Session 4 – Lists and List Controls

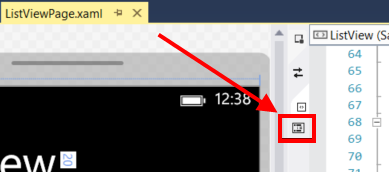
# Demo 1 – Create List Item Template

The goal of this demo is to familiarize the audience with working with the templates required for lists and list controls. We will create a new ItemTemplate in a ListView, change the colour in the ItemContainerStyle and demonstrate how to create a FooterTemplate.

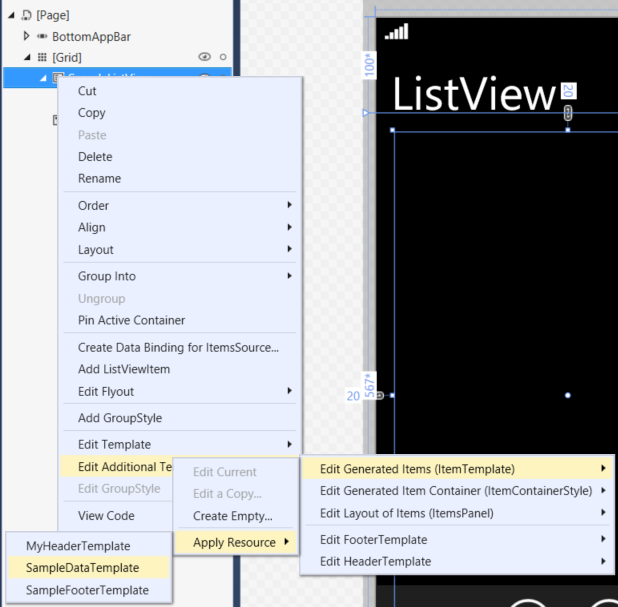
The solution consists of a Universal Windows and Windows Phone project. For the purpose of the demo, we’ll focus on the phone element, but the Windows project can also be referenced to show the ease of universal platform development.

Change the ItemTemplate

1. Open up the project and show that it is a Universal Project. Note that, as a universal project, it makes sense to keep our model objects in the shared element of the project.
2. In the Shared portion of the project, open Models/SampleItem.cs
3. This class is a simple class with a Title, a Subtitle, an Image and a TargetGroup property that we will use to help us group the items for display.
4. Run the application and tap on the “See ListView” button. Note that the default rendering of objects without any DataTemplate applied is to simply call the “ToString()” method on the object and display that. “So we need to define how this item should be displayed in this list.”
5. Open ListViewPage.xaml to the dual view (designer + XAML). Click on the “Document Outline” icon in the middle



1. Right-click on the SampleListView in the Document Outline and go down to “Edit Additional Templates”. Show how a developer can explore the templates available in the control. Select “Edit Generated Items (ItemTemplate)” 🡪 “Apply Resource” 🡪 SampleDataTemplate



1. Let’s look at the template. Right-click on the SampleListView, go down to “Edit Additional Templates” 🡪 ItemTemplate 🡪 Edit Current
2. The XAML pane will scroll to the template in the resources. Show the Bindings on the Image, and the two TextBlocks. If you’re comfortable with the XAML, feel free to break the bindings and show the audience how they can design the item with static data and paste the bindings back in.
3. Run the application, show how the items display properly
4. While still in the app, press on the list icon at the bottom. This changes the ListView from

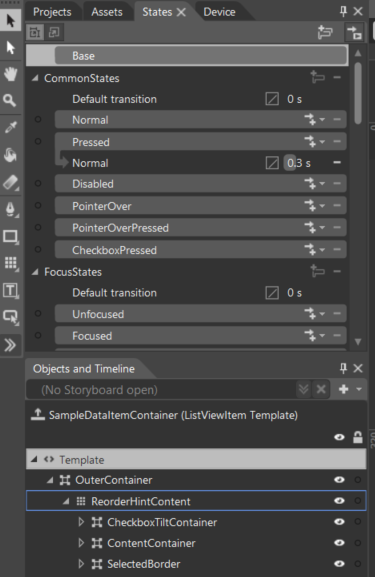
SelectionMode = ListViewSelectionMode.Single

To

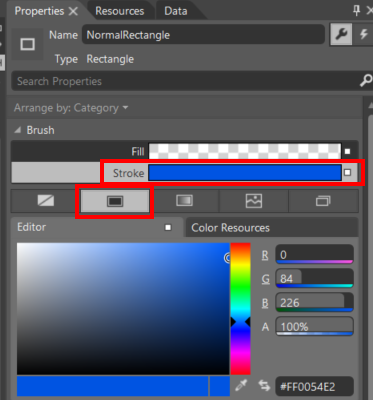
SelectionMode = ListViewSelectionMode.Multiple  
Note the visual change and mention that this change is driven by the ItemContainerStyle.

Changing the ItemContainerStyle

1. For the sake of the scenario, we’re going to change the ItemContainerStyle so that the checkbox is another color.
2. Right-click on the Windows Phone project and click “Open in Blend…” (second to the bottom).
3. The project will open in Blend. Go to the “Objects and Timeline” (will look like the Document Outline in Visual Studio). Right click on the SampleListView 🡪 Edit Additional Templates… 🡪 Edit Generated Item Container (ItemContainerStyle) 🡪 Edit a Copy…
4. Name it “SampleDataItemContainer” and press OK.
5. Click on “States” at the top left corner of Blend



1. Scroll down to the “MultiSelectStates” section and select the “ListMultiSelect” state. You will see the checkbox appear in the design pane. Select the “Base” state (which sticks to the top of the States tab).
2. Drill down into the CheckBoxTiltContainer until you reach the “NormalRectangle”
3. Select the rectangle and in the Properties pane on the right side of Blend, select “Stroke”, then “Solid Color Brush”. Change the color to blue.



1. Save and return to Visual Studio. Accept the changes and run the app again. The multi-select mode should look like this



# Demo 2 – Create Semantic Zoom JumpList

In this demo, we will implement a JumpList for a grouped GridView using a SemanticZoom control using one GridView for “ZoomedIn” view and another GridView for the “ZoomedOut” view.

1. Open the solution and open “GroupedGridView.xaml”. Starting at line 62, you’ll see a GridView that is bound to a grouped source.
2. Point out the GroupStyle and the GroupStyle.HeaderTemplate is where the grouping visuals are defined. The

Text="{Binding Key}"

binding inside the TextBlock is required to make the grouping visual work property.

1. NOTE: The grouped source was covered in the presentation, but if you want to walk through the details of creating a grouped data source, continue with step 4. If you want to skip it to save time, jump to step 8.
2. Scroll to the top of the page to see two CollectionViewSource resources. For the sake of convenient comparison, the top one is bound to a grouped data source and the bottom one is bounded to a simple list of items.
3. Open DataSource.cs from the Shared project section under the “Models” folder. Look at line 50 to see that the AlphaGrouped property is derived from the Ungrouped simple list.
4. Open the “alphaGroupSorting” method and show that we need to run our unsorted list through the AlphaKeyGroup<object>.CreateGroup method for sorting.
5. Note that this AlphaKeyGroup is the same thing we used to enable jumplists in Windows Phone 8 using the LongListSelector.
6. Run the application to show that grouping works, and that the headers act as responsive buttons, but that there is no jump list.
7. Past the following code into the application, just above the GridView (line 68)

<SemanticZoom

Grid.Row="1"

Margin="20,20,20,0" >

<SemanticZoom.ZoomedInView>

<GridView />

</SemanticZoom.ZoomedInView>

<SemanticZoom.ZoomedOutView>

<GridView />

</SemanticZoom.ZoomedOutView>

</SemanticZoom>

1. Point out that the SemanticZoom holds 2 List controls. The zoomed in and zoomed out views must be either a GridView or a ListView bound to the same grouped data.

Copy the existing GridView and paste it into the SemanticZoom.ZoomedInView, replacing the blank GridView already in there. Delete the Grid.Row="1" property from the GridView.

1. Copy the following to replace the empty GridView in the SemanticZoom.ZoomedOutView:

<GridView

ItemsSource="{Binding Source={StaticResource ItemsGroupedByGroup}, Path=CollectionGroups}"

Background="#AA000000"

>

<GridView.ItemTemplate>

<DataTemplate>

<Border Padding="5">

<Border Background="{Binding Converter={StaticResource BackgroundConverter}}"

Width="82" Height="82"

HorizontalAlignment="Left">

<TextBlock Text="{Binding Group.Key}"

Foreground="{Binding Converter={StaticResource ForegroundConverter}}"

FontSize="48"

Padding="6"

HorizontalAlignment="Left"

VerticalAlignment="Center"/>

</Border>

</Border>

</DataTemplate>

</GridView.ItemTemplate>

</GridView>

1. Note that the binding to this GridView is the same as the other GridView, but with Path=CollectionGroups added to the end.
2. Note that the binding on the TextBlock in the ItemTemplate is

Text="{Binding Group.Key}"

which corresponds to the

Text="{Binding Key}”

from the first GridView

1. Run the application to see that the JumpList is in place and is working.

# Demo 3 – Phased List Rendering

In the process