module 15

wpf resources, styles, control templates & animation

INTRODUCTION

Building on a basic knowledge of Windows Presentation Foundation, this module explains how to build advanced graphical applications. It covers ideal application structure as well as graphical customization and animation.

OUTCOMES

By the end of this module, you should be able to do the following:

* Create a well-organised WPF application with customized visual components.
* Explain how to define and use WPF Styles and the difference between WPF Styles and CSS Styles.
* Explain the difference between StaticResource and DynamicResource.
* Define a custom Control Template.

# module 15: resources, styles, control templates & animation

## 15.1 – Property values

Property values on WPF controls are similar to their Win Forms or Web Forms counterparts. A control has a number of properties which control the look and functionality of the controls (background colour, screen position, etc.). The property values of controls can be set either in the designer, or programmatically in code. The properties can also be altered with event triggers or over time via animations.

## 15.2 – Styles

XAML gives you the ability to theme and style your entire application easily. You can define application wide defaults for buttons, list boxes, etc. You could set the property element of each control individually, but if you want them to share similar properties you can create a style which can be applied to them with the properties set.

You can start off with everything a matte lime green, and then decide to make everything glossy peach. This is done easily by changing the XAML, no code changes are required. It is similar to CSS in how you can set up the style piece by piece, and use inheritance to build on top of styles already defined.

Extending Styles is similar to subclassing in C#/VB. You may want buttons and text boxes to have the same font size and colour, but would like the button to have some additional properties to specify a gradient across the background. This can be done by first specifying one style which has the Font properties, and then creating another style that is based on that one which then specifies the background property.

When defining a style, a key is always needed. If one is not specified, then the key name will be defaulted to the **TargetType** e.g. TargetType=“Button” will implicitly set the key to be x:Name=Typeof(Button). When this happens the style also becomes the default style for that control type. Explicitly setting the key is as simple as defining x:Key=“MyStyle”. All key names must be unique.

## 15.3 – Control Templates

Styles allow for changing properties on a control: font size, colour, etc. Templates differ by allowing you to override how a control renders itself.

Typically Buttons have a **ContentPresenter** nested inside their container which is used to display the content of the button (the button text). However you may want to create a new style of button that has an image on the left hand side and its text on the right hand side. Traditionally this type of functionality required the creation of a brand new custom control class to be created.

In WPF most controls have an appearance and a behaviour which are kept separately. If the behaviour of a control is appropriate (say the clicking functionality of a button) but you want to change the appearance then you can use a control template to define how the control should display itself whilst keeping the functionality provided. XAML allows you to define a new template for the button and define what should be created *inside* of it, saving the need to get into code and writing a brand new class.

### 15.3.1 – Triggers / Event Triggers

**Triggers** happen when a property value changes on a control, this is commonly used for simple things like changing the look of a control when a user is interacting with it, For example, when the control is selected, change the background colour to a dark colour.

**EventTriggers** are akin to responding to events firing in Win Forms. However instead of writing the code to handle it in C#/VB we can now write XAML code to respond to this, which can allow us to do animate the UI in response to events.

### 15.3.2 – Animation

WPF provides a framework for animating elements of the screen. Traditionally this required the use of rendering loops and methods dedicated to tracking where all the screen objects were. Now with WPF it is simply a matter of describing what an element should do over a time period. These animations can be activated from triggers in XAML or in C#/VB.

A wide variety of animations can be created easily in the XAML code, allowing a designer to worry about how screen elements react to the user completely independently of the programmer who is writing the code behind the scenes.

A designer can animate screen elements to slide in and out of view when they are activated, to slowly change colour when a MouseOver event occurs, or just have an animated company logo on the screen.

There are many elements that can be used to animate properties, which one you need to use is based on the value type of the property. Some common ones are

* ByteAnimation
* ColorAnimation
* DecimalAnimation
* DoubleAnimation
* Int(16/32/64)Animation
* ThicknessAnimation

For example, animating a brush’s colour would require ColourAnimation, but a Margin would require ThicknessAnimation.

### 15.3.3 – Storing resources

WPF stores resources directly in XAML. Resources can be defined in a Page or Window, in the App.xaml file, or in an external ResourceDictionary.xaml file. If resources are defined in an external Resource dictionary, then the page or window that wants to reference it simply needs to add it as a Resource location.

Styles can also be defined inline, and can be scoped at the element, window, page or application level. For example, you could define a style which is only accessible within a StackPanel, or just in one window, or can be accessed from the entire application.

When searching for styles to apply, WPF will take the most relevant one found. It will start at the control element, and then slowly move up and up until it finds the first one which can be applied (The control level, the parent control, the parent control, the page, the application)

* Resources can be defined as being **Static** or **Dynamic**.   
  **StaticResources** are loaded once into memory at runtime and cannot be modified afterwards.
* **DymanicResources** can be modified after being loaded, however they take up more resources.

For 90% of purposes, using StaticResources will suffice.

# module 15: demo notes

The demo just applies a new default style to some controls. Make sure you have XamlPadx installed.

<http://blogs.msdn.com/llobo/archive/2007/12/19/xamlpadx-v3-0.aspx>

### Styles

Open labs\styles\Styledemo.Xaml in XamlPad

Show the students that the XAML is plain and simple, just some buttons and panels.

Open the snipets.txt file and start to paste the blocks 1, 2 & 3. For each show the students that the snippets are just a few lines specifying. Each snippet just sets the type to a control type, and becomes the default style for that control type.

Once snippets 1-3 have been pasted in the screen will look very different. To quickly contrast it, select all of the styles and press delete to take it back to the default.

Press undo to get all the styles back again

Paste the 4th snippet, then update one of the buttons to have

Style="{StaticResource ButtonStyle1}"

### Control Templates

Open labs\ControlTemplate\Controltemplate.xaml

This just has 2 new custom butons. The imagebutton even has the general colour changes (mouse over, mouse click, etc.)

### Animation

Well it spins stuff around. What else is there to say?