFling(MotionEvent e1, MotionEvent e2, float velocityX,
            float velocityY) {
            Intent intent = new Intent(ContactosActivity.this, GruposActivity.class);
            if (Math.abs(e1.getY() - e2.getY()) > SWIPE_MAX_OFF_PATH) {
                return false;
            }
            // right to left swipe
            else if (e1.getX() - e2.getX() > SWIPE_MIN_DISTANCE
                && Math.abs(velocityX) >
                SWIPE_THRESHOLD_VELOCITY) {
                // Tu código para cuando hagas el gesto de izquierda a derecha
            }
            return false;
        }
        // It is necessary to return true from onDown for the onFling event to
        // register
        @Override
        public boolean onDown(MotionEvent e) {
            return true;
        }
    }
}
```


9. MULTIMEDIA

Daniel Sanz – Mariam Saucedo – Pilar Torralbo

MULTIMEDIA EN ANDROID

Hoy en día, los dispositivos móviles no solo se usan para hablar y mandar mensajes de texto. La gran mayoría de los mismos permiten, grabar y reproducir audio y video, así como realizar fotografías. En este tema, se intenta presentar todas las posibilidades mencionadas anteriormente, proporcionadas por Android, así como dar a conocer algunos ejemplos de implementación de pequeñas funciones para reproducir audio o hacer fotografías.

El Framework¹ proporcionado por Android para multimedia, permite capturar y reproducir audio, vídeo o imágenes en los formatos más comúnmente usados, pudiendo interactuar con archivos contenidos en el propio terminal Android, con archivos externos al dispositivo, o con aquellos dispuestos a través de Internet. Para la reproducción utiliza MediaPlayer o JetPlayer, siendo MediaRecorder el usado para grabar audio o vídeo, o hacer fotografías a través de la cámara del terminal. Android soporta varios formatos de audio, video e imágenes. Algunas de ellas como mp3, midi y flac para audio; 3gp y mpeg para videos; y jpg y png entre otros para imágenes.

REPRODUCCIÓN DE AUDIO

En la reproducción de audio, Android proporciona sus propias clases destinados a ello. A continuación, se describirá como usar la clase "**MediaPlayer**" para programar en nuestra propia aplicación y así crear un reproductor propio.

Para poder usar la clase "MediaPlayer", primero hay que especificar los permisos necesarios para que no dé problemas a la hora de compilar el programa. Para ello, en el AndroidManifest se incluye lo siguiente:

```
<uses-permission android:name="android.permission.INTERNET" />
```

Esto permitirá la conexión a Internet en el caso de que se reproduzca audio o video extraído de la web. Para llevar un mejor manejo de la reproducción, se puede incluir un par de botones como son "start" y "pause"

⁽¹⁾ Framework es un término que significa “marco”. Es decir, proporciona un conjunto de funciones para implementar aplicaciones con características semejantes.

Si además se quiere poder reproducir multimedia con el dispositivo en suspensión o bloqueado (usando los métodos `setScreenOnWhilePlaying()` o `setWakeMode()` que más adelante serán explicados), se añadirá la línea:

```
<uses-permission android:name="android.permission.WAKE_LOCK" />
```

Como ya se ha indicado antes, MediaPlayer permite reproducir audio o video desde diferente puntos:

- Recursos locales (los cuales se encontrarán guardados en la carpeta "raw" del proyecto).
- URI interna (Identificador Uniforme de Recurso en inglés, ya sea guardado en el dispositivo o en una tarjeta SD).
- URL externa (obtenida de una página web).

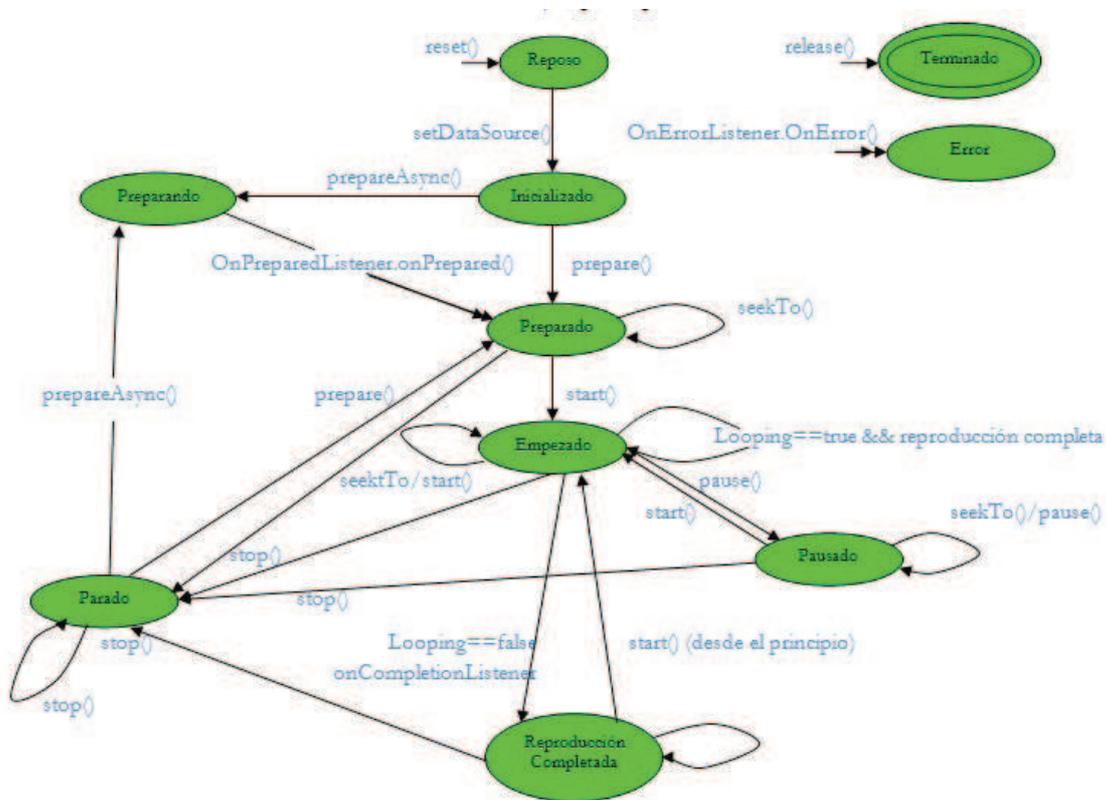


Figura 1. Funcionamiento de la clase MediaPlayer

El funcionamiento de esta clase funciona como una máquina de estados. Cuando el objeto MediaPlayer es creado mediante `new()` o después de llamar a la función `release()`, se dice que está en un estado de reposo. Hasta llegar a este estado mediante una de las dos llamadas, el objeto pasará por todo su ciclo de vida.

El ciclo de vida de una reproducción sin tener errores pasaría por los siguientes estados: **Reposo** (cuando se crea el objeto MediaPlayer), **Inicializado** (se le indica al objeto cuál va a ser la pista o video a reproducir), **Preparado** (el MediaPlayer ya tiene todo listo para reproducir), **Empezado** (MediaPlayer está reproduciendo), **Parado** (la pista terminó y MediaPlayer espera una nueva instrucción).

Para poder usar un objeto MediaPlayer, se invoca la constructora por defecto que proporciona Android. Una vez construido el objeto, se indica desde donde va a reproducir la pista. En el caso de que se quiera reproducir desde la carpeta raw del proyecto, hay que especificarle el lugar (en este caso la carpeta raw) y el nombre del archivo que se quiera reproducir:

```
MediaPlayer mp = new MediaPlayer();
mp = MediaPlayer.create(R.raw.cancion_prueba.mp3);
mp.start();
```

Si el archivo que se va a usar se encuentra en la memoria del dispositivo o en una tarjeta SD, mediante una variable de tipo URI se indica la ruta del teléfono en la cual se encuentra el archivo a reproducir:

```
Uri miUri = .... ;//Aquí se inicializa la variable apuntando al archivo deseado
MediaPlayer mp = new MediaPlayer();
mp.setDataSource(getApplicationContext(), miUri);
mp.prepare();
mp.start();
```

Si por el contrario el archivo se desea reproducir desde una url, mediante una conexión http indicamos al objeto desde donde descargarlo (hay que tener en cuenta que primero debe descargar el archivo, por lo que puede tardar un tiempo dependiendo de la velocidad de conexión):

```
String url = "http://...."; // Aquí se introduce la url desde la que operar
MediaPlayer mp = new MediaPlayer();
mp.setDataSource(url);
mp.prepare();
mp.start();
```

Como se ha podido comprobar, tanto para reproducir desde una URI interna como desde una url externa, es necesario preparar el MediaPlayer para obtener el archivo, al contrario que con un archivo local que se encuentra dentro del proyecto y listo para usarse. En el caso de querer manejar también el audio de la reproducción, se incluirá la clase "AudioManager" la cual primero hay que escribir los permisos necesarios para poder utilizarla:

```
<accion android:name="android.media.AUDIO_BECOMING_NOISY" />
```

Y así poder obtener y manejar a nuestro antojo el volumen de la aplicación. En el siguiente ejemplo se puede observar la forma de obtener el volumen máximo permitido por la aplicación y el volumen actual:

```
AudioManager aManager = (AudioManager) getSystemService(AUDIO_SERVICE);
//Captura las especificaciones del volumen de la aplicación
float actualVolumen = (float) aManager.getStreamVolume(aManager.STREAM_MUSIC);
float maxVolumen = (float) aManager.getStreamMaxVolume(aManager.STREAM_MUSIC);
```

Una vez obtenidos estos datos, se podrá manejar el volumen de forma sencilla.

REPRODUCCIÓN DE VIDEO

Para programar un reproductor de video en nuestra aplicación, será necesario incluir en la interfaz (archivo .xml) un control de "VideoView" que se encuentra en la paleta de "Images & Media". Para llevar un mejor manejo de la reproducción del video, además podemos añadir un par de botones para reproducir y pausar el video.

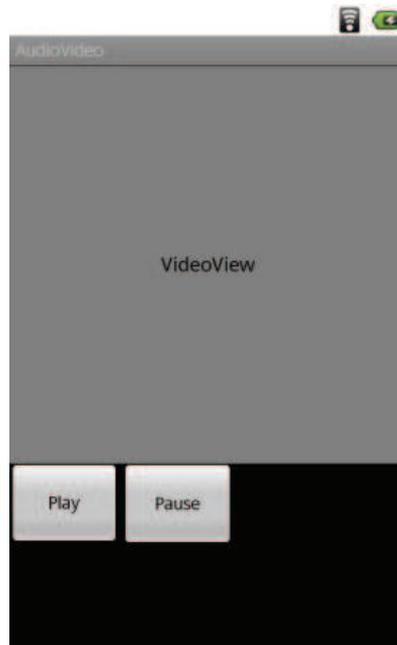


Figura 2. Reproducción de video

En este caso, podremos reproducir un video que esté alojado en la tarjeta SD de nuestro dispositivo (URI) o a través de una URL externa de la web. Para ello, hay que diferenciar los dos métodos de linkado para reproducir uno u otro video.

Para la reproducción a través de un objeto de tipo URI, escribimos las siguientes instrucciones:

```
Uri uri = ....; //Aquí se incluye la ruta del teléfono donde se encuentre el video
VideoView video = ..... //linkado con su id correspondiente al archivo .xml
video.setVideoUri(uri);
```

Y en el caso de que reproduzca a través de una url externa:

```
String url = ....; //URL desde donde se descargará el video
VideoView video = .....;
video.setVideoPath(url);
```

Para terminar, proporcionamos a los dos botones un `setOnClickListener()` para que cada uno realice su función. En el caso del botón play, la instrucción a escribir será `video.play()`. Y en el caso del botón pause la instrucción será `video.pause()`.

GRABAR SONIDO

El framework multimedia de Android incluye un soporte que permite capturar y codificar una gran variedad de formatos comunes de audio, de modo que estos se puedan integrar fácilmente en la aplicación. Para ello, se puede usar la API "MediaRecorder" siempre y cuando sea compatible con el hardware del dispositivo (que tenga micrófono o no). Sin embargo, el emulador de Android no posee la capacidad de capturar audio, por lo que toda prueba que se precie se deberá hacer en un dispositivo que disponga de micrófono.

Antes de nada, se deberán incluir los permisos necesarios para poder escribir desde una entrada externa en la tarjeta SD y para poder grabar audio. En el archivo AndroidManifest se escribirá:

```
<uses-permission  
android:name="android.permission.WRITE_EXTERNAL_STORAGE" />  
  
<uses-permission android:name="android.permission.RECORD_AUDIO" />
```

La captura de audio desde el dispositivo es más complicado que la reproducción de video o audio. Aún así, se puede programar de forma sencilla. En primer lugar, hay que crear un objeto de tipo MediaRecorder. Del mismo modo se puede crear otro objeto MediaPlayer (visto en este tema), para reproducir la pista capturada:

```
MediaRecorder mRecorder = new MediaRecorder();  
  
MediaPlayer mPlayer = new MediaPlayer();
```

A continuación, se definirá el micrófono como medio para capturar el audio y se define que el archivo se va a guardar con la especificación 3GPP, con extensión .3gp y el codec que se va a emplear:

```
mRecorder.setAudioSource(MediaRecorder.AudioSource.MIC);  
  
mRecorder.setOutputFormat(MediaRecorder.OutputFormat.THREE_GPP);  
  
mRecorder.setAudioEncoder(MediaRecorder.AudioEncoder.AMR_NB);
```

Una vez en este punto, hay que indicar la dirección de la tarjeta SD; indicándole al objeto MediaRecorder que, una vez realizada la captura de audio, debe almacenarse en esta dirección. A su vez, creamos un archivo temporal 3gp almacenado en la tarjeta SD donde se guardará el audio:

```
File direccion = new File(Environment.getExternalStorageDirectory().getPath());  
  
File archivo = File.createTempFile("temporal", ".3gp", direccion);  
  
mRecorder.setOutputFile(archivo.getAbsolutePath());
```

Ahora, ya podemos preparar la captura y empezar con ella. Y pararla una vez se haya terminado de grabar lo deseado:

```
mRecorder.prepare();  
  
mRecorder.start();
```

```
.....  
mRecorder.stop();  
mRecorder.release();
```

Al igual que se ha visto en la reproducción de video, se puede incluir en la interfaz unos botones para tener un mejor control del inicio, parado y reproducción de la grabación de audio.

GRABAR VIDEO

Al igual que pasaba en el apartado anterior, para la captura de video se necesita un dispositivo que disponga de una cámara. También se usará un objeto de tipo "MediaRecorder" solo que esta vez le indicaremos que la captura se obtendrá de la cámara. En este caso, a parte de los permisos que se declararon para la grabación de audio, hay que añadirle los permisos para grabar video y para usar la cámara:

```
<uses-permission  
android:name="android.permission.WRITE_EXTERNAL_STORAGE" />  
  
<uses-permission android:name="android.permission.RECORD_AUDIO" />  
  
<uses-permission android:name="android.permission.RECORD_VIDEO" />  
  
<uses-permission android:name="android.permission.CAMERA" />
```



Figura 3. Grabación de video

En la interfaz, hay que añadir un "SurfaceView", el cual permitirá tener un espacio dedicado a dibujar frames del vídeo para la vista y la reproducción (en la actividad principal hay que implementar la interfaz "SurfaceHolder.Callback"). Para este apartado, servirá para la vista previa de la grabación y para reproducirla posteriormente. Además, se podrá añadir tres botones que

implementen las funciones grabar, detener y reproducir del "SurfaceView". Como consejo, se puede configurar la rotación de la pantalla para que se quede bloqueada y no re-crea la actividad al girar el dispositivo. Para se escribe en el manifiesto la siguiente línea:

```
android:screenOrientation="portrait"
```

En esta aplicación se definirán cuatro variables importantes, el "MediaPlayer" y "MediaRecorder" vistos anteriormente y una variable con el nombre del archivo donde se va a guardar la grabación.

De los métodos heredados de la interfaz, importan dos de ellas. El método "surfaceCreated", en el cual se inicializaran las variables y se propone el "SurfaceHolder" como display para la grabación y la reproducción. Y el método "surfaceDestroyed", en el que se liberarán las variables usando el método `release()` de éstas.

```
mRecorder.setPreviewDisplay(holder.getSurface());  
  
mPlayer.setDisplay(holder);
```

Ahora se necesitará agregar la cámara para que el objeto "MediaRecorder" use ésta para la grabación. Se configura las fuentes de audio y video. Y se llama a los métodos `unlock()` y `lock()` de la cámara. Esto sirve para que ninguna otra aplicación pueda utilizar la cámara mientras esté la nuestra en funcionamiento. El primero se llamará cuando se quiera usar la cámara, y el segundo cuando se haya terminado.

```
mCamara = getCameraInstance();  
  
mCamara.unlock();  
  
mRecorder.setCamera(mCamara);  
  
mRecorder.setAudioSource(MediaRecorder.AudioSource.CAMCORDER);  
  
mRecorder.setVideoSource(MediaRecorder.VideoSource.CAMERA);  
  
mRecorder.setOutputFile(getOutputMediaFile(MEDIA_TYPE_VIDEO).toString());
```

Una vez llegados a este punto, ya se pueden configurar los botones para el uso de la grabación, usando los comandos `start()` y `stop()` del "MediaRecorder" para tener manejo sobre la grabación del video, los mismos comandos para el "MediaPlayer" para la previsualización de la grabación y, una vez terminado el uso de la cámara para grabar el video, el método `lock()` para bloquearla de nuevo.

10. CREACIÓN DE UN WIDGET

Álvaro Borrego – Francisco Hernández – David Palomero

CREACIÓN DE UN WIDGET SENCILLO

En esta sección del manual se va a explicar cómo crear un widget para Android. En primer lugar se mostrará y explicará con detalle la manera de crear un widget estático, es decir, sin ninguna funcionalidad. De esta manera se pretende que se consiga entender de manera sencilla y clara la estructura de este tipo de componentes muy utilizado en cualquier aplicación Android.

Una vez entendida la estructura de un widget, se añadirá una funcionalidad más avanzada, como puede ser, la realización de una determinada acción al ser pulsado por el usuario o su actualización automática.

El widget que se va a implementar en esta primera parte consistirá en una imagen de un reloj y un texto con un mensaje ('Hora Actual'). Con este ejemplo se pretende comprender de una manera sencilla la estructura de un widget en Android, sin tener en cuenta aspectos más avanzados, como puede ser la interacción con el usuario.

DEFINICIÓN DE LA INTERFAZ GRÁFICA

Como se ha comentado anteriormente, el widget consistirá en una imagen de un reloj y un mensaje de texto, que en un principio, solamente mostrará el mensaje 'Hora Actual'. La imagen general del widget en esta primera parte será:

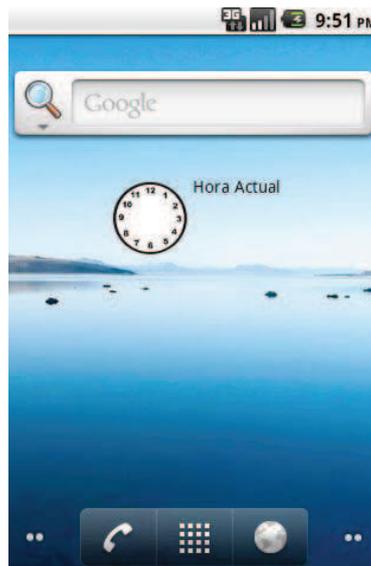


Figura 1. Creación de un widget

Para conseguir este aspecto, se creará en primer lugar, un layout xml llamado *widgethora.xml*. Éste layout está formado sencillamente por un contenedor *LinearLayout* en el que se sitúan una imagen *ImageView* y una etiqueta de texto *TextView* que muestra el mensaje.

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:orientation="horizontal"
    android:padding="5dip">
    <ImageView
        android:layout_width="wrap_content"
        android:src="@drawable/reloj"
        android:layout_height="wrap_content"
        android:layout_gravity="right"
        android:id="@+id/imageViewReloj"/>

    <TextView android:id="@+id/textViewHora"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:textColor="#000000"
        android:text="Hora Actual" />
</LinearLayout>
```

NOTA: En el ejemplo se ha utilizado una imagen llamada *reloj.png*, por lo que es necesario crear una imagen con ese nombre e incluirla en el proyecto. Otra opción posible, es sustituir `android:src="@drawable/reloj"` por `android:src="@drawable/icon"`, de esta manera el icono del widget no será el de un reloj sino el icono por defecto de Android.

ASIGNAR PROPIEDADES AL WIDGET

En esta parte se da la posibilidad de definir algunas de las propiedades que va a tener el widget, como por ejemplo, el tamaño en pantalla, el nombre que aparecerá en el menú etc... Para ello, se crea un nuevo xml (que se ubicará en la carpeta `\res\xml` del proyecto) llamado *widgethora_provider.xml*.

```
<?xml version="1.0" encoding="utf-8"?>
<appwidget-provider xmlns:android="http://schemas.android.com/apk/res/android"
    android:initialLayout="@layout/widgethora"
    android:label="Widget de la Hora"
    android:minWidth="146dip"
    android:minHeight="72dip"

/>
```

En este caso se han definido las siguientes propiedades:

- **initialLayout:** hace referencia al layout xml creado anteriormente, en este caso, *widgethora*.
- **label:** será el nombre que aparecerá en el menú de aplicaciones de Android.

- **minWidth/minHeight:** define el ancho y el alto respectivamente del widget en pantalla. En Android, la pantalla se divide en pequeñas celdas donde se pueden colocar iconos de aplicaciones, iconos de contactos, widgets, etc.

En el ejemplo, se quiere tener un widget con unas dimensiones de dos celdas de ancho y una celda de alto (2x1). Para conseguir estas dimensiones, existe una fórmula que indica el valor exacto que deben tener las propiedades minWidth y minHeight:

$$\text{dimensión} = (\text{número de celdas} \times 74) - 2$$

Teniendo en cuenta esta fórmula, y sustituyendo el número de celdas deseadas para cada dimensión, se obtienen los valores `minWidth="146dip"` y `minHeight="72dip"`. Existen otras propiedades muy interesantes que no se han tenido en cuenta en este ejemplo, pero que serían de gran utilidad en la mayoría de aplicaciones con widgets, como pueden ser:

- **icon:** icono que se muestra para el widget en el selector de AppWidget.
- **minResizeWidth/minResizeHeight:** anchura/altura mínima que se puede cambiar de tamaño para el widget.
- **resizeMode:** Las reglas por las que se puede cambiar el tamaño de un widget.
- **updatePeriodMillis:** frecuencia en la que se actualizará el widget, definida en milisegundos.

IMPLEMENTACIÓN DE LA CLASE PRINCIPAL

En este paso, se declarará la clase principal (que será la que se mostrará al pulsar sobre el icono en el menú). En este ejemplo, se mostrará una pantalla en negro con un mensaje que mostrará la forma de añadir el widget creado al escritorio.

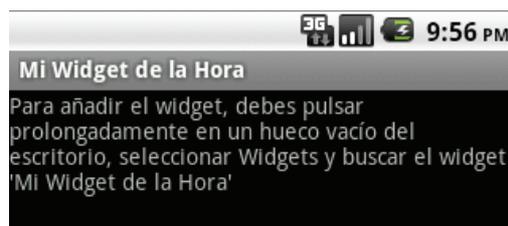


Figura 2. Información de un widget

El código de esta clase es el siguiente:

```
public class WidgetHoraMain extends Activity {
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
    }
}
```

El layout *main* estará formado únicamente por una etiqueta de texto, explicando la forma de añadir el widget al escritorio.

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    >
<TextView
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    android:text="@string/mensaje"
    />
</LinearLayout>
```

Se debe incluir en el archivo `\res\values\strings.xml` la siguiente entrada:

```
<string name="mensaje">
```

Para añadir el widget, se debe pulsar prolongadamente en un hueco vacío del escritorio, seleccionar Widgets y buscar el widget "Mi Widget de la Hora"

Como se ha comentado en repetidas ocasiones a lo largo de la explicación, en esta primera parte del ejemplo no se va a implementar ninguna funcionalidad del widget, no obstante, se va a declarar y a explicar brevemente la clase necesaria para esta tal efecto ya que será implementada más adelante. Es necesario crear la clase *MiWidgetHora.java*:

```
public class MiWidgetHora extends AppWidgetProvider {
    public void onUpdate(Context context, AppWidgetManager appWidgetManager, int[] appWidgetIds) {
    }
}
```

Los principales eventos en un widget son:

- **onEnabled()**: lanzado cuando se añade al escritorio la primera instancia de un widget.
- **onUpdate()**: lanzado periódicamente cada vez que se debe actualizar un widget.
- **onDeleted()**: lanzado cuando se elimina del escritorio una instancia de un widget.
- **onDisabled()**: lanzado cuando se elimina del escritorio la última instancia de un widget.

AÑADIR EL WIDGET AL MANIFEST DEL PROYECTO

Por último, queda declarar el widget en el archivo *manifest* del proyecto.

```
<receiver android:name=".MiWidgetHora" android:label="Mi Widget de la Hora">
    <intent-filter>
        <action android:name="android.appwidget.action.APPWIDGET_UPDATE" />
    </intent-filter>
    <meta-data
        android:name="android.appwidget.provider"
        android:resource="@xml/widgethora_provider" />
</receiver>
```

WIDGET CON FUNCIONALIDAD

Una vez explicadas las características más importantes para la creación de un widget, llega el momento de añadirle funcionalidad, permitiendo su actualización e interacción con el usuario. En esta segunda parte, se va a utilizar el widget creado anteriormente, añadiendo una serie de eventos que permitan realizar una acción concreta al ser pulsado por el usuario.

La idea de este widget más avanzado, será mostrar la hora actual del dispositivo al ser pulsado por el usuario. También se añadirá la opción de actualizarse automáticamente pasado un periodo de tiempo determinado. El layout *widgethora.xml* no se ha modificado, por lo que el widget seguirá teniendo la misma apariencia. El objetivo, es ahora, mostrar la hora actual del dispositivo en lugar de un mensaje estático.



Figura 3. Widget más avanzado

DEFINIR EL TIEMPO DE ACTUALIZACIÓN DEL WIDGET

Como una de las funciones del widget será la de actualizarse automáticamente, habrá que definir el tiempo de actualización (en milisegundos). Para ello, simplemente hay que añadir la siguiente propiedad en el *widgethora_provider.xml*:

```
android:updatePeriodMillis="60000"
```

Una opción muy interesante, pero que no se implementará en este caso debido a la simplicidad del ejemplo, es la de mostrar una pantalla de configuración del widget, de esta manera el usuario podrá seleccionar las características que le interesen. Para mostrar una pantalla de configuración, bastaría con añadir, a este mismo archivo:

```
android:configure="rutaDelPaquete.ConfigWidget"
```

Siendo `ConfigWidget` la actividad donde se definen las distintas opciones de configuración. Con esto, al colocar el widget en el dispositivo, se ejecutará automáticamente la pantalla de configuración antes de ser colocado.

AÑADIR FUNCIONALIDAD AL WIDGET

Una vez realizadas estas pequeñas modificaciones, es el momento de implementar la funcionalidad. En la primera parte, se creó una clase llamada `MiWidgetHora.java` que quedó prácticamente vacía (ya que no se deseaba implementar ninguna funcionalidad). Ahora es necesario completar esa clase, ya que va a ser la encargada de realizar todas las acciones relacionadas con el widget.

EVENTO `ON_UPDATE()`

El evento `onUpdate` (lanzado periódicamente cada vez que se debe actualizar un widget) recibe como parámetro una lista de todas las instancias añadidas al escritorio. Cuando se actualiza, es necesario actualizar todas estas instancias. Para ello, se recorre dicha lista y se van actualizando una a una llamando al método `actualizar`, que será explicado más adelante.

```
public class MiWidgetHora extends AppWidgetProvider {
    public void onUpdate(Context context, AppWidgetManager appWidgetManager, int[] appWidgetIds) {
        int i=0;
        while (i<appWidgetIds.length)
        {
            int id = appWidgetIds[i];
            actualizar(context, appWidgetManager, id);
            i++;
        }
    }
}
```

EVENTO `ON_RECEIVE()`

El evento `onReceive`, es el encargado de capturar los mensajes enviados por las componentes. Dependiendo del tipo de mensaje recibido, se realizará una función u otra. En este ejemplo sólo se va a capturar un tipo de mensaje, que será el enviado al pulsar sobre el icono del reloj de la interfaz. Para indicar esto, se creará al principio de la clase un atributo que hará referencia a este tipo de mensajes.

```
private static final String ACCION_PULSAR = "PULSAR";
```

El primer paso a realizar en éste método, es capturar la acción asociada al *intent*:

```
final String action = intent.getAction();
```

Una vez que se tiene esta acción, es necesario saber de qué tipo es. Como se ha comentado anteriormente, en este ejemplo, sólo se tiene un tipo de mensaje (`ACCION_PULSAR`), por lo que bastará con comprobar si el mensaje capturado es de este tipo. En caso de ser así, se debe conocer la id del widget pulsado, ya que es posible tener más de un widget de este mismo tipo en pantalla. Finalmente, sólo queda llamar al método `actualizar`, encargado de realizar la acción sobre el widget pulsado. Dicho método, necesita como uno de sus argumentos un widget manager, por lo que es necesario obtenerlo antes de realizar la llamada:

```
AppWidgetManager widgetManager = AppWidgetManager.getInstance(context);
```

El código completo de *onReceive* se muestra a continuación:

```
public void onReceive(Context context, Intent intent) {
    final String action = intent.getAction();

    if (action.equals(ACCION_PULSAR)) {
        final int id = intent.getIntExtra(AppWidgetManager.EXTRA_APPWIDGET_ID,
            AppWidgetManager.INVALID_APPWIDGET_ID);

        AppWidgetManager widgetManager = AppWidgetManager.getInstance(context);

        actualizar(context, widgetManager, id);
    }
    super.onReceive(context, intent);
}
```

MÉTODO ACTUALIZAR

El método *actualizar*, es el encargado de realizar la actualización del widget. Un widget está formado por una serie de componentes llamadas Remote Views. Para hacer referencia a cada una de estas vistas, es necesario acceder a la lista de componentes:

```
RemoteViews componentes = new RemoteViews(context.getPackageName(), R.layout.widgethora);
```

El objeto *componentes*, contiene una lista con todas las vistas que forman el widget, en este caso, un *TextView* y un *ImageView*. La forma de hacer referencia a cada una de ellas es la siguiente:

```
componentes.setTextViewText(R.id.textViewHora, "...");
```

Con esto, se consigue asignar al elemento de tipo *TextView* con id *textViewHora* (definido en el *layout widgethora.xml*) el string encerrado entre comillas. Ahora, toca el turno de hacer lo mismo con el componente *imageViewReloj* (la imagen del reloj). En este caso, lo que se desea es definir el evento *OnClick* sobre la imagen, para así poder actualizar el widget cuando es pulsada por el usuario. Como se ha comentado anteriormente en el *onReceive*, la forma de conseguir asignar un evento de tipo *OnClick* a una componente de un widget, es enviando un mensaje de tipo *ACCION_PULSAR*. Para ello, lo primero que hay que hacer es crear un *intent* y asociarle la acción comentada. También será necesario conocer el id del widget pulsado.

```
final Intent onClickIntent = new Intent(context, MiWidgetHora.class);
onClickIntent.setAction(MiWidgetHora.ACCION_PULSAR);
onClickIntent.putExtra(AppWidgetManager.EXTRA_APPWIDGET_ID, id);
```

Como la finalidad de este widget es la de mostrar la hora actual del dispositivo, va a ser necesario implementar un método que realice dicha tarea:

```
private static String dameHora() {
    Calendar calendario = new GregorianCalendar();
    return calendario.getTime().toLocaleString();
}
```

Para asegurar la correcta actualización de los componentes del widget en la interfaz, es necesario añadir al final de este método, la siguiente línea:

```
appWidgetManager.updateAppWidget(id, componentes);
```

El código completo del método *actualizar* queda de la siguiente manera:

```
public static void actualizar(Context context, AppWidgetManager appWidgetManager, int id)
{
    RemoteViews componentes = new RemoteViews(context.getPackageName(), R.layout.widgethora);

    final Intent onClickIntent = new Intent(context, MiWidgetHora.class);
    onClickIntent.setAction(MiWidgetHora.ACCION_PULSAR);
    onClickIntent.putExtra(AppWidgetManager.EXTRA_APPWIDGET_ID, id);

    final PendingIntent onclickPendingIntent = PendingIntent.getBroadcast(context, id, onClickIntent,
                                                                    PendingIntent.FLAG_UPDATE_CURRENT);

    componentes.setOnClickPendingIntent(R.id.imageViewReloj, onclickPendingIntent);
    String horaActual= dameHora();
    componentes.setTextViewText(R.id.textViewHora, horaActual);
    appWidgetManager.updateAppWidget(id, componentes);
}
```

11. PUBLICANDO EN EL MARKET

Álvaro Borrego – Francisco Hernández – David Palomero

EL ANDROID MARKET

El Android Market es la tienda de aplicaciones de Android. Tiene un acceso rápido y ágil a aplicaciones creadas por desarrolladores de todo el mundo. Desde el punto de vista del desarrollador, se puede usar el Market para publicar sus propias aplicaciones. Tiene las siguientes características destacables:

- Es libre, cualquier usuario puede apuntarse.
- Las aplicaciones publicadas pueden ser puntuadas y comentadas por los usuarios.
- Cuenta con estadísticas de cada aplicación, tales como número de descargas, puntuaciones, comentarios...
- Las aplicaciones publicadas en él, pueden ser de pago, gratuitas, gratuitas con publicidad o gratuitas con limitaciones. Éstas últimas cuentan con su versión completa de pago.

Una de las grandes ventajas que ofrece el Android Market es la posibilidad de instalar una aplicación desde un PC teniendo el dispositivo conectado.

ACCEDER AL ANDROID MARKET

Existen dos maneras por las que se puede acceder al Market para descargar aplicaciones a los dispositivos: mediante la aplicación preinstalada en los dispositivos Android, señalizada con el icono mostrado en la Figura 1., o mediante la Web oficial <https://market.android.com> cuya imagen inicial se muestra en al Figura 2.



Figura 1. Icono del Market

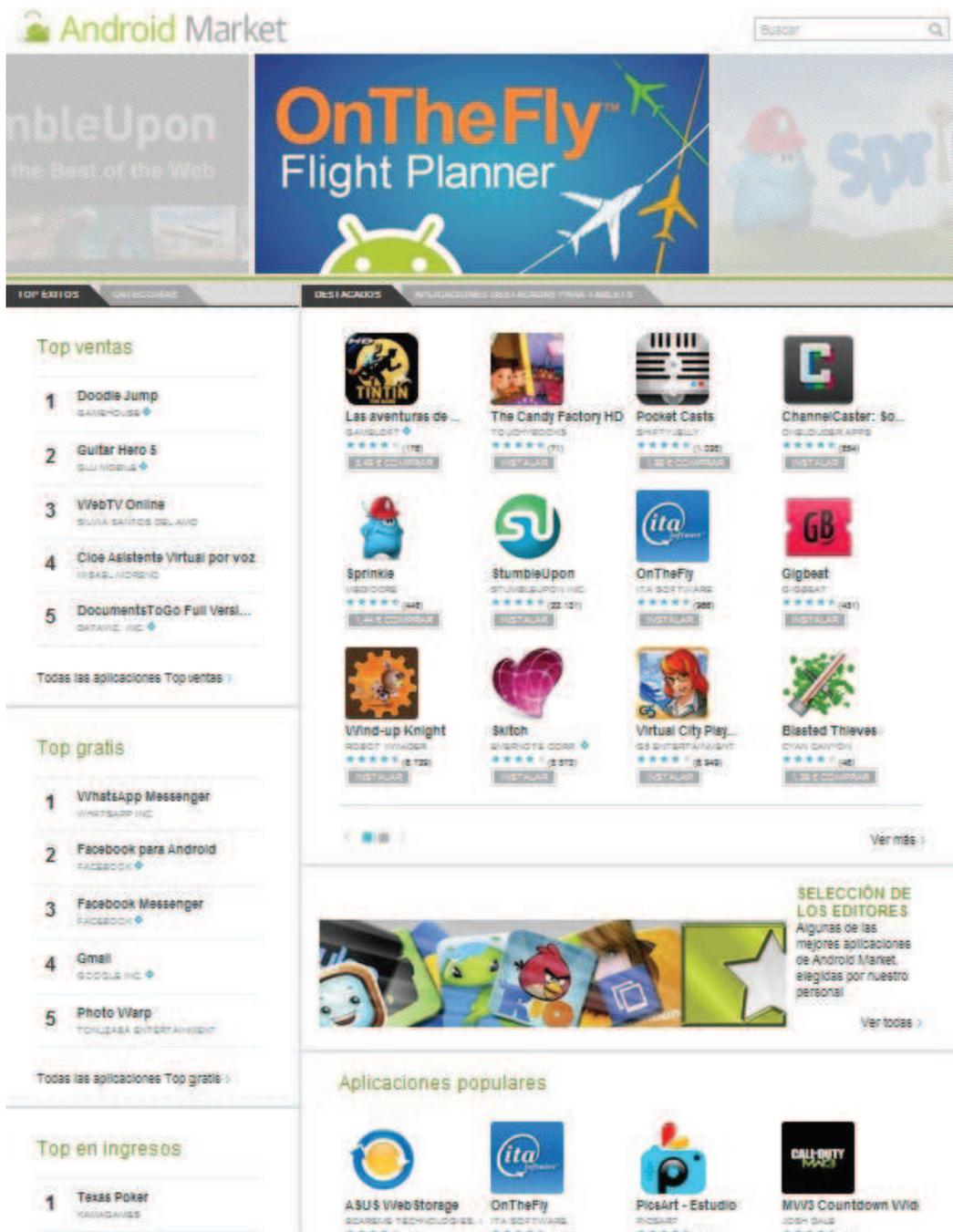


Figura 2. Web oficial de Android Market

En ambas, hay diferentes opciones y las aplicaciones están estructuradas por categorías. La primera opción que presenta es la búsqueda, mediante la cual se puede buscar la aplicación deseada. Una vez introducido el nombre, se mostrarán todas las aplicaciones encontradas, y se podrán clasificar según relevancia, popularidad, precio, etc. A continuación se muestran algunas de las aplicaciones clasificadas por categorías, destacados o aplicaciones top según ventas, nuevas, etc.

PUBLICAR EN ANDROID MARKET

REQUISITOS

Para publicar una aplicación en el Android Market, primero hay que registrarse con una cuenta de desarrollador de Google (Google Checkout 25 \$) de acuerdo con los términos del servicio. Se puede registrar como desarrollador en:

<http://market.android.com/publish>

Una vez registrado, se puede subir la aplicación, actualizar tantas veces como quiera, y publicarla cuando esté lista. Es necesario residir en uno de los países soportados: Australia, Austria, República Checa, Francia, Alemania, Italia, Países bajos, Polonia, Singapur, España, Reino Unido, Estados Unidos. Los requisitos impuestos por el servidor de Android Market son los siguientes:

1. La aplicación debe tener definidos dos atributos en el archivo <manifest>:
Android:versionCode y android:versionName
2. El atributo versionCode es un número entero creciente que lo utiliza el servidor para identificar internamente la versión de la aplicación y para el manejo de cambios.
3. El atributo versionName muestra a los usuarios la versión de la aplicación. Consiste en una cadena de texto que representa la versión tal y como la verán los usuarios.
4. Además, hay que definir otros dos atributos, android:icon y android:label, dentro del elemento <application> del archivo *manifest*.
5. La aplicación debe estar firmada con una clave criptografica privada

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.prueba"
    android:versionCode="1"
    android:versionName="1.0">
    <uses-sdk android:minSdkVersion="7" />

    <application android:icon="@drawable/icon" android:label="@string/app_name">
        <activity android:name=".pruebaActivity"
            android:label="@string/app_name">
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />
                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
    </application>
</manifest>
```

Figura 3. Requisitos de configuración en Manifest

PASOS

Se publica la aplicación a través de la herramienta web disponible en el portal de desarrolladores del Android market, cumplimentando los datos necesarios en tres partes:

1. La parte de “**subir recursos**“, donde se sube el .apk, las imágenes promocionales, un icono, un gráfico promocional, y un video de youtube (Figura 4).
2. La parte de “**especificación de detalles**” con la descripción de la aplicación en los diferentes idiomas, tipo, categoría, descripción (Figura 5).
3. Y la parte de ‘**opciones de publicación**’ con la protección y clasificación por edades, además de la información de contacto y aceptaciones correspondientes (Figuras 6 y 7).

Una vez publicada la aplicación (opción publish), en breves minutos ya se encontrará en el Market.

Subir recursos

Capturas de pantalla al menos 2	Añade una captura de pantalla: <input type="button" value="Seleccionar archivo"/> No se ha...archivo	<input type="button" value="Publicar"/>	Capturas de pantalla: archivo PNG o JPEG (no alpha) de 24 bits de 320 x 480, 480 x 800, 480 x 854, de 1280 x 720, 1280 x 800. Sangrado completo, sin bordes. Puedes subir capturas de pantalla en orientación horizontal. Parecerá que las miniaturas están giradas, pero se mantendrán la orientación y las imágenes reales.
Icono de aplicación de alta resolución [Más información]	Añade un icono de aplicación de alta resolución: <input type="button" value="Seleccionar archivo"/> No se ha...archivo	<input type="button" value="Publicar"/>	Icono de aplicación de alta resolución: imagen de 32 bits PNG o JPEG y 512 x 512, máximo: 1024 KB
Gráfico promocional opcional	Añade un gráfico promocional: <input type="button" value="Seleccionar archivo"/> No se ha...archivo	<input type="button" value="Publicar"/>	Gráfico promocional: archivo de 24 bits PNG o JPEG (no alpha) y 180 an x 120 al
Gráfico de funciones opcional	Añade un gráfico de funciones: <input type="button" value="Seleccionar archivo"/> No se ha...archivo	<input type="button" value="Publicar"/>	Gráfico de funciones: archivo de 24 bits PNG o JPEG (no alpha) y 1024 x 500; el tamaño se reducirá a mini o micro.
Video promocional opcional	Añade un enlace de video promocional: <input type="text" value="http://"/>	<input type="button" value="Publicar"/>	Video promocional: introducir URL de YouTube
Excluir marketing	<input checked="" type="checkbox"/> No promocionar mi aplicación salvo en Android Market y en los sitios web o para móviles propiedad de Google. Asimismo, soy consciente de que cualquier cambio relacionado con esta preferencia puede tardar sesenta días en aplicarse.		

Figura 4. Subir recursos

Especificación de detalles

Idioma | *English (en) |
[añadir idioma](#) El signo de estrella (*) indica el idioma predeterminado.

Title (Inglés)
0 caracteres (máximo 30)

Description (Inglés)
0 caracteres (máximo 4000)

Recent Changes (Inglés)
[\[Más información\]](#)
0 caracteres (máximo 500)

Promo Text (Inglés)
0 caracteres (máximo 80)

Tipo de aplicación

Categoría

Figura 5. Especificación de detalles

Opciones de publicación

Protección contra copias Desactivado (la aplicación se puede copiar desde el dispositivo)
 Activado (evita la copia de esta aplicación desde el dispositivo. Aumenta la cantidad de memoria del teléfono necesaria para instalar la aplicación).
La función de protección contra copias quedará obsoleta en poco tiempo; utiliza el [servicio de licencias](#) en su lugar.

Clasificación de contenido [\[Más información\]](#)
 Nivel de madurez alto
 Nivel de madurez medio
 Nivel de madurez bajo
 Para todos

Precios ¿Quieres vender aplicaciones? [Configura una cuenta de comerciante en Google Checkout.](#)

Dispositivos admitidos [\[Más información\]](#)
 Todos los países

<input checked="" type="checkbox"/> Alemania	<input checked="" type="checkbox"/> Islandia
<input checked="" type="checkbox"/> Argentina	<input checked="" type="checkbox"/> Israel
<input checked="" type="checkbox"/> Australia	<input checked="" type="checkbox"/> Italia
<input checked="" type="checkbox"/> Austria	<input checked="" type="checkbox"/> Japón
<input checked="" type="checkbox"/> Bélgica	<input checked="" type="checkbox"/> Kenia
<input checked="" type="checkbox"/> Brasil	<input checked="" type="checkbox"/> Letonia
<input checked="" type="checkbox"/> Bulgaria	<input checked="" type="checkbox"/> Lituania
<input checked="" type="checkbox"/> Camerún	<input checked="" type="checkbox"/> Luxemburgo
<input checked="" type="checkbox"/> Canadá	<input checked="" type="checkbox"/> Malta
<input checked="" type="checkbox"/> Chipre	<input checked="" type="checkbox"/> México
<input checked="" type="checkbox"/> Corea del Sur	<input checked="" type="checkbox"/> Nicaragua
<input checked="" type="checkbox"/> Costa de Marfil	<input checked="" type="checkbox"/> Noruega
<input checked="" type="checkbox"/> Dinamarca	<input checked="" type="checkbox"/> Nueva Zelanda
<input checked="" type="checkbox"/> Eslovaquia	<input checked="" type="checkbox"/> Países Bajos
<input checked="" type="checkbox"/> Eslovenia	<input checked="" type="checkbox"/> Polonia

Figura 6 Opciones de publicación

Información de contacto

Sitio web

Dirección de correo electrónico

Teléfono

Consentimiento

Esta aplicación cumple las [directrices para contenido de Android](#).

Acepto que mi aplicación pueda estar sujeta a las leyes de exportación de Estados Unidos, independientemente de mi ubicación o de mi nacionalidad. Asimismo, confirmo que he cumplido dichas leyes, incluidos los requisitos para software con funciones de encriptación. Certifico que mi aplicación se puede exportar desde Estados Unidos de acuerdo con las leyes de este país. [Más información](#)

Figura 7. Almacenamiento de información de contacto

FIRMA DIGITAL

Las aplicaciones deben estar firmadas digitalmente para poder distribuirse. Esto es una medida de seguridad, así solo el desarrollador de la aplicación puede modificarla y actualizarla. El certificador puede ser autofirmado, no es necesaria una autoridad certificadora. Existen dos modos para firmar las aplicaciones:

- Debug: mientras desarrollamos. El plug-in ADT se encarga de la firma automáticamente, gestionando claves generadas.
- Release: Crea una clave proporcionando el usuario un password. Para esto se puede utilizar el ADT export wizard desde eclipse.

Firmar aplicaciones desde eclipse a través de export wizard:

Se abre la aplicación que se quiere firmar con eclipse, y en el árbol de directorios se abre el archivo AndroidManifest. Una vez abierto, se abre la primera pestaña llamada *Manifest* y en la sección Exporting hay dos opciones para firmar la aplicación.

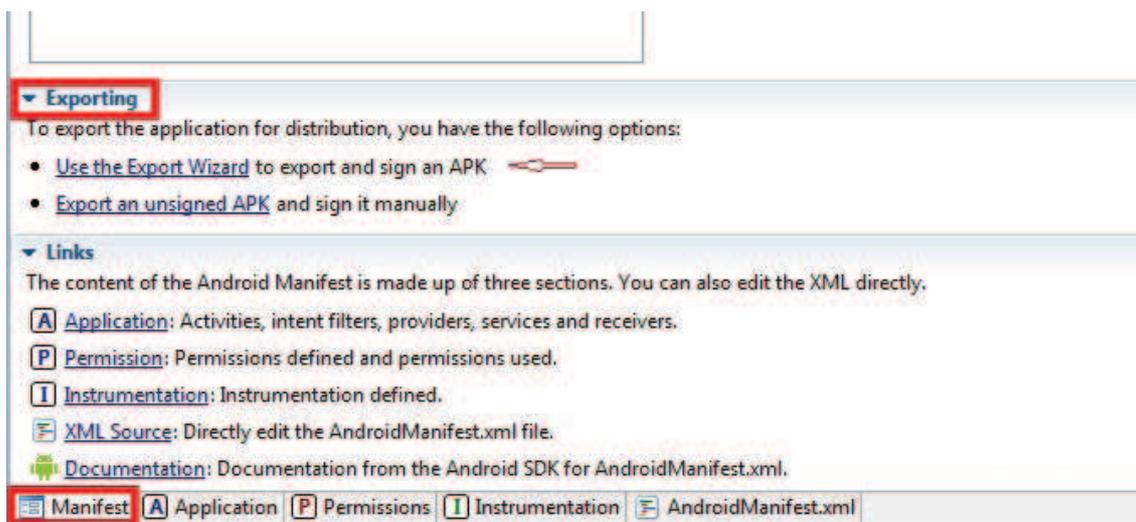


Figura 8. Sección Exporting de Manifest

Se selecciona la opción Use the Export Wizard.

A continuación, en la siguiente pantalla aparecerá automáticamente el proyecto a firmar, y si no se detecta ningún error, se pulsa en siguiente

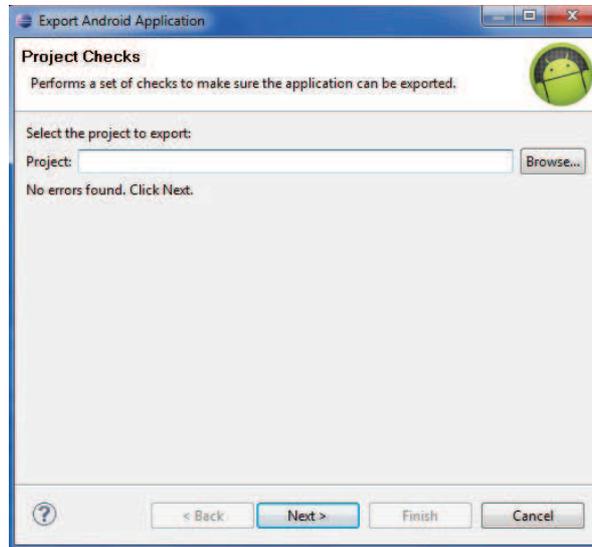


Figura 9. Firma proyecto (1)

Para firmar la aplicación es necesario tener una keystore. Si no se ha creado con anterioridad ninguna, hay que crear una nueva keystore con la opción Create new keystore, rellenando los datos solicitados: Location (directorio donde se guardará la keystore) y password.

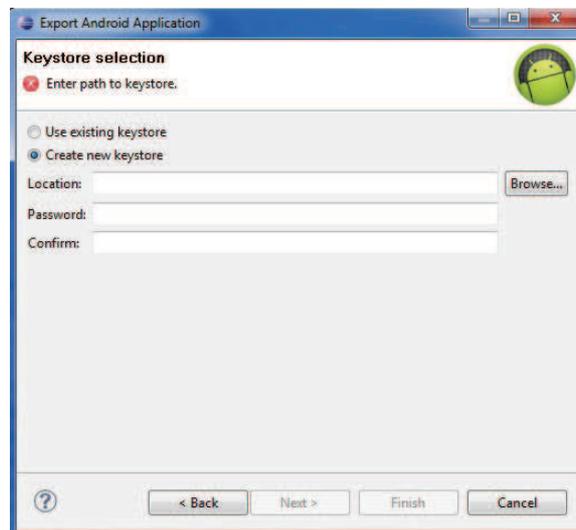


Figura 10. Firma proyecto (2)

En la siguiente pantalla, se rellena el formulario de datos de la keystore y del desarrollador de la aplicación.

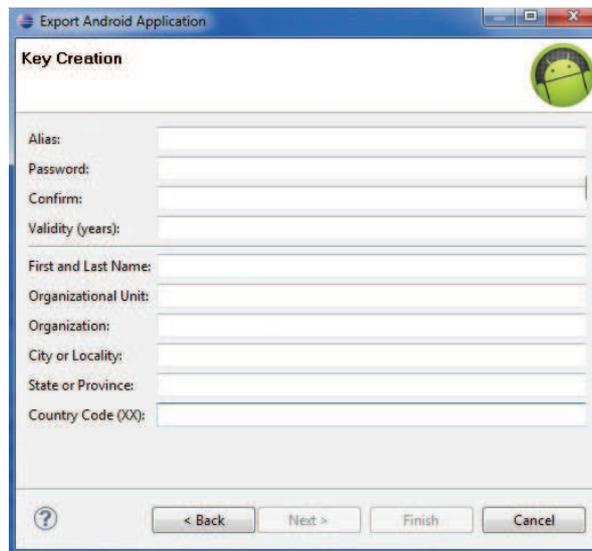


Figura 9. Firma proyecto (3)

Por último, se escoge el directorio en el que se guardara la aplicación ya firmada.

En el caso en el que se quieran firmar más aplicaciones, no será necesario volver a crear una keystore, sino que se podrá utilizar la creada anteriormente, sin necesidad de crear una keystore por aplicación.

ACTUALIZACION

Una vez publicadas las aplicaciones es recomendable actualizarlas corrigiendo errores detectados y añadiéndolas nuevas funcionalidades.

Esto es importante por dos motivos:

- Mantiene la aplicación en las primeras posiciones de la lista de aplicaciones del market, lo que da posibilidad a nuevos usuarios a conocer la aplicación y descargarla.
- Aumenta la confianza los usuarios en el desarrollador, ya que demuestra que tiene en cuenta sus opiniones para mejorar la aplicación.

Para actualizar la aplicación tan solo se debe tener en cuenta que hay que cambiar los atributos android:versionCode y android:versionName.

EL ANDROID MARKET PARA DESARROLLADORES

Cuando se accede al Market con una cuenta de desarrollador, se tienen opciones acerca de las aplicaciones desarrolladas.

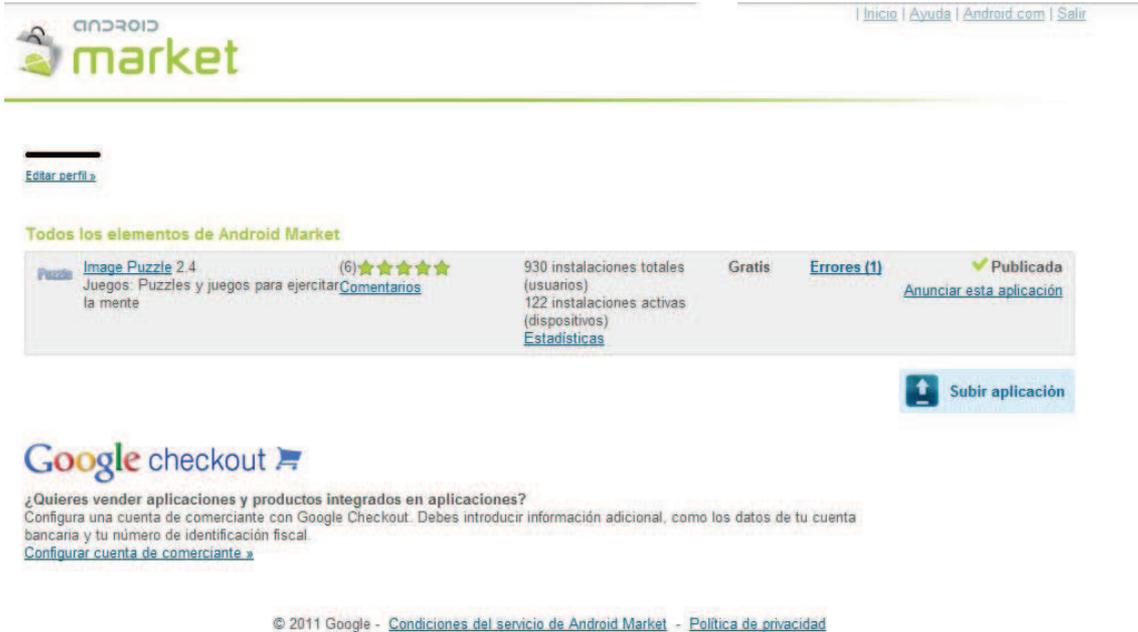


Figura 10. Acceso al Market de Android

La Figura 10 muestra dicho acceso. Algunas características y funcionalidades se describen a continuación.

1. **Listado de aplicaciones:** Muestra el número de descargas de cada aplicación, así como el número de descargas activas, que será el número más importante, ya que indica la cantidad de personas que tienen instalada la aplicación (importante si tiene publicidad la aplicación porque aumentará el beneficio).

Total de instalaciones activas



Figura 11. Cargas y descargas sobre las instalaciones activas

En la imagen anterior se observa un gran crecimiento en determinados momentos. Esto es debido a la publicación de actualizaciones de la aplicación, lo que conlleva un aumento del número de descargas al aparecer la aplicación entre las más nuevas, de ahí la importancia de desarrollar actualización de las aplicaciones.

2. **Comentarios:** Para cada aplicación, se puede acceder a las valoraciones y comentarios de los usuarios acerca de esa aplicación.

Comentarios de la aplicación



Figura 12. Comentarios de la aplicación

3. **Errores:** Muestra los errores que se producen en la aplicación y son enviados por los usuarios a los desarrolladores. Generalmente se producen porque determinadas partes de la aplicación no han sido debidamente testeadas, o son errores inesperados.

Informes de errores de la aplicación



Figura 13. Informe de errores de la aplicación

4. **Disponibilidad de dispositivo:** Indica que dispositivos son compatibles dependiendo del hardware utilizado pudiendo excluir algunos de ellos.
5. **Estadísticas:** Muestra estadísticas actualizadas diariamente de cada aplicación, tales como el número de instancias activas, distribución por versión de Android, por dispositivos, por país o por idioma.

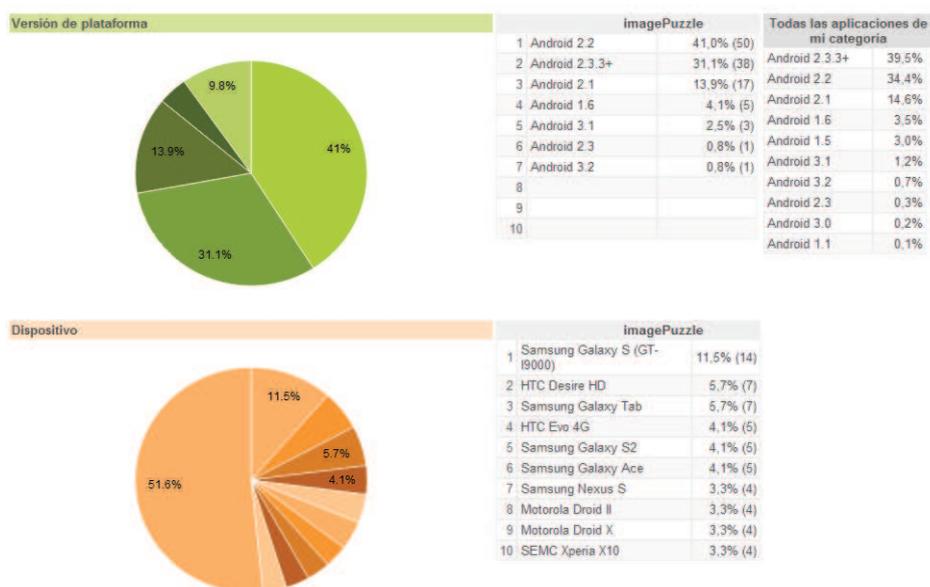


Figura 14. Estadísticas (1)

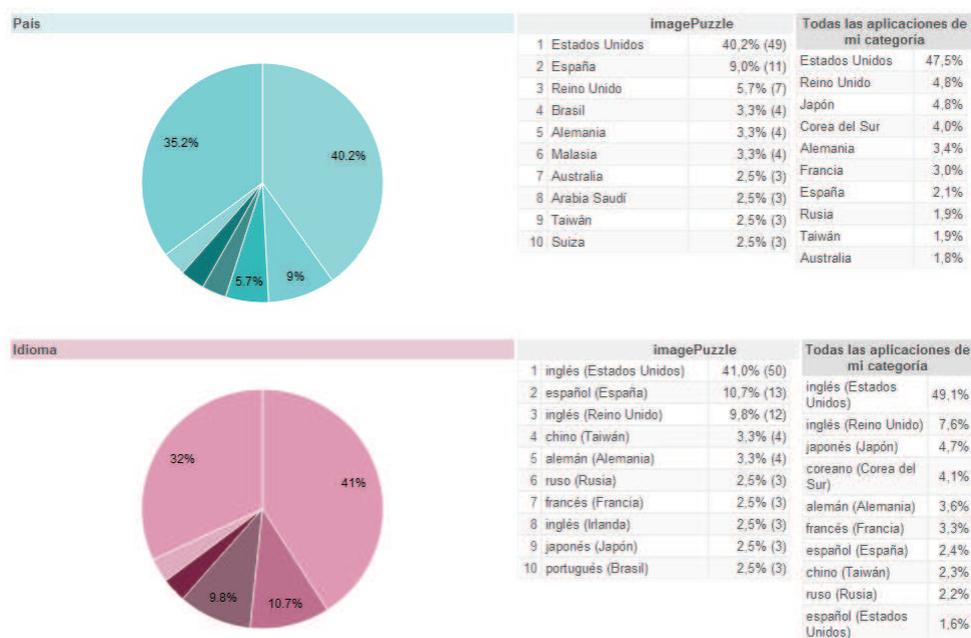


Figura 15. Estadísticas (2)