Estructura del Manifiesto

¡Hola! Te recibimos con mucho entusiasmo para analizar la estructura del manifiesto en más detalle.

La base para cualquier aplicación Android es el archivo de manifiesto: **AndroidManifest.xml**. Este se encuentra en el directorio **src/main/** del módulo de tu aplicación (el conjunto principal de código y otros elementos). Allí es donde declaramos lo que está dentro de nuestra aplicación: las actividades, los servicios, etc. Asimismo indicamos cómo estas piezas se unen e interactúan con el sistema Android en general; por ejemplo, indicamos qué actividad (o actividades) deben aparecer en el menú principal del dispositivo (también conocido como lanzador o launcher).

Cuando **creamos nuestra aplicación**, **obtendremos un manifiesto de inicio** generado por nosotros. Para una aplicación simple, que ofrece una sola actividad y nada más, el manifiesto autogenerado probablemente funcionará bien, o tal vez requiera de algunas modificaciones menores. En el otro extremo del espectro, el archivo de manifiesto para la suite de demostración del API de Android tiene más de 1,000 líneas de largo. Nuestras aplicaciones Android en producción, probablemente caerán en algún punto intermedio.

Elementos en Común entre el Manifiesto y Gradle

Hay algunos elementos claves que se pueden definir en el manifiesto y se pueden anular en las instrucciones en el **build.gradle**. Estos elementos también son muy importantes para el desarrollo y el funcionamiento de nuestras aplicaciones de Android. Vamos a verlos a continuación.

Nombre del paquete e ID de la aplicación

La raíz de todos los archivos de manifiesto es, como es lógico, un elemento del manifiesto:

```
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
package="com.nextu.android.elmanifiesto">
```

Hay que tener en cuenta la declaración de espacio de nombres de Android. Sólo se usará el espacio de nombres en muchos de los atributos, no en los elementos (por ejemplo, `<manifest>`, no `<android: manifest>`). La información más importante que se debe proporcionar sobre el elemento `<manifest>` es el atributo del paquete. El atributo del paquete siempre tendrá que estar en el manifiesto, incluso para proyectos de Android Studio. El atributo del paquete controlará dónde se genera algún código fuente, en particular algunas clases tales como R y BuildConfig.

Dado que el valor del paquete se utiliza para la generación de código Java, debe ser un nombre de paquete Java válido. La **convención de Java** dice que el nombre del paquete debe basarse en un nombre de dominio inverso (por ejemplo,

com.nextu.android.miaplicacion), donde seamos propietario del dominio en cuestión. De esa manera, es poco probable que alguien más, coloque accidentalmente el mismo nombre. El paquete también sirve como el "ID de aplicación" predeterminado. Este debe ser un identificador único, tal que:

No se pueden instalar dos aplicaciones en el mismo dispositivo, al mismo tiempo, con el mismo ID de aplicación.

No se pueden cargar dos aplicaciones en Play Store con el mismo ID de aplicación (además, otros canales de distribución pueden tener la misma limitación)

De manera predeterminada, el **ID de la aplicación** es el valor del paquete, pero los usuarios de Android Studio pueden anularlo en sus archivos de compilación de Gradle. Específicamente, dentro de la parte android **{}** del **build.gradle** de la app, puede haber un **defaultConfig**, y dentro de él puede haber una declaración **applicationId**:

```
android {
 //otras declaraciones
 defaultConfig {
 applicationId "com.nextu.android.otronombreapp"
 //otras declaraciones
}
}
```

Los **usuarios de Android Studio no sólo pueden anular el ID** de la aplicación en la configuración predeterminada, sino que también **pueden tener diferentes valores del ID de la aplicación** para diferentes escenarios. Por ejemplo, un ID para la compilación de depuración (debug) y uno para la compilación de producción (release).

Valores de minSdkVersion y targetSdkVersion

En la parte correspondiente a android **{}** en el archivo build.gradle, existen un par de propiedades llamadas **minSdkVersion** y **targetSdkVersion**. Técnicamente, estos valores pueden anular (override) lo que el usuario defina a través de un elemento **<uses-sdk>** en el manifiesto, aunque pocos proyectos tienen ese elemento en la actualidad. De los dos, el más crítico es **minSdkVersion**. Este indica cuál es la versión más antigua de Android en la que la aplicación se está probando. El valor del atributo es un número entero que representa el nivel API de Android. Por ejemplo, si sólo se está probando la aplicación en Android 4.1 y versiones más recientes de Android, hay que establecer el minSdkVersion en 16. El valor inicial es el que se solicitó, cuando Android Studio creó el proyecto.

También se puede especificar un **targetSdkVersion**. Esto indica en qué versión de Android estamos pensando mientras escribimos nuestro código. Si nuestra aplicación se ejecuta en una versión más reciente de Android, el sistema operativo Android puede hacer algunos ajustes para tratar de mejorar la compatibilidad de nuestro código con respecto a los cambios realizados en el Android más reciente. Hoy en día, la mayoría de los desarrolladores de Android deberían especificar una versión de SDK de 15 o superior.

Código de versión y nombre de versión

De la misma manera, la sección **defaultConfig** tiene las propiedades **versionCode** y versionName. En principio, ellas anulan (override) los atributos android: **versionName** y **android: versionCode** que se hayan definido en el elemento raíz `<manifest>` en el manifiesto, aunque no es común encontrar muchos proyectos que usen esos atributos XML.

Estos dos valores representan las versiones de nuestra aplicación. El valor de **versionName** es lo que verá el usuario como un indicador de versión en la pantalla de detalles de la aplicación, específicamente en la configuración (Settings) de la misma. También, Play Store usa el nombre de la versión, si estás distribuyendo tu aplicación por ese medio. El nombre de la versión puede ser cualquier string que desees. El **versionCode**, por otro lado, debe ser un número entero, y las versiones más nuevas deben tener códigos de versión más altos que las versiones anteriores. Android y Play Store compararán el código de versión de un nuevo APK con el código de versión de una aplicación instalada para determinar si el nuevo APK es realmente una actualización. El enfoque típico es iniciar el código de versión en 1 e incrementarlo con cada versión de producción de tu aplicación, aunque puede elegir otra convención si lo deseas. Durante el desarrollo puedes ignorarlos, pero cuando pasas a producción, estos dos atributos serán de gran importancia.

El resto del manifiesto

No todo lo que está en el manifiesto se puede anular en los archivos de compilación de Gradle. Existen algunos elementos claves que siempre se definirán en el Manifiesto, no en un archivo build.gradle.

Una aplicación para nuestra aplicación

En el manifiesto inicial de nuestro proyecto, el elemento secundario principal del elemento `<manifest>` es un elemento de tipo aplicación `<aplicación>`. De manera predeterminada, cuando creamos un nuevo proyecto de Android, obtenemos un único elemento `<activity>` dentro del elemento `<application>`:

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
   package="com.nextu.android.elmanifiesto">
   <application
       android:allowBackup="true"
       android:icon="@mipmap/ic_launcher"
       android:label="@string/app_name"
       android:roundIcon="@mipmap/ic_launcher_round"
       android:supportsRtl="true"
       android:theme="@style/AppTheme">
       <activity
           android:name=".MainActivity"
           android:label="@string/app_name"
           android:theme="@style/AppTheme.NoActionBar">
           <intent-filter>
               <action android:name="android.intent.action.MAIN" />
```

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

El elemento `<activity>` proporciona el **android:name** para la clase que implementa la actividad, **android:label** para el nombre que se mostrará de la actividad y (a veces), el elemento secundario `<intent-filter>` que describe en qué condiciones se mostrará esta actividad. El elemento `<activity>` configura nuestra actividad para que aparezca en el iniciador (launcher), de modo que los usuarios puedan elegir ejecutarla. Podemos tener varias actividades en un proyecto, si así lo necesitamos.

El atributo **android:name**, en este caso, tiene un nombre de clase Java simple (MainActivity).

A veces, veremos **android:nombre** con un nombre de clase totalmente calificado (por ejemplo, **com.nextu.android.elmanifiesto.MainActivity**). A veces, veremos un nombre de clase Java con un solo punto como prefijo (por ejemplo, .MainActivity). Tanto **MainActivity** como **.MainActivity** se refieren a una clase Java que estará en el paquete de nuestro proyecto, la misma que declaraste en el atributo del paquete del elemento `<manifest>`.

Soporte para múltiples pantallas (resoluciones)

Los dispositivos Android vienen con una amplia variedad de tamaños de pantalla, desde pequeños teléfonos inteligentes de 2.8 "hasta televisores de 65". Android los divide en cuatro categorías (que llama buckets), según el tamaño físico y la distancia a la que se ven generalmente:

- Pequeño (menos de 3 ")
- Normal (3" a alrededor de 4.5")
- Grande (4.5" a alrededor de 10")
- Extragrande (más de 10 ")

Por defecto, nuestra aplicación admitirá pantallas pequeñas y normales. También admitirá pantallas grandes y extragrandes a través de un código de conversión automatizado integrado en Android. Para admitir verdaderamente todos los tamaños de pantalla que deseemos, deberíamos considerar agregar un elemento `<supports-screens>` a nuestro manifiesto. Esto enumera los tamaños de pantalla para los que tiene soporte explícito. Por ejemplo, si nuestra aplicación proporciona soporte de IU personalizado para pantallas grandes o extragrandes, desearíamos tener el elemento `<supports-screens>`. De esa manera, tendremos en nuestro manifiesto algo como esto:

```
<supports-screens
    android:largeScreens="true"
    android:normalScreens="true"
```

```
android:smallScreens="true"
android:xlargeScreens="true" />
```

Otros elementos

Adicionalmente, podemos encontrar otros elementos que se agregan al manifiesto, tales como:

- `<uses-permission>` para decirle al usuario que necesita permiso para usar ciertas capacidades del dispositivo, tales como acceder a Internet
- `<uses-feature>` para indicarle a Android que necesita que el dispositivo tenga ciertas funciones (por ejemplo: una cámara) y por lo tanto, nuestra aplicación no debe instalarse en dispositivos que carecen de tales funciones
- `<meta-data>` para fragmentos de información que necesitan extensiones particulares de Android, tales como FileProvider.

Por el momento nos despedimos, pero estaremos contigo próximamente en otro interesante espacio de aprendizaje.