

Hands-on lab

Lab 2: Orientation, snapping, and   
Semantic Zoom

September 2012

Contents

[Overview 3](#_Toc334622630)

[Exercise 1: Device orientation 4](#_Toc334622631)

[Task 1 – Test the start page 4](#_Toc334622632)

[Task 2 – Customize the group-detail page 6](#_Toc334622633)

[Task 3 – Customize the item-detail page 12](#_Toc334622634)

[Exercise 2: Snapping 17](#_Toc334622635)

[Task 1 – Run Contoso Cookbook in snapped mode 18](#_Toc334622636)

[Task 2 – Modify the snapped grouped-items page 18](#_Toc334622637)

[Task 3 – Modify the snapped group-detail page 19](#_Toc334622638)

[Task 4 – Modify the snapped item-detail page 20](#_Toc334622639)

[Exercise 3: Semantic Zoom 22](#_Toc334622640)

[Task 1 – Add a SemanticZoom control to the start page 23](#_Toc334622641)

[Task 2 – Test the results 25](#_Toc334622642)

[Summary 26](#_Toc334622643)

Overview

* 1. One hallmark of a great Windows Store app is that it adapts to screens of various sizes and handles rotation between landscape mode and portrait mode. It must also respond to *snapping*, which enables two apps to share the screen and run side by side.
  2. Another feature related to screen size is *Semantic Zoom*. Unlike optical zoom, which simply scales content on the screen in response to user interactions such as pinch gestures, Semantic Zoom changes the representation of the content to show more or less detail as the user zooms in and out. Semantic Zoom lets users navigate long lists of content without excessive amounts of scrolling; users can zoom out, find what they want, and then select it to go directly to the corresponding location.
  3. In this lab, you will build upon Lab 1 by adding three important user-interface(UI) related features to Contoso Cookbook. First, you will customize the layout of the item-detail and group-detail pages when the screen is rotated. Next, you will customize the layout of the item-detail page when the app is snapped. Finally, you will implement Semantic Zoom in the start page, enabling users to zoom out and see all the recipe groups on a single screen.

# Objectives

* 1. This lab will show you how to:
  + Customize the UI when the device is rotated.
  + Customize the UI when your app is snapped.
  + Implement Semantic Zoom.

# System requirements

* 1. You must have the following items to complete this lab:
  + Microsoft Windows 8
  + Microsoft Visual Studio 2012

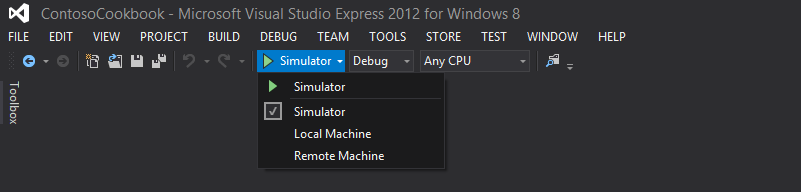
# Setup

* 1. You must perform the following steps to prepare your computer for this lab:
  2. Install Microsoft Windows 8.
  3. Install Microsoft Visual Studio 2012.

# Exercises

* 1. This hands-on lab includes the following exercises:
  2. Device orientation
  3. Snapping
  4. Semantic Zoom
  5. Estimated time to complete this lab: **40 to 60 minutes**.

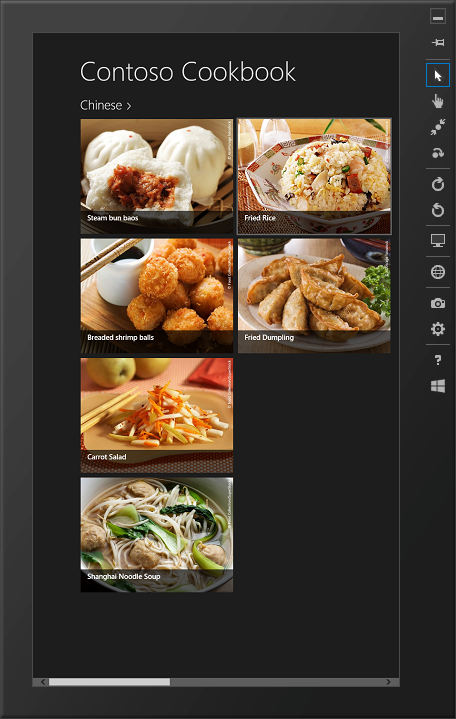
Exercise 1: Device orientation

1. Windows 8 is designed to run on a variety of devices, including tablets and other devices that, through the aid of on-board sensors, know whether they are in landscape mode or portrait mode. A page that looks great in landscape mode may need tweaking to look equally great in portrait mode, and vice versa. In this exercise, you’ll modify Contoso Cookbook to adapt to device orientation.
   1. **Note:** You do not have to have a tablet to perform Exercise 1. You can test orientation code in the Windows Simulator, which you can access directly from Visual Studio. To target the simulator, select **Simulator** from the drop-down list below. The next time you start the app, it will start in the Windows Simulator.
   2. 

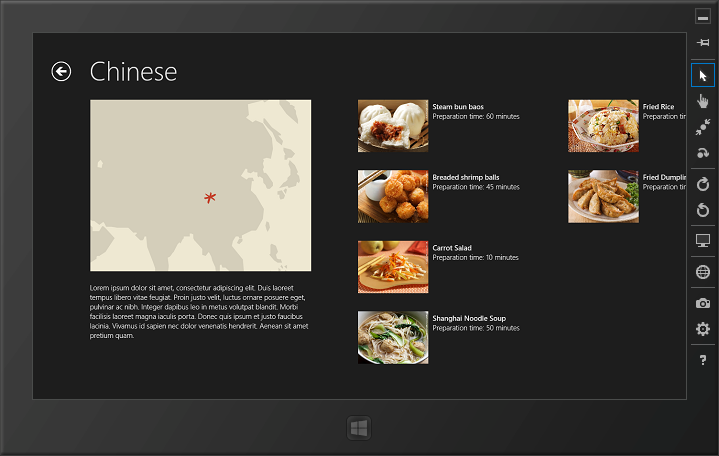
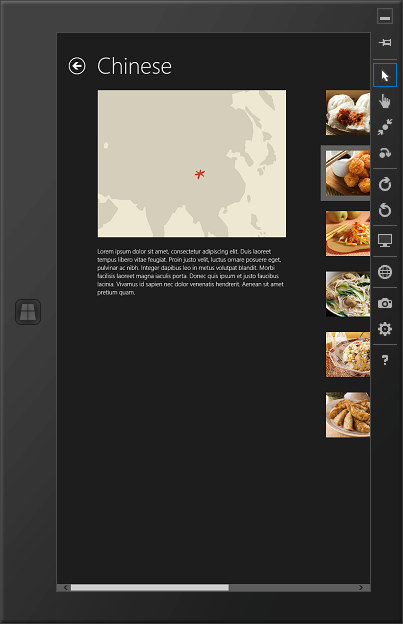
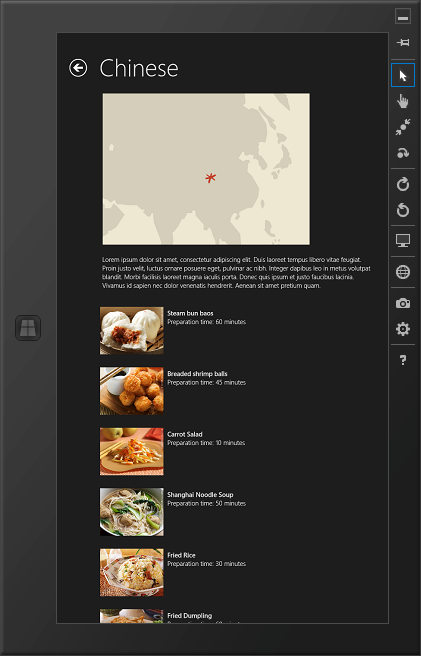
Task 1 – Test the start page

* 1. Let’s begin by examining the app’s start page in landscape mode and portrait mode to determine whether any changes are needed.
  2. Open the ContosoCookbook project you created in Lab 1 in Visual Studio. If you didn’t complete Lab 1 or would like to start with a reference copy, you can find a completed version of the lab in the starting materials.
  3. With the device in landscape mode, press F5 to run the app in the debugger. You should see the recipes start page shown in Figure 1, if you are using the simulator.
     1. 
     2. Figure 1
     3. The start page in landscape mode
  4. Now rotate the device to portrait mode. Make sure that the screen rotates 90 degrees, and that the start page rotates also and assumes the layout shown in Figure 2.
     1. **Note:** If you’re testing on a tablet and the start page doesn’t rotate, it might be because automatic rotation is not enabled. Some devices have a hardware switch that locks the screen into its current orientation. If you have such a device, make sure that switch is set to automatic rotation mode. You can also press Windows logo key+O to toggle automatic rotation on and off.

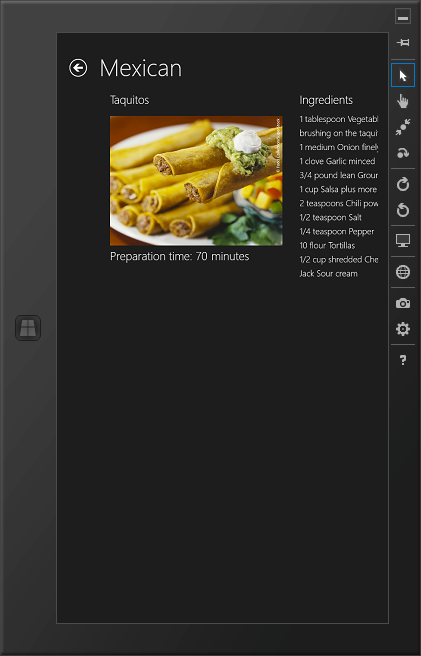
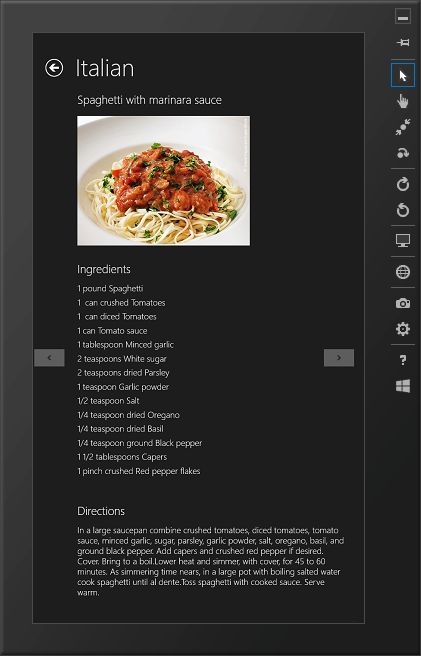
If the hardware you are testing on doesn’t support orientation changes, remember that you can use the Windows Simulator instead. To rotate the simulator, click either of the **Rotate** buttons in the simulator’s right margin.

* + 1. 
    2. Figure 2
    3. The start page in portrait mode
  1. Thanks to the **GridView** control that renders the recipe items, the start page looks fine in both landscape and portrait modes, so no additional work is required here.
  2. Return to Visual Studio and stop debugging.

Task 2 – Customize the group-detail page

* 1. The Contoso Cookbook app includes three pages: the start page, the group-detail page, and the item-detail page. The start page doesn’t require any modification for portrait mode, so let’s move on to the group-detail page—the one that appears when you tap a group title such as **Chinese** or **Italian**.
  2. Start the app again and tap **Chinese** to show the Chinese group-detail page. In landscape mode, the page assumes the layout shown in Figure 3.
     1. 
     2. Figure 3
     3. The group-detail page in landscape mode
  3. Now rotate the device to portrait mode (see Figure 4). Observe that the **GridView** control that renders most of the content on this page positions the group details (the recipes) to the right of the group header (the group title, image, and description), which leaves a lot of blank space in the lower half of the screen.
     1. 
     2. Figure 4
     3. The group-detail page in portrait mode
  4. Return to Visual Studio and stop debugging.
  5. Open GroupDetailPage.xaml and find the **ListView** control whose name is “itemListView.”
  6. Immediately after that **ListView** control, add another **ListView** named “portraitListView” that has the following properties.
     1. XAML
     2. <!-- Vertical scrolling list only used in portrait mode -->
     3. <ListView
     4. x:Name="portraitListView"
     5. AutomationProperties.AutomationId="ItemListView"
     6. AutomationProperties.Name="Items In Group"
     7. TabIndex="1"
     8. Grid.Row="1"
     9. Visibility="Collapsed"
     10. Padding="86,0,20,60"
     11. ItemsSource="{Binding Source={StaticResource itemsViewSource}}"
     12. ItemTemplate="{StaticResource Standard500x130ItemTemplate}"
     13. SelectionMode="None"
     14. IsItemClickEnabled="True"
     15. ItemClick="ItemView\_ItemClick">
     16. <ListView.Header>
     17. <StackPanel>
     18. <Image Source="{Binding Image}" Margin="20,-10,18,0" MaxWidth="480" Stretch="UniformToFill" HorizontalAlignment="Left"/>
     19. <TextBlock Margin="20,20,18,30" Text="{Binding Description}" Style="{StaticResource BodyTextStyle}"/>
     20. </StackPanel>
     21. </ListView.Header>
     22. </ListView>
  7. Scroll farther down in GroupDetailPage.xaml and find the **VisualState** element whose name is “FullScreenPortrait”.
  8. Inside that **VisualState** element, remove the following statements.
     1. XAML
     2. <ObjectAnimationUsingKeyFrames Storyboard.TargetName="itemGridView" Storyboard.TargetProperty="Padding">
     3. <DiscreteObjectKeyFrame KeyTime="0" Value="100,0,90,0"/>
     4. </ObjectAnimationUsingKeyFrames>
  9. Add the following statements after the statements you just deleted.
     1. XAML
     2. <ObjectAnimationUsingKeyFrames Storyboard.TargetName="itemGridView" Storyboard.TargetProperty="Visibility">
     3. <DiscreteObjectKeyFrame KeyTime="0" Value="Collapsed"/>
     4. </ObjectAnimationUsingKeyFrames>
     5. <ObjectAnimationUsingKeyFrames Storyboard.TargetName="portraitListView" Storyboard.TargetProperty="Visibility">
     6. <DiscreteObjectKeyFrame KeyTime="0" Value="Visible"/>
     7. </ObjectAnimationUsingKeyFrames>
     8. **Note:** The **ListView** control that you added defines how recipe groups are rendered in portrait mode. The animations you added hide the **GridView** control used in landscape mode and display the **ListView** when the view state changes to “FullScreenPortrait.” (View state in a Windows Store app has nothing to do with ASP.NET view state, so don’t let the term conjure up negative images in your mind!) One of the events that causes a view-state change is rotating the device from landscape mode to portrait or from portrait to landscape. The view state also changes when the app is “snapped.” Snapping is covered in the next exercise.
     9. The code that detects view-state changes is hidden away in the base class from which the app’s pages derive. That class—**LayoutAwarePage**—registers a handler for the main window’s **SizeChanged** events and uses the Windows Runtime’s **VisualStateManager** class to drive changes from one visual state to another.
  10. Run the app, tap **Chinese** to display the group-detail page, and rotate the device to portrait mode. Confirm that the page looks like the one in Figure 5.
      1. 
      2. Figure 5
      3. The finished portrait-mode layout
  11. Return to Visual Studio and stop debugging.

Task 3 – Customize the item-detail page

* 1. The next task is to tweak the item-detail page so that it is equally efficient in its use of space and pleasing to the eye in both landscape and portrait modes.
  2. Start the app again and tap a recipe to show the item-detail page. Figure 6 shows how the page looks in landscape mode.
     1. 
     2. Figure 6
     3. The item-detail page in landscape mode
  3. Now rotate the device to portrait mode (Figure 7). Once again, the layout that works well in landscape mode needs a bit of tweaking for portrait mode.
     1. 
     2. Figure 7
     3. The item-detail page in portrait mode
  4. Return to Visual Studio and stop debugging.
  5. Open ItemDetailPage.xaml and find the **FlipView** control named “flipView”.
  6. After that **FlipView** control, add another one control named “portraitFlipView”.
     1. XAML
     2. <!-- FlipView used in portrait mode -->
     3. <FlipView
     4. x:Name="portraitFlipView"
     5. AutomationProperties.AutomationId="ItemsFlipView"
     6. AutomationProperties.Name="Item Details"
     7. Grid.Row="1"
     8. Margin="0,-3,20,0"
     9. ItemsSource="{Binding Source={StaticResource itemsViewSource}}"
     10. Visibility="Collapsed">
     11. <FlipView.ItemTemplate>
     12. <DataTemplate>
     13. <UserControl Loaded="StartLayoutUpdates" Unloaded="StopLayoutUpdates">
     14. <ScrollViewer x:Name="scrollViewer" Style="{StaticResource VerticalScrollViewerStyle}" Grid.Row="1">
     15. <!-- Vertical StackPanel for item-detail layout -->
     16. <StackPanel Orientation="Vertical" Margin="100,0,20,0">
     17. <StackPanel Orientation="Vertical">
     18. <TextBlock FontSize="26.667" FontWeight="Light" Text="{Binding Title}" TextWrapping="Wrap"/>
     19. <Image x:Name="image" Width="400" Margin="0,20,0,40" Stretch="Uniform" Source="{Binding Image}" HorizontalAlignment="Left"/>
     20. </StackPanel>
     21. <StackPanel Orientation="Vertical">
     22. <TextBlock FontSize="26.667" FontWeight="Light" Text="Ingredients" Margin="0,0,0,16"/>
     23. <TextBlock FontSize="20" FontWeight="Light" LineHeight="32.5" Text="{Binding Ingredients, Converter={StaticResource ListConverter}}" TextWrapping="Wrap" />
     24. </StackPanel>
     25. <StackPanel Orientation="Vertical">
     26. <TextBlock FontSize="26.667" FontWeight="Light" Text="Directions" Margin="0,24,0,16"/>
     27. <ScrollViewer Style="{StaticResource VerticalScrollViewerStyle}">
     28. <Grid>
     29. <TextBlock FontSize="20" FontWeight="Light" Text="{Binding Directions}" TextWrapping="Wrap" />
     30. </Grid>
     31. </ScrollViewer>
     32. </StackPanel>
     33. </StackPanel>
     34. </ScrollViewer>
     35. </UserControl>
     36. </DataTemplate>
     37. </FlipView.ItemTemplate>
     38. </FlipView>
         1. **Note:** In landscape mode, the three content areas in the item-detail page—the recipe name, image, and description; the ingredients; and the directions—are laid out in columns of a multicolumn **Grid** element declared in the data template of the **FlipView** control named “flipView”. The **FlipView** control you just added is for portrait mode. It uses a vertical StackPanel element to stack the content areas on top of each other.
  7. Find the **VisualState** element whose name is “FullScreenPortrait”, near the bottom of ItemDetailPage.xaml. (The one you want is NOT the **VisualState** element with the same name embedded inside the first **FlipView** control.)
  8. Add the following statements to the **Storyboard** element inside the **VisualState** element. When the screen rotates to portrait mode, these statement switch from the control named “flipView” to the control named “portraitFlipView” that you just added.
     1. XAML
     2. <ObjectAnimationUsingKeyFrames Storyboard.TargetName="flipView" Storyboard.TargetProperty="Visibility">
     3. <DiscreteObjectKeyFrame KeyTime="0" Value="Collapsed"/>
     4. </ObjectAnimationUsingKeyFrames>
     5. <ObjectAnimationUsingKeyFrames Storyboard.TargetName="portraitFlipView" Storyboard.TargetProperty="Visibility">
     6. <DiscreteObjectKeyFrame KeyTime="0" Value="Visible"/>
     7. </ObjectAnimationUsingKeyFrames>
  9. Start the app and tap a recipe again to show the item-detail page. Rotate the display to portrait mode and verify that it assumes the single-column layout depicted in Figure 8.
     1. 
     2. Figure 8
     3. The modified portrait-mode layout
  10. Rotate back to landscape mode and make sure the page reverts to a three-column layout.
  11. Return to Visual Studio and stop debugging.

Exercise 2: Snapping

1. Snapping enables Windows 8 users to run two Windows Store apps side by side by splitting the screen horizontally. On a touch screen, you can demonstrate snapping by dragging your finger slowly across the screen, starting from the left edge and pausing momentarily until a divider—a vertical bar splitting the screen—appears. (If you don’t have a touch screen, press Windows logo key+Period to snap an app.)
2. Figure 9 shows snapping in action. Contoso Cookbook occupies the left 320 pixels of the screen, while Internet Explorer occupies the remaining portion. In this context, Contoso Cookbook is the *snapped app* and Internet Explorer is the *main app*. To reverse the apps’ roles, the user can drag the snap bar across the screen.
   * 1. 
   1. Figure 9
   2. Snapping in action

Contoso Cookbook already has some snapping behavior built in, thanks to the XAML code that Visual Studio included in the pages it generated for the project. In this exercise, you will apply a simple change to one of those pages to improve the user experience.

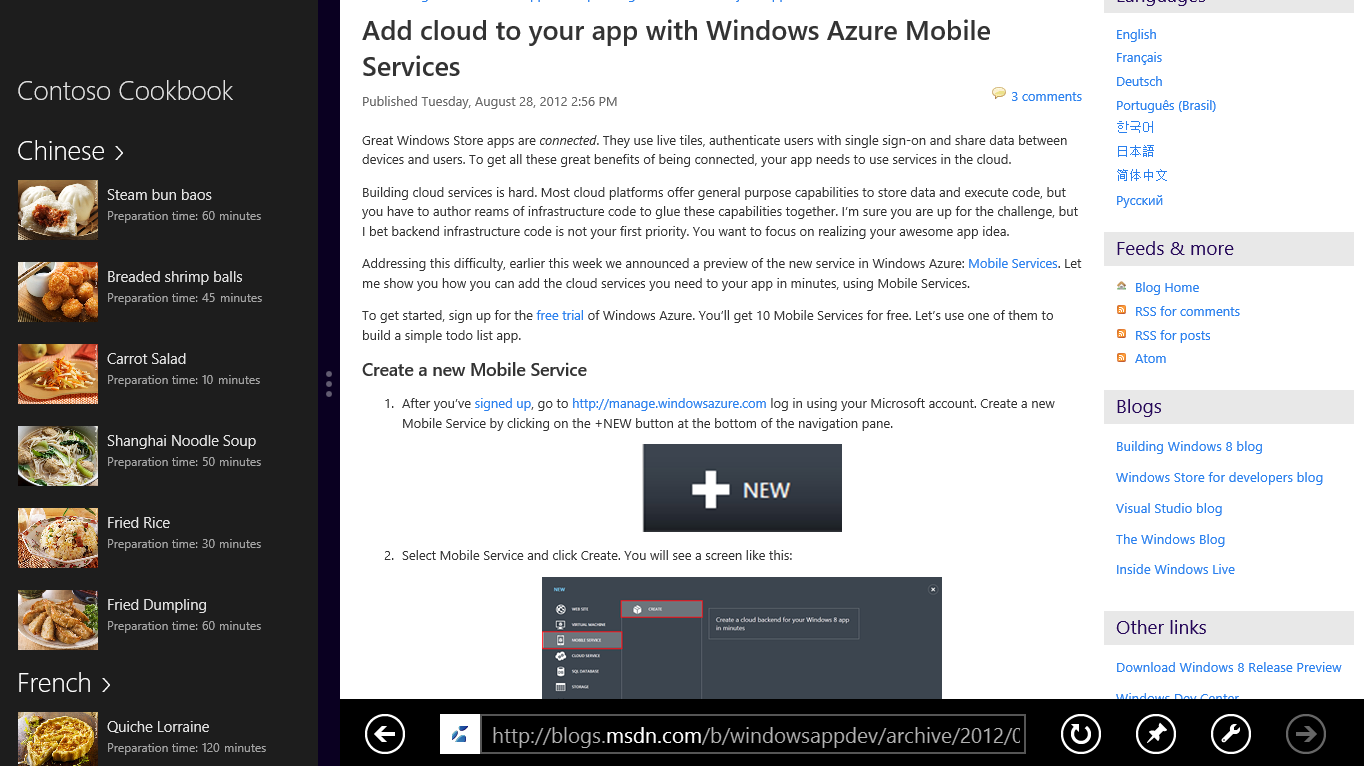
* 1. **Note:** To see snapping in action, you must be running Windows 8 on a device with a screen resolution of at least 1366 × 768 pixels. The Windows team chose 1366 as the minimum because it affords the snapped app a section of the screen that is 320 pixels wide (the same width as many smartphones) and the main app a section that is 1024 pixels wide. The divider consumes the extra pixels. If you’re running on a lower- resolution screen, use the Windows Simulator for this exercise and select a simulated screen resolution of at least 1366 × 768.

Task 1 – Run Contoso Cookbook in snapped mode

* 1. In this task, you will view each of Contoso Cookbook’s three pages when the app is snapped.
  2. Press F5 to start the app from Visual Studio. Then go back to the Windows Start screen and start another app such as Internet Explorer. With that app now occupying the screen, place a finger on the left edge of the screen and slowly drag it to the right. When Contoso Cookbook appears under your finger, pause until the divider appears. Then lift your finger to snap Contoso Cookbook into place.
  3. When Contoso Cookbook occupies only a portion of the screen, how does its start page differ from when it’s displayed full-screen?
  4. Tap one of the group names on the start page to see the group-detail page in snapped mode. How does it differ from its full-screen equivalent?
  5. Tap one of the recipes to see the item-detail page in snapped mode. How does its content and layout compare to when it’s running full screen?
  6. Return to Visual Studio and stop debugging.

Task 2 – Modify the snapped grouped-items page

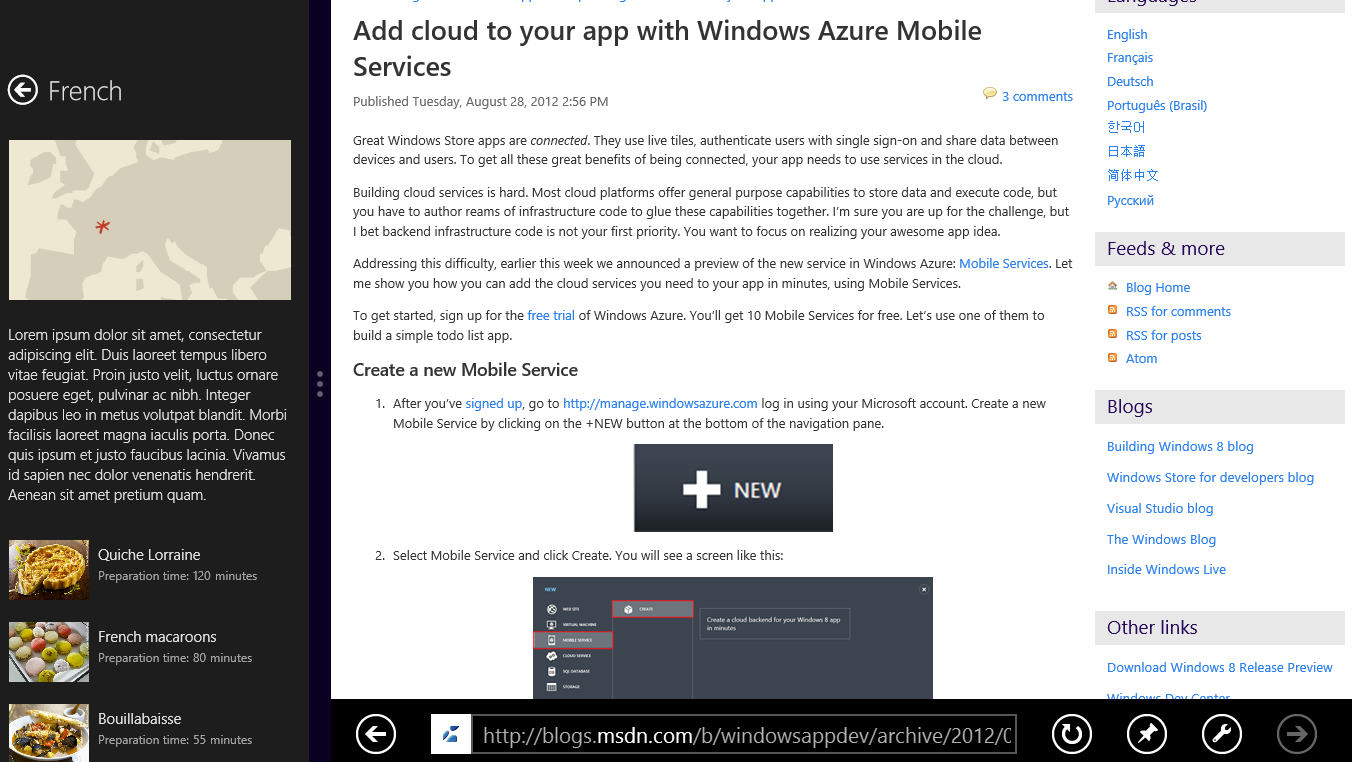
* 1. The default snapped layouts provided by Visual Studio were a great start, but you can customize them further. In Contoso Cookbook, we need to modify the snapped item-detail page, because the default layout leaves a lot of empty real estate in the portion of the screen that the page occupies—space that could be used to show additional info about the recipe. We also need to make minor adjustments to the grouped-items page and group-detail page. Let’s start with the grouped-items page.
  2. Open StandardStyles.xaml in the project’s Common folder and find the **DataTemplate** element named “Standard80ItemTemplate.”
  3. Replace that **DataTemplate** with this one:
     1. XAML
     2. <DataTemplate x:Key="Standard80ItemTemplate">
     3. <Grid Margin="6">
     4. <Grid.ColumnDefinitions>
     5. <ColumnDefinition Width="Auto"/>
     6. <ColumnDefinition Width="\*"/>
     7. </Grid.ColumnDefinitions>
     8. <Border Background="{StaticResource ListViewItemPlaceholderBackgroundThemeBrush}" Width="80">
     9. <Image Source="{Binding Image}" Stretch="UniformToFill"/>
     10. </Border>
     11. <StackPanel Grid.Column="1" Margin="10,0,0,0">
     12. <TextBlock Text="{Binding ShortTitle}" Style="{StaticResource ItemTextStyle}" MaxHeight="40"/>
     13. <StackPanel Orientation="Horizontal">
     14. <TextBlock Text="Preparation time:" Style="{StaticResource CaptionTextStyle}" />
     15. <TextBlock Text="{Binding PrepTime}" Style="{StaticResource CaptionTextStyle}" Margin="4,0,4,0" />
     16. <TextBlock Text="minutes" Style="{StaticResource CaptionTextStyle}" />
     17. </StackPanel>
     18. </StackPanel>
     19. </Grid>
     20. </DataTemplate>
  4. Press F5 to start the app.
  5. Snap the app and confirm that it looks like the one pictured in Figure 10.



* + 1. Figure 10
    2. The modified grouped-items page in snapped view
  1. Return to Visual Studio and stop debugging.

Task 3 – Modify the snapped group-detail page

* 1. The group-detail page needs some minor tweaking to improve its layout, too. Let’s make those changes and test the results.
  2. Open GroupDetailPage.xaml and find the **ListView** control named “itemListView.”
  3. Remove the first **TextBlock** element from the control’s **ListView.Header** element. Also change the left margin of the Image and the remaining **TextBlock** from 20 to 10. Here is the finished element.
     1. XAML
     2. <ListView.Header>
     3. <StackPanel>
     4. <Image Source="{Binding Image}" Margin="10,0,18,0" MaxHeight="160" Stretch="UniformToFill"/>
     5. <TextBlock Margin="10,20,18,30" Text="{Binding Description}" Style="{StaticResource BodyTextStyle}"/>
     6. </StackPanel>
     7. </ListView.Header>
  4. Press F5 to start the app.
  5. Snap the app. Click the **French** group header and confirm that the group-detail page looks like the one pictured in Figure 11.



* + 1. Figure 11
    2. The modified group-detail page in snapped view
  1. Return to Visual Studio and stop debugging.

Task 4 – Modify the snapped item-detail page

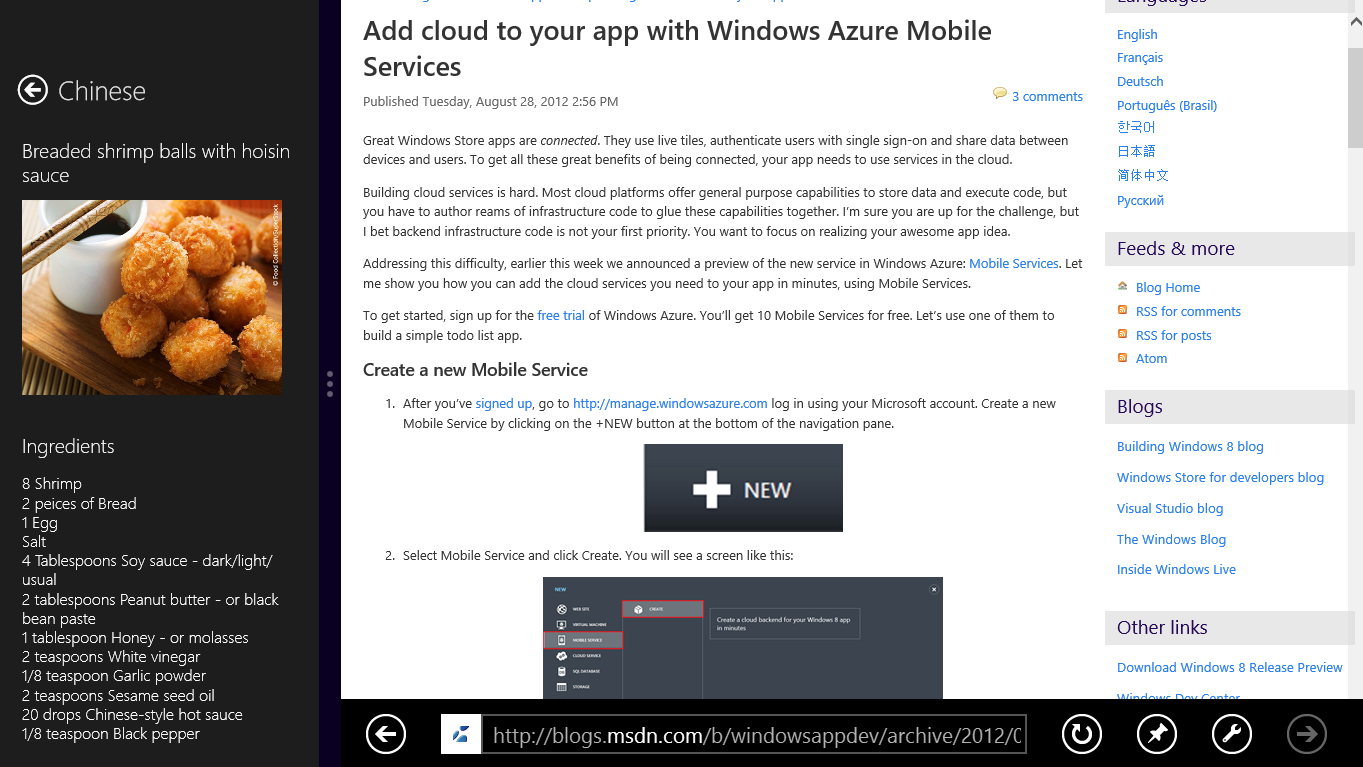
* 1. The final task is to modify the snapped item-detail page to improve the layout and include recipe ingredients.
  2. Open ItemDetailPage.xaml and add the following **FlipView** control to those that are already there.
     1. XAML
     2. <FlipView
     3. x:Name="snappedFlipView"
     4. AutomationProperties.AutomationId="ItemsFlipView"
     5. AutomationProperties.Name="Item Details"
     6. Grid.Row="1"
     7. Margin="0,-3,0,0"
     8. ItemsSource="{Binding Source={StaticResource itemsViewSource}}"
     9. Visibility="Collapsed">
     10. <FlipView.ItemTemplate>
     11. <DataTemplate>
     12. <UserControl Loaded="StartLayoutUpdates" Unloaded="StopLayoutUpdates">
     13. <ScrollViewer x:Name="scrollViewer" Style="{StaticResource VerticalScrollViewerStyle}" Grid.Row="1">
     14. <!-- Vertical StackPanel for item-detail layout -->
     15. <StackPanel Orientation="Vertical" Margin="20,0,20,0">
     16. <StackPanel Orientation="Vertical">
     17. <TextBlock FontSize="20" FontWeight="Light" Text="{Binding Title}" TextWrapping="Wrap"/>
     18. <Image x:Name="image" Width="260" Margin="0,12,0,40" Stretch="Uniform" Source="{Binding Image}" HorizontalAlignment="Left"/>
     19. </StackPanel>
     20. <StackPanel Orientation="Vertical">
     21. <TextBlock FontSize="20" FontWeight="Light" Text="Ingredients" Margin="0,0,0,16"/>
     22. <TextBlock FontSize="16" FontWeight="Light" TextWrapping="Wrap" Text="{Binding Ingredients, Converter={StaticResource ListConverter}}" />
     23. </StackPanel>
     24. </StackPanel>
     25. </ScrollViewer>
     26. </UserControl>
     27. </DataTemplate>
     28. </FlipView.ItemTemplate>
     29. </FlipView>
         1. **Note:** This is the second **FlipView** you’ve added to the item-detail page. The other one handles portrait-mode layout. This one is used only when the page is in snapped view.
  3. Find the **VisualState** element named “Snapped” near the bottom of ItemDetailPage.xaml.
  4. Add the following elements to the elements already present in the **VisualState** element’s **Storyboard** element to hide the default **FlipView** and show the snapped **FlipView** when the view state changes to “Snapped”.
     1. XAML
     2. <ObjectAnimationUsingKeyFrames Storyboard.TargetName="flipView" Storyboard.TargetProperty="Visibility">
     3. <DiscreteObjectKeyFrame KeyTime="0" Value="Collapsed"/>
     4. </ObjectAnimationUsingKeyFrames>
     5. <ObjectAnimationUsingKeyFrames Storyboard.TargetName="snappedFlipView" Storyboard.TargetProperty="Visibility">
     6. <DiscreteObjectKeyFrame KeyTime="0" Value="Visible"/>
     7. </ObjectAnimationUsingKeyFrames>
  5. Start the app again. Snap it so that it occupies the left side of the screen next to another app. The go to the item-detail page and confirm that the snapped page now shows recipe ingredients, as seen in Figure 12.
     + 1. 

Figure 12

* + 1. The modified item-detail page in snapped view
  1. Return to Visual Studio and stop debugging.

Exercise 3: Semantic Zoom

* 1. Many apps that run on touch screens enable users to zoom in and out using a two-fingered pinching motion. A photo-editing app, for example, might let you zoom in on a photo when two fingers touching the screen move apart, and zoom back out when two fingers touching the screen move together.
  2. Most zooms are optical zooms, meaning that they simply scale content that is displayed on the screen. It’s easy enough to include optical zoom in an app, but Windows 8 also supports *Semantic Zoom*. Semantic Zoom does not simply scale content up or down; it groups the content to provide a different, semantic view. For example, Semantic Zoom in a mapping app might reveal additional details such as street names and building names as the user zooms in, and remove those details when the user zooms back out.
  3. To aid you in implementing Semantic Zoom, the **Windows.UI.Xaml.Controls** namespace of the Windows Runtime includes a **SemanticZoom** control. The basic idea is that you provide the control with two views—a zoomed-in view and a zoomed-out view—and the control will switch between the two in response to user input. You don’t have to do the switching yourself, and you don’t have to write gesture-recognition code to take action when two fingers make contact with the screen and move together or apart. If you don’t have a touch screen, you can zoom in and out by holding down the Ctrl key and rolling the mouse wheel; if you don’t have a mouse, Ctrl+Plus Sign and Ctrl+Minus Sign work for zooming in and out. Sound appealing? Then let’s put Semantic Zoom to work.

Task 1 – Add a SemanticZoom control to the start page

* 1. Adding Semantic Zoom to Contoso Cookbook is as simple as declaring a **SemanticZoom** control and populating it with views representing the zoomed-in and zoomed-out views of the start page.
  2. Open GroupedItemsPage.xaml.
  3. Find the **GridView** element named “itemGridView”. Delete it and everything inside it.
  4. Replace the **GridView** you just deleted with the following statements.
     1. XAML
     2. <SemanticZoom Grid.Row="1">
     3. <SemanticZoom.ZoomedInView>
     4. <!-- Horizontal scrolling grid used in most view states -->
     5. <GridView
     6. x:Name="itemGridView"
     7. AutomationProperties.AutomationId="ItemGridView"
     8. AutomationProperties.Name="Grouped Items"
     9. Margin="0,-3,0,0"
     10. Padding="116,0,40,46"
     11. ItemsSource="{Binding Source={StaticResource groupedItemsViewSource}}"
     12. ItemTemplate="{StaticResource Standard250x250ItemTemplate}"
     13. SelectionMode="None"
     14. IsItemClickEnabled="True"
     15. ItemClick="ItemView\_ItemClick">
     16. <GridView.ItemsPanel>
     17. <ItemsPanelTemplate>
     18. <VirtualizingStackPanel Orientation="Horizontal"/>
     19. </ItemsPanelTemplate>
     20. </GridView.ItemsPanel>
     21. <GridView.GroupStyle>
     22. <GroupStyle>
     23. <GroupStyle.HeaderTemplate>
     24. <DataTemplate>
     25. <Grid Margin="1,0,0,6">
     26. <Button
     27. AutomationProperties.Name="Group Title"
     28. Content="{Binding Title}"
     29. Click="Header\_Click"
     30. Style="{StaticResource TextButtonStyle}"/>
     31. </Grid>
     32. </DataTemplate>
     33. </GroupStyle.HeaderTemplate>
     34. <GroupStyle.Panel>
     35. <ItemsPanelTemplate>
     36. <VariableSizedWrapGrid Orientation="Vertical" Margin="0,0,80,0"/>
     37. </ItemsPanelTemplate>
     38. </GroupStyle.Panel>
     39. </GroupStyle>
     40. </GridView.GroupStyle>
     41. </GridView>
     42. </SemanticZoom.ZoomedInView>
     43. <SemanticZoom.ZoomedOutView>
     44. <GridView x:Name="groupGridView" Margin="116,0,40,46">
     45. <GridView.ItemTemplate>
     46. <DataTemplate>
     47. <Grid Margin="0,0,24,0">
     48. <Image Source="{Binding Group.GroupImage}" Width="250" Height="500" Stretch="UniformToFill" />
     49. <TextBlock Text="{Binding Group.Title}" Foreground="{StaticResource ListViewItemOverlayForegroundThemeBrush}" Style="{StaticResource TitleTextStyle}" FontSize="28" Margin="12"/>
     50. <TextBlock Text="{Binding Group.RecipesCount}" Foreground="{StaticResource ApplicationSecondaryForegroundThemeBrush}" Style="{StaticResource TitleTextStyle}" FontSize="96" Margin="12,64,12,12" HorizontalAlignment="Right"/>
     51. </Grid>
     52. </DataTemplate>
     53. </GridView.ItemTemplate>
     54. </GridView>
     55. </SemanticZoom.ZoomedOutView>
     56. </SemanticZoom>
     57. **Note:** You just replaced the **GridView** that renders items to the start page with a **SemanticZoom** control containing two **GridView** elements: one representing the zoomed-in view of the start page and the other representing the zoomed-out view. The **SemanticZoom** control does the switching between the two.
  5. Near the bottom of GroupedItemsPage.xaml, find the **VisualState** element named “Snapped” and add the following statements to that element’s **Storyboard** to hide the zoomed-out view when the app is snapped.
     1. XAML
     2. <ObjectAnimationUsingKeyFrames Storyboard.TargetName="groupGridView" Storyboard.TargetProperty="Visibility">
     3. <DiscreteObjectKeyFrame KeyTime="0" Value="Collapsed"/>
     4. </ObjectAnimationUsingKeyFrames>
  6. Open GroupedItemsPage.xaml.cs and add the following statement to the end of the **LoadState** method.
     1. C#
     2. this.groupGridView.ItemsSource = this.groupedItemsViewSource.View.CollectionGroups;

Task 2 – Test the results

* 1. Now it’s time to see Semantic Zoom in action.
  2. Run the app and confirm that you see the same start page you’ve seen before.
  3. Put two fingers on the screen and move them together (or use the mouse wheel and the Ctrl key) to zoom out. Verify that the page changes to the one shown in Figure 13.
     1. 
     2. Figure 13
     3. The zoomed-out start page
  4. Put two fingers on the screen again, but this time move them away from each other (or use the mouse wheel and the Ctrl key again) to zoom in. What happens to the start page?
  5. Zoom out again and tap one of the recipe groups. Verify that you switch back to the zoomed-in view, and that it’s scrolled to the group you selected.
  6. Return to Visual Studio and stop debugging.

Summary

* 1. In this lab, you made some key UI-related enhancements to Contoso Cookbook. You adapted the layout for rotated displays, customized the layout when the app is snapped, and added Semantic Zoom to simplify the task of navigating among recipe groups on the start page. And you did all of this while writing remarkably little code.
  2. Most of the work in Labs 1 and 2 involved creating the user experience—getting the pages looking just right in all orientations and all modes. Now it’s time to do some work on the back end by adding support for sharing and search. Contoso Cookbook is about to become more tightly integrated with Windows and other apps.