User interface design

in C#, with Windows Presentation Foundation (WPF)

# Lab 4 – Using the Model-View-ViewModel Pattern

## Functional Goals

Architect the MyTVCompanion app using the MVVM pattern to allow for better scalability and separation of business logic and UI concerns.

## Learning Goals

* Understand the Model-View-ViewModel pattern
* Understand DataContexts

## Prerequisites

* You’ll need to install Visual Studio (2012 Ultimate was used to create this lab) from the MSDN/DreamSpark service on ANGEL’s RosePortal.
* Code from Lab3 (a complete version may be available from your instructor)

## Submission Instructions

Submit answers to the **5** (or **6**, with extra credit) questions in this lab as a .pdf to the appropriate Moodle submission form.

## Window-level appearance: Titles, Icons, and Positions

Open up Visual Studio and look at MainWindow in the designer. The title on the window is MainWindow, which isn’t very informative; so, let’s add a proper title.

1. In the Window tag at the beginning of MainWindow.xaml, find Title= and change the title to “MyTVCompanion”.

Now, launch the app. Notice how it appears somewhere in the top-left quadrant of your screen? Let’s make it so that the app appears right in the middle.

1. Again in the Window tag, add a new key-value pair: WindowStartupLocation="CenterScreen".

Launch the app and notice that it now appears right in the center of your (primary) screen.

Finally, let’s replace the default WPF application icon.

1. Right-click on the MyTVCompanion project, click Add -> New Item, and select Icon File.
2. Select the 32x32, 4bit, BMP on the left side of the icon editor, and create an icon for your project using the Text Tool ().
3. When you’re done, right-click on each of the other icon types and select Delete Image Type. Eventually, you’d want to have the same icon at each size to display properly in all of Windows Explorer’s views, but the 32x32 icon will suffice for the purposes of this lab.
4. Right-click on the MyTVCompanion project and select Properties.
   1. In the Resources box on the Application tab, choose your new icon file from the dropdown.

Launch your app. If you still see the default icon, that’s OK: launching your application through Visual Studio adds some debugging layers and loses your icon. If you run the application separately (in <project folder>\bin\debug), you should see the proper icon.

Question 1: Give the Settings Window a title (“Settings – MyTVCompanion”) and set it to start centered *over the parent window*. What snippet of code achieves the parent-centering? (3 points)

## Breathing room: Margins and padding

As it stands, your application is rather cramped – the buttons and ListViews are all mashed up against the edges of the window, and it doesn’t look very good. So, let’s change that by adding a 10px margin around the edges and between the views.

1. In the outermost Grid in MainWindow.xaml, add a new Margin attribute: <Grid Margin="10">.
2. Add a middle column in your <Grid.ColumnDefinitions> with a width of 10.
3. Change the second StackPanel’s Grid.Column to 2 to account for the new column.

That Settings button is better now, but it’s still a little long and narrow; let’s make it bigger.

1. In the Button tag, add a new Padding attribute: Padding="5".

Question 2: What’s the difference between Margin and Padding (i.e. where does the added space go relative to the control)? (3 points)

## Typography talks: Font styling

Right now, all the text in your app is the same size and style, which is rather boring. Good typography can help the user skim the layout and immediately determine what they need to do, so let’s add some.

While you can change the font, it’s generally safer to stay with the default (Segoe UI). It provides a lot of FontWeight and FontStyle attributes and resembles other Windows applications the user is already familiar with.

Protip

1. In the date Label in MainWindow.xaml, add the following attributes:

FontSize="30"

FontWeight="ExtraBlack"

Question 3: What XAML attribute would change the font if you decided to do so? (3 points)

## Reusing and standardizing style: ResourceDictionaries and Styles

Now, you’ve got a much cleaner-looking main page. Your next step is to apply all those styles to SettingsWindow. But, doing the same steps means you’re going to set a lot of identical style properties in multiple places, which isn’t good programming practice – what if you change your mind down the road and want your labels to be 36 pt.?

Thankfully, WPF has a solution for us: ResourceDictionaries. ResourceDictionaries are pretty much what they sound like – a key-based storage facility for Styles and other resources.

### Adding a ResourceDictionary

1. Right-click on the MyTVCompanion project, click Add -> New Item, and select Resource Dictionary (WPF).
2. Call your ResourceDictionary MyTVCompanionResources.xaml and save it in the root of your project.

<Application.Resources>

<ResourceDictionary>

<ResourceDictionary.MergedDictionaries>

<ResourceDictionary Source="MyTVCompanionResources.xaml" />

</ResourceDictionary.MergedDictionaries>

</ResourceDictionary>

</Application.Resources>

1. In App.xaml, add the following code inside the Application tag:

### Define a new Style

1. In MyTVCompanionResources.xaml, add the following code:

<Style x:Key="HeaderLabelStyle" TargetType="Label">

<Setter Property="FontSize" Value="30" />

<Setter Property="FontWeight" Value="ExtraBlack" />

</Style>

1. Examine the code you just inserted – this is a barebones Style that has the minimum amount of information to work correctly.
   * x:Key is how the XAML parser will locate your Style when you reference it elsewhere.
   * TargetType specifies which type of Control this style can apply to. This is especially useful as it allows IntelliSense to help you create Setters.

### Apply your new Style

1. Add the following to your Label in MainWindow.xaml: Style="{DynamicResource HeaderLabelStyle}".
2. In the designer, right-click on the Label, click Edit Template -> Apply Resource. Note that HeaderLabelStyle is checked.

Run your application – the custom styling from the previous steps should still appear. Note that you can use either of the above methods – type or point-and-click – to apply Styles.

Question 4: Create a new Style “GridStyle” that specifies the 10-point margins you created earlier, and apply the style to the MainWindow top-level Grid. Submit the XAML code for the style. If you are using a word processor that doesn’t retain Visual Studio’s text formatting on copy/paste, please take a screenshot of your code so that it remains properly formatted and colored. (3 points)

## Your turn: Settings window

Now that you’ve seen how to use ResourceDictionaries and Styles, apply the look of MainWindow (including margins, padding, and font options) to SettingsWindow. SettingsWindow.xaml controls may not directly specify their own styles if those styles are the same as MainWindow styles (in other words, use your ResourceDictionary).

Question 5: Submit the XAML code and a screenshot (if you’re not sure how to take a screenshot, type “Snipping Tool” in Start Search) of your Settings Window. If you are using a word processor that doesn’t retain Visual Studio’s text formatting on copy/paste, please take a screenshot of your code so that it remains properly formatted and colored. (12 points)

## Congratulations

You’re done, unless you choose to do the extra credit below. Don’t forget: submit answers to the **5** (or **6**, with extra credit) questions in this lab as a .pdf to the appropriate Moodle submission form.

## Above and beyond: Fixing the too-long list issue

From here on is extra credit. It is possible to earn full credit on the lab without doing this section.

Search for a vague term, like “Big”, in the Settings Window. Notice that the list of possibilities (and the Add button) disappear off the screen? That’s a consequence of one of our early design choices. Do a little research (the right queries in Bing will take you right to the Visual Studio forums and a fix to this issue) and redesign SettingsWindow so that vague queries have a scrolling ListView.

Hint: Different layout managing controls (Grids, StackPanels, etc.) take different approaches to managing the height of child controls. Some let the child have as much space as it requests, and others limit the child based on style definitions set in the parent.

When you’ve fixed the issue, make sure you apply consistent margins. It’s not necessary to use the ResourceDictionary, as you will likely require a more nuanced margin setting then the one you created for the top-level Grids.

Question 6: Submit the XAML code and a screenshot (if you’re not sure how to take a screenshot, type “Snipping Tool” in Start Search) of your Settings Window with enough search results to demonstrate your fix. If you are using a word processor that doesn’t retain Visual Studio’s text formatting on copy/paste, please take a screenshot of your code so that it remains properly formatted and colored. (6 points for fix, 3 points for applying consistent styling and margins.)