Interfaz de usuario en Android: Controles básicos (I)

En el capítulo anterior del curso vimos los distintos tipos de layouts con los que contamos en Android para distribuir los controles de la interfaz por la pantalla del dispositivo. En los próximos capítulos vamos a hacer un repaso de los diferentes controles que pone a nuestra disposición la plataforma de desarrollo de este sistema operativo. Empezaremos con los controles más básicos y seguiremos posteriormente con algunos algo más elaborados.

En este primer post sobre el tema nos vamos a centrar en los diferentes tipos de botones y cómo podemos personalizarlos. El SDK de Android nos proporciona tres tipos de botones: los clásicos de texto (`Button`), los que pueden contener una imagen (`ImageButton`), y los de tipo on/off (`ToggleButton` y `Switch`).

No vamos a comentar mucho sobre ellos dado que son controles de sobra conocidos por todos, ni vamos a enumerar todas sus propiedades porque existen decenas. A modo de referencia, a medida que los vayamos comentando iré poniendo enlaces a su página de la documentación oficial de Android para poder consultar todas sus propiedades en caso de necesidad.

Control Button [API]

Un control de tipo `Button` es el botón más básico que podemos utilizar y normalmente contiene un simple texto. En el ejemplo siguiente definimos un botón con el texto "*Click*" asignando su propiedad `android:text`. Además de esta propiedad podríamos utilizar muchas otras como el color de fondo (`android:background`), estilo de fuente (`android:typeface`), color de fuente (`android:textcolor`), tamaño de fuente (`android:textSize`), etc.

```
<FrameLayout
xmlns:android="http://schemas.android.com/apk/res/android"
android:layout_width="match_parent"
android:layout_height="match_parent">

<EditText android:id="@+id/TxtNombre"
    android:layout_width="match_parent"
```

```
    android:layout_height="match_parent"
    android:inputType="text" />

</FrameLayout>
```

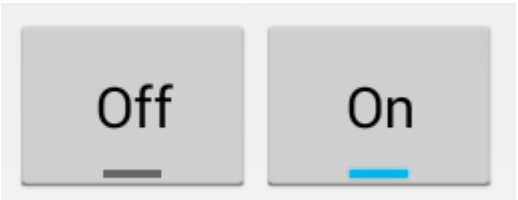
Este botón quedaría como se muestra en la siguiente imagen:

Control ToggleButton [API]

Un control de tipo `ToggleButton` es un tipo de botón que puede permanecer en dos posibles estados, pulsado o no_pulsado. En este caso, en vez de definir un sólo texto para el control definiremos dos, dependiendo de su estado. Así, podremos asignar las propiedades `android:textOn` y `android:textoOff` para definir ambos textos. Veamos un ejemplo a continuación.

```
<ToggleButton android:id="@+id/BtnToggle"
android:textOn="@string/on"
android:textOff="@string/off"
android:layout_width="wrap_content"
android:layout_height="wrap_content" />
```

El botón se mostraría de alguna de las dos formas siguientes, dependiendo de su estado:



Control Switch [API]

Un control `Switch` es muy similar al `ToggleButton` anterior, donde tan sólo cambia su aspecto visual, que en vez de mostrar un estado u otro sobre el mismo espacio, se muestra en forma de deslizador o interruptor. Su uso sería completamente análogo al ya comentado:

```
<Switch android:id="@+id/BtnSwitch"
android:textOn="@string/on"
android:textOff="@string/off"
android:layout_width="wrap_content"
android:layout_height="wrap_content" />
```

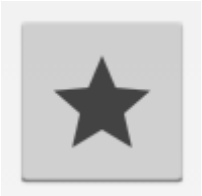
Y su aspecto sería el siguiente:

Control ImageButton [API]

En un control de tipo `ImageButton` podremos definir una imagen a mostrar en vez de un texto, para lo que deberemos asignar la propiedad `android:src`. Normalmente asignaremos esta propiedad con el descriptor de algún recurso que hayamos incluido en las carpetas */res/drawable*. Así, por ejemplo, en nuestro caso vamos a incluir una imagen llamada "*ic_estrella.png*" por lo que haremos referencia al recurso "`@drawable/ic_estrella`". Adicionalmente, al tratarse de un control de tipo imagen también deberíamos acostumbrarnos a asignar la propiedad `android:contentDescription` con una descripción textual de la imagen, de forma que nuestra aplicación sea lo más accesible posible.

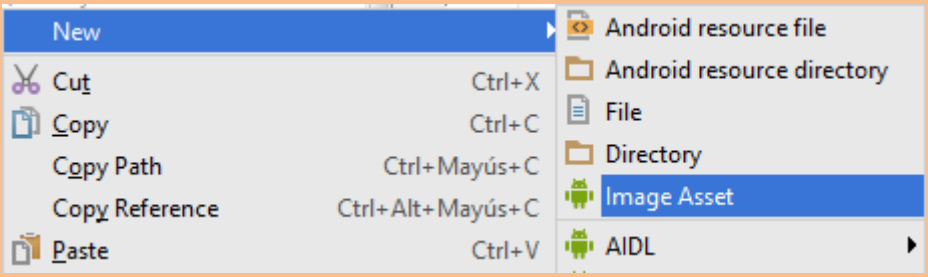
```
<ImageButton android:id="@+id/BtnImagen"
android:layout_width="wrap_content"
android:layout_height="wrap_content"
android:contentDescription="@string/icono_ok"
android:src="@drawable/ic_estrella" />
```

En una aplicación el botón anterior se mostraría de la siguiente forma:



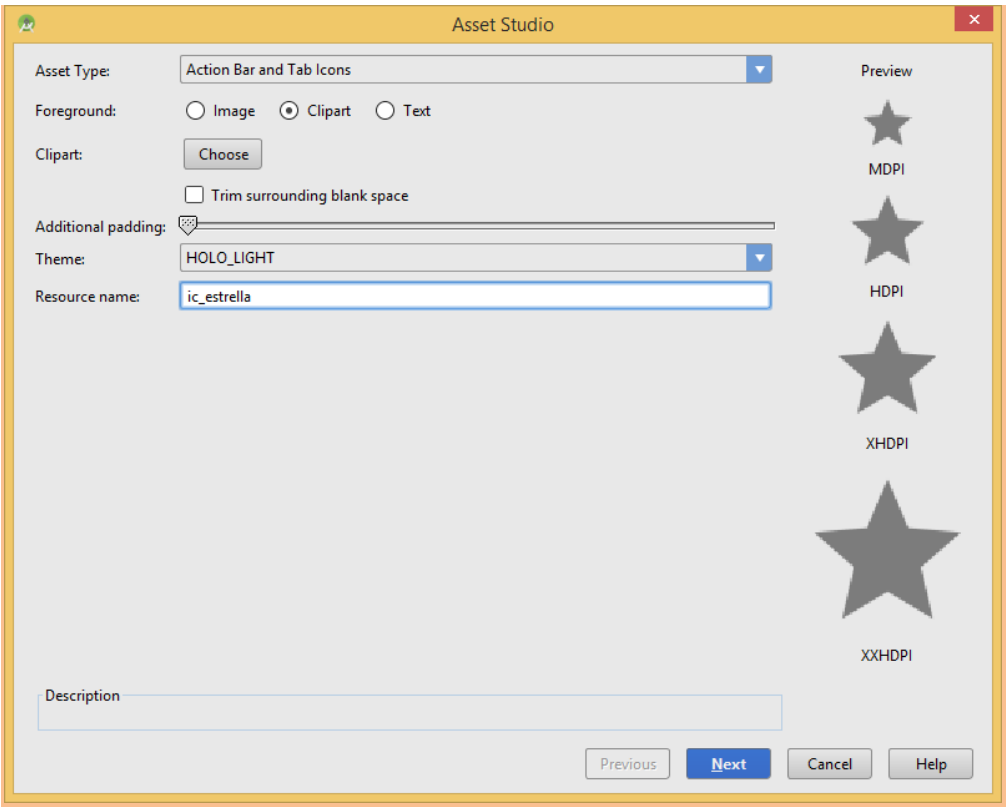
Añadir imágenes a un proyecto de Android Studio

Android Studio incorpora una utilidad llamada *Asset Studio* con la que podemos añadir rápidamente a un proyecto algunas imágenes o iconos estándar de entre una lista bastante amplia de muestras disponibles, o utilizar nuestras propias imágenes personalizadas. Podemos acceder a esta utilidad haciendo por ejemplo click derecho sobre la carpeta */main/res* del proyecto y seleccionando la opción New / Image Asset:

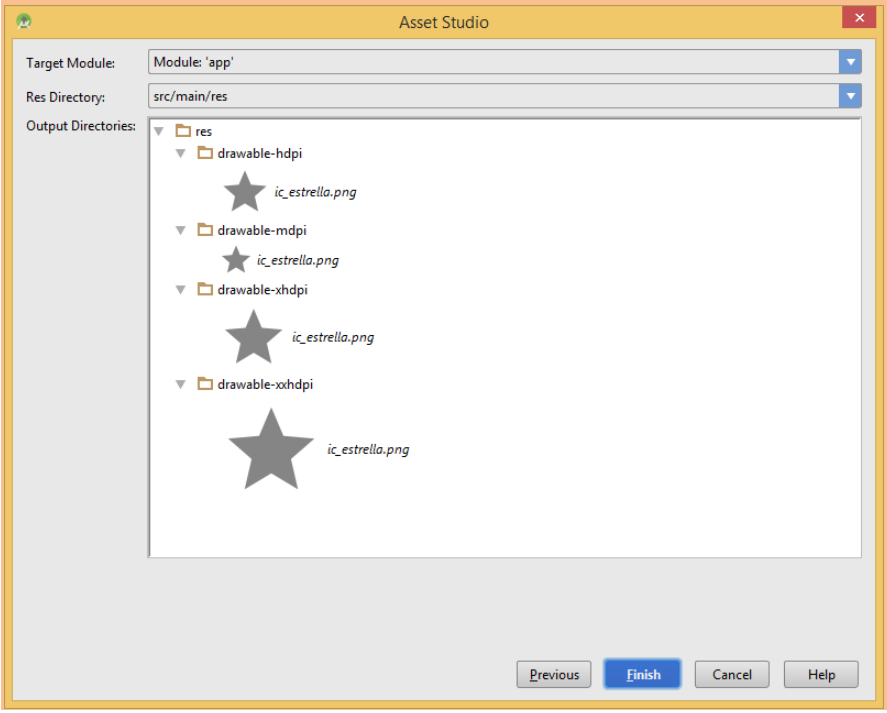


Esto nos da acceso a Asset Studio, donde podremos indicar el tipo de imagen a añadir (icono de lanzador, icono de action bar, icono de notificación, …), el origen de la imagen (*Image* = Fichero externo, *Clipart* = Colección de iconos estandar, *Text* = Texto personalizado), el tema de nuestra aplicación (lo que afectará al color de fondo y primer plano de los iconos seleccionados), y el nombre del recurso a incluir en el proyecto.

Así, en nuestro caso de ejemplo, seleccioné “Clipart” como origen de la imagen, seleccioné el icono de estrella mediante el botón “Choose”, e indiqué el nombre “ic_estrella”:



Al pulsar el botón Next el sistema consulta en qué módulo del proyecto y en qué carpeta de recursos del módulo colocará los recursos creados para el icono. Además, como podemos ver en la siguiente imagen Asset Studio se encarga de crear el icono para las distintas densidades de pixeles y colocarlo en su carpeta de recursos correspondiente.



Cabe decir además, que aunque existe este tipo específico de botón para imágenes, también es posible añadir una imagen a un botón normal de tipo `Button`, a modo de elemento suplementario al texto (*compound drawable*). Por ejemplo, si quisiéramos añadir un icono a la izquierda del texto de un botón utilizaríamos la propiedad `android:drawableLeft` indicando como valor el descriptor (ID) de la imagen que queremos mostrar, y si fuera necesario podríamos indicar también el espacio entre la imagen y el texto mediante la propiedad `android:drawablePadding`:

```
<Button android:id="@+id/BtnBotonMasImagen"
android:text="@string/click"
android:drawableLeft="@drawable/ic_estrella"
android:drawablePadding="5dp"
android:layout_width="wrap_content"
android:layout_height="wrap_content" />
```

El botón mostrado en este caso sería similar a éste:

Eventos de un botón

Como podéis imaginar, aunque estos controles pueden lanzar muchos otros eventos, el más común de todos ellos y el que querremos capturar en la mayoría de las ocasiones es el evento `onClick`, que se lanza cada vez que el usuario pulsa el botón. Para definir la lógica de este evento tendremos que implementarla definiendo un nuevo objeto `View.OnClickListener()` y asociándolo al botón mediante el método `setOnClickListener()`. La forma más habitual de hacer esto es la siguiente:

```
btnBotonSimple = (Button)findViewById(R.id.BtnBotonSimple);

btnBotonSimple.setOnClickListener(new View.OnClickListener() {
    public void onClick(View arg0)
    {
        lblMensaje.setText("Botón Simple pulsado!");
    }
});
```

En el caso de un botón de tipo `ToggleButton` o `Switch` suele ser de utilidad conocer en qué estado ha quedado el botón tras ser pulsado, para lo que podemos utilizar su método `isChecked()`. En el siguiente ejemplo se comprueba el estado del botón tras ser pulsado y se realizan acciones distintas según el resultado.

```
btnToggle = (ToggleButton)findViewById(R.id.BtnToggle);

btnToggle.setOnClickListener(new View.OnClickListener() {
    public void onClick(View arg0)
    {
        if(btnToggle.isChecked())
            lblMensaje.setText("Botón Toggle: ON");
        else
            lblMensaje.setText("Botón Toggle: OFF");
    }
});
```

Personalizar el aspecto un botón (y otros controles)

En las imágenes mostradas durante este apartado hemos visto el aspecto que presentan por defecto los diferentes tipos de botones disponibles. Pero, ¿y si quisiéramos personalizar su aspecto más allá de cambiar un poco el tipo o el color de la letra o el fondo?

Para cambiar la forma de un botón podríamos simplemente asignar una imagen a la propiedad `android:background`, pero esta solución no nos serviría de mucho porque siempre se mostraría la misma imagen incluso con el botón pulsado, dando poca sensación de elemento "clickable".

La solución perfecta pasaría por tanto por definir diferentes imágenes de fondo dependiendo del estado del botón. Pues bien, Android nos da total libertad para hacer esto mediante el uso de *selectores*. Un *selector* se define mediante un fichero XML localizado en la carpeta */res/drawable*, y en él se pueden establecer los diferentes valores de una propiedad determinada de un control dependiendo de su estado.

Por ejemplo, si quisiéramos dar un aspecto diferente a nuestro botón `ToggleButton`, para que sea de color azul y con esquinas redondeadas, podríamos diseñar las imágenes necesarias para los estados "pulsado" (en el ejemplo *toggle_on.9.png*) y "no pulsado" (en el ejemplo *toggle_off.9.png*) y crear un selector como el siguiente:

```
<selector xmlns:android="http://schemas.android.com/apk/res/android">
<item android:state_checked="false"
android:drawable="@drawable/toggle_off" />
<item android:state_checked="true"
android:drawable="@drawable/toggle_on" />
</selector>
```

En el código anterior vemos cómo se asigna a cada posible estado del botón una imagen (un elemento *drawable*) determinada. Así, por ejemplo, para el

estado “pulsado” (`state_checked=”true”`) se asigna la imagen `toggle_on`.

Este selector lo guardamos por ejemplo en un fichero llamado *toggle_style.xml* y lo colocamos como un recurso más en nuestra carpeta de recursos */res/drawable*. Hecho esto, tan sólo bastaría hacer referencia a este nuevo recurso que hemos creado en la propiedad `android:background` del botón:

```
<ToggleButton android:id="@+id/BtnPersonalizado"
android:textOn="@string/on"
android:textOff="@string/off"
android:layout_width="wrap_content"
android:layout_height="wrap_content"
android:background="@drawable/toggle_style" />
```

En la siguiente imagen vemos el aspecto por defecto de nuestro `ToggleButton` personalizado con los cambios indicados:

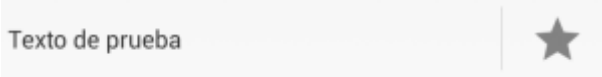
Botones sin borde

Otra forma de personalizar los controles en Android es utilizando *estilos*. Los estilos merecen un capítulo a parte, pero comentaremos aquí algunos muy utilizados en las últimas versiones de Android, concretamente en el tema que nos ocupa de los botones.

En determinadas ocasiones, como por ejemplo cuando se utilizan botones dentro de otros elementos como listas o tablas, es interesante contar con todas la funcionalidad de un botón pero prescindiendo sus bordes de forma que adquiera un aspecto plano y se intergre mejor con el diseño de la interfaz. Para ello, podemos utilizar el estilo `borderlessButtonStyle` como estilo del botón (propiedad `style`), de forma que éste se mostrará sin bordes pero conservará otros detalles como el cambio de apariencia al ser pulsado. Veamos cómo se definiría por ejemplo un `ImageButton` sin borde:

```
<ImageButton android:id="@+id/BtnSinBorde"
android:layout_width="wrap_content"
android:layout_height="wrap_content"
android:contentDescription="@string/icono_ok"
android:src="@drawable/ic_estrella"
style="?android:borderlessButtonStyle"/>
```

En la siguiente imagen vemos cómo quedaría este botón integrado dentro de un `LinearLayout` y alineado a la derecha:



El separador vertical que se muestra entre el texto y el botón se consigue utilizando las propiedades `showDividers`, `divider`, y `dividerPadding` del layout contenedor (para mayor claridad puede consultarse el código completo):

```
LinearLayout
android:id="@+id/BarraBtnSinBorde"
android:layout_width="match_parent"
android:layout_height="wrap_content"
android:orientation="horizontal"
android:showDividers="middle"
android:divider="?android:dividerVertical"
android:dividerPadding="6dp" >
```

Otro lugar muy habitual donde encontrar botones sin borde es en las llamadas barras de botones (*button bar*) que muestran muchas aplicaciones. Para definir una barra de botones, utilizaremos normalmente como contenedor un `LinearLayout` horizontal e incluiremos dentro de éste los botones (`Button`) necesarios, asignando a cada elelemento su estilo correspondiente, en este caso `buttonBarStyle` para el contenedor, y `buttonBarButtonStyle` para los botones. En nuetro ejemplo crearemos una barra con dos botones, *Aceptar* y *Cancelar*, que quedaría así:

```
LinearLayout
    android:id="@+id/BarraBotones"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:orientation="horizontal"
    android:layout_alignParentBottom="true"
    style="?android:attr/buttonBarStyle">

    <Button android:id="@+id/BtnAceptar"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_weight="1"
        android:text="@string/Aceptar"
```

```
        style="?android:attr/buttonBarButtonStyle" />

    <Button android:id="@+id/BtnCancelar"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_weight="1"
        android:text="@string/Cancelar"
        style="?android:attr/buttonBarButtonStyle" />

</LinearLayout>
```

Visualmente el resultado sería el siguiente:

Botones flotantes (Floating Action Button / FAB)

Como contenido extra de este capítulo voy a hacer mención a un nuevo "tipo de botón" aparecido a raiz de la nueva filosofía de diseño Android llamada *Material Design*. Me refiero a los botones de acción flotantes que están incorporando muchas aplicaciones, sobre todo tras su actualización a Android 5 Lollipop.

Con la reciente publicación por parte de Google de la librería *Design Support Library*, añadir este tipo de botones a nuestras aplicaciones es algo de lo más sencillo, y además aseguramos su compatibilidad no sólo con Android 5.x sino también con versiones anteriores. En esta librería se incluye un nuevo componente llamado `FloatingActionButton` con la funcionalidad y aspecto deseados.

Lo primero que tendremos que hacer para utilizarlo será añadir la librería indicada a nuestro proyecto.

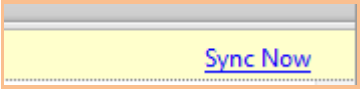
Añadir librerías externas a un proyecto de Android Studio

Para hacer uso de una librería externa en un proyecto de Android Studio tenemos dos posibilidades: añadir el fichero jar de la librería a la carpeta */libs* del módulo, o bien añadir la referencia a la librería (si está disponible) como dependencia en el fichero *build.gradle* del módulo (en la mayoría de

ocasiones usaremos esta segunda opción). En este caso, añadiremos a nuestro fichero *build.gradle* la siguiente línea en el apartado `dependencies`:

```
dependencies {
…
compile 'com.android.support:design:22.2.0'
}
```

Una vez añadida la referencia a la librería, salvamos el fichero y nos aseguramos de pulsar la opción "Sync Now" que nos aparecerá en la parte superior derecha del editor de código:



Tras esto, Android Studio se encargará de descargar automáticamente los ficheros necesarios y cuando sea necesario para que podamos hacer uso de la librería.

Una vez añadida la librería al proyecto como se describe en la nota anterior, podremos añadir un botón flotante a nuestra interfaz añadiendo un nuevo elemento a nuestro layout principal *activity_main.xml* de la siguiente forma:

```
<android.support.design.widget.FloatingActionButton
    android:id="@+id/fab"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_gravity="center"
    app:fabSize="normal"
    android:src="@drawable/ic_add" />
```

La propiedad más relevante, al igual que en el caso de un `ImageButton`, es `android:src`, con la que podemos asignar al control la imagen/icono a mostrar. En mi caso de ejemplo he utilizado un nuevo icono (`ic_add`) añadido al proyecto de la misma forma que explicamos al principio del artículo.

Vemos también la propiedad `app:fabSize`. Esta propiedad puede recibir los valores "`normal`" y "`mini`", que determinan el tamaño del control dentro de los dos tamaños estandar definidos en las especificaciones de *Material Design*.

¿Y como seleccionamos el color del botón? Pues bien, si no indicamos nada, el botón flotante tomará por defecto el *accent color* si lo hemos definido en el tema de la aplicación (más información en el artículo sobre la `Action Bar`) o el color de selección actual. Si queremos utilizar otro color debemos hacerlo desde el código java de la aplicación, por ejemplo en el `onCreate()` de la actividad principal, llamando al método `setBackgroundTintList()` del control `FloatingActionButton`.

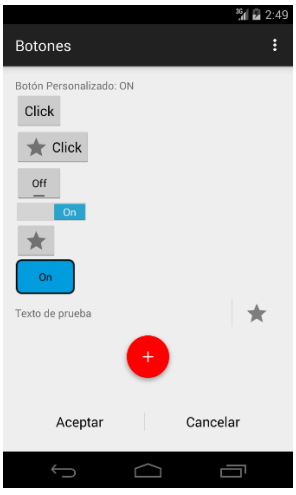
```
fabButton = (FloatingActionButton)findViewById(R.id.fab);
fabButton.setBackgroundTintList(
    getResources().getColorStateList(R.color.fab_color));
```

Como parámetro debemos pasarle un objeto de tipo `ColorStateList`, que no es más que un selector de color (similar al selector que creamos antes para el `ToggleButon` personalizado) que crearemos en la carpeta `/res/color` por ejemplo con el nombre `fab_color.xml` y que recuperaremos en nuestro código mediante el método `getResources().getColorStateList()`.

```
<selector xmlns:android="http://schemas.android.com/apk/res/android">
    <item android:state_pressed="true" android:color="#B90000" />
    <item android:color="#FF0000"/>
</selector>
```

Como podemos ver, en el selector hemos definido el color normal del botón y su color cuando está pulsado.

Para terminar, en la imagen siguiente se muestra la aplicación de ejemplo completa, donde se puede comprobar el aspecto de cada uno de los tipos de botón comentados:



Puedes consultar y/o descargar el código completo de los ejemplos desarrollados en este artículo accediendo a la página del curso en GitHub.

Enlaces de interés:

- Botones en Material Design
- Botones en Guía de diseño Android
- Botones en Guía de desarrollo Android
- Librería futuresimple/android-floating-action-button (GitHub)
- Librería makovkastar/FloatingActionButton (GitHub)