

# 56. Estilos - Consistencia de UI en la App

# El problema

Los valores XAML duplicados son propensos a errores y difíciles de mantener

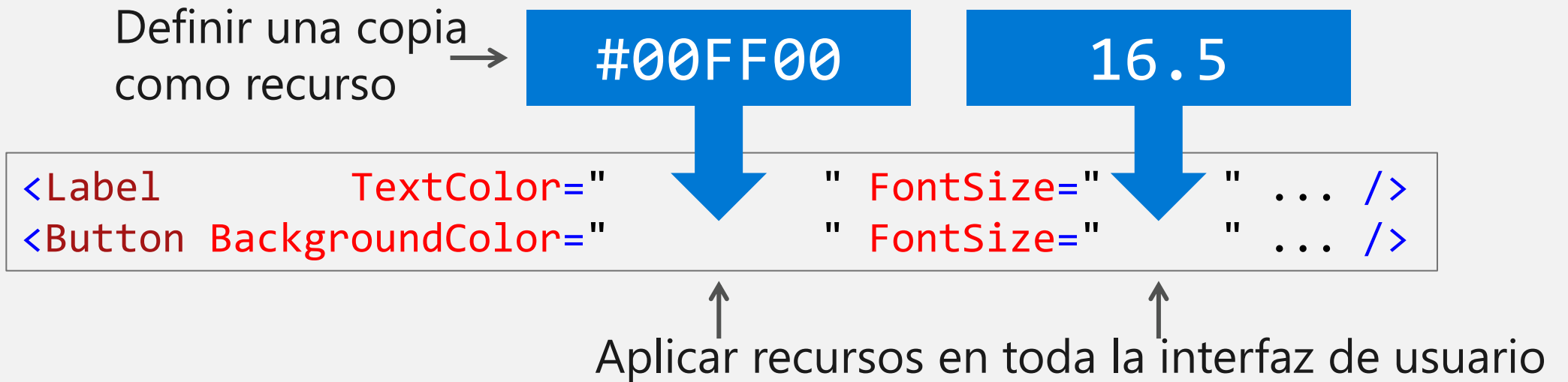
```
<StackLayout BackgroundColor="#FFFFFF">
  <Label      TextColor="#00FF00" FontSize="16.5" ... />
  <Entry      BackgroundColor="#FFFFFF" ... />
  <BoxView    BackgroundColor="#00FF00" ... />
  <Button     BackgroundColor="#00FF00" FontSize="16.5" ... />
</StackLayout>
```

Es común usar los mismos colores y tamaños  
en la interfaz de usuario de la app

# Recursos

# ¿Qué es un Recurso?

Un recurso es un objeto que se puede usar en varios lugares de la interfaz de usuario.



# ¿Qué es un ResourceDictionary?

ResourceDictionary es un diccionario clave/valor que se personaliza para su uso con los recursos de la interfaz de usuario

```
public sealed class ResourceDictionary : ...  
{ ...  
    public object this[string index] { get; set; }  
  
    public void Add(string key, object value);  
    public void Add(Style implicitStyle);  
}
```

# Resources a nivel de página

Cada página puede tener un diccionario de recursos, debe configurarse en código o XAML

Debes crear el  
diccionario

```
<ContentPage ...>  
    <ContentPage.Resources>  
        <ResourceDictionary>  
            ...  
        </ResourceDictionary>  
    </ContentPage.Resources>  
</ContentPage>
```

Asignar el diccionario  
en la propiedad Resources  
de la página

# Creando Resources

Los recursos creados en XAML deben usar la palabra clave del lenguaje XAML x:Key para establecer la clave.

Crear dentro  
del diccionario  
de la página

```
<ContentPage ...>  
  
  <ContentPage.Resources>  
    <ResourceDictionary>  
      <Thickness x:Key="myKey">10,20,40,80</Thickness>  
    </ResourceDictionary>  
  </ContentPage.Resources>  
  
</ContentPage>
```

Value

Key



Elije los nombres de los recursos en función del uso, no del valor; p.ej. use bgColor, no

# Usando static Resources

The StaticResource markup extension recupera un recurso, el valor se aplica una vez cuando se crea el objeto de destino

Definición

```
<ContentPage ...>

  <ContentPage.Resources>
    <ResourceDictionary>
      <Thickness x:Key="myKey">10,20,40,80</Thickness>
    </ResourceDictionary>
  </ContentPage.Resources>

</ContentPage>
```

Uso

```
<StackLayout Padding="{StaticResource myKey}">
  ...
</StackLayout>

</ContentPage>
```



# Tipos intrínsecos en XAML

La especificación XAML define muchos tipos que puede usar para los recursos XAML

String y  
Double son  
Útiles para  
muchas UIs  
porque son  
valores  
comunes

```
<ResourceDictionary>
→ <x:String    x:Key="...">Hello</x:String>
  <x:Char      x:Key="...">X</x:Char>
  <x:Single    x:Key="...">31.4</x:Single>
→ <x:Double    x:Key="...">27.1</x:Double>
  <x:Byte      x:Key="...">8</x:Byte>
  <x:Int16     x:Key="...">16</x:Int16>
  <x:Int32     x:Key="...">32</x:Int32>
  <x:Int64     x:Key="...">64</x:Int64>
  <x:Decimal   x:Key="...">12345</x:Decimal>
  <x:TimeSpan  x:Key="...">1.23:5959</x:TimeSpan>
  <x:Boolean   x:Key="...">True</x:Boolean>
</ResourceDictionary>
```

# Valores específicos por plataforma

Puede usar objetos OnPlatform en sus diccionarios de recursos para manejar valores específicos de la plataforma

```
<ResourceDictionary>  
  <OnPlatform x:Key="textColor"  
    x:TypeArguments="Color"  
    iOS="Red"  
    Android="Blue"  
    WinPhone="Green" />  
</ResourceDictionary>
```

```
<Label TextColor="{StaticResource textColor}" ... />
```

# Disponibilidad de recursos

Puede obtener valores de recursos después del inicio; sin embargo, los recursos aplicados con StaticResource fallarán si la clave no está en el diccionario

Lanzará una  
excepción si  
La clave no es  
encontrada

```
<ContentPage ...>

    <ContentPage.Resources>
        <ResourceDictionary>
            </ResourceDictionary>
        </ContentPage.Resources>

    <StackLayout BackgroundColor="{StaticResource bg}">
        ...
    </StackLayout>

</ContentPage>
```

# Disponibilidad de recursos

Los valores de los recursos pueden cambiar con el tiempo; sin embargo, los recursos aplicados con StaticResource no se actualizarán en respuesta al cambio

El valor es aplicado cuando se crea el objeto

```
<ContentPage ...>

  <ContentPage.Resources>
    <ResourceDictionary>
      <Color x:Key="bg">Blue</Color>
    </ResourceDictionary>
  </ContentPage.Resources>

  <StackLayout BackgroundColor="{StaticResource bg}">
    ...
  </StackLayout>

</ContentPage>
```

# ¿Cómo actualizar Resources?

Puede actualizar los valores de los recursos desde el código, útil cuando descarga nuevos valores o permite que el usuario seleccione colores preferidos, tamaños de fuente, etc.

Define el  
valor en  
XAML

```
<ResourceDictionary>  
→ <Color x:Key="bg">Blue</Color>  
</ResourceDictionary>
```

Actualizar  
a un nuevo  
valor

```
void OnChangeColor()  
{  
→ this.Resources["bg"] = Color.Green;  
}
```

# Usando dynamic Resources

The DynamicResource markup extension recupera un recurso cuando se crea el objeto de destino y lo actualiza a medida que cambia el valor

BackgroundColor  
Establecido a Blue  
inicialmente.



```
<ResourceDictionary>
  <Color x:Key="bg">Blue</Color>
</ResourceDictionary>

<StackLayout BackgroundColor="{DynamicResource bg}">
  ...
</StackLayout>
```

BackgroundColor  
Cambia a Green



```
void OnChangeColor()
{
  this.Resources["bg"] = Color.Green;
}
```

# Key not found

DynamicResource deja la propiedad sin establecer si no se encuentra la clave, no es un error y no se genera ninguna excepción

Key no  
definida

```
<ContentPage ...>  
  
  <ContentPage.Resources>  
    <ResourceDictionary>  
  </ResourceDictionary>  
  </ContentPage.Resources>
```

No se asigna el valor de  
BackgroundColor

```
<StackLayout BackgroundColor="{DynamicResource bg}">  
  ...  
</StackLayout>  
  
</ContentPage>
```

# Aplicando Resources desde código

Los recursos se pueden configurar en el código usando **SetDynamicResource**, permite que la lógica aplique diferentes recursos según lógica usada en tiempo de ejecución

```
var name = new Label { Text = "Name" };  
  
if (Device.OS == TargetPlatform.iOS)  
{  
    name.SetDynamicResource(Label.TextColorProperty, "h1Color");  
}
```

La BindableProperty a asignar

La Resource key a aplicar



Consistencia con estilos

# Código repetido

Los recursos le permiten evitar valores duplicados, pero aún debe establecer cada propiedad individualmente, lo que crea desorden y produce código repetido.

Asignar la propiedad lo repetimos constantmente

```
<Button
  BackgroundColor="{StaticResource highlightColor}"
  BorderColor     ="{StaticResource edgeColor}"
  BorderRadius    ="{StaticResource edgeRadius}"
  BorderWidth     ="{StaticResource edgeSize}"
  TextColor       ="{StaticResource textColor}"
  Text             ="OK" />
```

OK

```
<Button
  BackgroundColor="{StaticResource highlightColor}"
  BorderColor     ="{StaticResource edgeColor}"
  BorderRadius    ="{StaticResource edgeRadius}"
  BorderWidth     ="{StaticResource edgeSize}"
  TextColor       ="{StaticResource textColor}"
  Text             ="Cancel" />
```

Cancel

# Rendimiento

La búsqueda de recursos puede aumentar el tiempo de inicio de su aplicación, ya que la búsqueda lleva más tiempo que asignar un valor literal

```
<Button  
  TextColor="{StaticResource textColor}"  
  ... />
```

↑  
Slower

```
<Button  
  TextColor="White"  
  ... />
```

↑  
Faster

# ¿Qué es un Setter?

Un Setter es un contenedor para el par property→value

```
<Setter Property="TextColor" Value="White" />
```

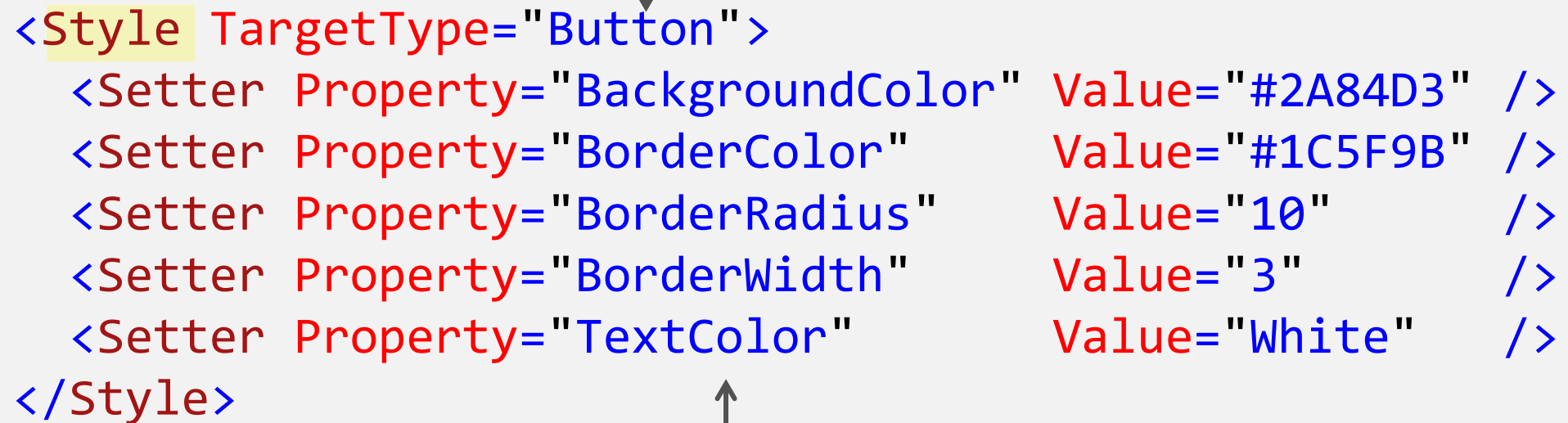
↑  
Una bindable  
property

↑  
Un valor apropiado  
Para la propiedad

# ¿Qué es un estilo?

Un estilo es una colección de setters para un tipo particular de View

TargetType debe establecerse (o excepción en runtime)



```
<Style TargetType="Button">
  <Setter Property="BackgroundColor" Value="#2A84D3" />
  <Setter Property="BorderColor" Value="#1C5F9B" />
  <Setter Property="BorderRadius" Value="10" />
  <Setter Property="BorderWidth" Value="3" />
  <Setter Property="TextColor" Value="White" />
</Style>
```

Las propiedades deben ser miembros de la clase TargetType (o excepción de tiempo de ejecución)

# Styles as Resources

Los estilos se pueden compartir, por lo que generalmente se definen como recursos

Definido en  
Dictionary

```
<ContentPage.Resources>
  <ResourceDictionary>

    <Style x:Key="MyButtonStyle" TargetType="Button">
      ...
    </Style>

  </ResourceDictionary>
</ContentPage.Resources>
```

# Usando un Style

Los estilos se establecen en un control a través de la propiedad Style, esto aplica todos los Setters en el estilo a ese control

```
<Button Text="OK" Style="{StaticResource MyButtonStyle}" />  
<Button Text="Cancel" Style="{StaticResource MyButtonStyle}" />
```

↑  
La propiedad Style está definida  
en la clase base de VisualElement  
para que esté disponible en todas las vistas

# Combinando estilos y recursos

Puede usar un recurso como valor para un setter, esto le permite compartir un valor con otros estilos

```
<Color x:Key="bgColor">White</Color>
<Color x:Key="fgColor">Black</Color>

<Style TargetType="Button" x:Key="AllButtons">
  <Setter Property="BackgroundColor" Value="{StaticResource bgColor}" />
  <Setter Property="TextColor" Value="{DynamicResource fgColor}" />
  ...
</Style>
```

Puedes combinar Static y Dynamic





# Estilos implícitos

Los estilos se pueden aplicar automáticamente a todos los controles de un tipo de destino omitiendo x:Key y colocando el estilo en un diccionario accesible

```
<ContentPage.Resources>
  <ResourceDictionary>
    → <Style TargetType="Button">
      <Setter Property="BackgroundColor" Value="Blue" />
      <Setter Property="BorderColor" Value="Navy" />
      ...
    </Style>
  </ResourceDictionary>
</ContentPage.Resources>
```

El tipo de objetivo todavía se especifica y se empareja exactamente, este estilo se aplicará a todos los botones de esta página

# Sobrecargar un setter

Los estilos proporcionan los valores predeterminados, los valores de propiedad explícitos en el control se aplican después del estilo y tienen prioridad.

```
<Style x:Key="MyButtonStyle" TargetType="Button">  
    <Setter Property="BackgroundColor" Value="Red" />  
</Style>
```

```
<Button  
    Style="{StaticResource MyButtonStyle}"  
    BackgroundColor="Blue" ✓  
    Text="Cancel"  
    ... />
```

El valor establecido anula directamente el valor del estilo



Background es blue, no red

# Antepasados

Un estilo puede apuntar a un tipo base del objeto al que se aplica

Este estilo es para VisualElement



```
<Style x:Key="MyVisualElementStyle" TargetType="VisualElement">  
  <Setter Property="BackgroundColor" Value="#2A84D3" />  
</Style>
```

```
<Button Style="{StaticResource MyVisualElementStyle}" ... />
```



Se puede aplicar a un botón desde el Botón  
la clase se deriva de VisualElement

# Creando un estilo en código

Los estilos se pueden crear en código para permitir personalizaciones en tiempo de ejecución

```
var s = new Style(typeof(Button));  
  
s.Setters.Add(new Setter {Property = Button.BackgroundColorProperty, Value = Color.Red});  
s.Setters.Add(new Setter {Property = Button.BorderRadiusProperty, Value = 4});
```



Luego puede aplicar Estilo a un botón directamente o agregarlo a los recursos para aplicar en XAML