XAML Composition Patterns

In this lab, we will code in bursts. Get the starting source code and follow along. I will commit changes after every burst, so you can get caught up if you get lost.

## Gitting the Source Code

If you have a git client installed, then clone the repository. Create a working branch. For example:

cd c:\projects

git clone git://github.com/dallasxaml/Composition.git

cd Composition

git checkout –b take1

After each successful burst, commit your changes. If you ever get lost, commit that branch, go back to master, and create a new one:

git add –A

git commit –m "I missed that."

git checkout master

git pull

git checkout –b take2

## Downloading the Source Code

If you don’t have git installed, then go to the following URL and click on the “ZIP” button:

https://github.com/dallasxaml/Composition

Unzip to your project folder. If you ever get lost, go back to the web page and download the zip again.

# User Controls

If you were to start coding up your XAML in the MainWindow, your app would quickly turn into a big ball of mud. Instead, use the main window to control your layout. Put all of the design inside of user controls.

To mock out your UI, create four WPF UserControls:

* CategoryView
* FeedView
* ProfileView
* TickerView

Within each of the controls, drop a differently colored rectangle. Give each rectangle a margin to keep it away from the edge of the control.

Then rough out the rows and columns of your grid in the MainWindow. Drag the controls onto the main window, right-click, and reset their layout. This will make them fill the grid cell. Drag the controls into the correct cells. You can span multiple cells for larger content controls, like the CategoryView.

When your controls are all laid out, you will notice that the spacing around the outside edge is narrower than the internal spacing. To balance this, give the MainWindow’s grid a margin. Now each of the grid lines goes right through the center of the whitespace. Each view contributes half of the space on all sides.

Put your name into the ProfileView. Then come back to the MainWindow and set the top row to auto size. Now the view controls the layout of the surrounding page.

# Data Templates

The FeedViewModel defines a list of activities. We want to display them in a list. So we data bind to the view model using the view model locator.

DataContext="{Binding Feed, Source={StaticResource Locator}}"

And then we data bind the Activities property to a list box.

<ListBox

ItemsSource="{Binding Activities}"

HorizontalContentAlignment="Stretch" />

The list box doesn’t display the items correctly. So let’s define data templates. There are two kinds of ActivityViewModels that we can display. Create a data template for each DateType. We can also change it to a ItemsControl, since we don’t want the selection behavior.

<UserControl.Resources>

<DataTemplate

DataType="{x:Type vm:RecruitingActivityViewModel}">

<v:RecruitingActivityView />

</DataTemplate>

<DataTemplate

DataType="{x:Type vm:UserGroupActivityViewModel}">

<v:UserGroupActivityView />

</DataTemplate>

</UserControl.Resources>

<Grid>

<ScrollViewer

Margin="5" VerticalScrollBarVisibility="Auto" >

<ItemsControl

ItemsSource="{Binding Activities}"

HorizontalContentAlignment="Stretch" VerticalAlignment="Top" />

</ScrollViewer>

</Grid>

# Control Templates

Drop a couple of buttons on the CategoryView. We want to give the buttons a modern look, so let’s apply a control template. In Blend, select one of the buttons and pull down “Button”, “Edit Template”, “Edit a Copy”.

Change the border to a grid so you can put more stuff in it. Add an ellipse, but keep the ContentPresenter.

Click on the States tab to change the look of the visual states. Add a slight blue background on mouse over, and a more prominent background on mouse down.

# Custom Controls

We want each button to have a unique icon. We want an area within the button that we can replace from the container. However, we’ve already used the Content property to control what appears in the ContentPresenter. How can we define a new property to use? By creating a Custom Control.

Create a Controls folder. In that folder, create a new WPF CustomControl. Name it “ImageButton”. It should inherit from Button and add an Image dependency property:

public class ImageButton : Button

{

public static DependencyProperty ImageProperty = DependencyProperty.Register(

"Image", typeof(FrameworkElement), typeof(ImageButton));

static ImageButton()

{

DefaultStyleKeyProperty.OverrideMetadata(typeof(ImageButton),

new FrameworkPropertyMetadata(typeof(ImageButton)));

}

public FrameworkElement Image

{

get { return (FrameworkElement)GetValue(ImageProperty); }

set { SetValue(ImageProperty, value); }

}

}

Delete the Themes folder that was created for you. This would be the place to put the default control template. However, we’ve already created a control template. So we’ll modify that one. Add a namespace reference:

xmlns:c="clr-namespace:DallasXAML.Composition.Controls"

Change the target type:

<Setter Property="Template">

<Setter.Value>

<ControlTemplate TargetType="{x:Type c:ImageButton}">

<!-- -->

</ControlTemplate>

</Setter.Value>

</Setter>

Change the control type and set the Image property:

<c:ImageButton Image="{StaticResource UserGroup}"/>

Now you can bind the content presenter to the Image property.

<ContentPresenter

Content="{TemplateBinding Image}"

Margin="10,10,10,25" />