Lesson 5 Guide:
More UI Design with XAML

Table of Contents

Examining XAML Controls (Part 2)

The StackPanel

The StackPanel is a container in which other objects may be placed. While it resembles a list display, it is really more of an organizational control. The StackPanel and does not maintain an items list as the various ItemsControl-derived classes do. It is derived from the Panel class, as are Grids and Canvases. It may display its contents vertically (the default) or horizontally.

Like any other control, it can be easily placed inside a ScrollView control to provide scrollable access to all its contents. Consider the following app, called My Photo Viewer:

Figure 1 *– Nine thumbnail images are available to the user in the StackPanel on the left by scrolling through its contents. Tapping any thumbnail displays the larger image on the right. Images used are in the public domain.*

In this example, there are nine PNG images embedded in the Solution Explorer in an Images folder, each with the three 100%, 140%, and 180% resolutions. There is also a smaller version of each at a 320x240 resolution, though in hindsight, these could have been 160x120. (The column in which the StackPanel resides began at 320 pixels wide. It was reduced to a 160-pixel width to present more thumbnails.) Even at the 160-pixel width, the StackPanel can only display four images. It also displays 20-pixel-high rectangle elements between the images to make them more distinct. One advantage of the StackPanel is its ability to display any objects (graphics, text controls, images, buttons, checkboxes, and more) of multiple types within its boundaries.

Consider the XAML code below. Note the use of both images and rectangles within the StackPanel. These are displayed from top to bottom. Placing the StackPanel control within a ScrollViewer wrapper enables the user to scroll through all the items in the StackPanel.

XAML Code

```
<Page
    x:Class="_05_StackPanel_Demo.MainPage"
    xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
    xmlns:local="using:_05_StackPanel_Demo"
    xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
    xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
    mc:Ignorable="d">

    <Grid Background="{StaticResource ApplicationPageBackgroundThemeBrush}">
        <Grid.ColumnDefinitions>
            <ColumnDefinition Width="140" />
            <ColumnDefinition Width="160" />
            <ColumnDefinition />
        </Grid.ColumnDefinitions>
        <Grid.RowDefinitions>
            <RowDefinition Height="140"/>
            <RowDefinition Height="*"/>
        </Grid.RowDefinitions>
        <TextBlock Text="My Photo Viewer" Grid.Column="1" Grid.ColumnSpan="2"
                   Style="{StaticResource PageHeaderTextStyle}"/>
        <ScrollViewer Grid.Column="1" Grid.Row="1" Margin="0,0,0,70" >
            <StackPanel>
                <Image x:Name="thumb0" Source="Images/_0000_320x240.png"
                       Tapped="ThumbnailTapped" />
                <Rectangle Height="20"/>
                <Image x:Name="thumb1" Source="Images/_0001_320x240.png"
                       Tapped="ThumbnailTapped"  />
                <Rectangle Height="20"/>
                <Image x:Name="thumb2" Source="Images/_0002_320x240.png"
                       Tapped="ThumbnailTapped" />
                <Rectangle Height="20"/>
                <Image x:Name="thumb3" Source="Images/_0003_320x240.png"
                       Tapped="ThumbnailTapped" />
                <Rectangle Height="20"/>
                <Image x:Name="thumb4" Source="Images/_0004_320x240.png"
                       Tapped="ThumbnailTapped" />
                <Rectangle Height="20"/>
                <Image x:Name="thumb5" Source="Images/_0005_320x240.png"
                       Tapped="ThumbnailTapped" />
                <Rectangle Height="20"/>
                <Image x:Name="thumb6" Source="Images/_0006_320x240.png"
                       Tapped="ThumbnailTapped" />
                <Rectangle Height="20"/>
                <Image x:Name="thumb7" Source="Images/_0007_320x240.png"
                       Tapped="ThumbnailTapped" />
                <Rectangle Height="20"/>
                <Image x:Name="thumb8" Source="Images/_0008_320x240.png"
                       Tapped="ThumbnailTapped" />
            </StackPanel>
        </ScrollViewer>
        <Image x:Name="imgDisplay" Grid.Column="2" HorizontalAlignment="Left"
```

```
                Height="550" Margin="80,0,0,0" Grid.Row="1" VerticalAlignment="Top"
                Width="734" Source="Images/_0000_.png"/>
    </Grid>
</Page>
```

Each of the Image controls in the StackPanel have a Tapped event handler. All share the same event handler, ThumbnailTapped, though this could have been developed where each Image responded via its own unique event handler. The code behind the page is:

C# Code

```
        private void ThumbnailTapped(object sender, TappedRoutedEventArgs e)
        {
            //include directive:  using Windows.UI.Xaml.Media.Imaging;
            Image myImage = (Image)sender;
            String imageFile = "ms-appx:///Images/_0000_.png";
            switch (myImage.Name)
            {
                case "thumb0":
                    imageFile = "ms-appx:///Images/_0000_.png";
                    break;
                case "thumb1":
                    imageFile = "ms-appx:///Images/_0001_.png";
                    break;
                case "thumb2":
                    imageFile = "ms-appx:///Images/_0002_.png";
                    break;
                case "thumb3":
                    imageFile = "ms-appx:///Images/_0003_.png";
                    break;
                case "thumb4":
                    imageFile = "ms-appx:///Images/_0004_.png";
                    break;
                case "thumb5":
                    imageFile = "ms-appx:///Images/_0005_.png";
                    break;
                case "thumb6":
                    imageFile = "ms-appx:///Images/_0006_.png";
                    break;
                case "thumb7":
                    imageFile = "ms-appx:///Images/_0007_.png";
                    break;
                case "thumb8":
                    imageFile = "ms-appx:///Images/_0008_.png";
                    break;
            }
            Uri myUri = new Uri(imageFile);
            BitmapImage bmi = new BitmapImage(myUri);
            imgDisplay.Source = bmi;
        }
```

VB Code

```
    Private Sub ThumbnailTapped(sender As Object, e As TappedRoutedEventArgs)
        'include directive:  imports Windows.UI.Xaml.Media.Imaging
        Dim myImage As Image = CType(sender, Image)
        Dim imageFile As String = "ms-appx:///Images/_0000_.png"
        Select (myImage.Name)
```

```
            Case "thumb0"
                imageFile = "ms-appx:///Images/_0000_.png"
            Case "thumb1"
                imageFile = "ms-appx:///Images/_0001_.png"
            Case "thumb2"
                imageFile = "ms-appx:///Images/_0002_.png"
            Case "thumb3"
                imageFile = "ms-appx:///Images/_0003_.png"
            Case "thumb4"
                imageFile = "ms-appx:///Images/_0004_.png"
            Case "thumb5"
                imageFile = "ms-appx:///Images/_0005_.png"
            Case "thumb6"
                imageFile = "ms-appx:///Images/_0006_.png"
            Case "thumb7"
                imageFile = "ms-appx:///Images/_0007_.png"
            Case "thumb8"
                imageFile = "ms-appx:///Images/_0008_.png"
        End Select
        Dim myUri As Uri = New Uri(imageFile)
        Dim bmi As BitmapImage = New BitmapImage(myUri)
        imgDisplay.Source = bmi
    End Sub
```

You can change the orientation of the StackPanel from its default vertical display to a horizontal one by adding an Orientation=“Horizontal” attribute. Likewise, you can change the ScrollViewer to have a horizontal orientation by adding attributes to disable and enable the VeriticalScrollBarVisibility and the HorizontalScrollBarVisibility respectively. Finally, the Height value of the rectangles between the images was modified to be a Width value. The result is the same app, but the StackPanel/ScrollViewer runs across the top of the page rather than down the left side. No change was necessary in the underlying C# or VB code.

Figure 2 *– The StackPanel (and ScrollViewer) can have a horizontal orientation. Images used are in the public domain.*

The Modified XAML code:

```
<Page
    x:Class="_05_StackPanel_Demo__VB_.MainPage"
    xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
    xmlns:local="using:_05_StackPanel_Demo__VB_"
    xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
    xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
    mc:Ignorable="d">

    <Grid Background="{StaticResource ApplicationPageBackgroundThemeBrush}">
        <Grid.ColumnDefinitions>
            <ColumnDefinition Width="140" />
            <ColumnDefinition Width="160" />
            <ColumnDefinition />
        </Grid.ColumnDefinitions>
        <Grid.RowDefinitions>
            <RowDefinition Height="140"/>
            <RowDefinition Height="*"/>
        </Grid.RowDefinitions>

        <TextBlock Text="My Photo Viewer" Grid.Column="1" Grid.ColumnSpan="2"
                   Style="{StaticResource PageHeaderTextStyle}"/>
        <ScrollViewer Grid.Row="1" Grid.Column="1" Grid.ColumnSpan="2" Height="120"
                      Margin="0,0,50,0" VerticalAlignment="Top"
                      HorizontalScrollBarVisibility="Auto"
                      VerticalScrollBarVisibility="Disabled">
            <StackPanel Orientation="Horizontal" Height="120">
```

```
                <Image x:Name="thumb0" Source="Images/_0000_320x240.png"
                       Tapped="ThumbnailTapped"/>
                <Rectangle Width="20"/>
                <Image x:Name="thumb1" Source="Images/_0001_320x240.png"
                       Tapped="ThumbnailTapped"/>
                <Rectangle Width="20"/>
                <Image x:Name="thumb2" Source="Images/_0002_320x240.png"
                       Tapped="ThumbnailTapped"/>
                <Rectangle Width="20"/>
                <Image x:Name="thumb3" Source="Images/_0003_320x240.png"
                       Tapped="ThumbnailTapped"/>
                <Rectangle Width="20"/>
                <Image x:Name="thumb4" Source="Images/_0004_320x240.png"
                       Tapped="ThumbnailTapped"/>
                <Rectangle Width="20"/>
                <Image x:Name="thumb5" Source="Images/_0005_320x240.png"
                       Tapped="ThumbnailTapped"/>
                <Rectangle Width="20"/>
                <Image x:Name="thumb6" Source="Images/_0006_320x240.png"
                       Tapped="ThumbnailTapped"/>
                <Rectangle Width="20"/>
                <Image x:Name="thumb7" Source="Images/_0007_320x240.png"
                       Tapped="ThumbnailTapped"/>
                <Rectangle Width="20"/>
                <Image x:Name="thumb8" Source="Images/_0008_320x240.png"
                       Tapped="ThumbnailTapped"/>
            </StackPanel>
        </ScrollViewer>
        <Image x:Name="imgDisplay" Grid.Column="2" HorizontalAlignment="Left"
               Height="425" Margin="104,152,0,0" Grid.Row="1" VerticalAlignment="Top"
               Width="568" Source="Images/_0000_.png"/>
    </Grid>
</Page>
```

The FlipView

The FlipView is an ItemsControl that presents one item at a time and automatically provides the ability to scroll left or right (or up and down if configured) to see the previous or next item in the list.

The FlipView might be used to present the same images from the StackPanel demo. In this sense, it functions more as a Panel control. For mouse-controlled devices, scroll arrow buttons appear at the left and right.

Figure 3 – *The FlipView is used to present one data item at a time. The user can scroll left or right to view the previous or next item. The Barns grand tetons mountains.jpg by Jon Sullivan is from Wikimedia and is in the public domain.*

XAML Code:

```
<Page
    x:Class="WrapGrid_Demo__VB_.MainPage"
    xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
    xmlns:local="using:WrapGrid_Demo__VB_"
    xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
    xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
    mc:Ignorable="d">

    <Grid Background="{StaticResource ApplicationPageBackgroundThemeBrush}">
        <Grid.ColumnDefinitions>
            <ColumnDefinition Width="140" />
            <ColumnDefinition Width="160" />
            <ColumnDefinition />
        </Grid.ColumnDefinitions>
        <Grid.RowDefinitions>
            <RowDefinition Height="140"/>
            <RowDefinition Height="*"/>
            <RowDefinition Height="50" />
         </Grid.RowDefinitions>

        <TextBlock Text="FlipView Demo" Grid.Column="1" Grid.ColumnSpan="2"
                   Style="{StaticResource PageHeaderTextStyle}"/>
        <FlipView Grid.Row="1" Grid.Column="0" Grid.ColumnSpan="3">
            <Image Source="Images/_0000_.png"/>
            <Image Source="Images/_0001_.png"/>
```

```
            <Image Source="Images/_0002_.png"/>
            <Image Source="Images/_0003_.png"/>
            <Image Source="Images/_0004_.png"/>
            <Image Source="Images/_0005_.png"/>
            <Image Source="Images/_0006_.png"/>
            <Image Source="Images/_0007_.png"/>
            <Image Source="Images/_0008_.png"/>
        </FlipView>
    </Grid>
</Page>
```

There is no C# or VB code -behind.

There are disadvantages to using a FlipView. For this image presentation, the user cannot preview the images as easily as when multiple images were presented in a StackPanel. To access a particular image, the end user has to scroll left or right until they find the image.

The Canvas

The Canvas control is a fixed dimensional panel. It is often used as an interactive painting area, thus the name Canvas. The Canvas has its own coordinate system and it is easiest to position objects in the Canvas using its coordinates. Using a Canvas lends itself well to a game where all the actions are not visible at one time.

Consider an archery game where the bow and arrow are on the left, but the target is on the right. Consider the following app:

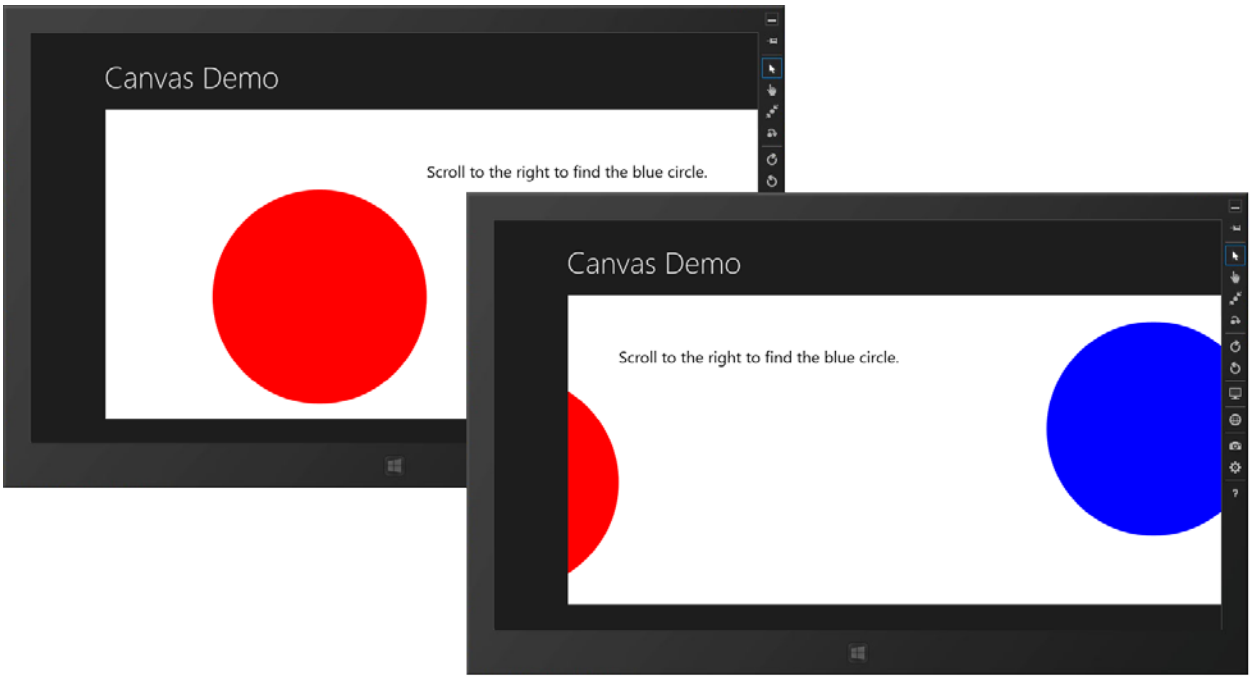

Figure 4 *– The white box is a Canvas (inside a ScrollView). It is 2000 pixels in width and too long to view all at one time.*

The XAML code:

```
<Page
    x:Class="_05_Canvas_Demo.MainPage"
```

```
    xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
    xmlns:local="using:_05_Canvas_Demo"
    xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
    xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
    mc:Ignorable="d">

        <Grid Background="{StaticResource ApplicationPageBackgroundThemeBrush}">
            <Grid.ColumnDefinitions>
                <ColumnDefinition Width="140" />
                <ColumnDefinition Width="160" />
                <ColumnDefinition />
            </Grid.ColumnDefinitions>
            <Grid.RowDefinitions>
                <RowDefinition Height="140"/>
                <RowDefinition Height="*"/>
                <RowDefinition Height="50" />
            </Grid.RowDefinitions>

            <TextBlock Text="Canvas Demo" Grid.Column="1" Grid.ColumnSpan="2"
                   Style="{StaticResource PageHeaderTextStyle}"/>
            <ScrollViewer Grid.Row ="1" HorizontalScrollBarVisibility="Auto"
                      Grid.ColumnSpan="2" Grid.Column="1"
                      VerticalScrollBarVisibility="Disabled">
            <Canvas HorizontalAlignment="Left"
                Height="578" Grid.Row="1" VerticalAlignment="Top" Width="2000"
                Background="White">
                <Ellipse Canvas.Left="200" Canvas.Top="150" Fill="Red" Height="400"
                         Width="400" />
                <TextBlock Text="Scroll to the right to find the blue circle."
                           FontSize="30" Canvas.Top="100" Canvas.Left="600"
                           Foreground="Black" />
                <Ellipse Canvas.Left="1400" Canvas.Top="50" Fill="Blue" Height="400"
                         Width="400" />
            </Canvas>
        </ScrollViewer>
    </Grid>
</Page>
```

The following example uses a Canvas control as a space for freehand painting. The PointerPressed and PointerReleased events are handled, allowing the canvas to toggle the drawing activity on or off. The PointerMoved event causes a 12-pixel diameter ellipse to be drawn using a blue-color brush—both actions are set as class-level variables. In the practice at the end of this lesson, you will duplicate this app and enhance it by adding the ability to change colors and brush sizes.

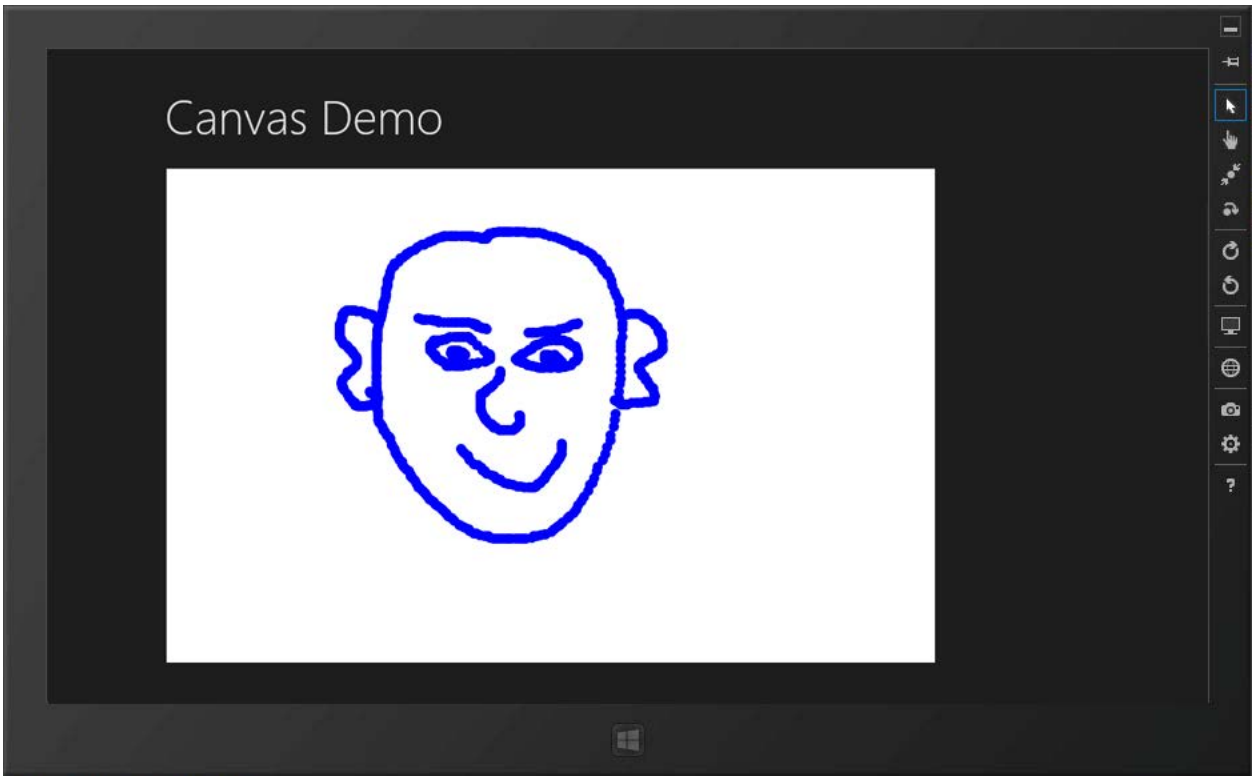

Figure 5 *– The Canvas control in this app is used as the area in which freehand painting can occur.*

XAML Code

```
<Page
    x:Class="_05_Canvas_Demo_2__Painting__VB_.MainPage"
    xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
    xmlns:local="using:_05_Canvas_Demo_2__Painting__VB_"
    xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
    xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
    mc:Ignorable="d">

    <Grid Background="{StaticResource ApplicationPageBackgroundThemeBrush}">
        <Grid.ColumnDefinitions>
            <ColumnDefinition Width="140" />
            <ColumnDefinition Width="160" />
            <ColumnDefinition />
        </Grid.ColumnDefinitions>
        <Grid.RowDefinitions>
            <RowDefinition Height="140"/>
            <RowDefinition Height="*"/>
            <RowDefinition Height="50" />
        </Grid.RowDefinitions>

        <TextBlock Text="Canvas Demo" Grid.Column="1" Grid.ColumnSpan="2"
                   Style="{StaticResource PageHeaderTextStyle}"/>
        <Canvas x:Name="myCanvas" Grid.Row ="1"  Grid.Column="1"
                Grid.ColumnSpan="2" HorizontalAlignment="Left"
                Height="580" VerticalAlignment="Top" Width="900"
                Background="White" PointerPressed="StartPaint" PointerMoved="PaintOnMe"
```

```
                PointerReleased="StopPaint">
        </Canvas>
    </Grid>
</Page>
```

C# Code:

```
using System;
using System.Collections.Generic;
using System.IO;
using System.Linq;
using Windows.Foundation;
using Windows.Foundation.Collections;
using Windows.UI.Xaml;
using Windows.UI.Xaml.Controls;
using Windows.UI.Xaml.Controls.Primitives;
using Windows.UI.Xaml.Data;
using Windows.UI.Xaml.Input;
using Windows.UI.Xaml.Media;
using Windows.UI.Xaml.Navigation;
using Windows.UI.Input;
using Windows.UI.Xaml.Shapes;
using Windows.UI;


// The Blank Page item template is documented at
http://go.microsoft.com/fwlink/?LinkId=234238

namespace _05_Canvas_Demo_2__Painting__VB_
{
    /// <summary>
    /// An empty page that can be used on its own or navigated to within a Frame.
    /// </summary>
    public sealed partial class MainPage : Page
    {
        bool painting = false;
        Brush brushColor = new SolidColorBrush(Colors.Blue);
        double brushDiameter = 12.0;
        double lastX, lastY;

        public MainPage()
        {
            this.InitializeComponent();
        }

        /// <summary>
        /// Invoked when this page is about to be displayed in a Frame.
        /// </summary>
        /// <param name="e">Event data that describes how this page was reached.  The
Parameter
        /// property is typically used to configure the page.</param>
        protected override void OnNavigatedTo(NavigationEventArgs e)
        {
        }

        private void StartPaint(object sender, PointerRoutedEventArgs e)
        {
            painting = true;
```

```
            lastX = -1;
            lastY = -1;
        }

        private void PaintOnMe(object sender, PointerRoutedEventArgs e)
        {
            if (painting)
            {
                PointerPoint myLoc = e.GetCurrentPoint(myCanvas);
               if (lastX != -1 )
                {
                    Line myLine = new Line();
                    myLine.StrokeThickness = brushDiameter;
                    myLine.StrokeEndLineCap = PenLineCap.Round;
                    myLine.Stroke = brushColor;
                    myLine.X1 = lastX;
                    myLine.Y1 = lastY;
                    myLine.X2 = myLoc.Position.X;
                    myLine.Y2 = myLoc.Position.Y;
                    myCanvas.Children.Add(myLine);
                }
                lastX = myLoc.Position.X;
                lastY = myLoc.Position.Y;
            }
        }

        private void StopPaint(object sender, PointerRoutedEventArgs e)
        {
            painting = false;
        }
    }
}
```

VB Code:

```
Imports Windows.UI.Input
Imports Windows.UI.Xaml.Shapes
Imports Windows.UI


Public NotInheritable Class MainPage
    Inherits Page

    Dim painting As Boolean = False
    Dim brushColor As Brush = New SolidColorBrush(Colors.Blue)
    Dim brushDiameter As Double = 12.0
    Dim lastX, lastY As Double

    Protected Overrides Sub OnNavigatedTo(e As Navigation.NavigationEventArgs)

    End Sub

    Private Sub PaintOnMe(sender As Object, e As PointerRoutedEventArgs)
        If (painting) Then
            Dim myLoc As PointerPoint = e.GetCurrentPoint(myCanvas)
            If (lastX <> -1) Then
                Dim myLine As Line = New Line()
                myLine.StrokeThickness = brushDiameter
```

```
                myLine.StrokeEndLineCap = PenLineCap.Round
                myLine.Stroke = brushColor
                myLine.X1 = lastX
                myLine.Y1 = lastY
                myLine.X2 = myLoc.Position.X
                myLine.Y2 = myLoc.Position.Y
                myCanvas.Children.Add(myLine)
            End If
            lastX = myLoc.Position.X
            lastY = myLoc.Position.Y
        End If
    End Sub

    Private Sub StartPaint(sender As Object, e As PointerRoutedEventArgs)
        painting = True
        lastX = -1
        lastY = -1
    End Sub

    Private Sub StopPaint(sender As Object, e As PointerRoutedEventArgs)
        painting = False
    End Sub
End Class
```

ListBoxes and ComboBoxes

Windows 8 app ListBoxes and ComboBoxes function much like their desktop application counterparts. Both are derived from the ItemsControl class and thus display a collection of items. Item can be hardcoded in the XAML, added via user interaction using C# or VB coded instructions, bound to a database, or read from a text file.

One interesting note is that on touch devices, ComboBox items take on a circular display. Also in the ComboBox, the items list is displayed on top of the control by default rather than underneath.

TIP: Try the ComboBox function in the simulator with the Touch simulation.

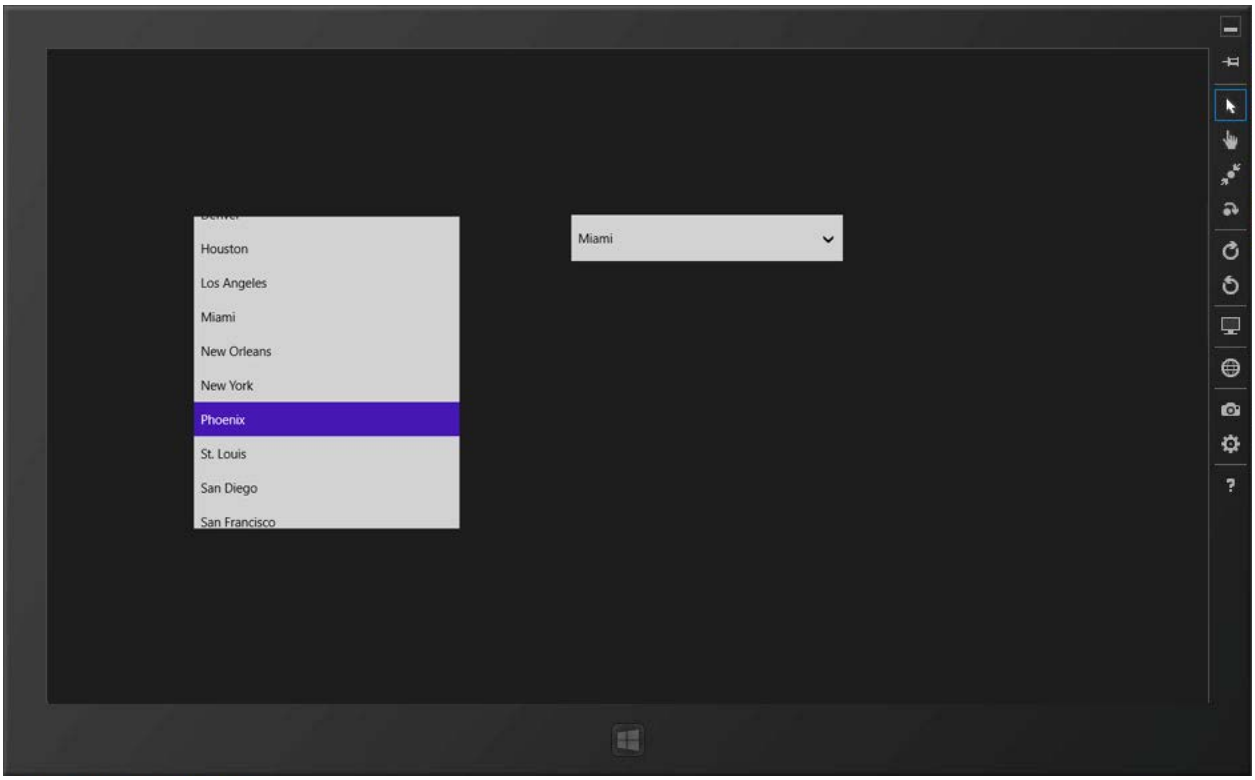

Figure 6 *– Both the ListBox (left) and ComboBox (right) are used to display a selectable list of items.*

XAML Code

```
    <Grid Background="{StaticResource ApplicationPageBackgroundThemeBrush}">
        <ListBox x:Name="lbCities" HorizontalAlignment="Left" Height="369"
                 Margin="167,191,0,0" VerticalAlignment="Top" Width="315">
            <ListBoxItem>Atlanta</ListBoxItem>
            <ListBoxItem>Boston</ListBoxItem>
            <ListBoxItem>Chicago</ListBoxItem>
            <ListBoxItem>Dallas</ListBoxItem>
            <ListBoxItem>Denver</ListBoxItem>
            <ListBoxItem>Houston</ListBoxItem>
            <ListBoxItem>Los Angeles</ListBoxItem>
            <ListBoxItem>Miami</ListBoxItem>
            <ListBoxItem>New Orleans</ListBoxItem>
            <ListBoxItem>New York</ListBoxItem>
            <ListBoxItem>Phoenix</ListBoxItem>
            <ListBoxItem>St. Louis</ListBoxItem>
            <ListBoxItem>San Diego</ListBoxItem>
            <ListBoxItem>San Francisco</ListBoxItem>
            <ListBoxItem>Seattle</ListBoxItem>
        </ListBox>
        <ComboBox x:Name ="cmbCities" HorizontalAlignment="Left" Height="54"
                  Margin="611,191,0,0" VerticalAlignment="Top" Width="318">
            <ComboBoxItem>Atlanta</ComboBoxItem>
            <ComboBoxItem>Boston</ComboBoxItem>
            <ComboBoxItem>Chicago</ComboBoxItem>
            <ComboBoxItem>Dallas</ComboBoxItem>
            <ComboBoxItem>Denver</ComboBoxItem>
            <ComboBoxItem>Houston</ComboBoxItem>
```

```
            <ComboBoxItem>Miami</ComboBoxItem>
            <ComboBoxItem>New Orleans</ComboBoxItem>
            <ComboBoxItem>New York</ComboBoxItem>
            <ComboBoxItem>Phoenix</ComboBoxItem>
            <ComboBoxItem>St. Louis</ComboBoxItem>
            <ComboBoxItem>San Diego</ComboBoxItem>
            <ComboBoxItem>San Francisco</ComboBoxItem>
            <ComboBoxItem>Seattle</ComboBoxItem>
        </ComboBox>
    </Grid>
```

A unique feature of ComboBoxes and ListBoxes is the ability to format individual items in their display lists. You can modify the XAML code as follows:

XAML Code

```
  <ListBoxItem FontWeight="Bold">New York</ListBoxItem>
  <ListBoxItem FontSize="26">Phoenix</ListBoxItem>
  <ListBoxItem Foreground="Red">Saint Louis</ListBoxItem>
  <ListBoxItem FontFamily="Impact" FontSize="26"
               Foreground="Blue">San Diego</ListBoxItem>
```

Figure 7 *– Individual items can be formatted.*

The ListView Control

ListViews are preferred by most Windows Store developers to the ListBox control inherited from older versions of Windows. The spacing of items better supports touch devices. It also includes built-in animations when selecting, adding, or removing items.

ListViews are developed in much the same way as ListBoxes in XAML.

```
        <ListView HorizontalAlignment="Left" Height="369"
                  Margin="645,191,0,0" VerticalAlignment="Top"
                  Width="252" SelectionMode="Multiple">
            <ListViewItem>Atlanta</ListViewItem>
            <ListViewItem>Boston</ListViewItem>
            <ListViewItem>Chicago</ListViewItem>
            <ListViewItem>Dallas</ListViewItem>
            <ListViewItem>Denver</ListViewItem>
            <ListViewItem>Houston</ListViewItem>
            <ListViewItem>Los Angeles</ListViewItem>
            <ListViewItem>Miami</ListViewItem>
```

```
            <ListViewItem>New Orleans</ListViewItem>
            <ListViewItem>New York</ListViewItem>
            <ListViewItem>Phoenix</ListViewItem>
            <ListViewItem>Saint Louis</ListViewItem>
            <ListViewItem>San Diego</ListViewItem>
            <ListViewItem>San Francisco</ListViewItem>
            <ListViewItem>Seattle</ListViewItem>
        </ListView>
```

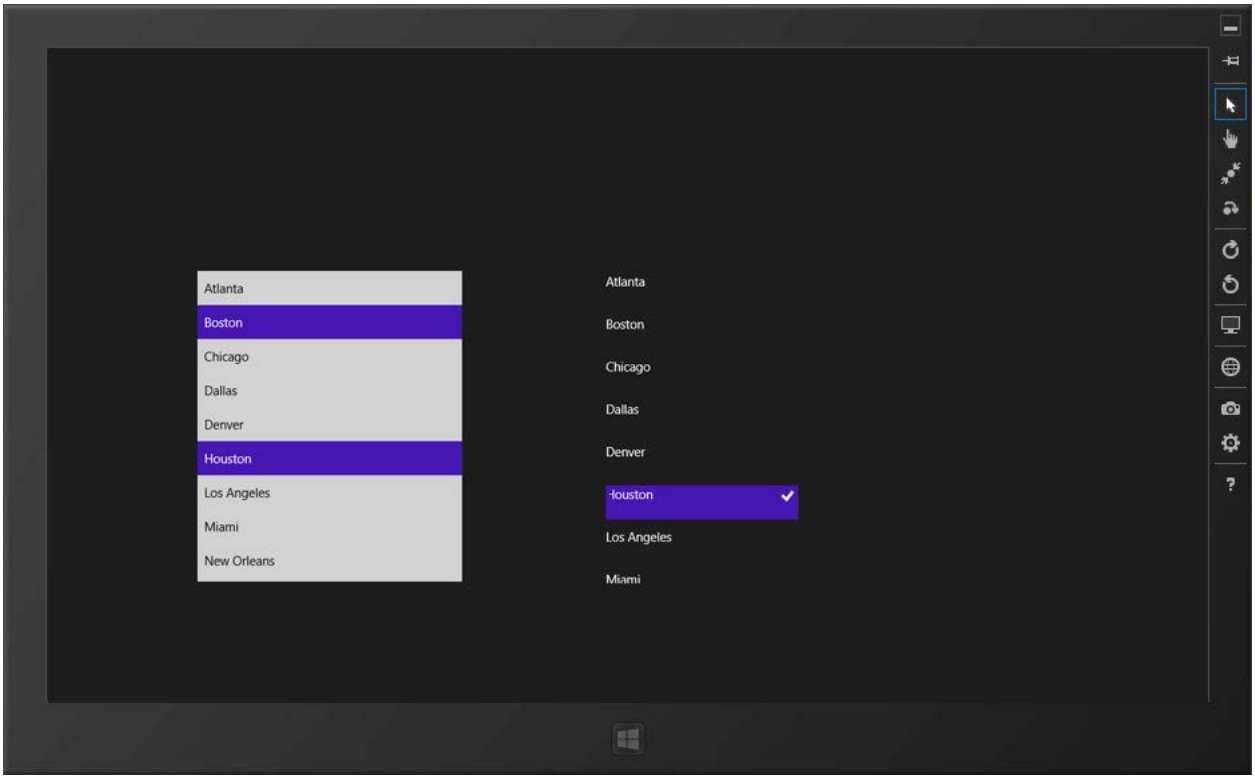

Figure 8 *– The ListView on the right provides better spacing than a ListBox (left) for Touch devices. It also includes built-in animations when selecting, adding, or removing items.*

Items can be styled in a custom Resource Dictionary. Consider the following XAML Style code:

```
    <Style TargetType="ListViewItem">
        <Setter Property="Background" Value="DarkTurquoise"/>
        <Setter Property="Foreground" Value="White"/>
        <Setter Property="BorderBrush" Value="White"/>
        <Setter Property="BorderThickness" Value="3"/>
        <Setter Property="FontSize" Value="26"/>
        <Setter Property="Margin" Value="5"/>
    </Style>
```

Do not forget to add a reference to the add Resource Dictionary to the app.xaml code!

```
    <ResourceDictionary Source="Assets/Dictionary1.xaml"/>
```

The result is a modified display of the ListView's items.

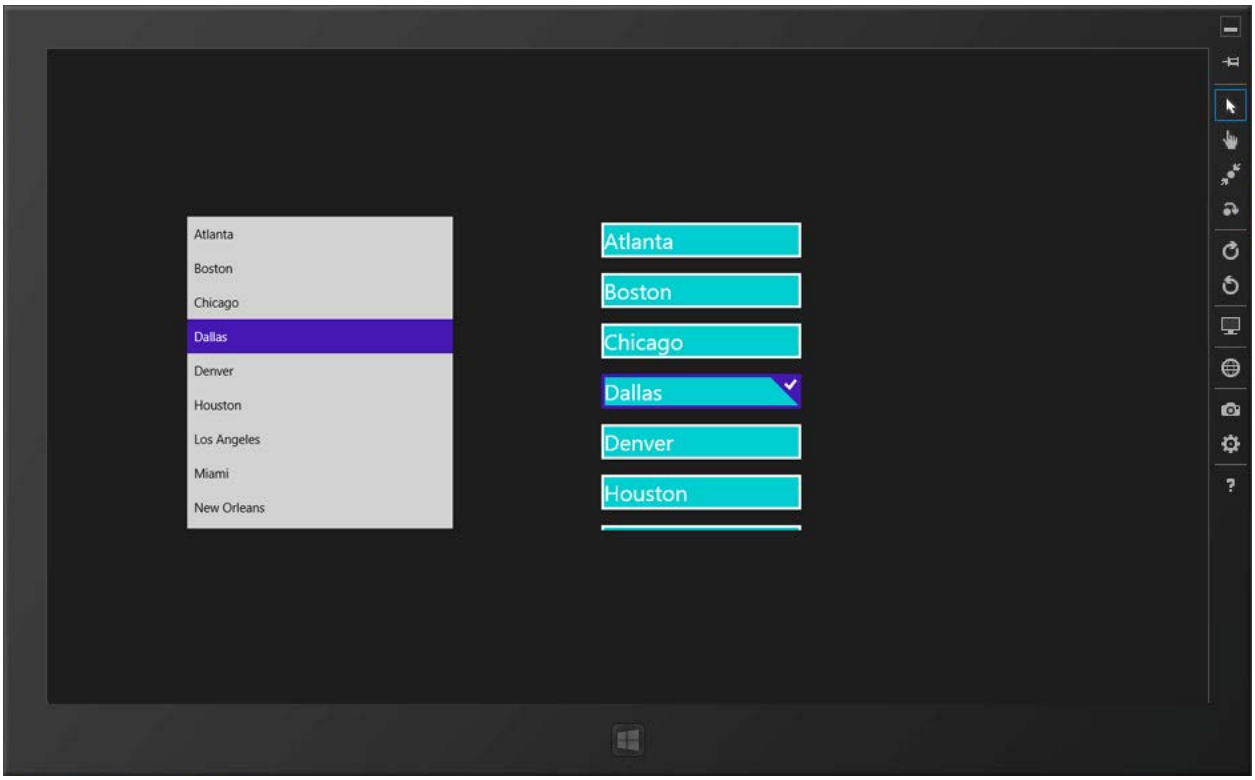

Figure 9 *– A custom style targeting the ListViewItem changes the visual appearance of the items.*

ListViews (as well as ListBoxes and ComboBoxes) are predominantly coded by handling the SelectionChanged event and utilizing the control's SelectedIndex (integer value) or SelectedItem (string value) property. The following app utilizes a ListView to select which embedded image is displayed.

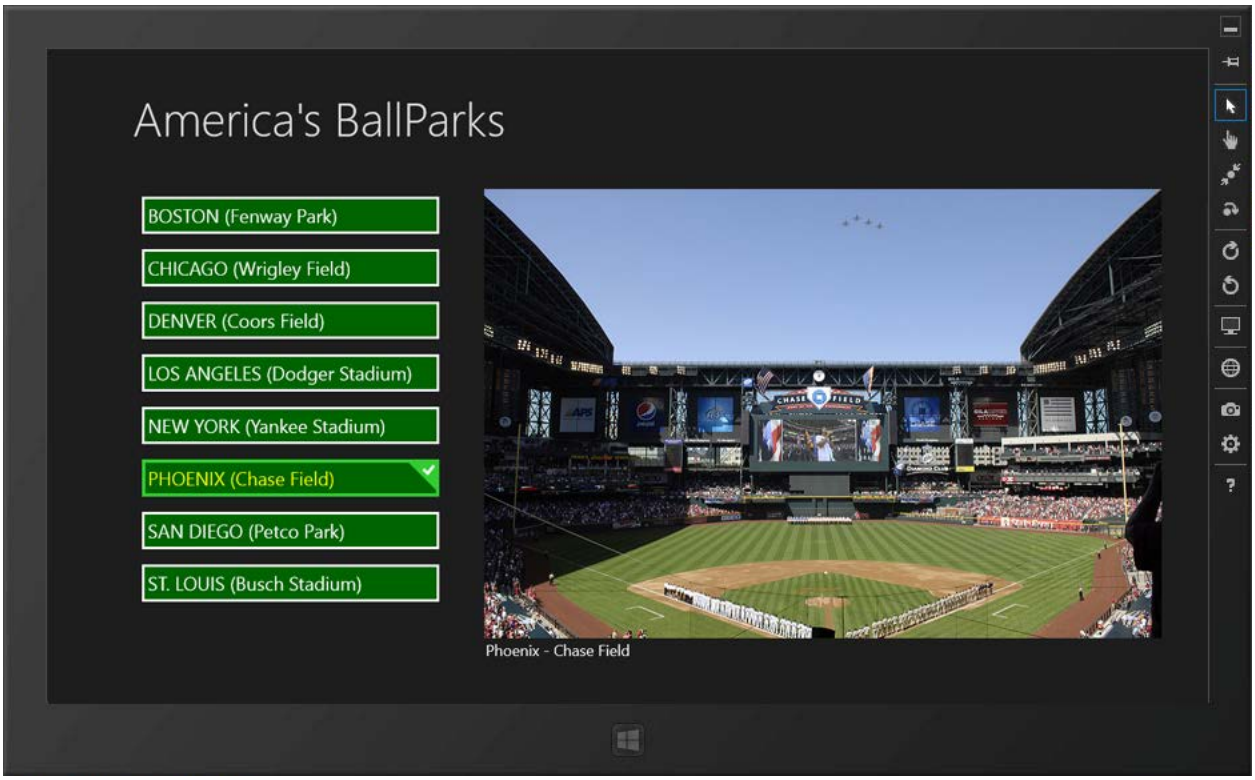

Figure 10 *- In this app, the ListView's SelectionChanged event is handled to change the displayed image. Note the highlighted item—the SelectedBackground color is set to LimeGreen and the SelectedForeground is set to Yellow by setting these SolidColorBrush styles in the App.xaml file's code.*

XAML Code (MainPage.xaml):

```
    <Grid Background="{StaticResource ApplicationPageBackgroundThemeBrush}">
        <ListView x:Name="lstBallParks" HorizontalAlignment="Left" Height="501"
                  Margin="100,162,0,0" VerticalAlignment="Top" Width="369"
                  SelectionChanged="BallparkChosen" >
            <ListViewItem>BOSTON (Fenway Park)</ListViewItem>
            <ListViewItem>CHICAGO (Wrigley Field)</ListViewItem>
            <ListViewItem>DENVER (Coors Field)</ListViewItem>
            <ListViewItem>LOS ANGELES (Dodger Stadium)</ListViewItem>
            <ListViewItem>NEW YORK (Yankee Stadium)</ListViewItem>
            <ListViewItem>PHOENIX (Chase Field)</ListViewItem>
            <ListViewItem>SAN DIEGO (Petco Park)</ListViewItem>
            <ListViewItem>ST. LOUIS (Busch Stadium)</ListViewItem>
        </ListView>
        <Image x:Name="imgBallpark" HorizontalAlignment="Left" Height="530"
               Margin="513,162,0,0" VerticalAlignment="Top" Width="800"
               Source="Images/ballpark_title.png" Stretch="Fill" />
        <TextBlock HorizontalAlignment="Left" Height="66" Margin="100,60,0,0"
                   Text="America's BallParks" VerticalAlignment="Top" Width="740"
                   Style="{StaticResource HeaderTextStyle}"/>
        <TextBlock x:Name="txtCaption" HorizontalAlignment="Left" Height="24"
                   Margin="515,697,0,0" Text="" VerticalAlignment="Top" Width="616"
                   FontSize="18"/>
    </Grid>
```

Dictionary1.xaml:

```
<ResourceDictionary
    xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
    xmlns:local="using:_05_ListBox_v_ListView.Assets">

    <Style TargetType="ListViewItem">
        <Setter Property="Background" Value="DarkGreen"/>
        <Setter Property="Foreground" Value="White"/>
        <Setter Property="BorderBrush" Value="White"/>
        <Setter Property="BorderThickness" Value="3"/>
        <Setter Property="FontSize" Value="22"/>
        <Setter Property="Margin" Value="5"/>
        <Setter Property="Padding" Value="4"/>
    </Style>

</ResourceDictionary>
```

The Selected state colors were altered in the **App.xaml** file.

```
<Application
    x:Class="_05_ListView_Ballparks__CS_.App"
    xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
    xmlns:local="using:_05_ListView_Ballparks__CS_">

    <Application.Resources>
        <ResourceDictionary>

            <SolidColorBrush x:Key="ListViewItemSelectedBackgroundThemeBrush"
                    Color="LimeGreen" />
            <SolidColorBrush x:Key="ListViewItemSelectedForegroundThemeBrush"
                     Color="Yellow" />

            <ResourceDictionary.MergedDictionaries>

                <!--
                    Styles that define common aspects of the platform look and feel
                    Required by Visual Studio project and item templates
                 -->
                <ResourceDictionary Source="Common/StandardStyles.xaml"/>
                <ResourceDictionary Source="Assets/Dictionary1.xaml"/>
            </ResourceDictionary.MergedDictionaries>

        </ResourceDictionary>
    </Application.Resources>
</Application>
```

C# Code:

```
        private void BallparkChosen(object sender, SelectionChangedEventArgs e)
        {
            //add: using Windows.UI.Xaml.Media.Imaging;
            String imageFile;
            switch (lstBallParks.SelectedIndex)
            {
                case 0:
                    imageFile = "ms-appx:///Images/Ballpark_BOS.png";
                    txtCaption.Text = "Boston - Fenway Park";
```

```
                    break;
                case 1:
                    imageFile = "ms-appx:///Images/Ballpark_CHI.png";
                    txtCaption.Text = "Chicago - Wrigley Field";
                    break;
                case 2:
                    imageFile = "ms-appx:///Images/Ballpark_DEN.png";
                    txtCaption.Text = "Denver - Coors Field";
                    break;
                case 3:
                    imageFile = "ms-appx:///Images/Ballpark_LAD.png";
                    txtCaption.Text = "Los Angeles - Dodger Stadium";
                    break;
                case 4:
                    imageFile = "ms-appx:///Images/Ballpark_NYY.png";
                    txtCaption.Text = "New York - Yankee Stadium";
                    break;
                case 5:
                    imageFile = "ms-appx:///Images/Ballpark_AZ.png";
                    txtCaption.Text = "Phoenix - Chase Field";
                    break;
                case 6:
                    imageFile = "ms-appx:///Images/Ballpark_SD.png";
                    txtCaption.Text = "San Diego - Petco Park";
                    break;
                case 7:
                    imageFile = "ms-appx:///Images/Ballpark_STL.png";
                    txtCaption.Text = "St. Louis - Busch Stadium";
                    break;
                default:
                    imageFile = "ms-appx:///Images/ballpark_title.png";
                    txtCaption.Text = "";
                    break;
            }
            Uri myUri = new Uri(imageFile);
            BitmapImage bmi = new BitmapImage(myUri);
            imgBallpark.Source = bmi;
        }
```

VB Code:

```
Private Sub BallparkChosen(sender As Object, e As SelectionChangedEventArgs)
        'add: imports Windows.UI.Xaml.Media.Imaging
        Dim imageFile As String
        Select Case lstBallParks.SelectedIndex
            Case 0
                imageFile = "ms-appx:///Images/Ballpark_BOS.png"
                txtCaption.Text = "Boston - Fenway Park"
            Case 1
                imageFile = "ms-appx:///Images/Ballpark_CHI.png"
                txtCaption.Text = "Chicago - Wrigley Field"
            Case 2
                imageFile = "ms-appx:///Images/Ballpark_DEN.png"
                txtCaption.Text = "Denver - Coors Field"
            Case 3
                imageFile = "ms-appx:///Images/Ballpark_LAD.png"
                txtCaption.Text = "Los Angeles - Dodger Stadium"
            Case 4
```

```
                imageFile = "ms-appx:///Images/Ballpark_NYY.png"
                txtCaption.Text = "New York - Yankee Stadium"
            Case 5
                imageFile = "ms-appx:///Images/Ballpark_AZ.png"
                txtCaption.Text = "Phoenix - Chase Field"
            Case 6
                imageFile = "ms-appx:///Images/Ballpark_SD.png"
                txtCaption.Text = "San Diego - Petco Park"
            Case 7
                imageFile = "ms-appx:///Images/Ballpark_STL.png"
                txtCaption.Text = "St. Louis - Busch Stadium"
            Case Else
                imageFile = "ms-appx:///Images/ballpark_title.png"
                txtCaption.Text = ""
        End Select
        Dim myUri As Uri = New Uri(imageFile)
        Dim bmi As BitmapImage = New BitmapImage(myUri)
        imgBallpark.Source = bmi
    End Sub
```

ListViews are most often used with data. This example uses a ListView with a small array of structured data that is hardcoded into the underlying code-behind file. The previous example can be rewritten to dynamically provide the items for the ListView using the array data.

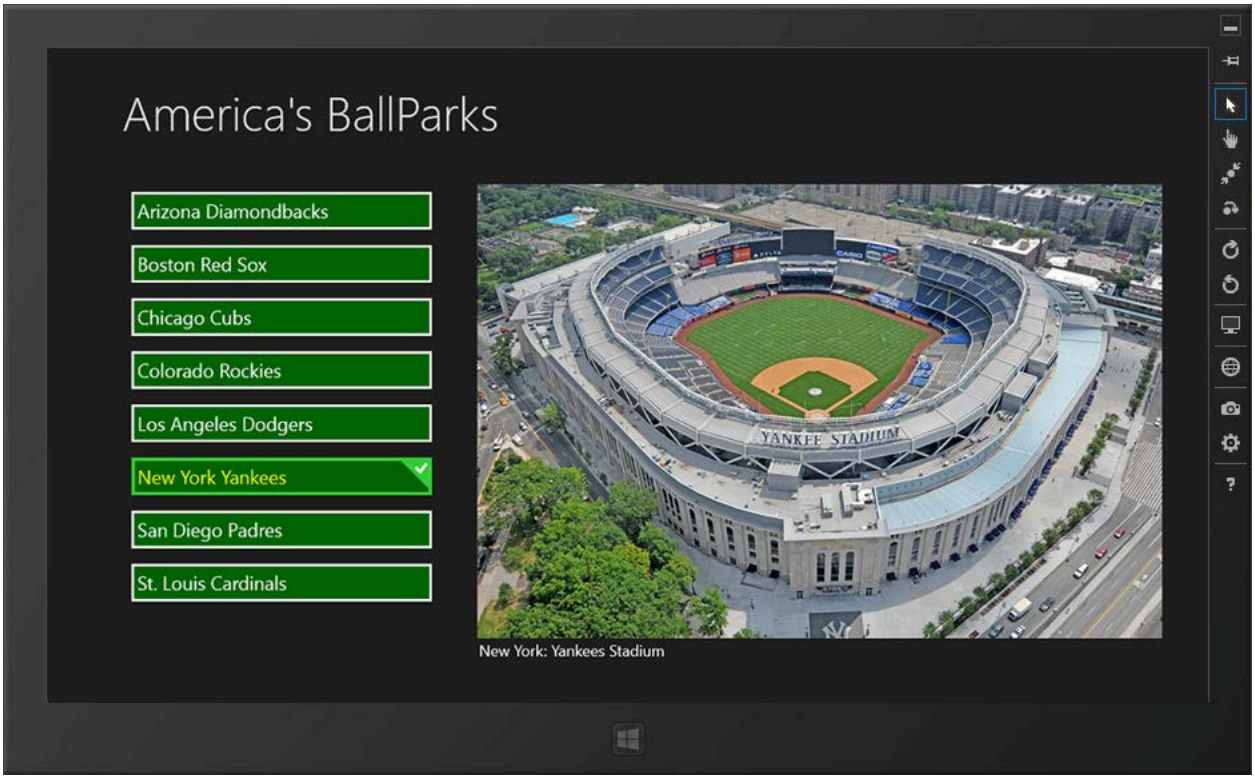

Figure 11 *– The items of the ListView are dynamically provided from an array in the code-behind file.*

This simplifies the XAML code for the page.

```
    <Grid Background="{StaticResource ApplicationPageBackgroundThemeBrush}">
        <ListView x:Name="lstBallParks" HorizontalAlignment="Left" Height="501"
                  Margin="100,162,0,0" VerticalAlignment="Top" Width="369"
                  SelectionChanged="BallparkChosen" >

        </ListView>
        <Image x:Name="imgBallpark" HorizontalAlignment="Left" Height="530"
               Margin="513,162,0,0" VerticalAlignment="Top" Width="800"
               Source="Images/ballpark_title.png" Stretch="Fill" />
        <TextBlock HorizontalAlignment="Left" Height="66" Margin="100,60,0,0"
                   Text="America's BallParks" VerticalAlignment="Top" Width="740"
                   Style="{StaticResource HeaderTextStyle}"/>
        <TextBlock x:Name="txtCaption" HorizontalAlignment="Left" Height="24"
                   Margin="515,697,0,0" Text="" VerticalAlignment="Top" Width="616"
                   FontSize="18"/>
    </Grid>
```

The style for the ListViewItems is derived from the same custom Dictionary1.xaml resource dictionary used previously, as well as the Brush styles added to the App.xaml code. The code now defines a structure and an array of the structure, populates the array, and then dynamically adds the array values to the ListView items.

C# Code:

```
using System;
using System.Collections.Generic;
using System.IO;
using System.Linq;
using Windows.Foundation;
using Windows.Foundation.Collections;
using Windows.UI.Xaml;
using Windows.UI.Xaml.Controls;
using Windows.UI.Xaml.Controls.Primitives;
using Windows.UI.Xaml.Data;
using Windows.UI.Xaml.Input;
using Windows.UI.Xaml.Media;
using Windows.UI.Xaml.Navigation;
using Windows.UI.Xaml.Media.Imaging;

namespace _05_ListView_bound_to_array__CS_
{
    public sealed partial class MainPage : Page
    {
        struct BallPark
        {
            public String team;
            public String imageFile;
            public String caption;
        };

        BallPark[] myArray = new BallPark[8];

        public MainPage()
        {
            this.InitializeComponent();
        }

        protected override void OnNavigatedTo(NavigationEventArgs e)
```

```
        {
            myArray[0].team = "Arizona Diamondbacks";
            myArray[0].imageFile = "ms-appx:///Images/Ballpark_AZ.png";
            myArray[0].caption = "Phoenix: Chase Field";
            myArray[1].team = "Boston Red Sox";
            myArray[1].imageFile = "ms-appx:///Images/Ballpark_BOS.png";
            myArray[1].caption = "Boston: Fenway Park";
            myArray[2].team = "Chicago Cubs";
            myArray[2].imageFile = "ms-appx:///Images/Ballpark_CHI.png";
            myArray[2].caption = "Chicago: Fenway Park";
            myArray[3].team = "Colorado Rockies";
            myArray[3].imageFile = "ms-appx:///Images/Ballpark_DEN.png";
            myArray[3].caption = "Denver: Coors Field";
            myArray[4].team = "Los Angeles Dodgers";
            myArray[4].imageFile = "ms-appx:///Images/Ballpark_LAD.png";
            myArray[4].caption = "Los Angeles: Dodger Stadium";
            myArray[5].team = "New York Yankees";
            myArray[5].imageFile = "ms-appx:///Images/Ballpark_NYY.png";
            myArray[5].caption = "New York: Yankees Stadium";
            myArray[6].team = "San Diego Padres";
            myArray[6].imageFile = "ms-appx:///Images/Ballpark_SD.png";
            myArray[6].caption = "San Diego: Petco Park";
            myArray[7].team = "St. Louis Cardinals";
            myArray[7].imageFile = "ms-appx:///Images/Ballpark_STL.png";
            myArray[7].caption = "St. Louis: Busch Stadium";
            lstBallParks.Items.Clear();
            for (int i=0; i<myArray.Length; i++)
            {
                lstBallParks.Items.Add(myArray[i].team);
            }
        }

        private void BallparkChosen(object sender, SelectionChangedEventArgs e)
        {
            int i = lstBallParks.SelectedIndex;
            Uri myUri = new Uri(myArray[i].imageFile);
            BitmapImage bmi  = new BitmapImage(myUri);
            imgBallpark.Source = bmi;
            txtCaption.Text = myArray[i].caption;
        }
    }
}
```

VB Code:

```
Imports Windows.UI.Xaml.Media.Imaging

Public NotInheritable Class MainPage
    Inherits Page

    Structure BallPark
        Dim team As String
        Dim imageFile As String
        Dim caption As String
    End Structure

    Dim myArray(7) As BallPark
```

```
    Protected Overrides Sub OnNavigatedTo(e As Navigation.NavigationEventArgs)
        myArray(0).team = "Arizona Diamondbacks"
        myArray(0).imageFile = "ms-appx:///Images/Ballpark_AZ.png"
        myArray(0).caption = "Phoenix: Chase Field"
        myArray(1).team = "Boston Red Sox"
        myArray(1).imageFile = "ms-appx:///Images/Ballpark_BOS.png"
        myArray(1).caption = "Boston: Fenway Park"
        myArray(2).team = "Chicago Cubs"
        myArray(2).imageFile = "ms-appx:///Images/Ballpark_CHI.png"
        myArray(2).caption = "Chicago: Fenway Park"
        myArray(3).team = "Colorado Rockies"
        myArray(3).imageFile = "ms-appx:///Images/Ballpark_DEN.png"
        myArray(3).caption = "Denver: Coors Field"
        myArray(4).team = "Los Angeles Dodgers"
        myArray(4).imageFile = "ms-appx:///Images/Ballpark_LAD.png"
        myArray(4).caption = "Los Angeles: Dodger Stadium"
        myArray(5).team = "New York Yankees"
        myArray(5).imageFile = "ms-appx:///Images/Ballpark_NYY.png"
        myArray(5).caption = "New York: Yankees Stadium"
        myArray(6).team = "San Diego Padres"
        myArray(6).imageFile = "ms-appx:///Images/Ballpark_SD.png"
        myArray(6).caption = "San Diego: Petco Park"
        myArray(7).team = "St. Louis Cardinals"
        myArray(7).imageFile = "ms-appx:///Images/Ballpark_STL.png"
        myArray(7).caption = "St. Louis: Busch Stadium"
        lstBallParks.Items.Clear()
        For i = 0 To 7
            lstBallParks.Items.Add(myArray(i).team)
        Next
    End Sub

    Private Sub BallparkChosen(sender As Object, e As SelectionChangedEventArgs)
        Dim i As Integer = lstBallParks.SelectedIndex
        Dim myUri As Uri = New Uri(myArray(i).imageFile)
        Dim bmi As BitmapImage = New BitmapImage(myUri)
        imgBallpark.Source = bmi
        txtCaption.Text = myArray(i).caption
    End Sub
End Class
```

GridView

Gridviews display data much like ListViews, but rather than one column of data, the GridView displays data in multiple columns. Data can be grouped together as well. The Start Screen utilizes a GridView to present all your installed apps. These can be grouped together and items also can be repositioned within the GridView if configured and programmed to allow for such behavior.

As with ListViews, the GridView is used primarily to display dynamic data from some bound source. Binding will be covered in more detail in a later lesson, but in the following app, a custom class is created and data is placed in an ObservableCollection object. ObservableCollections provide for auto-updating of the displayed data as the data is modified and updated.

Figure 12 *– The GridView displays data in a series of columns and can be scrolled horizontally to view more data if necessary.*

The names, teams, and positions of each data item is styled via a custom resource dictionary.

ResourceDictionary (Dictionary1.xaml)

```
<ResourceDictionary
    xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
    xmlns:local="using:_05_ListBox_v_ListView.Assets">


    <Style x:Key="PlayerName" TargetType="TextBlock">
        <Setter Property="Foreground" Value="White"/>
        <Setter Property="FontFamily" Value="SegoeUI"/>
        <Setter Property="FontSize" Value="18"/>
        <Setter Property="FontWeight" Value="Bold"/>
        <Setter Property="HorizontalAlignment" Value="Center"/>
    </Style>

    <Style x:Key="PlayerTeam" TargetType="TextBlock">
        <Setter Property="Foreground" Value="Gray"/>
        <Setter Property="FontFamily" Value="SegoeUI"/>
        <Setter Property="FontSize" Value="14"/>
        <Setter Property="FontWeight" Value="Normal"/>
        <Setter Property="HorizontalAlignment" Value="Center"/>
    </Style>

    <Style x:Key="PlayerPosition" TargetType="TextBlock">
        <Setter Property="Foreground" Value="Cyan"/>
```

```
        <Setter Property="FontFamily" Value="SegoeUI"/>
        <Setter Property="FontSize" Value="14"/>
        <Setter Property="FontWeight" Value="Bold"/>
        <Setter Property="HorizontalAlignment" Value="Center"/>
    </Style>

</ResourceDictionary>
```

The XAML code of the page provides a named DataTemplate structure, which is named in the example as 'DataTemplate1'. This DataTemplate is bound to the ItemTemplate attributes as a StaticResource.

XAML Code (MainPage.xaml)

```
<Page
    x:Class="_05_GridView_Demo__CS_.MainPage"
    xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
    xmlns:local="using:_05_GridView_Demo__CS_"
    xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
    xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
    mc:Ignorable="d">

    <Page.Resources>
        <DataTemplate x:Key="DataTemplate1" >
            <Border BorderBrush="Red" BorderThickness="2">
                <Grid Height="80" Width="240" >
                    <Grid.ColumnDefinitions>
                        <ColumnDefinition Width="80" />
                        <ColumnDefinition Width="*" />
                    </Grid.ColumnDefinitions>
                    <Image Width="70" Height="70" Stretch="Uniform"
                           Source="{Binding Logo}" />
                    <StackPanel Grid.Column="1" VerticalAlignment="Center">
                        <TextBlock  Height="25" Text="{Binding FullName}"
                           VerticalAlignment="Center"
                            Style="{StaticResource PlayerName}"/>
                        <TextBlock  Height="15" Text="{Binding Team}"
                           VerticalAlignment="Center"
                            Style="{StaticResource PlayerTeam}"/>
                        <TextBlock  Height="15" Text="{Binding Position}"
                           VerticalAlignment="Center"
                            Style="{StaticResource PlayerPosition}"/>
                    </StackPanel>
                </Grid>
            </Border>
        </DataTemplate>

    </Page.Resources>

    <Grid Background="{StaticResource ApplicationPageBackgroundThemeBrush}">
        <Grid.RowDefinitions>
            <RowDefinition Height="140" />
            <RowDefinition Height="*" />
        </Grid.RowDefinitions>
        <Grid.ColumnDefinitions>
            <ColumnDefinition Width="140"/>
            <ColumnDefinition Width="*"/>
```

```
        </Grid.ColumnDefinitions>
        <TextBlock x:Name="pageTitle" Grid.Column="1" Text="Central City Little League"
                   Style="{StaticResource PageHeaderTextStyle}"/>
        <GridView x:Name="gvRoster" Grid.Row="1" Grid.Column="1"
                  ItemTemplate="{StaticResource DataTemplate1}">

        </GridView>
    </Grid>
</Page>
```

In the code-behind, a custom class (Player) is created with a constructor that accepts four string values: first name, last name, team, and position played. Instances of each data object are added to an ObservableCollection, and this collection is bound to the GridView. The TextBlocks and Image controls of the DataTemplate in the page's xaml code are bound to properties of the Player class (i.e. the FullName, Team, and Position values) respectively. Note that the data source (ItemsSource attribute) for the GridView (gvRoster) is established and bound here in code.

C# Code

```
using System;
using System.Collections.Generic;
using System.IO;
using System.Linq;
using Windows.Foundation;
using Windows.Foundation.Collections;
using Windows.UI.Xaml;
using Windows.UI.Xaml.Controls;
using Windows.UI.Xaml.Controls.Primitives;
using Windows.UI.Xaml.Data;
using Windows.UI.Xaml.Input;
using Windows.UI.Xaml.Media;
using Windows.UI.Xaml.Navigation;
using System.Collections.ObjectModel;


// The Blank Page item template is documented at
http://go.microsoft.com/fwlink/?LinkId=234238

namespace _05_GridView_Demo__CS_
{

    public sealed partial class MainPage : Page
    {

        public ObservableCollection<Player> Rosters = new ObservableCollection<Player>();

        public MainPage()
        {
            this.InitializeComponent();
        }

       protected override void OnNavigatedTo(NavigationEventArgs e)
        {
            // Add items to the collection.
            Rosters.Add(new Player("Carter","Chris","Cardinals","2B"));
            Rosters.Add(new Player("Baker", "Mark", "Pirates", "LB"));
```

```
            Rosters.Add(new Player("Turner", "Glenn", "Cubs", "SS"));
            Rosters.Add(new Player("Simpson", "Roger", "Cubs", "P"));
            Rosters.Add(new Player("Grant", "Tanner", "Tigers", "P"));
            Rosters.Add(new Player("Smith", "Stan", "Cardinals", "C"));
            Rosters.Add(new Player("Harris", "Blake", "Reds", "RF"));
            Rosters.Add(new Player("Jackson", "Greg", "Pirates", "P"));
            Rosters.Add(new Player("Stevenson", "William", "Giants", "1B"));
            Rosters.Add(new Player("Lincoln", "Michael", "Cardinals", "SS"));
            Rosters.Add(new Player("Brewer", "James", "Cubs", "C"));
            Rosters.Add(new Player("Peterson", "Alex", "Tigers", "RF"));
            Rosters.Add(new Player("Sherman", "Vince", "Cardinals", "CF"));
            Rosters.Add(new Player("Robertson", "Doug", "Reds", "3B"));
            Rosters.Add(new Player("Carlson", "Leonard", "Reds", "SS"));
            Rosters.Add(new Player("Marshall", "Harvey", "Pirates", "1B"));
            Rosters.Add(new Player("Patterson", "Dan", "Tigers", "LF"));
            Rosters.Add(new Player("Oswalt", "Caleb", "Cardinals", "LF"));
            Rosters.Add(new Player("Willis", "Brent", "Cubs", "P"));
            Rosters.Add(new Player("O'Grady", "Shawn", "Giants", "P"));
            Rosters.Add(new Player("Edwards", "Tony", "Tigers", "Ss"));
            Rosters.Add(new Player("Brown", "Lance", "Cubs", "CF"));
            Rosters.Add(new Player("Young", "Johnny", "Reds", "P"));
            Rosters.Add(new Player("Perez", "Sergio", "Giants", "SS"));
            Rosters.Add(new Player("Clifton", "Robbie", "Cubs", "P"));
            Rosters.Add(new Player("Saunders", "Jeff", "Cardinals", "2B"));
            Rosters.Add(new Player("Prentiss", "Shane", "Cubs", "1B"));
            Rosters.Add(new Player("Nance", "Evan", "Reds", "CF"));
            Rosters.Add(new Player("McDaniels", "Dave", "Giants", "3B"));
            Rosters.Add(new Player("Franklin", "Kyle", "Tigers", "C"));
            Rosters.Add(new Player("Olson", "Walt", "Cubs", "2B"));
            Rosters.Add(new Player("Garcia", "Jose", "Cardinals", "P"));
            Rosters.Add(new Player("Wiliams", "Sandy", "Tigers", "CF"));
            Rosters.Add(new Player("Anderson", "Rudy", "Reds", "1B"));
            Rosters.Add(new Player("Perry", "Randolph", "Giants", "C"));
            Rosters.Add(new Player("Thompson", "Victor", "Reds", "C"));
            Rosters.Add(new Player("Harrison", "Nolan", "Giants", "P"));
            Rosters.Add(new Player("Burrows", "David", "Pirates", "P"));
            Rosters.Add(new Player("Baker", "Edgar", "Cardinals", "3B"));
            Rosters.Add(new Player("Vasquez", "Jose", "Giants", "RF"));

            gvRoster.ItemsSource = Rosters;

        }
    }

    public class Player
    {
        public string LastName { get; set; }
        public string FirstName { get; set; }
        public string FullName { get; set; }
        public string Team { get; set; }
        public string Position { get; set; }
        public string Logo { get; set; }

        public Player() { }

        public Player(string lastN, string firstN, string myTeam, string pos)
        {
            LastName = lastN;
```

```
            FirstName = firstN;
            FullName = firstN + " " + lastN;
            Team=myTeam;
            Position = pos;
            switch (myTeam)
            {
                case "Cardinals":
                    Logo = "ms-appx:///Images/cardinals.png";
                    break;
                case "Cubs":
                    Logo = "ms-appx:///Images/cubs.png";
                    break;
                case "Giants":
                    Logo = "ms-appx:///Images/giants.png";
                    break;
                case "Pirates":
                    Logo = "ms-appx:///Images/pirates.png";
                    break;
                case "Reds":
                    Logo = "ms-appx:///Images/reds.png";
                    break;
                case "Tigers":
                    Logo = "ms-appx:///Images/tigers.png";
                    break;
                default:
                    Logo = "ms-appx:///Images/freeAgent.png";
                    break;
            }
        }

        // Override the ToString method.
        public override string ToString()
        {
            return FirstName + " " + LastName + " plays " + Position + " for the " +
                     Team;
        }
    }
}
```

Items can respond to a mouse click or finger tab. Set the *IsItemClickEnabled* property to *True* and assign an event handler to handle the *ItemClicked* event.

XAML (Modifications highlighted)

```
      <GridView x:Name="gvRoster" Grid.Row="1" Grid.Column="1"
                ItemTemplate="{StaticResource DataTemplate1}" ItemClick="PlayerChosen"
                IsItemClickEnabled="True" >
```

C# Code (Event procedure added to display a message box)

```
        private async void PlayerChosen(object sender, ItemClickEventArgs e)
        {
            // add directive: using Windows.UI.Popups
            string myString = e.ClickedItem.ToString();
            MessageDialog md = new MessageDialog(myString, "Details");
            await md.ShowAsync();
        }
```

This displays a message dialog for whichever item was clicked. Note in the code that the passed variable e contains the data of the clicked item. The data is of type Player and thus e.ClickedItem.ToString( ) calls the ToString() method of the Player.cs class.

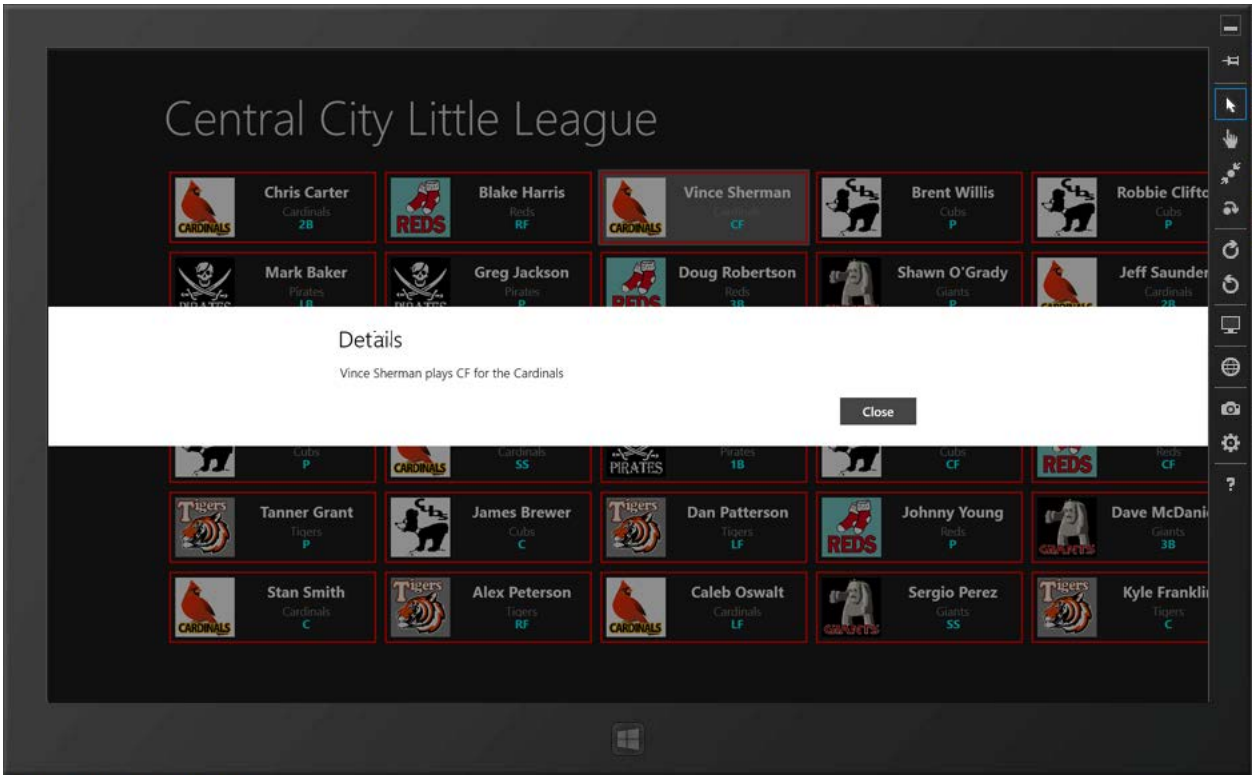

Figure 13 *– The ItemClicked event of the GridView is easy to handle, but remember to set the IsItemClickEnabled property off the GridView to True.*

You can take this a step further by having the ItemClicked event navigate to a second, more-detailed page.

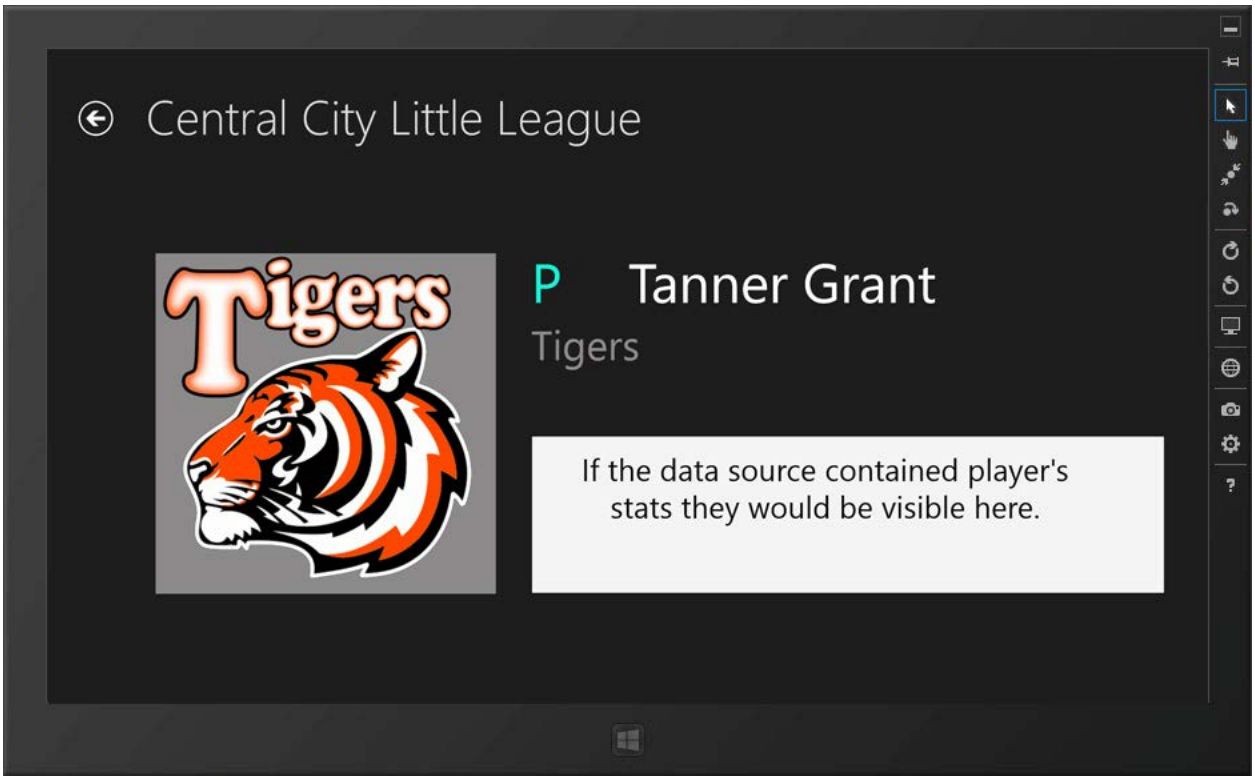

Figure 14 *– A BasicPage object was added to the project to provide details of the player clicked in the GridView of the MainPage.*

First, add a new Blank Page object to the project, naming it PlayerDetail.xaml.

Change the C# code for the MainPage.xaml.cs file to comment out the Message Dialog code while adding a statement to navigate to the new PlayerDetail page. You pass the ClickedItem information to the new page as a parameter.

C# Code:

```
        private async void PlayerChosen(object sender, ItemClickEventArgs e)
        {
            // add directive: using Windows.UI.Popups
            /* string myString = e.ClickedItem.ToString();
            MessageDialog md = new MessageDialog(myString, "Details");
            await md.ShowAsync(); */
            this.Frame.Navigate(typeof(PlayerDetail), e.ClickedItem);
        }
```

Build the XAML for the PlayerDetail page. The DataContext for the page will be set in the C# or VB code-behind for the page, referencing the Player class data object. Thus, the Text property of the three TextBlocks displaying the player's position, name, and team—as well as the Logo image string—can be bound just as the data was on the MainPage. In essence, the data object of the player's information is being passed from the first page (MainPage) to this page (PlayerDetail).

XAML Code:

```
<common:LayoutAwarePage
    x:Name="pageRoot"
    x:Class="_05_GridView_Demo__CS_.PlayerDetail"
```

```
    DataContext="{Binding DefaultViewModel, RelativeSource={RelativeSource Self}}"
    xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
    xmlns:local="using:_05_GridView_Demo__CS_"
    xmlns:common="using:_05_GridView_Demo__CS_.Common"
    xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
    xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
    mc:Ignorable="d">

    <Page.Resources>
             <x:String x:Key="AppName">Central City Little League</x:String>
    </Page.Resources>


    <Grid Style="{StaticResource LayoutRootStyle}">
        <Grid.RowDefinitions>
            <RowDefinition Height="140"/>
            <RowDefinition Height="*"/>
        </Grid.RowDefinitions>

        <!-- Back button and page title -->
        <Grid>
            <Grid.ColumnDefinitions>
                <ColumnDefinition Width="Auto"/>
                <ColumnDefinition Width="*"/>
            </Grid.ColumnDefinitions>
            <Button x:Name="backButton" Click="GoBack"
                    IsEnabled="{Binding Frame.CanGoBack, ElementName=pageRoot}"
                    Style="{StaticResource BackButtonStyle}"/>
            <TextBlock x:Name="pageTitle" Grid.Column="1" Text="{StaticResource AppName}"
                    Style="{StaticResource PageHeaderTextStyle}"
        </Grid>
        <Image x:Name ="imgTeamLogo" Grid.Row="1" HorizontalAlignment="Left" Height="400"
                 Margin="129,100,0,0" VerticalAlignment="Top" Width="400"
                 Source="{Binding Logo}"/>
        <TextBlock x:Name="txtDetailName" Grid.Row="1" HorizontalAlignment="Left"
                   Height="78" Margin="685,100,0,0" Text="{Binding FullName}"
                   VerticalAlignment="Top" Width="651" FontFamily="Segoe UI Semibold"
                   FontSize="64"/>
        <TextBlock x:Name="txtDetailTeam" Grid.Row="1" HorizontalAlignment="Left"
                   Height="57" Margin="571,183,0,0" FontFamily="Segoe UI Semibold"
                   FontSize="48" Text="{Binding Team}" VerticalAlignment="Top"
                   Width="720" Foreground="#FF918C8C"/>
        <TextBlock x:Name="txtDetailPosition" Grid.Row="1" Text="{Binding Position}"
                   HorizontalAlignment="Left" FontFamily="Segoe UI Semibold"
                   FontSize="64"  Height="78" Margin="571,100,0,0"
                   VerticalAlignment="Top"  Width="95" Foreground="#FF13F7DA"/>
        <Rectangle Grid.Row="1" Fill="#FFF4F4F5" HorizontalAlignment="Left" Height="186"
                   Margin="571,314,0,0"  Stroke="Black" VerticalAlignment="Top"
                   Width="745"/>

        <TextBlock HorizontalAlignment="Left" Height="116" Margin="609,336,0,0"
                   Grid.Row="1"TextWrapping="Wrap" Text="If the data source contained
                   player's stats they would be visible here."
                   VerticalAlignment="Top" Width="614" Foreground="#FF0C0D0C"
                   FontSize="36" TextAlignment="Center"/>
```

```
        <VisualStateManager.VisualStateGroups>

        //  . . . . . code removed in this listing to save space

        </VisualStateManager.VisualStateGroups>
    </Grid>
</common:LayoutAwarePage>
```

The C# code-behind for the PlayerDetail.xaml page handles the OnNavigatedTo event with an overridden method that sets the DataContext property for the page to the passed data object and then routes the event to the base method.

C# Code:

```
using System;
using System.Collections.Generic;
using System.IO;
using System.Linq;
using Windows.Foundation;
using Windows.Foundation.Collections;
using Windows.UI.Xaml;
using Windows.UI.Xaml.Controls;
using Windows.UI.Xaml.Controls.Primitives;
using Windows.UI.Xaml.Data;
using Windows.UI.Xaml.Input;
using Windows.UI.Xaml.Media;
using Windows.UI.Xaml.Navigation;


namespace _05_GridView_Demo__CS_
{

    public sealed partial class PlayerDetail :
                                _05_GridView_Demo__CS_.Common.LayoutAwarePage
    {
        public PlayerDetail()
        {
            this.InitializeComponent();
        }

        protected override void LoadState(Object navigationParameter, Dictionary<String,
                                                                       Object> pageState)
        {
        }

        protected override void SaveState(Dictionary<String, Object> pageState)
        {
        }

        protected override void OnNavigatedTo(NavigationEventArgs e)
        {
            this.DataContext = e.Parameter;
            base.OnNavigatedTo(e);
        }
    }
}
```

Note that the backButton is fully functional and returns the user to the main page. It is coded in the LayoutAwarePage.cs code found in the Common folder, in the collapsed region for Navigation Support.

Using LINQ to Manipulate the Data

LINQ (Language Integrated Query) is a powerful and simple query language developed by Microsoft. It is similar to SQL (Structured Query Language) and is simple to use.

Using Language Integrated Query (LINQ)

LINQ is easy to use for a variety of data sources, including collections such as the ObservableCollection used in the previous project. Consult a C# or VB reference book, the Microsoft Developer Network (MSDN), or the Internet for more information on LINQ.

You can use LINQ to alphabetize your players by their last names. Make the following modifications to the C# code-behind for MainPage.xaml.cs:

C# Code for for MainPage.xaml.cs:

```
        protected override void OnNavigatedTo(NavigationEventArgs e)
        {
            // Add items to the collection.
            Rosters.Add(new Player("Carter","Chris","Cardinals","2B"));
              // . . .  code removed in this listing to save space
            Rosters.Add(new Player("Baker", "Edgar", "Cardinals", "3B"));
            Rosters.Add(new Player("Vasquez", "Jose", "Giants", "RF"));

           //Use LINQ to query the collection and create sub collections
           var sortedNames = from value in Rosters orderby value.LastName select value;
           gvRoster.ItemsSource = sortedNames;
        }
```

Run the project again and the names should now appear sorted.

Figure 15 *– LINQ (Language Integrated Query) is a quick, simple, and powerful way to sort data. In this example, the players have been alphabetized by last name.*

Likewise, to sort players alphabetically by their last names but to group teams together, the LINQ statement could be changed to:

```
var sortedNames = from value in Rosters orderby value.Team + value.LastName +
value.FirstName select value;
```

Figure 16 *– Multiple fields can be concatenated to produce primary, secondary, and tertiary sorts. In this example, the data is sorted primarily by team, secondarily by the player's last name, and thirdly by the first name.*

LINQ statements can also filter data to create a subset of data. You can create multiple LINQ statements to generate subsets for each team's rosters.

```
    var cardsRoster = from value in Rosters where value.Team.Equals("Cardinals")
                      orderby value.LastName select value;
    var cubsRoster = from value in Rosters where value.Team.Equals("Cubs")
                     orderby value.LastName select value;
    var giantsRoster = from value in Rosters where value.Team.Equals("Giants")
                       orderby value.LastName select value;
    var piratesRoster = from value in Rosters where value.Team.Equals("Pirates")
                        orderby value.LastName select value;
    var redsRoster = from value in Rosters where value.Team.Equals("Reds")
                     orderby value.LastName select value;
    var tigersRoster = from value in Rosters where value.Team.Equals("Tigers")
                       orderby value.LastName select value;

    gvRoster.ItemsSource = cardsNames; //change the source here to view other teams
    /* In the next section we will add an App Bar to view the different teams,
       But you could also add a ListView or ComboBox to the page to accomplish the same
       thing */
```

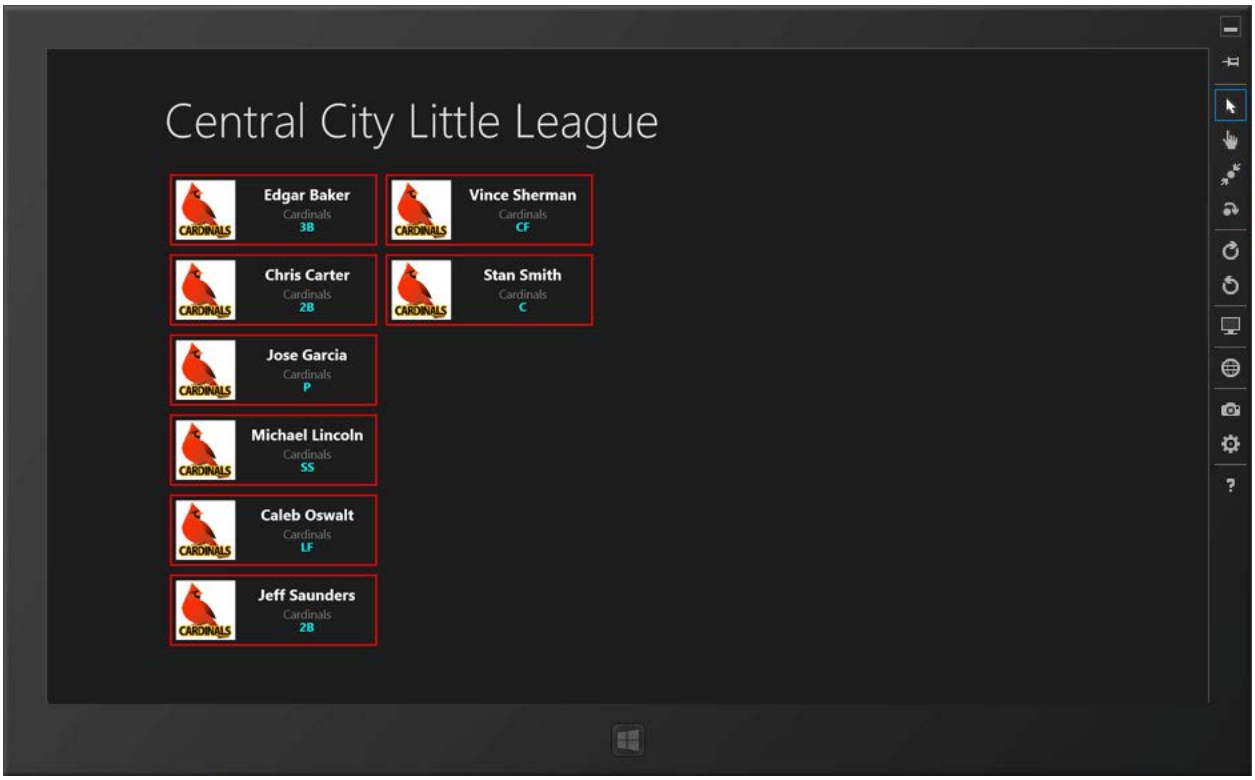

Figure 17 *– LINQ statements can query a data collection to create subsets of the data.*

Adding an App Bar

The App Bar provides users with additional commands (just like menu bars used to do in the past). They difference is that the App Bar commands are more streamlined. On a touch device, the user swipes up from the bottom with his or her finger or stylus. The App Bar(s) can be accessed using the following methods:

- With a mouse by right-clicking
- With the keyboard using the Windows key and the letter Z
- With the keyboard using the Shift key and the F10 function key

App Bars are tailored to the app and can be context-sensitive. They may be placed on the bottom and/or top of the app. This is in contrast to the Charms Bar, which is universal in all apps and settings and always contains the same five buttons. However, the content in the Charms bar may be modified for a specific app, such as placing settings information for the app in the Settings area of the Charms Bar.

Both the App Bar and Charms bar are designed to maintain a clean appearance to the app's environment and provide maximum real-estate where users can use the full screen for content. This is especially important for the user experience on lower-resolution devices.

App Bar Guidelines

App Bar buttons should not contain content that are critical to the mission of the app, such as performing some calculation. Such control should exist on the page itself.

App Bars should follow Windows 8 conventions, such as:

- Top App Bars are predominantly used for navigation of the app. Bottom App Bars are mostly used for issuing commands such as sorting or filtering data.
- Navigation and view-switching buttons should be grouped together on the far left.
- Commands that are not context-related or are persistent should be on the right.
- Commands to create or add content should be on the farthest right with commands to delete, clear, or remove content to the immediate left of the Add command.
- If providing user response buttons, similar to the desktop MessageBox, the options for Accept, Yes, or OK should be to the left of the options for Reject, No, or Cancel.
- Utilize the far left and far right first in design since most touch device users will select with their thumbs while holding the device.
- Group similar commands with flyouts and menus.
- Consider Snapped and Filled views as well as Portrait rotations.

For additional information, refer to the "Guidelines for app bars (Windows Store apps)" from the Windows Dev Center – Windows Store Apps website for best practices.

Most App Bars utilize round glyph buttons. There are over 150 glyph buttons already styled for developers to use. Exploring the StandardStyles.xaml file reveals that most of these are commented out. To use any of these standard glyphs (and you are greatly encouraged to do so), simply uncomment the style in the StandardStyles document and set the button's Style attribute in the page's xaml code to use that particular static resource.

Examples of round glyph buttons include the followiong:
- SaveAppBarButtonStyle
- AddAppBarButtonStyle
- RemoveAppBarButtonStyle
- SearchAppBarButtonStyle
- FilterAppBarButtonStyle
- SortAppBarButtonStyle
- HelpAppBarButtonStyle
- PlayAppBarButtonStyle
- PauseAppBarButtonStyle
- NextAppBarButtonStyle
- PreviousAppBarButtonStye

The Segoe UI Symbol font (or other symbol fonts) is a great place to find icons for custom buttons.

Now, you can add an App Bar to the previous Little League rosters project. An App Bar is added as a XAML control. In this example, a BottomAppBar is added to the page, but we could add a TopAppBar in the same way. The App Bar control here is divided into three columns using a Grid. Horizontal StackPanels are used in the first and last columns to contain the button controls. Each button uses one of the StandardStyles that have been uncommented, although the page XAML overrides the displayed name for each button.

The Tapped event is handled for each button. You can use LINQ to sort and filter the data within the various button event handlers. A MessageDialog was utilized to provide the contact info for the far-right button.

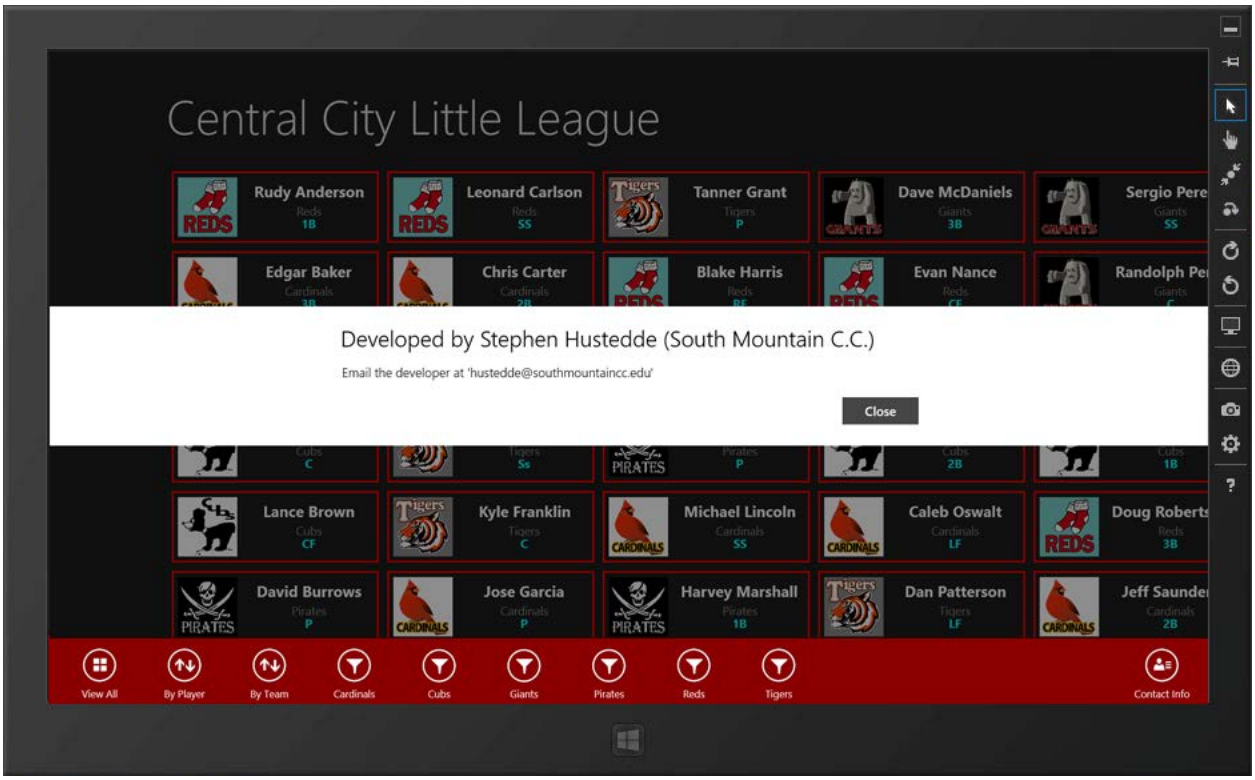

Figure 18 *– The Contact Info button at the far right of the App Bar displays a 'Developed by…' MessageDialog.*

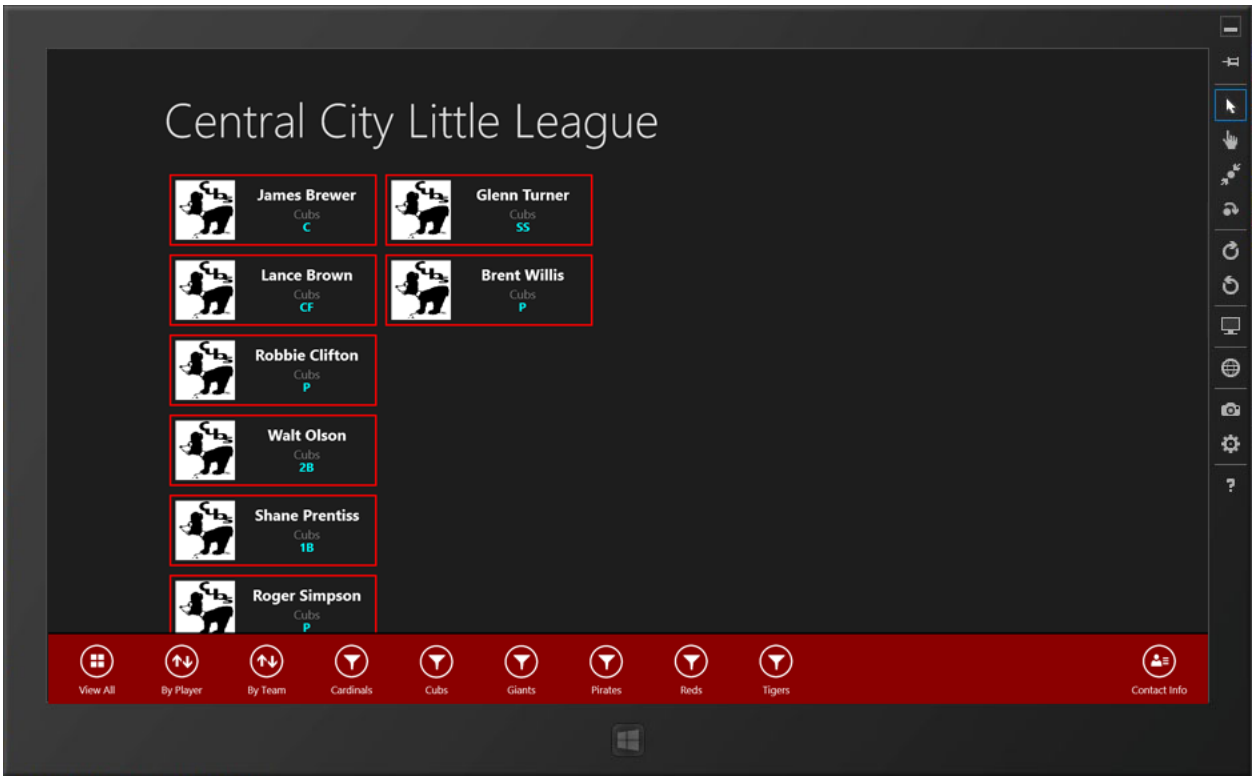

Figure 19 *– Sorting and Filtering operations are provided via an App Bar.*

StandardStyles.xaml (Added styles were copied and pasted from the commented code)

```
    <Style x:Key="ContactInfoAppBarButtonStyle" TargetType="ButtonBase"
        BasedOn="{StaticResource AppBarButtonStyle}">
        <Setter Property="AutomationProperties.AutomationId"
                  Value="ContactInfoAppBarButton"/>
        <Setter Property="AutomationProperties.Name" Value="Contact Info"/>
        <Setter Property="Content" Value="&#xE136;"/>
    </Style>
    <Style x:Key="FilterAppBarButtonStyle" TargetType="ButtonBase"
        BasedOn="{StaticResource AppBarButtonStyle}">
        <Setter Property="AutomationProperties.AutomationId" Value="FilterAppBarButton"/>
        <Setter Property="AutomationProperties.Name" Value="Filter"/>
        <Setter Property="Content" Value="&#xE16E;"/>
    </Style>
    <Style x:Key="SortAppBarButtonStyle" TargetType="ButtonBase"
        BasedOn="{StaticResource AppBarButtonStyle}">
        <Setter Property="AutomationProperties.AutomationId" Value="SortAppBarButton"/>
        <Setter Property="AutomationProperties.Name" Value="Sort"/>
        <Setter Property="Content" Value="&#xE174;"/>
    </Style>
    <Style x:Key="ViewAllAppBarButtonStyle" TargetType="ButtonBase"
        BasedOn="{StaticResource AppBarButtonStyle}">
        <Setter Property="AutomationProperties.AutomationId"
                      Value="ViewAllAppBarButton"/>
        <Setter Property="AutomationProperties.Name" Value="View All"/>
        <Setter Property="Content" Value="&#xE138;"/>
    </Style>
```

XAML (MainPage XAML code added for the App Bar)

```
    <Page.BottomAppBar>
        <AppBar Background="DarkRed">
            <Grid>
                <Grid.ColumnDefinitions>
                    <ColumnDefinition Width="Auto"/>
                    <ColumnDefinition Width="Auto"/>
                    <ColumnDefinition Width="*"/>
                </Grid.ColumnDefinitions>
                <StackPanel Orientation="Horizontal">
                    <Button Style="{StaticResource ViewAllAppBarButtonStyle}"
                            Tapped="ShowAllPlayers" />
                    <Button Style="{StaticResource SortAppBarButtonStyle}"
                            AutomationProperties.Name="By Player" Tapped="SortPlayers"/>
                    <Button Style="{StaticResource SortAppBarButtonStyle}"
                            AutomationProperties.Name="By Team" Tapped="SortTeams"/>
                    <Button Style="{StaticResource FilterAppBarButtonStyle}"
                            AutomationProperties.Name="Cardinals"
                            Tapped="FilterCardinals" />
                    <Button Style="{StaticResource FilterAppBarButtonStyle}"
                            AutomationProperties.Name="Cubs" Tapped="FilterCubs"/>
                    <Button Style="{StaticResource FilterAppBarButtonStyle}"
                            AutomationProperties.Name="Giants" Tapped="FilterGiants"/>
                    <Button Style="{StaticResource FilterAppBarButtonStyle}"
                            AutomationProperties.Name="Pirates" Tapped="FilterPirates"/>
                    <Button Style="{StaticResource FilterAppBarButtonStyle}"
                            AutomationProperties.Name="Reds" Tapped="FilterReds"/>
                    <Button Style="{StaticResource FilterAppBarButtonStyle}"
```

```
                        AutomationProperties.Name="Tigers" Tapped="FilterTigers"/>
            </StackPanel>
            <StackPanel Orientation="Horizontal" Grid.Column="2"
                        HorizontalAlignment="Right">
                <Button Style="{StaticResource ContactInfoAppBarButtonStyle}"
                        Tapped="ShowInfo"  />
            </StackPanel>
        </Grid>
    </AppBar>
</Page.BottomAppBar>
```

C# Code (The added event handler methods for the App Bar buttons)

```
private async void ShowInfo(object sender, TappedRoutedEventArgs e)
{
    string myString = "Email the developer at 'hustedde@southmountaincc.edu'";
    MessageDialog md = new MessageDialog(myString, "Developed by Stephen Hustedde
                (South Mountain C.C.)");
    await md.ShowAsync();
}

private void ShowAllPlayers(object sender, TappedRoutedEventArgs e)
{
    gvRoster.ItemsSource = Rosters;
}

private void SortPlayers(object sender, TappedRoutedEventArgs e)
{
    var sortedNames = from value in Rosters orderby value.LastName +
                            value.FirstName select value;
    gvRoster.ItemsSource = sortedNames;
}

private void SortTeams(object sender, TappedRoutedEventArgs e)
{
    var teamGroups = from value in Rosters orderby value.Team +
                           value.LastName + value.FirstName select value;
    gvRoster.ItemsSource = teamGroups;
}

private void FilterCardinals(object sender, TappedRoutedEventArgs e)
{
    var cardsRoster = from value in Rosters
                      where value.Team.Equals("Cardinals")
                      orderby value.LastName
                      select value;
    gvRoster.ItemsSource = cardsRoster;
}

private void FilterCubs(object sender, TappedRoutedEventArgs e)
{
    var cubsRoster = from value in Rosters
                     where value.Team.Equals("Cubs")
                     orderby value.LastName
                     select value;
    gvRoster.ItemsSource = cubsRoster;
}
```

```
        private void FilterGiants(object sender, TappedRoutedEventArgs e)
        {
            var giantsRoster = from value in Rosters
                               where value.Team.Equals("Giants")
                               orderby value.LastName
                               select value;
            gvRoster.ItemsSource = giantsRoster;
        }

        private void FilterPirates(object sender, TappedRoutedEventArgs e)
        {
            var piratesRoster = from value in Rosters
                               where value.Team.Equals("Pirates")
                               orderby value.LastName
                               select value;
            gvRoster.ItemsSource = piratesRoster;
        }

        private void FilterReds(object sender, TappedRoutedEventArgs e)
        {
            var redsRoster = from value in Rosters
                               where value.Team.Equals("Reds")
                               orderby value.LastName
                               select value;
            gvRoster.ItemsSource = redsRoster;
        }

        private void FilterTigers(object sender, TappedRoutedEventArgs e)
        {
            var tigersRoster = from value in Rosters
                               where value.Team.Equals("Tigers")
                               orderby value.LastName
                               select value;
            gvRoster.ItemsSource = tigersRoster;
        }
```