Layout on Large Screens

The **Nokia Lumia 1520** brings a new dimension into design of Windows Phone 8 applications. Large screens deserve dedicated design to display content in an even more beautiful way.

Note: This is an entry in the Nokia Imaging and Big UI Wiki Competition 2013Q4.

How screens are growing

The original Windows Phone 7 lineup offered just one single **WVGA resolution – 480x800 pixels**. This resultion was carefully chosen as the optimal for all 3.7 to 4.3 inch devices that were on market.

Windows Phone 8

When new Windows Phone 8 devices were introduced, the offering was expanded to allow for **WXGA (768x1280)** and **720p (1280x720) resolutions**. The great thing about the changes was, that all older apps were completely compatible with the new devices thanks to OS scaling. Windows Phone design is mostly vector based, so scaling elements up to the newer screen sizes didn't introduce any loss of quality in most apps. There was still a reason to release a new version of apps specifically targeted on Windows Phone 8 – **aspect ratio**.

As we can see, the original WVGA screens had the ratio of **15:9** which is exactly the same as the ratio of WXGA, but differs from 720p displays, whose ratio is **16:9**. The base resolution that is used to scale up apps for 720p is **not 480x800, but 480x853**. Because of this original Windows Phone 7 apps are displayed with **letterboxing black edge** on top that helps to hide the difference.

GDR3

As the market develops today, we start to see yet another trend in smartphone devices, which are usually called **phablets**. These devices offer the mobile capabilities of a smartphone, while offering larger screens, that are almost in the range of smallest tablets and hence offer great mobility, productivity and readability. As the first phablet member of the Lumia familiy, Nokia now introduced the latest and greatest **Nokia Lumia 1520**, that offers a beautiful, **stunning 6 inch display with crisp Full HD resolution – 1920x1080 pixels**. This new resolution is enabled in the upcoming update to Windows Phone 8 OS, GDR3.

Once again FullHD resolution has the aspect ratio of 16:9 and so the elements on the screen are vector-scaled up from **480x853 resolution**. This works exactly the same way as for 720p displays. But is simple scaling **really enough** for the large displays as the one Nokia Lumia 1520 offers?

Thinking big

The Nokia Lumia 1520 has an amazing 6 inch screen. If we compare this with a 3.7 inch Windows Phone 7 device – for example Nokia Lumia 800, we can truly see the amazing opportunity we are being offered – **the psychical area is 2.5 larger**!

By scaling apps just as they are to the larger size, we would end up with apps that **work perfectly**, but could be **more visually compelling** when designed to take advantage of all the screen estate they are show on. And that is the purpose of this article.

Display properties and details

To make our code simpler and more reusable, we will build a simple static helper class that will quickly return all important device information.

```
    public enum DisplayResolution
    {
        WVGA,
        WXGA,
        HD720p,
        HD1080p
    }

    public static class DeviceScreenHelper
    {
        static DeviceScreenHelper()
        {
            object data = null;
            if ( DeviceExtendedProperties.TryGetValue( "PhysicalScreenResolution", out
data ) )
            {
```


(C) Copyright Nokia 2014. All rights reserved.

```
                physicalScreenSize = data as Size?;
            }
            if ( DeviceExtendedProperties.TryGetValue( "RawDpiX", out data ) )
            {
                dpi = data as double?;
            }
            if ( dpi != null && physicalScreenSize != null )
            {
                isGDR3 = true;
            }
        }

        public static double BaseWidth
        {
            get
            {
                return App.Current.Host.Content.ActualWidth;
            }
        }

        public static double BaseHeight
        {
            get
            {
                return App.Current.Host.Content.ActualHeight;
            }
        }


        private static Size? physicalScreenSize = null;
        private static double? dpi = null;
        private static bool isGDR3 = false;
        private static double scaleFactor = App.Current.Host.Content.ScaleFactor /
100.0;

        public static double ScaleFactor
        {
            get
            {
                return scaleFactor;
            }
        }

        public static double RawScaleFactor
        {
            get
            {
                if ( !isGDR3 )
                {
                    return ScaleFactor;
                }
                else
                {
                    return PhysicalResolutionWidth / BaseWidth;
                }
            }
        }

        public static bool IsGDR3
        {
            get
            {
                return isGDR3;
```


(C) Copyright Nokia 2014. All rights reserved.

```
        }
    }

    public static double PhysicalResolutionWidth
    {
        get
        {
            if ( !isGDR3 )
            {
                return BaseWidth * ScaleFactor;
            }
            else
            {
                return physicalScreenSize.Value.Width;
            }
        }
    }

    public static double PhysicalResolutionHeight
    {
        get
        {
            if ( !isGDR3 )
            {
                return BaseHeight * ScaleFactor;
            }
            else
            {
                return physicalScreenSize.Value.Height;
            }
        }
    }

    public static double? PhysicalScreenDiagonal
    {
        get
        {
            if ( isGDR3 )
            {
                return Math.Sqrt(
                    Math.Pow( physicalScreenSize.Value.Width / dpi.Value, 2.0 ) +
                    Math.Pow( physicalScreenSize.Value.Height / dpi.Value, 2.0 ) );
            }
            else
            {
                return null;
            }
        }
    }

    public static DisplayResolution ResolutionType
    {
        get
        {
            if ( PhysicalResolutionWidth > 1000 )
            {
                return DisplayResolution.HD1080p;
            }
            else if ( PhysicalResolutionWidth > 760 )
            {
                return DisplayResolution.WXGA;
            }
            else if ( PhysicalResolutionWidth > 700 )
```


(C) Copyright Nokia 2014. All rights reserved.

```
                {
                    return DisplayResolution.HD720p;
                }
                else
                {
                    return DisplayResolution.WVGA;
                }
            }
        }
    }
```

Making things crisp on 1080p

Before we tackle the question of grid layouts and adapting to large diagonals, we should focus first on adapting assets to the FullHD resolution.

Local assets

Bitmap image assets on 1080p deserve a new level of clarity – all assets are **scaled by 2.25** from the base resolution. To achieve beautiful results, you should offer the system the right images for each occassion.

The easiest way to differentiate between different screen resolutions is to add a special suffix to each image asset's filename. For example – instead of one Eample.jpg file we will have these:

- Example.screen-wvga.jpg
- Example.screen-wxga.jpg
- Example.screen-720p.jpg
- Example.screen-1080p.jpg

Note: Adding four instances of each image adds additional data to the **app's package size**. To make the demands on phone storage and app's download size smaller, you can offer just the 1080p asset, and the other sizes will automatically downscale. This is a reasonable compromise, but of course it is best to offer dedicated resources for each resolution, especially when we are talking about important in-app elements, that are heavily visible on many screens (e.g. the app's logo, appbar icons).

We can create a simple helper method that will return a specific asset name, when we supply the file name and extension.

```
    public static class AssetFileNameHelper
    {
        /// <summary>
        /// Transforms a simple asset file name to resolution specific
        /// </summary>
        /// <param name="baseName">The main part of the file name (e.g.
'example')</param>
        /// <param name="extension">The file extension (e.g. 'jpg')</param>
        /// <returns>Resolution specific file name (e.g. 'example.screen-
1080p.jpg')</returns>
        public static string GetName( string baseName, string extension )
        {
            string dotExtension = extension.Length > 0 ? "." + extension : "";
            string displayTypePart = "";
            switch ( DeviceScreenHelper.ResolutionType )
            {
                case DisplayResolution.HD1080p:
                    {
                        displayTypePart = "1080p";
                        break;
                    }
                case DisplayResolution.HD720p:
                    {
                        displayTypePart = "720p";
                        break;
                    }
                case DisplayResolution.WXGA:
                    {
                        displayTypePart = "wxga";
```


(C) Copyright Nokia 2014. All rights reserved.

```
                        break;
                    }
                default:
                    {
                        displayTypePart = "wvga";
                        break;
                    }
            }
            return string.Format( "{0}.screen-{1}{2}", baseName, displayTypePart,
dotExtension );
        }
    }
```

Remote assets

The same method we used for local assets can be used for querying differentely sized remote assets. Alternatively, if you generate the images on server side, the device can include it's resolution in a **query parameter**.

The only difference with remote assets comes with the fact, that users can rely on **limited data connection** and also the speed of data transfer can be very low on some celluar data connections.

Because the number of pixels the 1080p screens display is more than **five times bigger** than that of WVGA screens and therefore the size difference of assets here is quite significant. To offer the best experience, you should offer the user a choice of which image sizes the app should use for both celluar connections and Wi-Fi connections. This way the **user can decide** herself which compromise between speed and image quality is the most suitable.

The large diagonal

As we discovered in the introduction, it is useful to optimize the apps according to the physical screen diagonal and resolution of the display.

Dynamic layout

Nokia Lumia Developer Library offers a great starting point for **dynamic layout** in dedicated developer resource article - Dynamic Layout. This sample code will help you create **different XAML style dictionaries** based on the size of the screen. In most cases, you will create just a normal and big screen layout.

As you can see in the sample, XAML is a powerful tool that can help you create dynamic and original layout, that can at the same time be updated and improved by adding more columns for the large screen. This is a nice way to start, but we can go even further by **mimicking the Start Screen** approach and creating a beautiful and variable design for grid displays.

The Tile Canvas control

Adding more columns in a grid is certainly a way to adjust for large screens, but we can add a point of variability by introducing **different layout of items** inside.

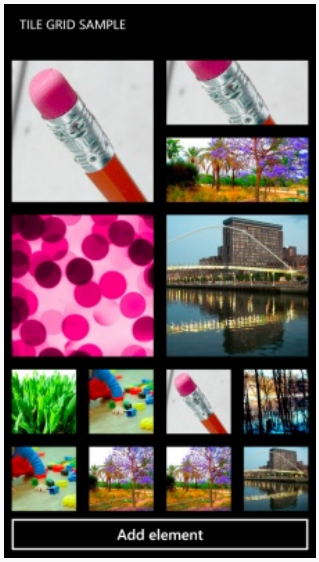
In this sample we will create a simple control, derived from a canvas, that will act as a variable design grid. This control will automatically **arrange the controls inside** it so that the result looks very compelling and natural on larger screens and also you will find that the result is very similar to the Windows Phone Start Screen design (although we also add a large square tile, similarly to the Windows 8.1 Start Screen). This will work best for **image galleries**, where all images have stretching set to UniformToFill to make them fit the resulting tiles. The resulting control can be easily combined with the Dynamic Layout sample to make sure larger screens display more columns than smaller ones.

Here are some sample screenshots with three, two and four unit columns. More columns are suitable more for larger screens, less for smaller ones. Combining TileCanvas control with the Dynamic Layout sample you will obtain the optimal experience for each device.

(C) Copyright Nokia 2014. All rights reserved.

Three unit columns

Two unit columns



Four unit columns

Code

```
/// <summary>
/// Represents a "block" with certain layout in the TileCanvas
/// </summary>
public class LayoutDesign
{
    public int UnitColumnWidth;
    public LayoutDesignTile[] TileConfigurations;
}

/// <summary>
/// Individual tile in a layout design
/// </summary>
public struct LayoutDesignTile
{
    public int UnitLeft;
    public int UnitTop;
    public int UnitWidth;
    public int UnitHeight;
}
```


(C) Copyright Nokia 2014. All rights reserved.

```
    /// <summary>
    /// TileCanvas control for visually compelling grid layout
    /// </summary>
    public class TileCanvas : Canvas
    {
        /// <summary>
        /// Some predefined layout designs, can be expanded with more layouts
        /// </summary>
        private static readonly LayoutDesign[] PREDEFINED_DESIGNS = new LayoutDesign[]{
            new LayoutDesign(){ //Two unit size squares on top of each other
                UnitColumnWidth = 1,
                TileConfigurations = new LayoutDesignTile[]{
                    new LayoutDesignTile(){
                        UnitLeft=0,
                        UnitTop = 0,
                        UnitHeight = 1,
                        UnitWidth = 1 },
                    new LayoutDesignTile(){
                        UnitLeft = 0,
                        UnitTop = 1,
                        UnitHeight = 1,
                        UnitWidth = 1
                    } }
            },
            new LayoutDesign(){ //Unit high wide tiles on top of each other
                UnitColumnWidth = 2,
                TileConfigurations = new LayoutDesignTile[]{
                    new LayoutDesignTile(){
                        UnitLeft=0,
                        UnitTop = 0,
                        UnitHeight = 1,
                        UnitWidth = 2 },
                    new LayoutDesignTile(){
                        UnitLeft = 0,
                        UnitTop = 1,
                        UnitHeight = 1,
                        UnitWidth = 2
                    } }
            },
            new LayoutDesign(){ //One wide tile and two unit tiles below
                UnitColumnWidth = 2,
                TileConfigurations = new LayoutDesignTile[]{
                    new LayoutDesignTile(){
                        UnitLeft=0,
                        UnitTop = 0,
                        UnitHeight = 1,
                        UnitWidth = 2 },
                    new LayoutDesignTile(){
                        UnitLeft = 0,
                        UnitTop = 1,
                        UnitHeight = 1,
                        UnitWidth = 1
                    },
                    new LayoutDesignTile(){
                        UnitLeft = 1,
                        UnitTop = 1,
                        UnitHeight = 1,
                        UnitWidth = 1
                    }
                }
            },
            new LayoutDesign(){ //Two unit tiles and one wide tile below
                UnitColumnWidth = 2,
```


(C) Copyright Nokia 2014. All rights reserved.

```
                TileConfigurations = new LayoutDesignTile[]{
                    new LayoutDesignTile(){
                        UnitLeft=0,
                        UnitTop = 0,
                        UnitHeight = 1,
                        UnitWidth = 1 },
                    new LayoutDesignTile(){
                        UnitLeft=1,
                        UnitTop = 0,
                        UnitHeight = 1,
                        UnitWidth = 1 },
                    new LayoutDesignTile(){
                        UnitLeft = 0,
                        UnitTop = 1,
                        UnitHeight = 1,
                        UnitWidth = 2
                    } }
            },
            new LayoutDesign(){ //One large square tile
                UnitColumnWidth = 2,
                TileConfigurations = new LayoutDesignTile[]{
                    new LayoutDesignTile(){
                        UnitLeft=0,
                        UnitTop = 0,
                        UnitHeight = 2,
                        UnitWidth = 2 }
                }
            }
        };

        /// <summary>
        /// Inner margin between the tiles in TileCanvas
        /// </summary>
        public double InnerMargin
        {
            get { return ( double )GetValue( InnerMarginProperty ); }
            set { SetValue( InnerMarginProperty, value ); }
        }

        // Using a DependencyProperty as the backing store for InnerMargin.  This
enables animation, styling, binding, etc...
        public static readonly DependencyProperty InnerMarginProperty =
            DependencyProperty.Register( "InnerMargin", typeof( double ), typeof(
TileCanvas ), new PropertyMetadata( 0.0, MarginChanged ) );

        /// <summary>
        /// Respond to the change of inner margins by recalculating the tile positions
and sizes
        /// </summary>
        /// <param name="d"></param>
        /// <param name="e"></param>
        public static void MarginChanged( DependencyObject d,
DependencyPropertyChangedEventArgs e )
        {
            ( ( TileCanvas )d ).PositionChildren( false );
        }

        /// <summary>
        /// Number of unit columns in the TileCanvas
        /// </summary>
        public int ColumnCount
        {
            get { return ( int )GetValue( ColumnCountProperty ); }
```


(C) Copyright Nokia 2014. All rights reserved.

```
            set { SetValue( ColumnCountProperty, value ); }
        }

        public static readonly DependencyProperty ColumnCountProperty =
            DependencyProperty.Register( "ColumnCount", typeof( int ), typeof(
TileCanvas ), new PropertyMetadata( 1, ColumnCountChanged ) );

        /// <summary>
        /// Respond to the change of column count by recalculating the tile positions
and sizes, this requires generating new layouts
        /// </summary>
        /// <param name="d"></param>
        /// <param name="e"></param>
        public static void ColumnCountChanged( DependencyObject d,
DependencyPropertyChangedEventArgs e )
        {
            ( ( TileCanvas )d ).PositionChildren( true );
        }

        public TileCanvas()
        {
            LayoutUpdated += TileCanvas_LayoutUpdated;
        }

        /// <summary>
        /// We need to update layout
        /// </summary>
        /// <param name="sender"></param>
        /// <param name="e"></param>
        void TileCanvas_LayoutUpdated( object sender, EventArgs e )
        {
            PositionChildren( false );
        }

        /// <summary>
        /// Cached list of layout designs in the TileCanvas
        /// </summary>
        private List<LayoutDesign> designList = new List<LayoutDesign>();

        private static Random random = new Random();

        /// <summary>
        /// Respositiones and resizes tiles in the TileCanvas
        /// </summary>
        /// <param name="changeColumns">True when we need to generate new list of layout
designs (breaking change in number of columns)</param>
        private void PositionChildren( bool changeColumns = false )
        {
            if ( this.ActualWidth != 0 )
            {
                //number of columns has changed, rebuild layout designs
                if ( changeColumns )
                {
                    designList.Clear();
                }

                //default sizes of tiles
                double singleSize = ( this.ActualWidth - ( ColumnCount * InnerMargin ) +
InnerMargin ) / ColumnCount;
                double doubleSize = singleSize * 2 + InnerMargin;

                //is size valid?
                if ( singleSize > 0 )
```


(C) Copyright Nokia 2014. All rights reserved.

```
                {
                    //number of tiles we need to process
                    int totalTileCountToPosition = Children.Count;
                    //number of columns to fill in the current row
                    int currentRowColumnsLeft = 0;
                    //the layout design being processed now
                    int currentLayoutDesign = 0;

                    while ( totalTileCountToPosition > 0 )
                    {
                        //are we at the end of row?
                        if ( currentRowColumnsLeft == 0 )
                        {
                            currentRowColumnsLeft = ColumnCount;
                        }
                        if ( currentLayoutDesign < designList.Count )
                        {
                            //a design is already in place
                            totalTileCountToPosition -= designList[ currentLayoutDesign
].TileConfigurations.Length;
                            currentRowColumnsLeft -= designList[ currentLayoutDesign
].UnitColumnWidth;
                        }
                        else
                        {
                            //we need to add a suitable design, precodnition -
PREDEFINED_DESIGNS contain at least one 1 unit wide layout design
                            LayoutDesign ld = (
                                           from d in PREDEFINED_DESIGNS
                                           where d.UnitColumnWidth <=
currentRowColumnsLeft
                                           orderby random.NextDouble()
                                           select d
                                       ).FirstOrDefault();
                            designList.Add( ld );
                            currentRowColumnsLeft -= designList[ currentLayoutDesign
].UnitColumnWidth;
                            totalTileCountToPosition -= designList[ currentLayoutDesign
].TileConfigurations.Length;
                        }
                        currentLayoutDesign++;
                        int designTileId = 0;
                        int designId = 0;
                        double designLeft = 0;
                        double designTop = 0;
                        int rowColumnsLeft = ColumnCount;
                        int currentRow = 0;
                        //ok, render
                        foreach ( var c in Children )
                        {
                            FrameworkElement f = c as FrameworkElement;
                            //have we already crossed the available number of tiles for
this design
                            if ( designList[ designId ].TileConfigurations.Length <=
designTileId )
                            {
                                //end of design, move on row
                                rowColumnsLeft -= designList[ designId
].UnitColumnWidth;
                                designLeft = designLeft + designList[ designId
].UnitColumnWidth * ( singleSize + InnerMargin );
                                if ( rowColumnsLeft <= 0 )
                                {
```


(C) Copyright Nokia 2014. All rights reserved.

```
                                    designLeft = 0;
                                    rowColumnsLeft = ColumnCount;
                                    currentRow++;
                                    designTop = ( currentRow * ( InnerMargin +
doubleSize ) );
                                }
                                designTileId = 0;
                                designId++;
                            }
                            //set position and size
                            f.SetValue( TileCanvas.LeftProperty,
                                    designLeft + designList[ designId
].TileConfigurations[ designTileId ].UnitLeft * ( singleSize + InnerMargin ) );
                            f.SetValue( TileCanvas.TopProperty,
                                    designTop + designList[ designId
].TileConfigurations[ designTileId ].UnitTop * ( singleSize + InnerMargin ) );
                            f.Width = ( designList[ designId ].TileConfigurations[
designTileId ].UnitWidth * ( singleSize + InnerMargin ) ) - InnerMargin;
                            f.Height = ( designList[ designId ].TileConfigurations[
designTileId ].UnitHeight * ( singleSize + InnerMargin ) ) - InnerMargin;
                            designTileId++;
                        }
                        //update control's height
                        this.Height = ( currentRow + 1 ) * ( doubleSize + InnerMargin )
- InnerMargin;
                    }
                }
            }
        }
    }
```

Explanation

LayoutDesignTile

This class represents an individual tile in the tile canvas. The properties represent the position and dimensions of the tile in it's containing design.

LayoutDesign

The *LayoutDesign* class represents a predefined layout of one **block of tiles** in the canvas. The UnitColumnWidth represents the number of columns this block covers and the *TileConfigurations* array defines all tiles in the block. Each block has a **height of 2 units**, so that we can easily add blocks on individual rows and do not have to care about the vertical space they will cover.

TileCanvas – *PREDEFINED_DESIGNS*

In this constant static array we can define any number of **different tile layouts** we want to utilize. To work properly, we need at least one one-column wide design, which is just two unit sized squares on top of each other.

For our sample, we defined five sample layouts – two unit square tiles on top of each other, two wide tiles, one large square tile and two unit tiles and wide tile (in two different layouts).

TileCanvas – *InnerMargin* property

This dependency property sets out how wide will be the **inner margins** between individual tiles. The higher the value, the smaller will be the tiles themselves.

TileCanvas – *ColumnCount* property

ColumnCount is a very important property of the TileCanvas, as it defines the total number of unit-wide columns we want to be displayed.

TileCanvas – *LayoutUpdated* event handler

Because we want our control to respond to layout changes (including different space available horizontally, changing orientation, etc.), we need to subscribe to this event. When something changes, we can recalculate the position and sizes of the tiles.

TileCanvas – *DesignList*

(C) Copyright Nokia 2014. All rights reserved.

This variable is used for caching the selected tile layout designs, so that when a redraw or recalculation is needed, we just add more designs if needed, but all the original tiles will stay on their places.

TileCanvas – *PositionChildren*

This is the heart of the whole control. We will slowly go trhough the code of this method to make sure everything is clear. First we check, if we have any space horizontally and if it even makes sense to calculate anything. If we are ready, we check if the *changeColumns* parameter is true. This parameter indicates, that the number of **columns** of the TileCanvas has **changed** and so we need to create a new list of tile design layouts, to make sure each row is fully filled and content doesn't overflow the right side. After that we calculate the **size of one unit** and for faster retrieval also size of two-unit element, Our goal now is to make sure all children can be fit in the tile layout designs defined in the *designList* variable. If they fit, we can proceed to rendering. Otherwise, we need to add as many layout designs as needed to **accomodate for all children** elements of the canvas.

We go through all children and **calculate their position and size** according to the values set in selected tile layout designs. We also keep track of our position on **current row and the row's top position**. Finally we set the height the canvas needs on the page according to the number of rows we added.

Usage

Here is a sample XAML page that contains the TileCanvas control and a button to add a new tile to it.

```
<phone:PhoneApplicationPage
    x:Class="TileCanvasSample.MainPage"
    xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
    xmlns:phone="clr-namespace:Microsoft.Phone.Controls;assembly=Microsoft.Phone"
    xmlns:shell="clr-namespace:Microsoft.Phone.Shell;assembly=Microsoft.Phone"
    xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
    xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
    mc:Ignorable="d"
    xmlns:ctls="clr-namespace:TileCanvasSample.Controls"
    FontFamily="{StaticResource PhoneFontFamilyNormal}"
    FontSize="{StaticResource PhoneFontSizeNormal}"
    Foreground="{StaticResource PhoneForegroundBrush}"
    SupportedOrientations="PortraitOrLandscape" Orientation="Portrait"
    shell:SystemTray.IsVisible="False">

    <!--LayoutRoot is the root grid where all page content is placed-->
    <Grid x:Name="LayoutRoot" Background="Transparent">
        <Grid.RowDefinitions>
            <RowDefinition Height="Auto"/>
            <RowDefinition Height="*"/>
            <RowDefinition Height="Auto"/>
        </Grid.RowDefinitions>
        <!--TitlePanel contains the name of the application and page title-->
        <StackPanel x:Name="TitlePanel" Grid.Row="0" Margin="12,17,0,28">
            <TextBlock Text="TILE GRID SAMPLE" Style="{StaticResource
PhoneTextNormalStyle}" Margin="12,0"/>
        </StackPanel>

        <!--ContentPanel - place additional content here-->
        <ScrollViewer Grid.Row="1">
            <ctls:TileCanvas x:Name="ContentPanel" Margin="12,0,12,0" InnerMargin="20"
ColumnCount="4">
            </ctls:TileCanvas>
        </ScrollViewer>
        <Button Grid.Row="2" Click="Button_Click">Add element</Button>
    </Grid>

</phone:PhoneApplicationPage>
```

And here is the code-behind – handling the button's click event. You can either display images or just solid color rectangles (commented out).

(C) Copyright Nokia 2014. All rights reserved.

```
    public partial class MainPage : PhoneApplicationPage
    {
        public MainPage()
        {
            InitializeComponent();
        }

        Random rand = new Random();

        private void Button_Click( object sender, RoutedEventArgs e )
        {
            //Rectangle r = new Rectangle();
            //r.Fill = new SolidColorBrush( Color.FromArgb( 255, ( byte )rand.Next( 256
), ( byte )rand.Next( 256 ), ( byte )rand.Next( 256 ) ) );
            //ContentPanel.Children.Add( r );

            Image i = new Image();
            i.Source = new BitmapImage( new Uri( "/Images/" + rand.Next( 10 ) + ".jpg",
UriKind.Relative ) );
            i.Stretch = Stretch.UniformToFill;
            ContentPanel.Children.Add( i );
        }
    }
```

TileCanvas Source Code download

You can download the source code of this sample control here - File:TileCanvas.zip

Summary

In this article we shown some useful tips that will simplify your work with large screen design. We created a simple control, that can be used to make grid data display **more alive and natural**. The TileCanvas is a simple control, that can be expanded and built upon, to create great experiences with great look on devices like the **Nokia Lumia 1520**.

(C) Copyright Nokia 2014. All rights reserved.