Modernize Your App

Overview

Line-of-business applications often survive generations of development teams and technologies. As every technology matures, it brings with it new framework advancements, productivity benefits, and user experience improvements. With the release of Visual Studio 2013 and the .NET Framework 4.5 (and the more recent .NET 4.5.1), WPF developers have an even better platform to build great software.

In our scenario, our expense reporting environment has been updated to take advantage of the latest features of the .NET Framework and WPF stack. In this lab, we'll take a look at some of the benefits developers were able to take advantage of, including in Visual Studio, the .NET Framework, and WPF. We'll also take a look at some of the great work Microsoft's partners have delivered to enable an even better WPF experience for developers and users.

Objectives

In this hands-on lab, you will learn how to:

- Take advantage of some of the new features of Visual Studio 2013
- Integrate some of the features of the .NET Framework and WPF shipped since version 4.0
- Work with WPF components from some of Microsoft's leading partners

Prerequisites

The following is required to complete this hands-on lab:

- Microsoft Visual Studio 2013

Setup

This lab picks up where module 3 left off.

Exercises

This hands-on lab includes the following exercises:

1. A quick tour of Visual Studio 2013, the .NET Framework 4.5.1, and WPF

Exercise 1: A quick tour of Visual Studio 2013, the .NET Framework 4.5.1, and WPF

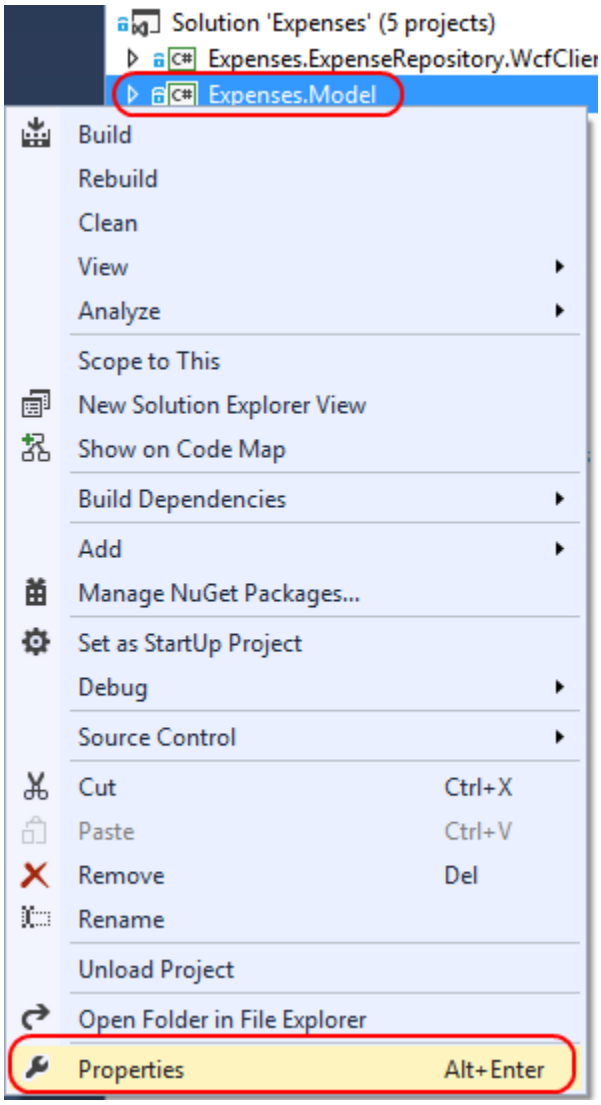
In this exercise, we'll check out some of the new features in Visual Studio 2013, the .NET Framework 4.5.1, and WPF.

Task 1: Portable class libraries

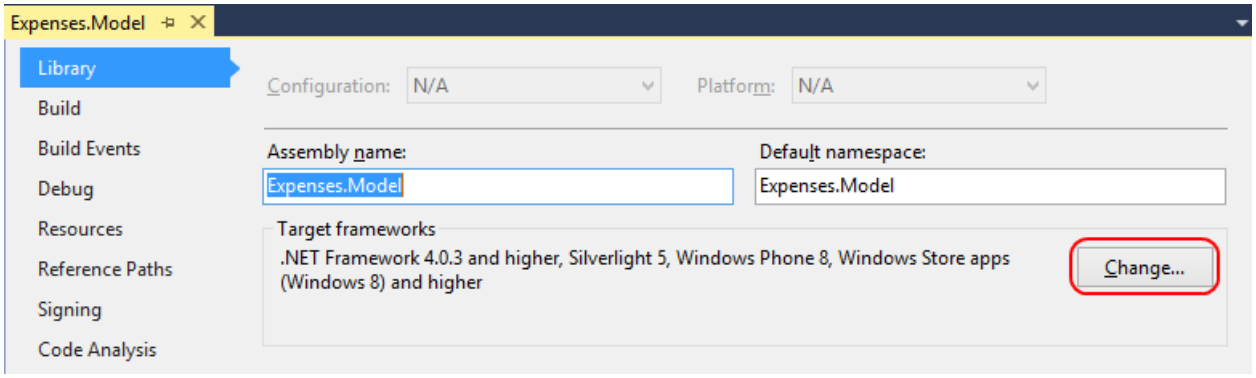
In this task, we'll take a look at portable class library support in the .NET Framework.

1. Open **Source\Begin\Expenses\Expenses.sln** in Visual Studio 2013.

2. Right-click the **Expenses.Model** project and select **Properties**.

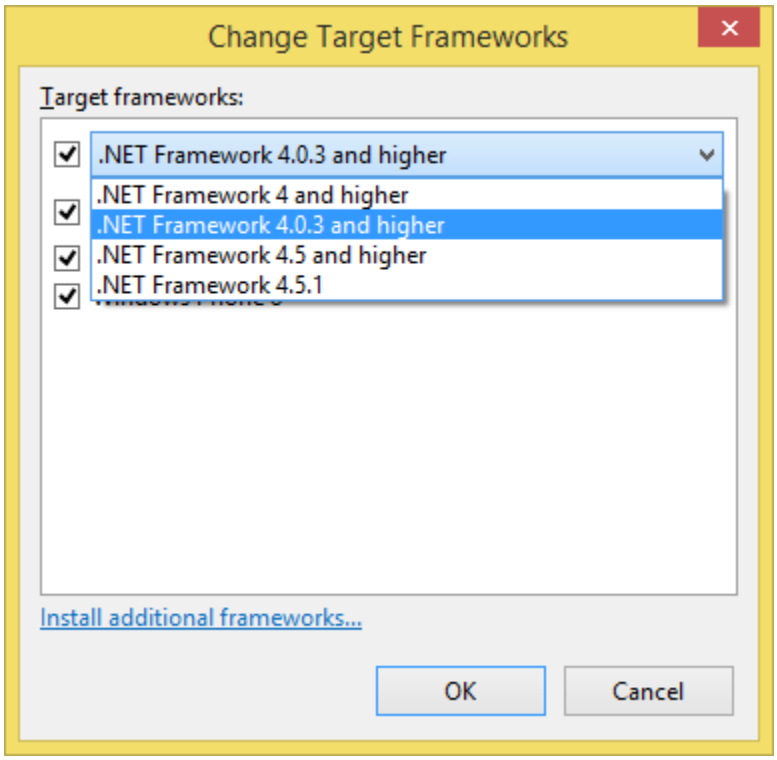


3. On the project properties page, click the **Change…** button.



4. In a portable class library, you compile your project down into a binary assembly that can be shared as-is across multiple platforms. On the **Change Target Frameworks** dialog you can see that we're building to support multiple targets:
    a. .NET Framework 4.0.3 and higher
    b. Windows Store apps (Windows 8) and higher
    c. Silverlight 5
    d. Windows Phone 8

5. On the **Change Target Frameworks** dialog, expand the first drop down to review the various .NET Framework options. If you select one of these options, Visual Studio will adjust its references to limit the available objects and APIs to only those that are available to every selected platform. While it can be great to support as many platforms as possible, there are tradeoff risks that involve losing support for key features. You can also install additional frameworks to broaden the support of platforms you can target with PCLs. Cancel out of the dialog to avoid applying and changes.



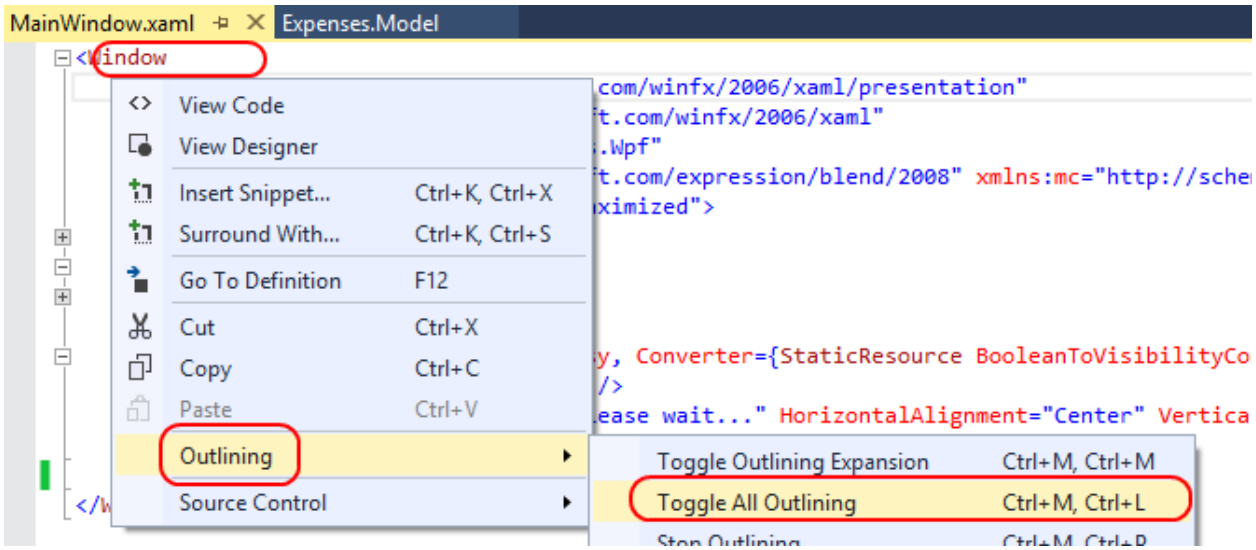
Task 2: Ribbon control

In this task, we'll take a look at the Ribbon control included with WPF.

1. From the **Solution Explorer**, double-click **MainWindow.xaml** from the **Views** folder.

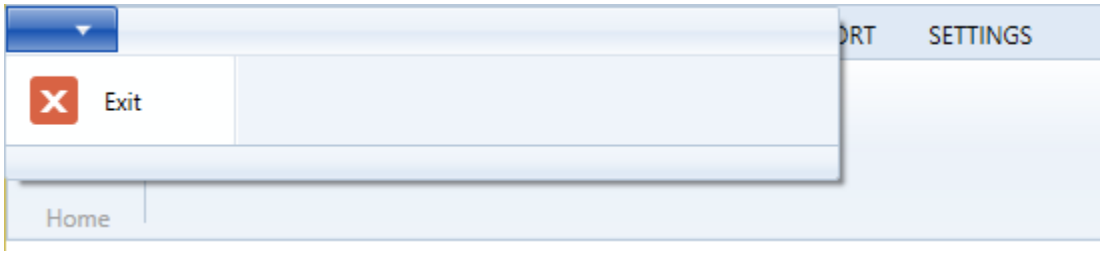
2. Switch to **XAML** view if the editor opens in design view.

3. If the **<Ribbon>** tag is not visible, right-click inside the editor window and select **Outlining | Toggle All Outlining**.

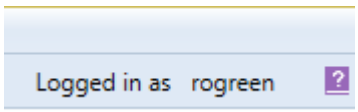


4. Locate the **<Ribbon>** tag. Note that you'll need to add a reference to **System.Windows.Controls.Ribbon** in order to use the **Ribbon**.

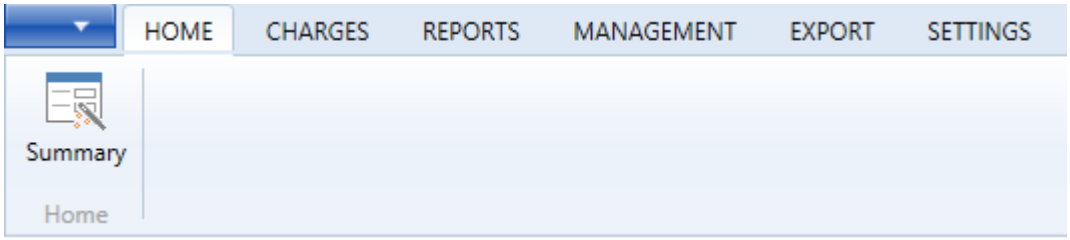
5. Locate the **<Ribbon.ApplicationMenu>** tag. The **Ribbon** is extremely flexible and provides some great options, such as the **ApplicationMenu**, which we've put a single **Exit** button into. The markup is also very easy to lay out and bind to. In our case, we've bound the button's **Command** to the current **DataContext's ExitCommand** and given it a simple PNG image.



6. Locate the **<Ribbon.HelpPaneContent >** tag. There is also a place to define the **HelpPaneContent**, where we've put the current logged-in user and a cosmetic help button. Note that we're using standard WPF content in addition to the Ribbon-centric controls.



7. The remainder of our **Ribbon** includes **RibbonTabs**, which have **RibbonGroups** with various kinds of buttons and controls. In the **Home RibbonTab**, there is a single **Home RiggonGroup** with a single **Summary RibbonButton**. All of the logic is wired up using WPF commands, allowing for very convenient integration with MVVM-style architectures. There is also a wide variety of **Ribbon** controls, including check boxes, combo boxes, drop-downs, toggle buttons, and more.

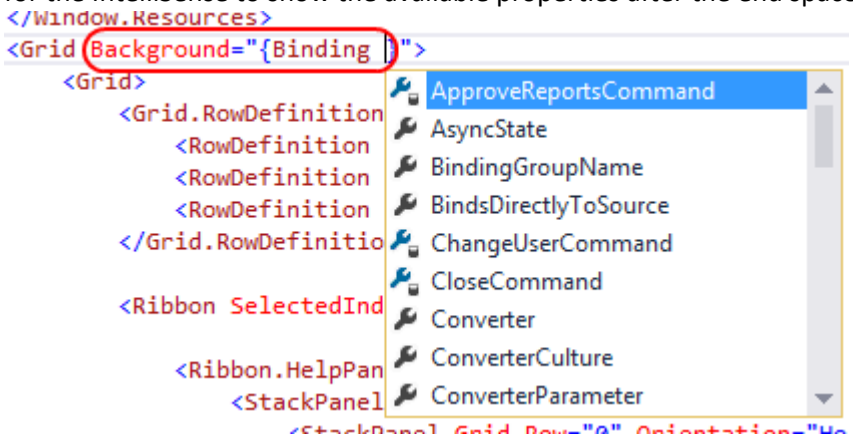


Task 3: XAML editor improvements

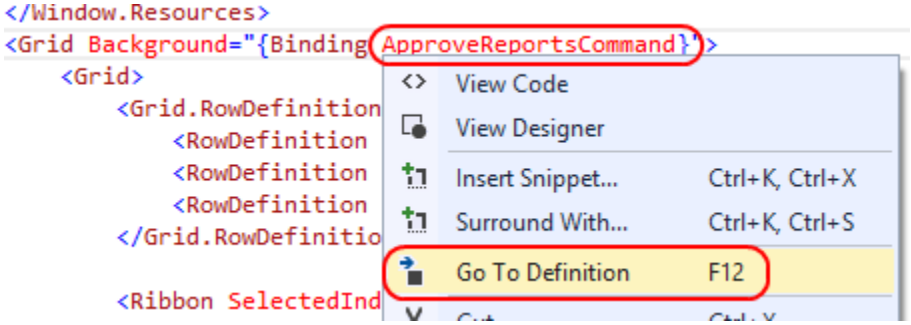
In this task, we'll take a look at some of the XAML editor improvements available in Visual Studio 2013.

1. In the XAML view of **MainWindow.xaml**, locate the **d:DataContext="{d:DesignInstance Type=uc:MainWindowViewModel}"** attribute in the **<Window>** tag. This is a design-time attribute that allows us to specify the **DataContext** type for design purposes. In this case, we're using the **MainWindowViewModel**.

2. Begin to add a **Background** attribute to the top-level **Grid**. Type **"Background={Binding "** and wait for the IntelliSense to show the available properties after the end space key is pressed.



3. Select the first option, **ApproveReportsCommand**. It doesn't really apply here, but select it for the purposes of this lab. Right-click **ApproveReportsCommand** and select **Go To Definition**.

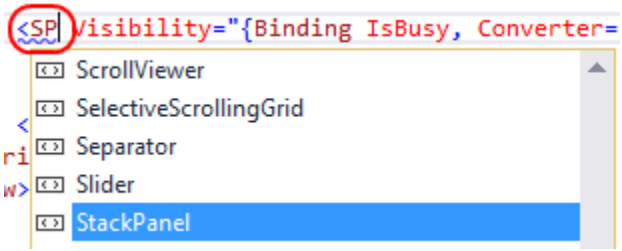


4. The **Go To Definition** feature takes us directly to the definition of the command in code. Having the ability to move seamlessly from design to code saves a lot of hunting time for developers. If the element were defined in XAML, it would bring us there, instead. For system types, it takes us to the **Object Browser**.

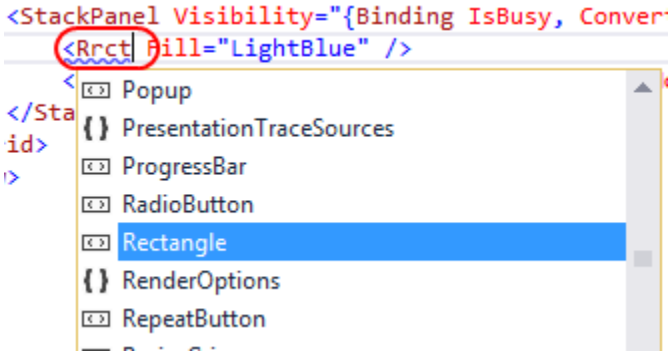
5. Switch back to **MainWindow.xaml**.

6. Scroll to the bottom of the XAML to find the last **Grid** element. Change the **<Grid>** tag to **<StackPanel**> and notice how the closing tag is automatically renamed as well.

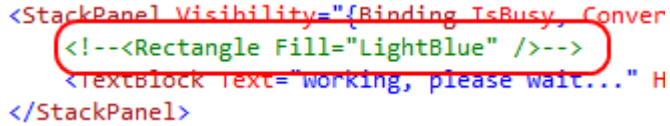
7. Alternatively, we could have taken advantage of the casing inference of the XAML editor by just typing the uppercase letters. If you replace the **"StackPanel"** with **"SP"** it will offer **StackPanel** as the first option.



8. Highlight the **"Rectangle"** in the first tag inside the **StackPanel** we just changed. Type **"Rrct"** (intentionally misspelling "Rect") and notice that the fuzzy matching employed by the XAML editor recognizes that we were probably looking for the **Rectangle**. Select the **Rectangle** option.



9. One of the challenges in working with the XAML editor was that it was annoying to comment out large blocks of XAML if the block contained comments already, such as for documentation. In Visual Studio 2013, this experience has been drastically improved. Highlight the **<Rectangle>** tag and press **Ctrl+C** and then **Ctrl+K**. This will comment out the selection.



10. Now highlight the entire enclosing **<StackPanel>** and press **Ctrl+C** and then **Ctrl+K** to comment it out. Notice how the XAML editor retains the original comment, but properly comments out the remaining XAML.

```
<!--<StackPanel Visibility="{Binding IsBusy, Converter={StaticResourc
    --><!--<Rectangle Fill="LightBlue" />--><!--
    <TextBlock Text="Working, please wait..." HorizontalAlignment="Ce
</StackPanel>-->
```

11. Close the file without saving changes.