# How to add a loading screen to your Windows Phone Application using data binding and behaviors in Blend.

## Introduction

This code sample will show you how to easily add a loading screen to your Windows Phone Application using data binding and behaviors in Blend.   
The scenario we are focusing on for this code sample is an application that loads data that may take some time to obtain and you want the user interface to display some indication while the data is being loaded. Once the application data is ready, it transitions from the loading screen to the screen displaying the data.

## Building the Sample

Right click the solution name, then click "Enable NuGet Package Restore", and then build the sample normally. You may notice that it will try to download Microsoft Http Client Libraries from NuGet if you haven't installed it before.

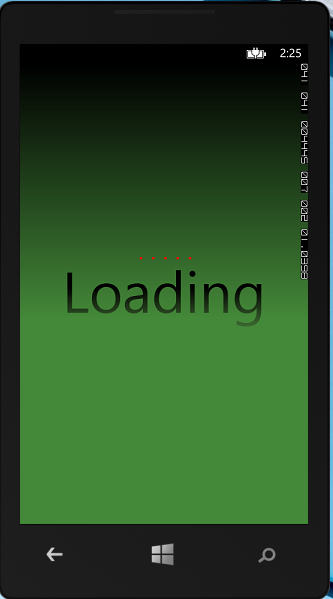
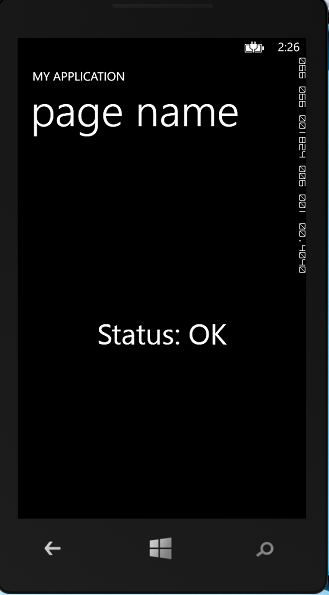
NOTE: If you see an error message like *“The build restored NuGet packages. Build the project again to include these packages in the build.”* when building the sample, please close the solution, then re-open and rebuild it.

To install Microsoft HTTP Client Libraries manually, please refer to the following link:

<https://www.nuget.org/packages/Microsoft.Net.Http/2.2.18>

## Running the Sample

Press Ctrl + F5 to run the sample, you will first see screenshot (a): it indicates the data is being loaded; once the data is ready, you will see screenshot (b).

(a) (b)

## Using the Code

The sample application is fairly simple and we're using the Model-View-ViewModel pattern.

We simulate loading data by specifying a URL to the OneCode site. This site loads relatively fast, so you may want provide a URL that takes a few seconds to respond.

To modify the URL, you can open the MainViewModel.cs(MainViewModel.vb), and replace the URL with yours.

|  |
| --- |
| -Code block start-  --C# code snippet start--  HttpResponseMessage response = await \_httpClient.GetAsync("http://blogs.msdn.com/b/onecode/");  --C# code snippet end--  Insert other Programming Language Code Snippet here  --VB code snippet start--  Dim response As HttpResponseMessage = Await \_httpClient.GetAsync("http://blogs.msdn.com/b/onecode/")  --VB code snippet end--  -Code block end- |

This will be sufficient for the purpose of this sample which is focusing on the user interface, and not the application logic.

Step 1. Create MainViewModel.

The ViewModel contains the DataLoaded property, which is a Boolean. A value of True indicates the data is loaded. A value of False indicates it is not.

The Status property is a simple string where we'll populate the Http Status response we get from our sample web service call, which is done in the LoadViewModelData routine.

|  |
| --- |
| -Code block start-  --C# code snippet start--  public class MainViewModel : INotifyPropertyChanged  {  HttpClient \_httpClient;  public MainViewModel()  {  \_dataLoaded = false;  \_status = "";  \_httpClient = new HttpClient();  LoadViewModelData();  }  public event PropertyChangedEventHandler PropertyChanged;  private void NotifyPropertyChanged(string propName)  {  if (PropertyChanged != null)  PropertyChanged(this, new PropertyChangedEventArgs(propName));  }  private bool \_dataLoaded;  public bool DataLoaded  {  get { return \_dataLoaded; }  set { \_dataLoaded = value; NotifyPropertyChanged("DataLoaded"); }  }  private string \_status;  public string Status  {  get { return \_status; }  set { \_status = value; NotifyPropertyChanged("Status"); }  }    public async void LoadViewModelData()  {  DataLoaded = false;  HttpResponseMessage response = await \_httpClient.GetAsync("http://blogs.msdn.com/b/onecode/");  Status = String.Format("Status: {0}", response.StatusCode);  DataLoaded = true;  }    }  --C# code snippet end--  Insert other Programming Language Code Snippet here  --VB code snippet start--  Public Class MainViewModel  Implements INotifyPropertyChanged  Private \_httpClient As HttpClient  Public Sub New()  \_dataLoaded = False  \_status = ""  \_httpClient = New HttpClient()  LoadViewModelData()  End Sub  Public Event PropertyChanged(sender As Object, e As PropertyChangedEventArgs) Implements INotifyPropertyChanged.PropertyChanged  Private Sub NotifyPropertyChanged(propName As String)  RaiseEvent PropertyChanged(Me, New PropertyChangedEventArgs(propName))  End Sub  Private \_dataLoaded As Boolean  Public Property DataLoaded() As Boolean  Get  Return \_dataLoaded  End Get  Set(value As Boolean)  \_dataLoaded = value  NotifyPropertyChanged("DataLoaded")  End Set  End Property  Private \_status As String  Public Property Status() As String  Get  Return \_status  End Get  Set(value As String)  \_status = value  NotifyPropertyChanged("Status")  End Set  End Property  Public Async Sub LoadViewModelData()  DataLoaded = False  Dim response As HttpResponseMessage = Await \_httpClient.GetAsync("http://blogs.msdn.com/b/onecode/")  Status = [String].Format("Status: {0}", response.StatusCode)  DataLoaded = True  End Sub  End Class  --VB code snippet end--  -Code block end- |

Step 2. Create ViewModelLocator.

ViewModelLocator exposes our main ViewModel through the main property on the ViewModelLocator.

|  |
| --- |
| -Code block start-  --C# code snippet start--  public class ViewModelLocator  {  /// <summary>  /// Initializes a new instance of the ViewModelLocator class.  /// </summary>  static ViewModelLocator()  {  \_main = new MainViewModel();  }  public static MainViewModel \_main;  public MainViewModel Main  {  get  {  return \_main;  }  }  public static void Cleanup()  {  // TODO Clear the ViewModels  }  }  --C# code snippet end--  Insert other Programming Language Code Snippet here  --VB code snippet start--  Public Class ViewModelLocator  ''' <summary>  ''' Initializes a new instance of the ViewModelLocator class.  ''' </summary>  Shared Sub New()  \_main = New MainViewModel()  End Sub  Public Shared \_main As MainViewModel  Public ReadOnly Property Main() As MainViewModel  Get  Return \_main  End Get  End Property  Public Shared Sub Cleanup()  ' TODO Clear the ViewModels  End Sub  End Class  --VB code snippet end--  -Code block end- |

This is created as a static resource in our App.xaml file so any page in our application can access the ViewModelLocator and obtain a ViewModel from it.

|  |
| --- |
| -Code block start-  --XAML code snippet start--  <Application.Resources>  <local:LocalizedStrings xmlns:local="clr-namespace:SplashScreen" x:Key="LocalizedStrings"/>  <vm:ViewModelLocator x:Key="Locator"/>  </Application.Resources>  --XAML code snippet end--  Insert other Programming Language Code Snippet here  -Code block end- |

Step 3. Customize MainPage.

In this case our MainPage.xaml sets its data context equal to the main property of our ViewModelLocator. And our MainPage.xaml only has a simple TextBlock that will display the status from our ViewModel.

|  |
| --- |
| -Code block start-  --XAML code snippet start--  <phone:PhoneApplicationPage  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"  xmlns:i="clr-namespace:System.Windows.Interactivity;assembly=System.Windows.Interactivity" xmlns:ec="clr-namespace:Microsoft.Expression.Interactivity.Core;assembly=Microsoft.Expression.Interactions"  x:Class="SplashScreen.MainPage"  mc:Ignorable="d"  FontFamily="{StaticResource PhoneFontFamilyNormal}"  FontSize="{StaticResource PhoneFontSizeNormal}"  Foreground="{StaticResource PhoneForegroundBrush}"  SupportedOrientations="Portrait" Orientation="Portrait"  shell:SystemTray.IsVisible="True"  DataContext="{Binding Source={StaticResource Locator}, Path=Main}">  --XAML code snippet end--  Insert other Programming Language Code Snippet here  -Code block end- |

We also need to create a splash screen. We can create that in Blend and wire it up to trigger off the binding to our DataLoaded property on our ViewModel.

Open the MainPage.xaml in Blend. You can do this by right clicking MainPage.xaml in Visual Studio and choosing open in Blend. Creating the loading screen will be done in three parts.

a. We will create the visual we want to see when data is loading.

The first thing we need to do is create what we want to see when the data is loading.

To do this we'll use a StackPanel that contains a ProgressBar and LoadingText.

We'll add the StackPanel at the same level as the ContentPanel.

And then we will add the ProgressBar and the TextBlock to the StackPanel.

We want the StackPanel to take up the entire screen, so we need to change a couple properties on it.

First, we need to make sure it spans both rows. So we'll set the auto Width and auto Height proerty and then stretch it.

And lastly for this StackPanel we want to give it a background color so it stands out and covers up the main screen.

We're going to just choose a dark green here with a gradient brush and give it a little bit of a gradient here.

Next we want to position our loading text and the ProgressBar towards the middle of the screen.

So for the ProgressBar we'll set its top margin halfway down the screen.

And we'll also set the **IsIndeterminate** property to true.

This means that the loading dots that come across the screen will always be there.

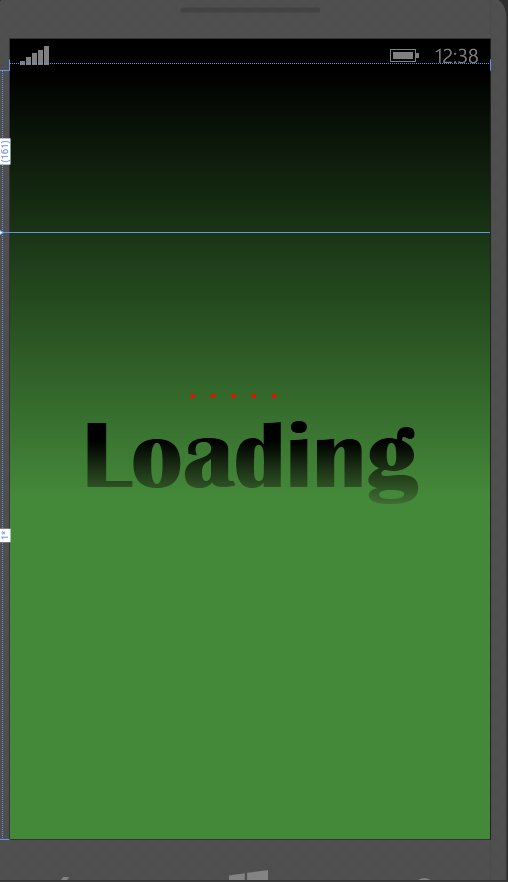
Lastly we want to change some properties on the TextBlock. We'll Start with centering it.

Then we're going to set the font to the same one we're using for our StatusText on the main screen.

And for that one we used Britannic Bold. And then we're going to set the font size to 72 and change the text to Loading.

And let's give the text a different color as well. Keep it in the green theme here. And change the brush to the linear gradient brush.

So this is basically what it will look like when data is loading.



b) Then we will create a VisualState group to define a Loading and data Loaded VisualState.

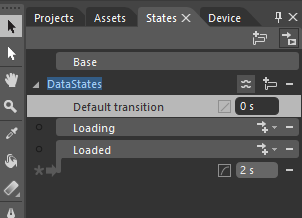
We do this from the States tab in Blend. First thing you need to do is create a VisualStateGroup which I will call DataStates. And we'll have two different states, one for the Loading state and one for the Loaded state.

When you choose either of these states, basically it will record the properties and values you want for this specific VisualState. For the loading state we don't want the content panel to be displayed; so we'll set it to collapsed. But for the StackPanel we created, this is the screen we want to see when we're loading data, so we want to make sure the opacity is 100% and its visibility is set to visible.

Now once the data is loaded, we want the splash screen to go away. So we'll set its opacity to zero and the visibility to collapsed so we can see the main screen. And that basically sets up what it looks like between the two states.

If we have transition preview on, we can see what it looks like to transition from the Loading to the Loaded state. And by default it's just going to be an abrupt change.

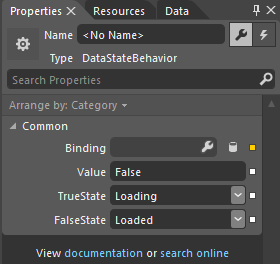
To fix that we'll add a transition, and we'll choose just a very simple curved EasingFunction and let the transition take two seconds. Now that we have our states and transitions taken care of, let's turn off recording.



c) And finally we will add a DataState behavior and bind it to our DataLoaded property.

Then we need to hook up our ViewModel's DataLoaded property so that it controls the VisualStates in our UI.

To do this we add a DataStateBehavior to our page. The behaviors can be found on the Assets tab under behaviors. And in this case we'll add the DataStateBehavior to our page. We'll bind the DataStateBehavior to the DataLoadedProperty on our ViewModel. The value of the DataLoaded property will trigger our state changes on when in False.



So when our DataLoaded property value is False, it means that we want to set the True state to Loading. And when the DataLoaded property is True, it means that we want to set the False state to Loaded.

You can use these DataStates for more complex scenarios as well.

For example a game screen on which we want to show scores between levels while the next level is loading, or something similar like that.

The following is the complete XAML code of the MainPage:

|  |
| --- |
| -Code block start-  --XAML code snippet start--  <phone:PhoneApplicationPage  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"  xmlns:i="clr-namespace:System.Windows.Interactivity;assembly=System.Windows.Interactivity" xmlns:ec="clr-namespace:Microsoft.Expression.Interactivity.Core;assembly=Microsoft.Expression.Interactions"  x:Class="SplashScreen.MainPage"  mc:Ignorable="d"  FontFamily="{StaticResource PhoneFontFamilyNormal}"  FontSize="{StaticResource PhoneFontSizeNormal}"  Foreground="{StaticResource PhoneForegroundBrush}"  SupportedOrientations="Portrait" Orientation="Portrait"  shell:SystemTray.IsVisible="True"  DataContext="{Binding Source={StaticResource Locator}, Path=Main}">  <i:Interaction.Behaviors>  <ec:DataStateBehavior Binding="{Binding DataLoaded}" Value="False" TrueState="Loading" FalseState="Loaded"/>  </i:Interaction.Behaviors>  <!--LayoutRoot is the root grid where all page content is placed-->  <!--LayoutRoot is the root grid where all page content is placed-->  <!--LayoutRoot is the root grid where all page content is placed-->  <Grid x:Name="LayoutRoot" Background="Transparent">  <VisualStateManager.VisualStateGroups>  <VisualStateGroup x:Name="DataStates">  <VisualStateGroup.Transitions>  <VisualTransition GeneratedDuration="0:0:2" To="Loaded">  <VisualTransition.GeneratedEasingFunction>  <CubicEase EasingMode="EaseOut"/>  </VisualTransition.GeneratedEasingFunction>  </VisualTransition>  </VisualStateGroup.Transitions>  <VisualState x:Name="Loading">  <Storyboard>  <ObjectAnimationUsingKeyFrames Storyboard.TargetProperty="(UIElement.Visibility)" Storyboard.TargetName="ContentPanel">  <DiscreteObjectKeyFrame KeyTime="0">  <DiscreteObjectKeyFrame.Value>  <Visibility>Collapsed</Visibility>  </DiscreteObjectKeyFrame.Value>  </DiscreteObjectKeyFrame>  </ObjectAnimationUsingKeyFrames>  </Storyboard>  </VisualState>  <VisualState x:Name="Loaded">  <Storyboard>  <DoubleAnimation Duration="0" To="0" Storyboard.TargetProperty="(UIElement.Opacity)" Storyboard.TargetName="stackPanel" d:IsOptimized="True"/>  </Storyboard>  </VisualState>  </VisualStateGroup>  </VisualStateManager.VisualStateGroups>  <Grid.RowDefinitions>  <RowDefinition Height="Auto"/>  <RowDefinition Height="\*"/>  </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="MY APPLICATION" Style="{StaticResource PhoneTextNormalStyle}" Margin="12,0"/>  <TextBlock Text="page name" Margin="9,-7,0,0" Style="{StaticResource PhoneTextTitle1Style}"/>  </StackPanel>  <!--ContentPanel - place additional content here-->  <Grid x:Name="ContentPanel" Grid.Row="1" Margin="12,0,12,0">  <TextBlock HorizontalAlignment="Center" TextWrapping="Wrap" Text="{Binding Status}" VerticalAlignment="Center" FontFamily="Britannic Bold" FontSize="48"/>  </Grid>  <StackPanel x:Name="stackPanel" Grid.RowSpan="2">  <StackPanel.Background>  <LinearGradientBrush EndPoint="0.5,1" StartPoint="0.5,0">  <GradientStop Color="Black" Offset="0"/>  <GradientStop Color="#FF448939" Offset="0.552"/>  </LinearGradientBrush>  </StackPanel.Background>  <ProgressBar Height="10" Margin="0,320,0,0" IsIndeterminate="True"/>  <TextBlock TextWrapping="Wrap" Text="Loading" TextAlignment="Center" FontFamily="Britannic Bold" FontSize="96">  <TextBlock.Foreground>  <LinearGradientBrush EndPoint="0.5,1" StartPoint="0.5,0">  <GradientStop Color="Black" Offset="0.375"/>  <GradientStop Color="#FF427434" Offset="1"/>  <GradientStop Color="#FF417333" Offset="1"/>  </LinearGradientBrush>  </TextBlock.Foreground>  </TextBlock>  </StackPanel>  </Grid>  </phone:PhoneApplicationPage>  --XAML code snippet end--  Insert other Programming Language Code Snippet here  -Code block end- |

## More Information

Video blog: Implementing a Loading Splash Screen for your Windows Phone App

<http://channel9.msdn.com/Series/Windows-Store-Developer-Solutions/Implementing-a-Loading-Splash-Screen-for-your-Windows-Phone-App>

Implementing the Model-View-ViewModel pattern in a Windows Phone App  
<http://msdn.microsoft.com/en-us/library/windowsphone/develop/gg521153(v=vs.105).aspx>

Designing your XAML UI with Blend Jumpstart  
<http://www.microsoftvirtualacademy.com/training-courses/designing-your-xaml-ui-with-blend-jump-start#?fbid=Y4cvgt4Hy8E>