* 1. 

Hands-On Lab

Building Applications in Silverlight 4

Lab 06 - Great UX with Blend

**Contents**

[Lab 6 - Creating a Great User Experience 3](#_Toc276644866)

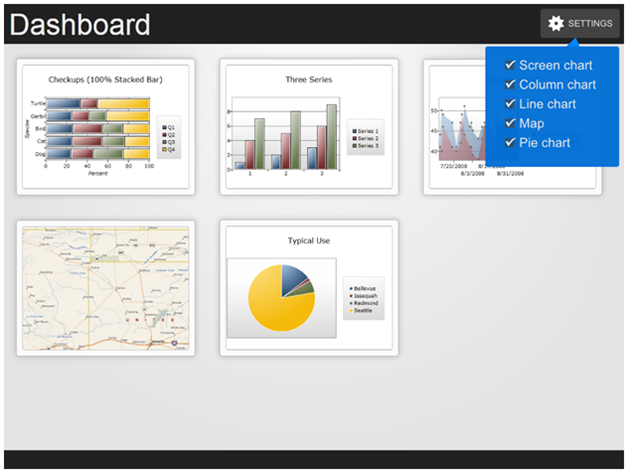
[Exercise 1: Create a State 5](#_Toc276644867)

[Exercise 2: Add Behaviors 11](#_Toc276644868)

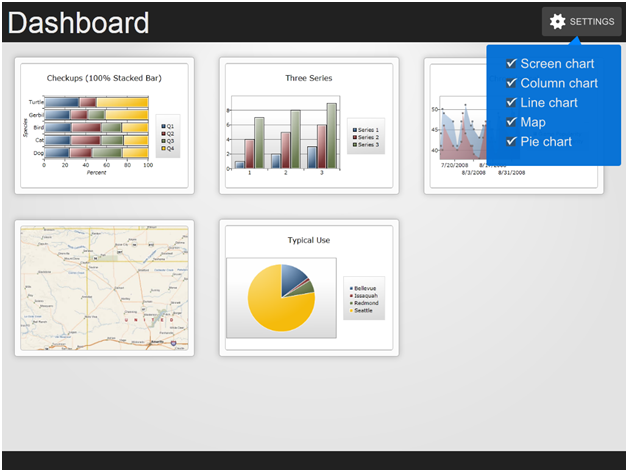
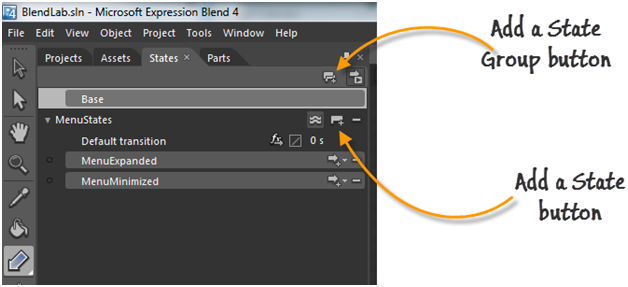
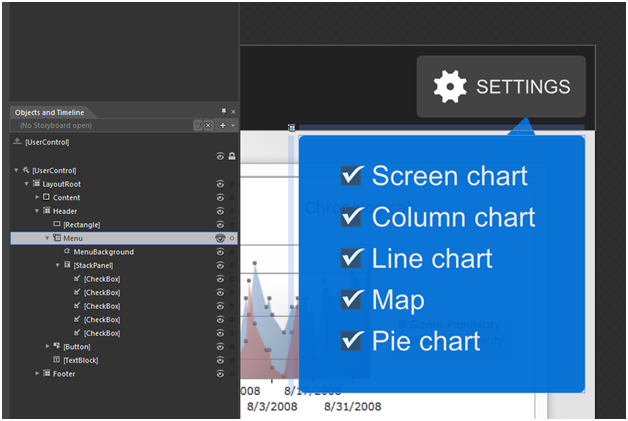
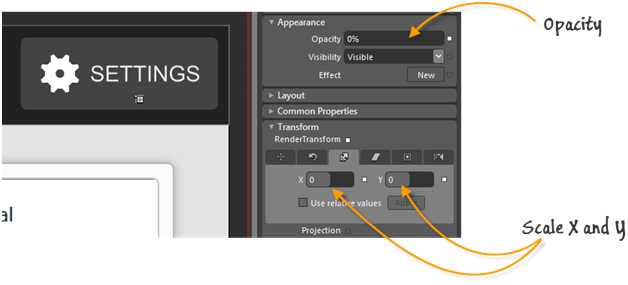
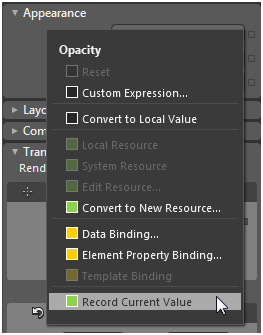
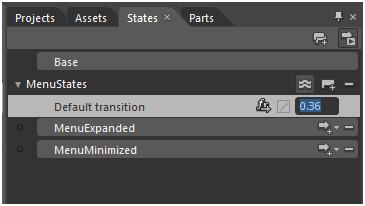
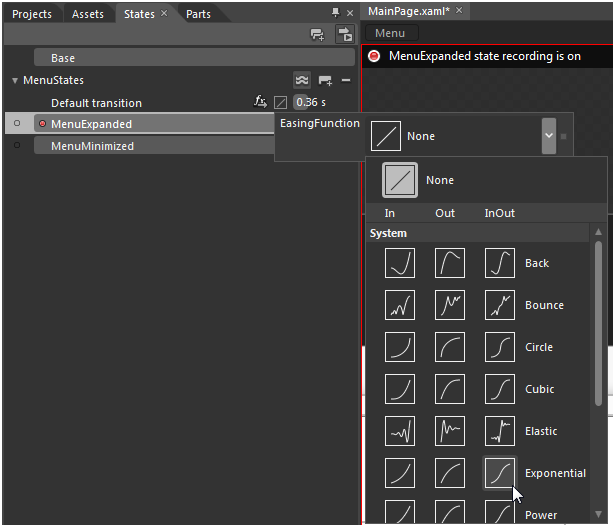
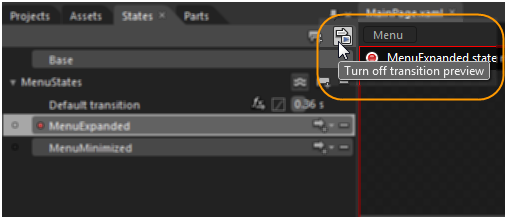
[Exercise 3: Blend playing nice with MVVM 13](#_Toc276644869)

[Summary 20](#_Toc276644870)

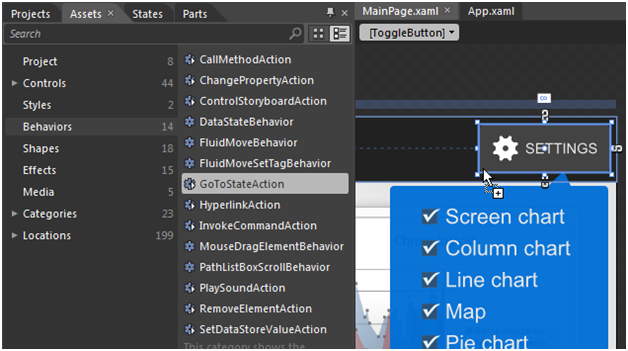
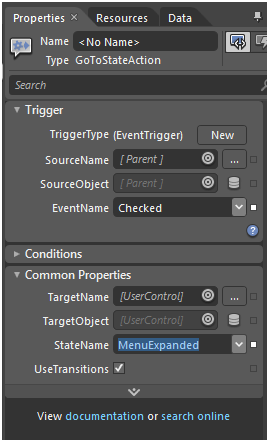
Lab 6 - Creating a Great User Experience

* 1. A user’s experience with an application is the perception of how it works. No matter how solid the backend is, if it’s easy to use the user will think it’s a great product. Silverlight enables you to quickly add small pieces of functionality that can greatly improve the user experience. This lab is designed to introduce developers to Expression Blend.
  2. In the lab you'll create a menu that transitions between an opened and closed states. Along the way you'll learn how to use Blend to create states, work with behaviors, and integrate MVVM with Blend.
  3. You'll start by create opened and closed states for the menu. Next, you’ll add behaviors to the Settings button to toggle between states without writing code. Finally, you’ll integrate the MVVM pattern into Blend. The Silverlight application that you'll create is shown next:
     1. 
     2. Figure 1
  4. **You Will Benefit from this Lab if:**
  + You are interested in adding interactivity to your application with little cost
  + You have little experience with Expression Blend
  + You want to integrate existing application design patterns with Blend
  1. **You Will Learn:**
  + Key features of Expression Blend
  + How to create visual states
  + Add interactivity without adding additional code
  + About behaviors that integrate with MVVM
  + How data binding works within Blend
  1. **Business Requirements for the Silverlight application include:**
* Create visual states for a settings menu
* Add interactivity with behaviors
* Integrate an existing ViewModel using Blend
* Change the states based on values in the ViewModel

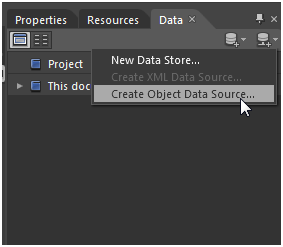
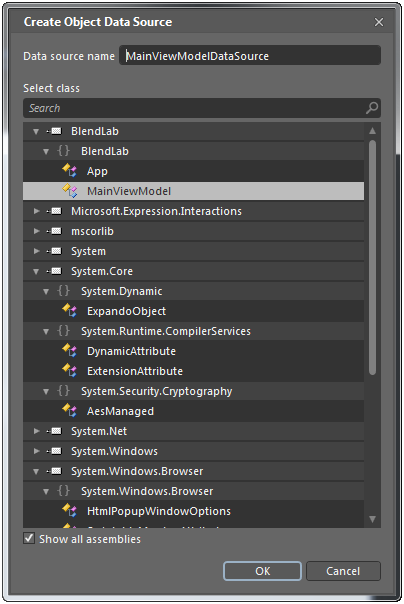
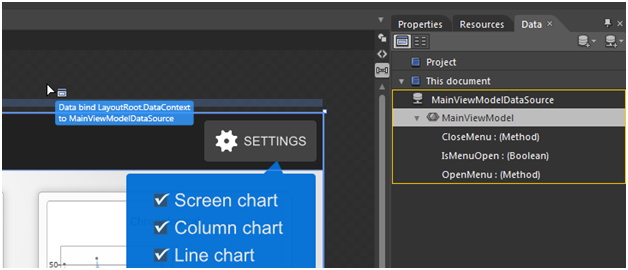
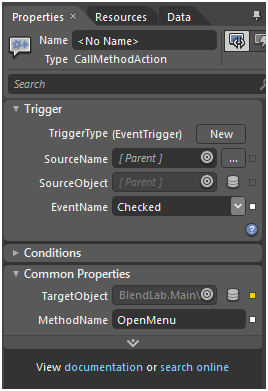
Exercise 1: Create a State

* 1. A visual state represents the visual appearance of controls in a given state. For example, take a Button. It has a MouseOver state, Pressed state, and a Disabled state. In each state there is a definition of how the control should look in that state. What you may not know is that you can create your own states.
  2. In this exercise you will create an opened and closed state for the menu in the starter solution. You will also configure the time it takes to transition between states resulting in an animation transition. Finally, you will add easing to the animation. Although a small touch, easing makes the animation feel more natural to the user.
  3. Use a pre-baked UI. The UI will be a menu that we’ll have the student expand and collapse. The goal of this lab is to add interactivity without any code.
     1. 
     2. Figure 2
     3. UI example
  4. We’re going to focus on opening and closing the menu. A fairly small task that could be accomplished with code by toggling the Visibility attribute, however the power of Silverlight is enabling you to create interactive experiences in little time and little risk to your code base.
  5. In order to create the opening and closing of the menu, you need to define two states. Do this by opening the States panel; click the Add a State Group button. Name the State Group MenuStates. Now add two states, MenuExpanded and MenuMinimized, by clicking the Add a State Button.
     1. 
     2. Figure 3
     3. Add states
  6. Now the states are defined, next you need to define what happens when the UI is in that state. In the States panel select the MenuMinimized State and change the Opacity, ScaleX and ScaleY of the Menu Panel.
     1. 
     2. Figure 4
     3. MenuMinimized State
     4. 
     5. Figure 5
     6. Change Properties
  7. Select the MenuExpanded State and set the Opacity, ScaleX, and ScaleY back to the original size (if the value is already set, then click the small advanced button and select Record Value
     1. 
     2. Figure 6
     3. MenuExpanded state
  8. At this point you have defined what the UI looks like in each state. There are a couple more properties you can set to improve the experience. The first is timing. In the States panel change the time from 0 to .36 seconds. This will interpolate the changed values.
     1. 
     2. Figure 7
     3. Set Time
  9. Add easing makes the animations for more natural. By default the animations are linear, making the transitions mechanic. Although subtle, easing makes a big difference.
     1. 
     2. Figure 8
     3. Easing function
  10. Finally, to see the transition in action, click the preview button and click between states. Running the application won’t produce anything since the states have not been wired up. The next section will discover how to use behaviors, instead of code, to change the states.
      1. 
      2. Figure 9
      3. Preview Button

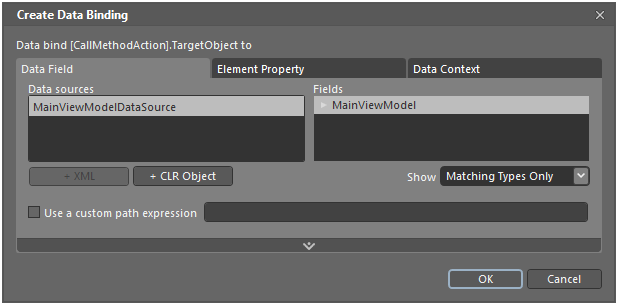
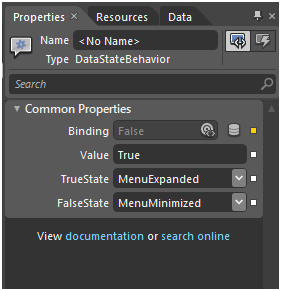
Exercise 2: Add Behaviors

* 1. A behavior is a self-contained piece of functionality. Expression Blend offers a number of prebuilt behaviors. The one this exercise focuses on is the GoToStateAction. You’ll attach it to the Settings button to toggle between states defined in the previous section.
  2. Using the states created above, navigate to the Assets Panel. Select the GoToState Behavior. Drag and drop it onto the Settings button.
     1. 
     2. Figure 10
     3. Behavior tab
  3. In the Properties panel change the Event to Checked and the State to MenuExpanded.
     1. 
     2. Figure 11
     3. Properties
  4. Now add another GoToStateAction for the MenuMinimized. Again, drag and drop another GoToState Behavior onto the Toggle Button. This time use Unchecked for the EventName and MenuMinimized for the StateName. At this point your menu will open and close without any Code. Press F5 or Project > Run Project to see the menu in action.

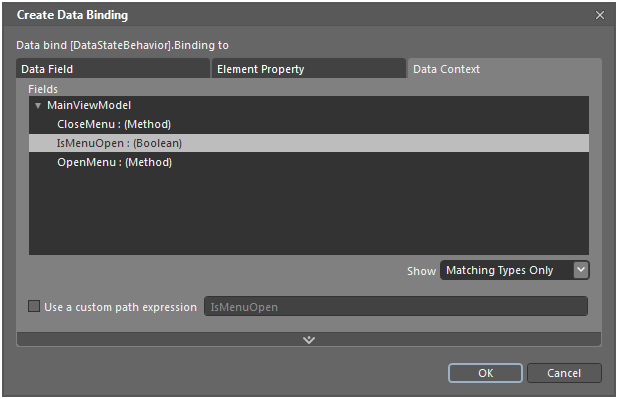
Exercise 3: Blend playing nice with MVVM

* 1. Behaviors are not limited to playing animations or changing states. As a developer there are a few behaviors specific to us; namely the DataStateBehavior, CallMethodAction, InvokeCommand. This exercise uses the DataStateBehavior and CallMethodAction to integrate into our ViewModel.
  2. In the application you’ve downloaded, there is a ViewModel already defined: MainViewModel.cs. Included is a property called IsMenuOpen and two methods OpenMenu() and CloseMenu().
     1. C#
     2. private bool isMenuOpen;
     3. public bool IsMenuOpen
     4. {
     5. get { return isMenuOpen; }
     6. set
     7. {
     8. isMenuOpen = value;
     9. NotifyPropertyChanged("IsMenuOpen");
     10. }
     11. }
     12. public void OpenMenu()
     13. {
     14. IsMenuOpen = true;
     15. }
     17. public void CloseMenu()
     18. {
     19. IsMenuOpen = false;
     20. }
     21. Visual Basic
     22. Private isMenuOpen\_Renamed As Boolean
     23. Public Property IsMenuOpen() As Boolean
     24. Get
     25. Return isMenuOpen\_Renamed
     26. End Get
     27. Set(ByVal value As Boolean)
     28. isMenuOpen\_Renamed = value
     29. NotifyPropertyChanged("IsMenuOpen")
     30. End Set
     31. End Property
     32. Public Sub OpenMenu()
     33. IsMenuOpen = True
     34. End Sub
     35. Public Sub CloseMenu()
     36. IsMenuOpen = False
     37. End Sub
  3. In Blend add the MainViewModel as a DataSource. Do this by clicking the Create DataSource button in the Data panel and then selecting Create Object Data Source.
     1. 
     2. Figure 12
     3. Create Object Data Source
  4. You’ll be prompted with a Create Object Data Source dialog box. Select the MainViewModel from dialog box. This will create a DataSource in XAML.
     1. 
     2. Figure 13
     3. Data Source dialog box
  5. After the data source is created, click and dragging it onto the Artboard. Notice the cursor below indicating bind the data source to the LayoutRoot DataContext.
     1. 
     2. Figure 14
     3. Bind the data source
  6. Instead of using the GoToState behavior for the Checked and Unchecked events, use the ViewModel to do the work. There are a number of reasons for this. For example if you need to track when a user opens or closes the menu, that functionality is better served in the VM. Or if need to request data for a dynamic menu, again this is better done in the VM.
  7. Remove the GoToState behaviors for the ToggleButton and replace them with CallMethodAction behaviors. From the Asset panel drag and drop two CallMethodAction behaviors onto the ToggleButton. From the properties panel set the EventName, TargetObject, and MethodName.
     1. 
     2. Figure 15
     3. Change Properties

|  |  |
| --- | --- |
| Property name | value |
| EventName | select Checked from the drop down |
| TargetObject | click the Advanced Options square button to the right and select Data Binding. This will bring up the Create Data Binding dialog. From this dialog, select MainViewModel and click OK. |
| MethodName | Type in OpenMenu. This is the method defined in the ViewModel |

* + 1. 
    2. Figure 16
    3. Binding window
  1. Do the same for the CloseMethod.
  2. The final step before getting this working is adding the DataStateBehavior. This will listen to the IsMenuOpen property and change the state based on the value.
  3. Back in the UI add a DataStateBehavior to the LayoutRoot and bind it to the Boolean property in the VM.
     1. 
     2. Figure 17
     3. Change Properties

|  |  |
| --- | --- |
| Property name | value |
| Binding | Like the TargetObject, click the Advanced Options button. Bind to the IsMenuOpen property of the MainViewModel (below figure) |
| Value | Set this to “True”. This is the compared value |
| TrueState | Select MenuExpanded from the drop down. |
| FalseState | Select MenuMinimized from the drop down. |

* + 1. 
    2. Figure 18
    3. Binding window
  1. At this point you’ve wired up the ViewModel to the View using the DataStateBehavior and the CallMethodAction. Run the project to see the menu open and close.

Summary

* 1. In this exercise you examined key features of Expression Blend. Visual states, behaviors, and data behaviors are three topics that will enable you to create a better user experience in minimal time and without risking your code base. This application satisfied the following requirements:
  + Create visual states for a settings menu
  + Add interactivity with behaviors
  + Integrate an existing ViewModel using Blend
  + Change the states based on values in the ViewModel
  1. Although Expression Blend has many features outside of what was covered in this lab, the topics covered will enable you to quickly add interactivity with minimal risk to your existing code. Additionally this lab showed how you can tie your user experience into your architecture.