course: COMP6215 week: 04 lesson: 01 topic: styling

DOWNLOAD PDF CLICK HERE

WPF Styling

An important aspect of GUI programming - and UI design in general - is the styling of components.

Throughout the course we have done a little bit of this using the attributes that you can access in each component. An example of this might look like:

```
<TextBox FontSize="30" Width="200" Height="50"
VerticalContentAlignment='Center' />
```

Although this may be the easiest one to implement, it comes with some serious drawbacks:

- Everything is inline, so you have to repeat the same style multiple times to multiple objects.
- A change in a single object might mean a change in other objects
- Messy code - mixing style and content makes it hard to navigate through code

Styling can be separated from the UI object themselves by using the Style tag in the associative resource tag of any element. The important parts to this way of styling are the use of the `<Style>` tag and the `<Setter>` tag with the `Property` and `Value` attributes.

An example of this would be:

```
<Image Source="Images/cat01.jpg" >
    <Image.Resources>
        <Style TargetType="Image">
            <Setter Property="Margin" Value="70, 0, 40, 40" />
            <Setter Property="Width" Value="100" />
        </Style>
    </Image.Resources>
</Image>
```

Yes you end up with more code, but it is more readable code - which is more important.

The above code, although better than what we had initially - is still not ideal. What if there was a way where can group all of the styles together?

For that we can move all of our style tags into the `<Window.Resources>` tag like this:

```
<Window.Resources>
    <Style TargetType="Border">
        <Setter Property="BorderThickness" Value="5" />
```

```
            <Setter Property="Margin" Value="10" />
            <Setter Property="Background" Value="Red" />
        </Style>
        <Style TargetType="Image" >
            <Setter Property="Margin" Value="10, 10, 10, 10" />
            <Setter Property="MaxWidth" Value="100" />
            <Setter Property="Height" Value="100" />
        </Style>
        <!-- ...More styling ... -->
    </Window.Resources>
```

The benefit of this is that we can apply more generic styles. One `Style` tag can be applied to multiple objects.

The best way arguably is to remove any styling from a XAML page into its own file, so that they can be imported into multiple files and for that we can use a `ResourceDictionary`

```
<ResourceDictionary
xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
                    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
                    xmlns:local="clr-namespace:Styling">
    <Brush x:Key="backgroundBrush">#DCE8F3</Brush>
    <Style TargetType="Border">
        <Setter Property="BorderThickness" Value="5" />
        <Setter Property="Margin" Value="10" />
        <Setter Property="Background" Value="Red" />
    </Style>
</ResourceDictionary>
```

We then only need to import the dictionary into each of the XAML files where we want to apply it by doing this:

```
<Window.Resources>
    <ResourceDictionary>
        <ResourceDictionary.MergedDictionaries>
            <ResourceDictionary Source="StyleDictionary.xaml"/>
        </ResourceDictionary.MergedDictionaries>
    </ResourceDictionary>
</Window.Resources>
```

Using Keys

The above methods have been directly targetting tags - and in some cases this is a valid approach. However what if you have a style that you only want to apply to some objects of the same type?

For that we can use the `x:Key` attribute and pass in a value, which would be the referencee name to a style that has the same reference name.

```
<Style TargetType="Image" x:Key="SomeCats" >
    <Setter Property="Margin" Value="10, 10, 10, 10" />
    <Setter Property="Width" Value="300" />
    <Setter Property="Height" Value="200" />
</Style>
```

You can reference the above style to the object you want by using a `StaticResource` value.

```
<Image Style="{StaticResource SomeCats}" Source="Images/cat03.jpg" />
```

You still need to have the `TargetType` specified, so that you don't apply the styles for a button to an image.

Cascading Styles

The style tags have a cascading order to them. What comes last counts, so if you do an inline style or apply a style using associative tags then they override the value of the `Windows.Resources` or the external dictionary.