



# Crear estilo personalizado

El diseño visual de una **aplicación Android** es representado a través de reglas contenidas en **themes.xml** y **styles.xml**. Estas herramientas permiten que los programadores y diseñadores generen una interfaz más amigable y personalizada de sus apps, para establecer una identidad que impacte al usuario final.

Un estilo es un conjunto de reglas que determinan la apariencia y formato de un View o Layout. El **color de fondo**, cambiar el **tamaño del texto**, definir el **alto** y **ancho**, etc., son características que hacen parte de los estilos.

Aunque las propiedades se pueden especificar en nuestro mismo layout (como lo hemos hecho hasta ahora), es posible independizarlos del diseño a través de un archivo de recurso de estilos. Este concepto es muy similar cuando desarrollamos websites, separando los **archivos html** de los **estilos css**

Para definir estilos propios, crearemos un fichero de recursos (new Files/Values Recurse Files) en el directorio *res/values* llamado **styles.xml**.

Ahora, para definir un estilo usaremos el elemento `<style>` y le asignaremos un nombre único a través de su atributo `name`. Para definir las reglas que lo componen crearemos elementos `<item>` en su interior, detallando el nombre del atributo a modificar y su respectivo valor.

Veamos un ejemplo:

```
<?xml version="1.0" encoding="utf-8"?>
<resource>
  <style name="buttonStyle">
    <item name="android:layout_width">wrap_content</item>
    <item name="android:layout_height">wrap_content</item>
    <item name="android:textColor">#AEC6CF</item>
  </style>
</resource>
```

Si deseáramos implementar este estilo en un botón dentro de un layout, entonces referenciamos un acceso a los recursos de estilos con la convención `@style/nombreEstilo` como se muestra a continuación:

```
<Button
  style="@style/buttonStyle"
  text="Clickeame"/>
```

## Herencia de estilos

El elemento `<style>` también puede heredar propiedades de otro estilo a través de su atributo `parent`. Esta relación permite copiar las reglas del estilo padre y sobrescribir o añadir propiedades. Veamos un ejemplo

```
<style name="buttonStyle" parent="@style/parentStyle">
```

## ¿Qué es un tema?

Un tema es un estilo genérico que se asigna a una aplicación completa o actividad. Esto permite que todos los componentes sigan un mismo patrón de diseño y personalización para mantener consistencia en la UI.

Si deseamos añadir un tema a una aplicación debemos dirigirnos al archivo *AndroidManifest.xml* y agregar al elemento `<application>` el atributo **android.theme** con la referencia del tema solicitado.

Veamos:

```
<application android:theme="@style/MiTema">
```

Si te das cuenta, en el *AndroidManifest.xml* pone que estamos usando el tema: **android:theme="@style/Theme.MyLayoutsEjemplo"**, que **está definido en el fichero *res/values/themes/themes.xml***.

Si fuese una actividad entonces haríamos exactamente lo mismo:

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

Se puede aplicar incluso a un componente (view) específico, usando igualmente el atributo **android.theme**.

## Crear tu propio tema

Para facilitar la personalización de un tema nuevo es recomendable extender las propiedades de los temas que Android contiene. Esto nos permitirá ahorrarnos tiempo en definición y escritura, por lo que solo se implementan las reglas que deseamos modificar en particular.

Supongamos que deseamos usar el tema *Holo.Light* en nuestra aplicación pero deseamos todo el formato de texto *itálico*. Para conseguir este resultado nuestro tema heredará la mayoría de características del tema y solo tendremos que editar el atributo `android:textStyle`.

```
<style name="Italic" parent="@android:style/Theme.Holo.Light">
    <item name="android:textStyle">italic</item>
</style>
```

Otro ejemplo podría ser:

```
<style name="NewTheme" parent="Theme.AppCompat.Light.NoActionBar">
    <item name="android:textStyle">italic</item>
</style>
```

Y después actualizamos en el fichero *AndroidManifest.xml* donde pondremos:

```
<application android:theme="@style/Italic">
```

## Cambiar el fondo de nuestras actividades

Es normal que deseemos cambiar el aspecto con que se proyecta una actividad en su interior por un color llamativo o una imagen de fondo. Para hacerlo, acudimos a la propiedad **windowBackground**.

Los atributos que empiezan por el prefijo `window` no son aplicables a un `view` en concreto. Ellos se aplican a una `app` o actividad como si se tratase de un todo o un solo objeto.

Este atributo recibe por referencia un color sólido, una forma o una imagen de nuestros recursos. Normalmente los colores se deben declarar como ítems `<color>`, cuyo valor es un número hexadecimal.

Ejemplo:

```
<style name="NewTheme" parent="Theme.AppCompat.Light.NoActionBar">
    <item name="android:windowBackground">@android:color/holo_blue_light</item>
</style>
```

Para setear una imagen simplemente usamos una referencia drawable como ya hemos hecho antes:

```
<style name="NewTheme" parent="Theme.AppCompat.Light.NoActionBar">
    <item name="android:windowBackground">@drawable/background</item>
</style>
```

### Color del texto

Para cambiar el color del texto tendremos que modificar el atributo **android:textColor** y asignarle el valor del color. Veamos:

```
<style name="NewTheme" parent="Theme.AppCompat.Light.NoActionBar">
<item name="android:textColor">#25383C</item>
</style>
```

### Color CheckBox

El color que usan interruptores, controles deslizantes, casillas de verificación, botones de opción y otros widgets, se derivan del atributo **android.colorAccent** que se declara en el tema principal de la app.

```
<style name="NewTheme" parent="Theme.AppCompat.Light.NoActionBar">
    <item name="colorAccent">@color/colorPrimary</item>
</style>
```

### Ejemplo tema propio:

Define dentro del fichero themes.xml, un estilo propio con las siguientes características:

AppNewTheme: Es el nombre del tema general de la aplicación y hereda sus atributos del tema Theme.AppCompat.Ligth.NoActionBar. Atributos nuevos:

- android:editTextStyle: Estilo visual de los Edit Texts. Aquí hicimos referencia al estilo EditTextStyle creado más abajo.
- windowFullscreen: true, para que las actividades de la aplicación se ejecuten en pantalla completa.
- textAppearance: Tamaño de la fuente del view. Normalmente nos referiremos a tres tamaños: Small (Pequeño), Medium (Mediano) y Large (Grande). Como son valores definidos en el tema actual, se accederá por ejemplo de la siguiente manera: ?android:attr/textAppearanceMedium
- layoutMarginTop: Se refiere a la margen superior del textview con respecto a los elementos dentro del layout.
- textStyle: Representa la modalidad de texto, cuyo valor puede ser italic, bold o normal.

EditTextStyle: Contiene el estilo de los edit texts de nuestro formulario. Atributos nuevos:

- android:background: Se refiere al color de fondo

```

<resources>
    <style name="AppNewTheme" parent="Theme.AppCompat.Light.NoActionBar">
        <!-- Estilos para Edit Texts-->
        <item name="android:editTextStyle">@style/EditTextStyle</item>
        <!--Estilos de ventana-->
        <item name="android:windowFullscreen">true</item>
        <!-- Estilos textView →
        <item name="android:layout_width">match_parent</item>
        <item name="android:layout_height">wrap_content</item>
        <item
name="android:textAppearance">?android:attr/textAppearanceMedium</item>
        <item
name="android:textColor">@android:color/holo_blue_bright</item>
        <item name="android:layout_marginTop">10dp</item>
        <item name="android:textStyle">italic</item>
    </style>
    <style name="EditTextStyle"
parent="@android:style/Widget.Holo.Light.EditText">
        <item name="android:background">@android:color/white</item>
    </style>
</resources>

```

**Nota:** Se pueden definir temas específicos para una versión en concreto de Android.

Ejemplo: values-v23/themes.xml

**Nota:** Las reglas de estilo para la versión oscuro, se definirá sobreescribiendo los estilos definidos para claro, pero dentro del archivo [themes.xml](#) que hay en la carpeta [values-night](#).

## Cambiar a estilo oscuro:

Para cambiar al estilo oscuro en tu aplicación Android, puedes seguir estos pasos:

1. **Define un tema oscuro en tu archivo styles.xml:** Crea un nuevo tema que herede de un tema base oscuro, como Theme.AppCompat.DayNight.DarkActionBar o Theme.MaterialComponents.DayNight.DarkActionBar.
2. **Aplica el tema oscuro en tu Activity:** En tu Activity, llama a `AppCompatActivity.setDefaultNightMode()` con el modo nocturno deseado. Puedes usar `MODE_NIGHT_YES` para activar el modo oscuro, `MODE_NIGHT_NO` para desactivarlo, o `MODE_NIGHT_FOLLOW_SYSTEM` para seguir la configuración del sistema.

Para que los cambios de tema se apliquen, debes recrear la Activity. Puedes hacerlo llamando a `recreate()`, si estás aplicando el cambio desde un Fragment llama a `requireActivity().recreate()`

Ejemplo:

```
fun switchToDarkMode(isDarkModeEnabled: Boolean) {
    if (isDarkModeEnabled) {
        AppCompatActivity.setDefaultNightMode(AppCompatActivity.MODE_NIGHT_YES)
    } else {
        AppCompatActivity.setDefaultNightMode(AppCompatActivity.MODE_NIGHT_NO)
    }
    recreate()
}
```

Puedes guardar la preferencia del usuario para el modo oscuro en [SharedPreferences](#) y aplicarla al iniciar la aplicación.

Puedes escuchar los cambios de configuración del sistema (como el modo nocturno) y actualizar el tema de tu aplicación en consecuencia.

## Cambiar de un tema a otro:

Teniendo en cuenta que tenemos definidos diferentes temas en nuestro fichero [styles.xml](#), guarda la preferencia del usuario para el tema en [SharedPreferences](#), aplica el tema al iniciar la aplicación, en el método `onCreate()` de tu Activity, lee la preferencia del tema y aplica el tema correspondiente antes de llamar a `super.onCreate()`. Aplica el tema especificado a la Activity usando `setTheme()`.

Ejemplo aplicación en [MainActivity](#), del tema guardado en [SharedPreferences](#) al iniciar la aplicación:

```
class MainActivity: AppCompatActivity() {  
    override fun onCreate(savedInstanceState: Bundle?) {  
        val sharedPreferences =  
PreferenceManager.getDefaultSharedPreferences(this)  
        val themeName = sharedPreferences.getString("pref_theme",  
"AppTheme") // Lee el nombre del tema  
        when (themeName) {  
            "AppTheme" -> setTheme(R.style.AppTheme)  
            "AppTheme.Dark" -> setTheme(R.style.AppTheme_Dark)  
            "AppTheme.Red" -> setTheme(R.style.AppTheme_Red)  
            // ... otros temas  
        }  
        super.onCreate(savedInstanceState)  
        setContentView(R.layout.activity_main)  
        // ...  
    }  
}
```

Recordar que para que los cambios de tema se apliquen, debes recrear la Activity. Puedes hacerlo llamando a `recreate()`, si estás aplicando el cambio desde un Fragment llama a `requireActivity().recreate()`.

También es importante, establecer el tema antes de inflar las vistas, fíjate como se llama antes de llamar al `super.onCreate` y a `setContentView`...