Preliminares

Antes de realizar el empaquetado final de la aplicación, hay que verificar algunos puntos tales como la versión de la aplicación y los filtros configurados en la aplicación que permiten ponerla a disposición solamente de aquellos dispositivos compatibles.

1. Versión de la aplicación

Es costumbre, en informática, adjuntar un número de versión a todo software. Este número de versión debe ser un identificador único, de otro modo no tendría mucho interés.

En Android, existen dos campos en el manifiesto que tienen como objetivo determinar este número de versión:`android:versionCode` y `android:versionName`

a. android:versionCode

`android:versionCode` es un número entero. Este número no se muestra al usuario. Permite a las demás aplicaciones y a Play Store poder comparar dos versiones de la aplicación para determinar cuál de las dos se corresponde con la actualización, a saber, la que tenga un número de versión mayor.

Es habitual comenzar la numeración por 1 e incrementar el valor con cada actualización oficial de la aplicación, sea mayor o menor.

El sistema Android no verifica este valor, y deja total libertad al desarrollador a la hora de especificar este valor. Corresponde al desarrollador velar por que la última versión disponible posea, siempre, el valor mayor.

Sintaxis

```
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    android:versionCode="entero" >
  ...
</manifest>
```

Ejemplo

```
<manifest
    xmlns:android="http://schemas.android.com/apk/res/android"
    package="es.midominio.android.miaplicacion"
    android:versionCode="42" >
</manifest>
```

b. android:versionName

`android:versionName` es el número de versión que se muestra al usuario en Play Store. Es un elemento clave de carácter informativo que sirve tanto al desarrollador como al usuario.

Para el desarrollador, este número de versión permite especificar las versiones mayores, las versiones menores, las versiones correctoras… Permite a su vez, cuando la aplicación se ha publicado bajo diversas versiones, asociar bugs a una u otra o versión.

Para el usuario, el número de versión permite verificar que se está utilizando la última actualización de la aplicación. Le permite juzgar la conveniencia de una actualización según la importancia del cambio de versión, mayor o menor. Por último, le permite también distinguir la versión en las conversaciones con otros usuarios o con el desarrollador.

El desarrollador tiene libertad en cuanto a la sintaxis que quiera adoptar para la numeración de las

versiones. El valor debe ser una cadena de caracteres.

Este número de versión debe ser único y diferente, como mínimo, con cada publicación de la aplicación. En caso contrario nadie sabrá de qué variante de la aplicación se trata.

Sintaxis

```
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    android:versionName="cadena de caracteres" >
    ...
</manifest>
```

Ejemplo

```
<manifest
  xmlns:android="http://schemas.android.com/apk/res/android"
  package="es.midominio.android.miaplicacion"
  android:versionName="13.3.7" >
</manifest>
```

En este ejemplo, la sintaxis utilizada respeta la siguiente convención:*número_mayor.número_menor.número_build*.

2. Filtros para el mercado

Pensemos, por ejemplo, en una aplicación de fotografía que, por definición, requiere la presencia de una cámara fotográfica instalada en el dispositivo Android. Si esta aplicación se le propusiera a un usuario cuyo dispositivo Android no poseyera cámara, correría el riesgo de no funcionar correctamente.

En este caso, el usuario podría verse motivado a expresar su descontento en forma de comentario y de nota mediocre, los cuales son públicos en Play Store y pueden ser consultados por los demás usuarios antes de decidirse a la hora de instalar una aplicación. Es, por tanto, muy importante no permitir la instalación de la aplicación sobre un dispositivo incompatible.

Para evitar este tipo de disgustos, Play Store filtra las aplicaciones para proponer solamente aquellas que son compatibles con el dispositivo del usuario. Para hacer esto, se basa en la información proporcionada por la aplicación indicando las características de hardware y software de las que depende la aplicación.

Esta información se provee en el manifiesto en las etiquetas `uses-feature` y `uses-configuration`.

Esta información tiene un carácter meramente informativo. El sistema Android no las utiliza, ni siquiera para filtrar la instalación de una aplicación o para autorizar su uso en el dispositivo. Se utilizan, principalmente, en programas específicos como Play Store para filtrar los dispositivos compatibles. No reemplazan, en ningún caso, a las solicitudes de permisos necesarios para utilizar algún hardware o componente de software.

Play Store utiliza también la etiqueta `uses-sdk` y su atributo `android:minSdkVersion`, las etiquetas `uses-library` y la etiqueta `supports-screen` para filtrar las aplicaciones compatibles con el dispositivo del usuario.

a. uses-feature

La etiqueta `uses-feature` permite indicar las características de hardware y software de las que depende la aplicación. Se recomienda informarlas todas. Los valores son de tipo`android.hardware.*` y `android.software.*`.

El atributo `android:required` permite indicar si la característica correspondiente es obligatoria o no. Por defecto, lo es. No obstante, puede ser interesante que la aplicación gestione el caso en que la característica no esté presente. Esto permite abrir la aplicación a más usuarios, en concreto a aquellos que utilicen tabletas táctiles y que no dispongan, necesariamente, de las mismas capacidades de hardware que, por ejemplo, los teléfonos móviles.

Play Store utiliza las declaraciones de permisos requeridos por la aplicación para verificar la lista de características requeridas obligatoriamente y completarla automáticamente. Por ejemplo, si una aplicación solicita el permiso `ACCESS_FINE_LOCATION`, Play Store deducirá lascaracterísticas `android.hardware.location` y `android.hardware.location.gps` y, por tanto, la presencia de un componente de hardware de localización GPS, con carácter obligatorio.

Sintaxis

```
<uses-feature android:name="cadena de caracteres"
  android:required="booleano"
  ... />
```

Ejemplo

```
<uses-feature android:name="android.hardware.location" />
<uses-feature android:name="android.hardware.wifi"
  android:required="false" />
```

En el caso en que la característica no se requiera obligatoriamente, la aplicación debe probar dinámicamente si el dispositivo la posee o no, para adaptar su comportamiento. Para ello, es preciso usar el método `hasSystemFeature` de la clase `PackageManager` cuya instancia devuelve el método `getPackageManager`. Este método recibe como parámetro la característica que se quiere probar como cadena de caracteres y devuelve un valor booleano.

Sintaxis

```
public abstract boolean hasSystemFeature (String name)
```

Ejemplo

```
boolean gpsPresent = getPackageManager()
.hasSystemFeature(PackageManager.FEATURE_LOCATION_GPS);
```

b. uses-configuration

La etiqueta `uses-configuration` permite, a su vez, indicar las combinaciones de hardware requeridas por la aplicación. Cada uso de esta etiqueta corresponde con una combinación de hardware y software compatible.

Esta etiqueta incluye varios atributos, dos de los cuales describimos a continuación. Por ejemplo, el atributo `android:reqHardKeyboard` permite especificar si la aplicación requiere un teclado físico o no. El atributo `android:reqTouchScreen` permite indicar si la aplicación requiere una pantalla táctil.

Por defecto, en ausencia de esta etiqueta, se considera que la aplicación no requiere ni teclado físico, ni pantalla táctil.

Sintaxis

```
<uses-configuration android:reqHardKeyboard="booleano"
  android:reqTouchScreen="[finger|notouch|stylus|undefined]"
/>
```

Ejemplo

```
<uses-configuration android:reqHardKeyboard="true" />
```