Lesson 10 Guide: Deployment via the Windows Store

Table of Contents

Designing for the Store

While an automated program called the Windows Application Certification Kit (WACK) is used to ensure technical conformation, human testers evaluate the submitted app's design, usability, performance, and worthiness.

Here's a brief checklist of some design considerations:

- ☐ Strive to use the Segoe font family.
- ☐ Use a title header that is 120–140 pixels in height.
- ☐ Maintain a left margin of 120 pixels in full landscape view.
- ☐ Maintain at least 20 pixels between interactive controls.
- ☐ Provide alternative designs or modifications for filled and snapped views. If the app will support portrait orientation, it must provide alternative screen design for that state.
- ☐ Manage data persistence to provide a continuous experience if the app is terminated and then re-activated.

In this lesson, the user interface of the Take Me Out to the Ballgame assignment will be modified for Store deployment.

In the Take Me Out to the Ballgame app, a 120-pixel left margin was established by providing an additional column of 120 pixels. Column 0 is blank. Column 1 contains all the UI controls (except the map), and the third column, Column 2, contains the Bing map control.

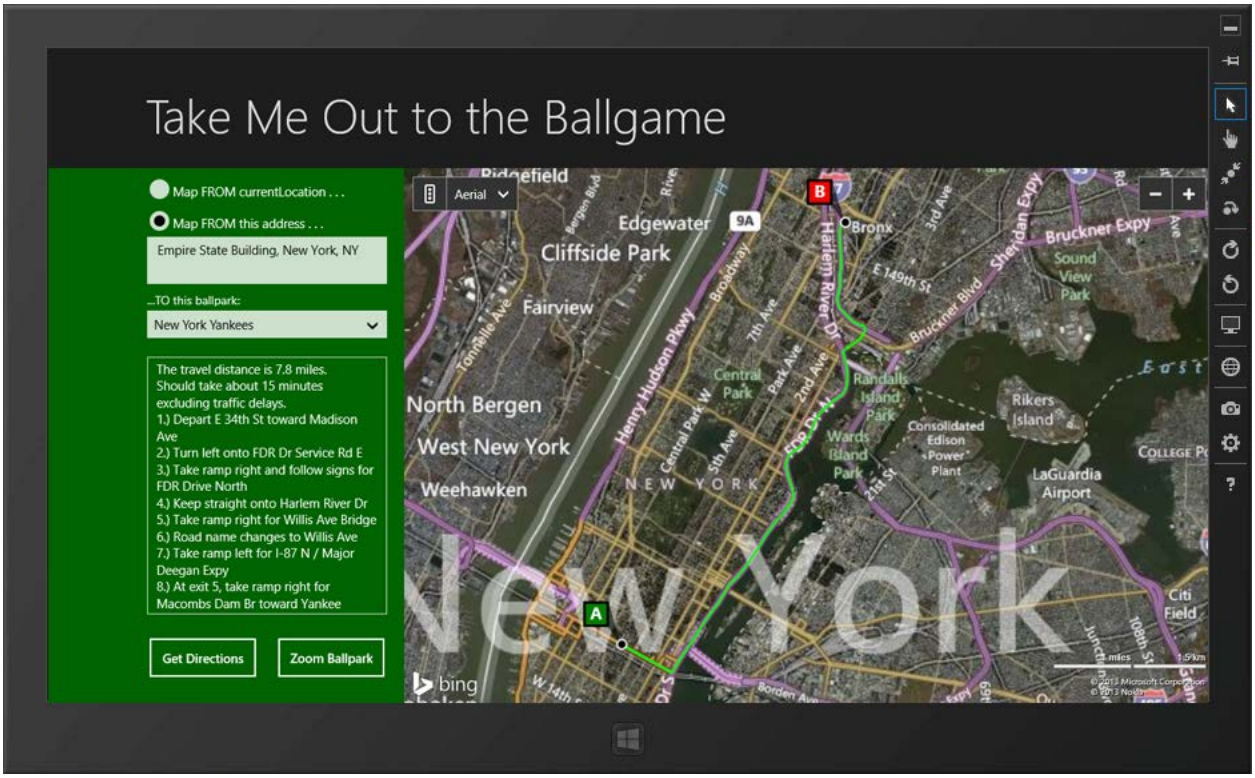


Figure 1 *– The app conforms to the standard 120-pixel left margin in the full landscape state.*

XAML Code Snippet for BasicPage1.xaml

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

            <TextBlock x:Name="pageTitle" Margin="120,0,0,40"
                       Text="Take Me Out to the Ballgame" Style="{StaticResource
                       PageHeaderTextStyle}" TextWrapping="Wrap"/>

            <Grid Grid.Row="1">
                <Grid.ColumnDefinitions>
                    <ColumnDefinition x:Name="Col1A" Width="120"/>
                    <ColumnDefinition x:Name="Col1B" Width="300"/>
                    <ColumnDefinition Width="*"/>
                </Grid.ColumnDefinitions>
                <Rectangle Grid.Row="1"  Fill="DarkGreen"/>
                <Rectangle Grid.Row="1"  Grid.Column="1" Fill="DarkGreen"/>
                <RadioButton x:Name="rbFromLocation" Grid.Row="1"  Grid.Column="1"
                        Content="Map FROM currentLocation . . ."
                        HorizontalAlignment="Left" Height="23" Margin="0,13,0,0"
                        VerticalAlignment="Top" Width="280" IsChecked="True"/>
                <RadioButton x:Name="rbFromAddress" Grid.Row="1"  Grid.Column="1"
                        Content="Map FROM this address . . . " HorizontalAlignment="Left"
                        Height="23" Margin="0,50,0,0" VerticalAlignment="Top" Width="280"/>
                <TextBox x:Name ="txtAddress" Grid.Row="1"  Grid.Column="1"
                        HorizontalAlignment="Left" Height="55" Margin="0,80,0,0"
                        TextWrapping="Wrap" Text="7050 S. 24th Street, Phoenix, AZ, 85042"
                        VerticalAlignment="Top" Width="280"/>
                <TextBlock Grid.Row="1"  Grid.Column="1" HorizontalAlignment="Left"
                        Height="15" Margin="0,146,0,0" TextWrapping="Wrap" Text="...TO this
                        ballpark:" VerticalAlignment="Top" Width="168" FontSize="14"/>
                <ComboBox x:Name="cmbTeam" Grid.Row="1"  Grid.Column="1"
                        HorizontalAlignment="Left" Margin="0,166,0,0" VerticalAlignment="Top"
                        Width="280"/>
                <TextBox x:Name="txtItinerary" Grid.Row="1"  Grid.Column="1"
                        HorizontalAlignment="Left" Height="300"
                       Margin="0,220,0,0" TextWrapping="Wrap" Text="Directions will be shown
                        here." Foreground="White" Background="DarkGreen"
                        VerticalAlignment="Top" BorderThickness="1"
                        ScrollViewer.VerticalScrollBarVisibility="Auto" Width="280" />
                <Button x:Name="btnGetDirections" Grid.Row="1"  Grid.Column="1" Content="Get
                        Directions" HorizontalAlignment="Left" Height="51" Margin="0,545,0,0"
                        VerticalAlignment="Top" Width="130"  Click="GetDirections"/>
                <Button x:Name="btnCenterZoom" Grid.Row="1"  Grid.Column="1" Content="Zoom
                        Ballpark" HorizontalAlignment="Left" Height="51" Margin="150,545,0,0"
                        VerticalAlignment="Top" Width="130"  Click="ZoomBallpark"/>
                <bm:Map Grid.Row="1" Grid.Column="2" Credentials="Aj3Kc7F3dXDGy0-bmbnmurrKJYf-
                         ylvzIuxzXproUTKIpyUblyfxgLTzgi9W_hy5" x:Name="myMap" />
            </Grid>
        </Grid>
```

The use of the blank column (“Col1A”) makes it easy to shift all the controls to the left in the snapped visual state. A VisualStateManager is added to the XAML code. For the filled state, the width of the left column is reduced to 20 pixels. Likewise the “pageTitle” textbox is shifted left from 120 pixels to 20. These same animations occur in the Snapped state, along with resizing and wrapping the title (“pageTitle” textbox). Note that the Visibility of the map object (“myMap”) is set to Collapsed. This was done to compensate for a known

bug in the Bing maps that causes the app to crash if the map object has children levels in the snapped view in Windows 8. Collapsing it seems to bypass the problem.

Addition to the XAML Code for BasicPage1.xaml

```
        <VisualStateManager.VisualStateGroups>
            <!-- Visual states reflect the application's view state -->
            <VisualStateGroup x:Name="ApplicationViewStates">
                <VisualState x:Name="FullScreenLandscape"/>
                <VisualState x:Name="Filled">
                    <Storyboard>
                        <ObjectAnimationUsingKeyFrames Storyboard.TargetName="Col1A"
                                  Storyboard.TargetProperty="Width">
                            <DiscreteObjectKeyFrame KeyTime="0" Value="20"/>
                        </ObjectAnimationUsingKeyFrames>

                        <ObjectAnimationUsingKeyFrames Storyboard.TargetName="pageTitle"
                                   Storyboard.TargetProperty="Margin">
                            <DiscreteObjectKeyFrame KeyTime="0" Value="20,0,0,40"/>
                        </ObjectAnimationUsingKeyFrames>
                    </Storyboard>
                </VisualState>
                <VisualState x:Name="FullScreenPortrait"/>

                <VisualState x:Name="Snapped">
                    <Storyboard>
                        <!—Collapse column 0 to 20 pixels -->
                        <ObjectAnimationUsingKeyFrames Storyboard.TargetName="Col1A"
                                 Storyboard.TargetProperty="Width">
                            <DiscreteObjectKeyFrame KeyTime="0" Value="20"/>
                        </ObjectAnimationUsingKeyFrames>
                        <ObjectAnimationUsingKeyFrames Storyboard.TargetName="myMap"
                                 Storyboard.TargetProperty="Visibility">
                            <DiscreteObjectKeyFrame KeyTime="0" Value="Collapsed"/>
                        </ObjectAnimationUsingKeyFrames>
                         <!-- The title gets smaller and wraps when snapped -->
                        <ObjectAnimationUsingKeyFrames Storyboard.TargetName="pageTitle"
                                 Storyboard.TargetProperty="Width">
                            <DiscreteObjectKeyFrame KeyTime="0" Value="290"/>
                        </ObjectAnimationUsingKeyFrames>
                        <ObjectAnimationUsingKeyFrames Storyboard.TargetName="pageTitle"
                                 Storyboard.TargetProperty="VerticalAlignment">
                            <DiscreteObjectKeyFrame KeyTime="0" Value="Center"/>
                        </ObjectAnimationUsingKeyFrames>
                        <ObjectAnimationUsingKeyFrames Storyboard.TargetName="pageTitle"
                                 Storyboard.TargetProperty="Margin">
                            <DiscreteObjectKeyFrame KeyTime="0" Value="20,0,0,0"/>
                        </ObjectAnimationUsingKeyFrames>
                        <ObjectAnimationUsingKeyFrames Storyboard.TargetName="pageTitle"
                                 Storyboard.TargetProperty="FontSize">
                            <DiscreteObjectKeyFrame KeyTime="0" Value="38"/>
                        </ObjectAnimationUsingKeyFrames>
                    </Storyboard>
                </VisualState>
            </VisualStateGroup>
        </VisualStateManager.VisualStateGroups>

    </Grid>
```

```
</common:LayoutAwarePage>
```

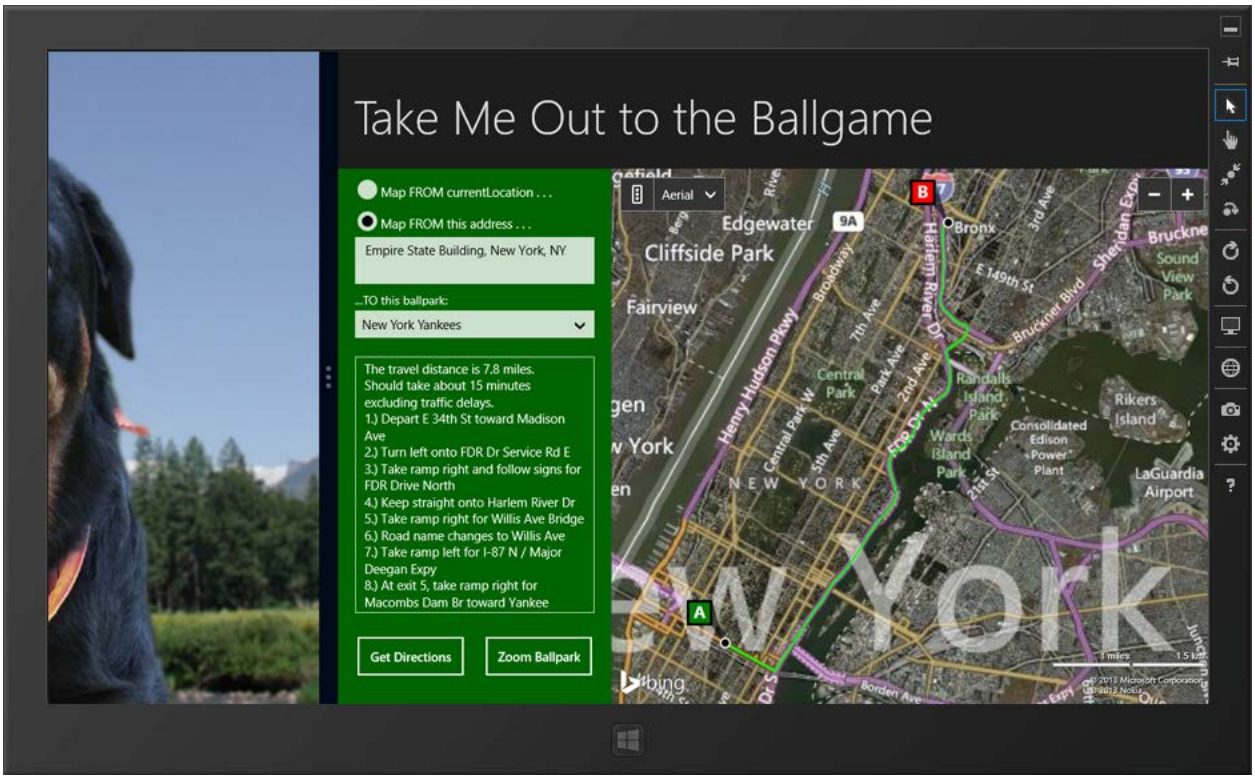


Figure 2 *– In the Filled state, the controls and title are shifted to the left by changing the margins of the title and reducing the width of the left-most column in row 1.*

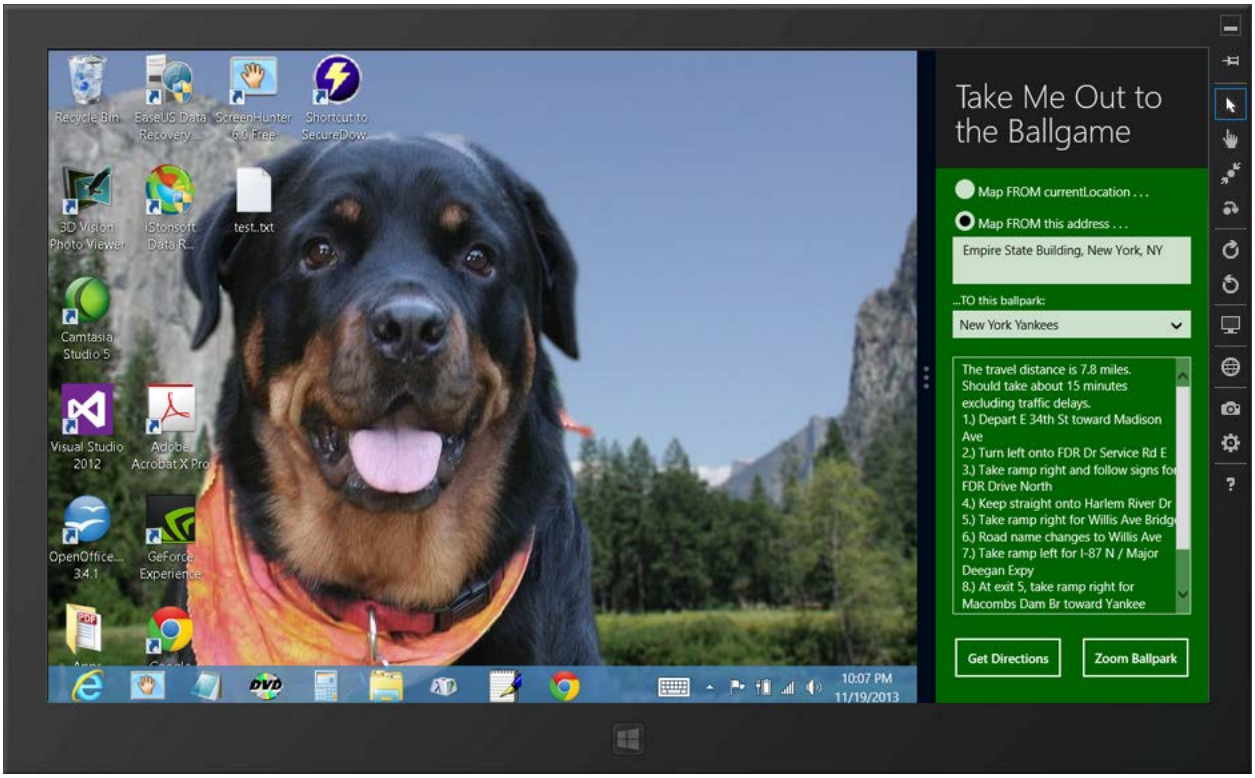


Figure 3 *– In the Snapped state, the title and controls are shifted to the left as they were in the Filled state. The map is no longer visible, but directions can still be obtained.*

Code was added to provide for data persistence if the app is terminated. The address, selections, map, and directions are returned to their previous status when the app is re-activated. The status of the controls and the latitude, longitude, and zoom levels of the map are stored in the Save State method and re-instituted in the LoadState method. Note that a timer was utilized to provide a slight 2-second delay in updating the map, allowing for the screen to first be refreshed. See Lesson 8 for more information on application lifecycle management techniques.

C# Code Snippet for BasicPage1.xaml Showing the Data Persistence / Application Lifecycle Management Portion.

```
 . . . .
double startLat, startLng, destLat, destLng;
Location xyz;
double zoomB;
DispatcherTimer timer = new DispatcherTimer();
 . . . .
 protected override void LoadState(Object navigationParameter, Dictionary<String,
                                                                Object> pageState)
 {
     if (pageState != null)
     {
         if (pageState.ContainsKey("FromLocation"))
         {
             if (pageState["FromLocation"] as String == "1")
             {
                 rbFromLocation.IsChecked = true;
                 rbFromAddress.IsChecked = false;
             }
             else
             {
                 rbFromLocation.IsChecked = false;
                 rbFromAddress.IsChecked = true;
             }
         }
         if (pageState.ContainsKey("Address"))
         {
             txtAddress.Text = pageState["Address"] as string;
         }
         if (pageState.ContainsKey("Ballpark"))
         {
             cmbTeam.SelectedIndex = int.Parse(pageState["Ballpark"] as string);
         }
         RouteDirections();
         txtItinerary.Text = pageState["Directions"] as string;

         myMap.MapType = MapType.Aerial;
         double zoomA = myMap.ZoomLevel;
         xyz = new Location(myMap.Center);
         if (pageState.ContainsKey("Lat") && pageState.ContainsKey("Lng"))
         {
             xyz.Latitude = double.Parse(pageState["Lat"] as string);
             xyz.Longitude = double.Parse(pageState["Lng"] as string);
         }
         zoomB = zoomA;
         if (pageState.ContainsKey("ZoomLevel"))
         {
```

```
                zoomB = double.Parse(pageState["ZoomLevel"] as string);
                //ShowMessage(zoomA.ToString() + "   " + zoomB.ToString(), "Zoom Levels");
            }
            // zoom and center is set vai timer so myMap has a chance to
            // first refresh the route directions
            timer.Interval = TimeSpan.FromSeconds(2);
            timer.Tick += timer_Tick;
            timer.Start();
        }
    }

    private void timer_Tick(object sender, object e)
    {
        //ShowMessage("Timer fired.", "Test");
        myMap.Center = xyz;
        myMap.SetZoomLevel(zoomB);
        timer.Stop();
    }

    protected override void SaveState(Dictionary<String, Object> pageState)
    {
        if (rbFromLocation.IsChecked == true)
        {
            pageState["FromLocation"] = "1";
        }
        else
        {
            pageState["FromLocation"] = "0";
        }
        pageState["Address"] = txtAddress.Text;
        pageState["Ballpark"] = cmbTeam.SelectedIndex.ToString();

        pageState["ZoomLevel"] = myMap.ZoomLevel.ToString();
        pageState["Lat"] = myMap.Center.Latitude.ToString();
        pageState["Lng"] = myMap.Center.Longitude.ToString();
        pageState["Directions"] = txtItinerary.Text;
    }
```

Branding: Name, Logos, and Splash Screens

While it is important to keep technical requirements and guidelines in mind, branding is a critical element that affects the success of an app in the Windows Store. Branding involves choosing a name and designing logos and splash screens for your app.

Your app needs a name. While names up to 256 characters are allowed, you should select a name that is much shorter than that. Your name must be unique to any other that is in the Windows Store. Once you come up with a name, search the store to verify that the name is not already taken. However, since names can be reserved by a developer up to 1 year in advance of submission, a registered name may not yet appear in the Store. It is therefore a good idea to perhaps have a couple backup names as well. The name should be consistent throughout the program wherever it appears. If more than one name appears in different places or if it varies from the registered name, the app will likely be rejected for inclusion in the Windows Store.

Logos are a very important part of branding the app. In the Assets folder are several default logo graphics: Logo.png, SmallLogo.png, SplashScreen.png, and StoreLogo.png. These are generic with a logo of a square containing an X. It is recommended that these be saved as a

transparent PNG file in the Assets folder of your project, though a JPG image may be substituted if transparency is not necessary. The background will automatically be set to the background color that you choose in the *package.appxmanifest* when displayed.

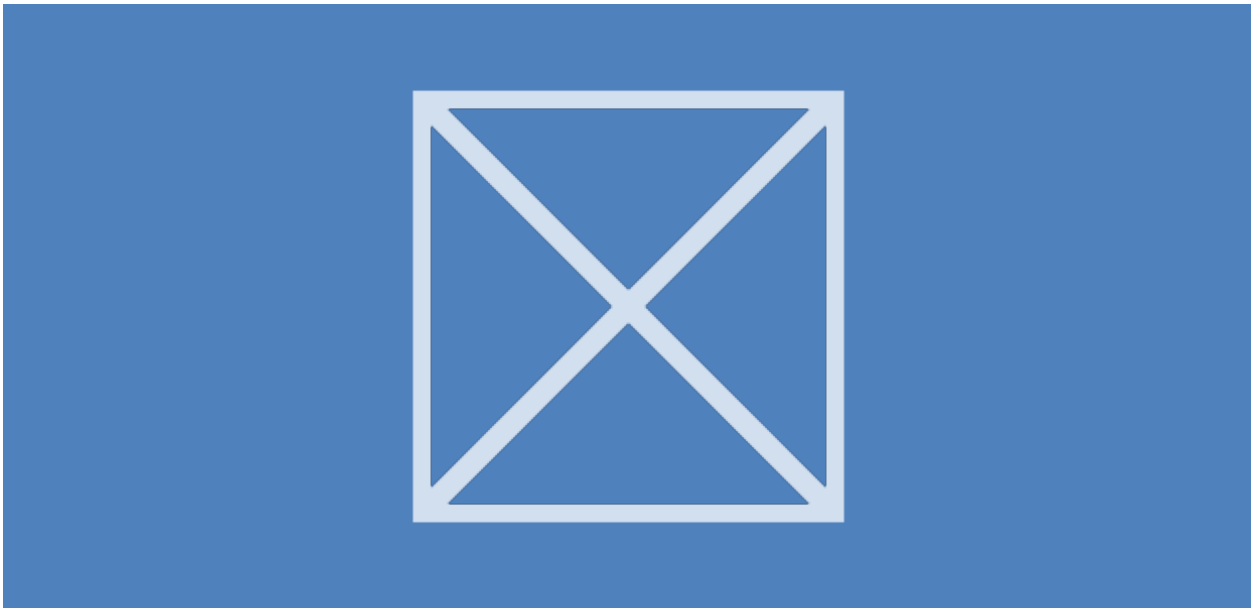


Figure 4 *– The default SplashScreen.png file with generic logo. The blue background is actually transparent so the image is chosen against the default background color.*

While the images can be edited directly in Visual Studio simply by opening them, it may be preferable to use another bitmap editor that you are comfortable with. Adobe Photoshop is a good option. Use the same logo throughout the images for consistency.

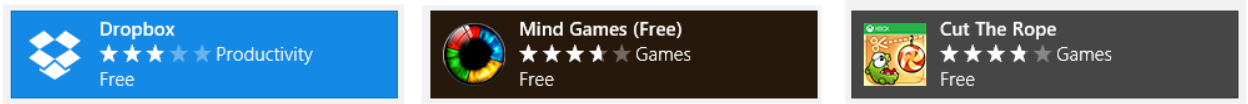


Figure 6 *– The 50x50 pixel StoreLogo.png image is used for the tile display in the Windows Store. Note that the image may be simply a white (or other single color) logo on a transparent background (left), a multicolor logo on a transparent background (center), or an image with no transparent pixels.*

The chart below shows the sizes for each image. Four images are required (highlighted and in bold in the table), the others are optional. Recall from Lesson 4 that the different resolution images should be distinguished in their names as follows:

imageName.Scale-80.png (for device that has a height resolution of 600 pixels)
imageName.Scale-100.png (for device that has a height resolution of 768 or 800 pixels)
imageName.Scale-140.png (for device that has a height resolution of 1080 pixels)
imageName.Scale-180.png (for device that has a height resolution of 1440 pixels)

Image	80% size	100% size	140% size	180% size	Use
Store Logo	**None**	**50x50**	70x70	90x90	Used by Windows Store in the Details section and search results
Small Logo	**24x24**	**30x30**	42x42	54x54	Used as the square tile image of the app in the Start screen

Logo	**120x120**	**150x150**	210x210	270x270	Used as the square tile image of the app in the Start screen
Wide Logo	248x120	310x150	434x210	558x270	Used as the wide tile on the Start screen
Badge Logo	None	24x24	33x33	43x43	Used for Badge apps on the Lock Screen
Splash Screen	None	**620x300**	868x420	1116x540	Displayed while the app is launching

There are also optional target sizes of 256x256, 48x48, 32x32 and 16x16.

Up to four promotional images may also be provided during the app submission process, in addition to those included in your app's package. These must be .PNG images and sized as 414x180, 414x168, 558x756, and 846x468. These are used for marketing in the store such as those apps featured in the Store's spotlight. The first two sizes are the most frequently used, and while all four are optional, it is recommended that you provide all four images. Providing these does not guarantee that your app will be featured, but not providing them could limit your promotional opportunities.

It is recommended that the Splash Screen and other logos should be kept to a minimum (remember the Bauhaus design influence discussed in Lesson 2). Use transparent graphics that can blend in with the background color. Consider starting your design with the 180% Splash Screen image at a resolution of 1116x540. Create a simple logo that communicates the main idea of your app.

Figure 5 *– The logo for the Take Me Out to the Ballgame app is white with a transparent background. It is shown here as the app's SplashScreen image with the selected background color, Sedona Red.*

TIP: Here is a look at how the Take Me Out to the Ballgame logo was created.
The chosen background color for the manifest was Sedona Red (#C51230), which is the principal team color of the Arizona Diamondbacks. Starting with a solid background layer of this color in Photoshop, the developer created a new image that was 1116x540 in size. He added the baseball as a separate layer and selected the stitching, then deleted it so the background color

showed through. A third layer consisted of the arrow (a preset shape in Photoshop's custom shape tool) using a white fill. He stroked the arrow on the outside with a 4-pixel-width stroke. Then, that stroke was selected (using the magic wand tool) and deleted. With the stroke area still selected, the baseball layer underneath was made the active layer and the stroke area was deleted so the underlying background could be seen. The background layer's visibility was turned off, leaving only the white logo of the baseball and arrow layers visible. The image was saved as a PSD file first (for future use) then saved out as a PNG file keeping its transparency. It was saved with a filename of SplashScreen.Scale-180.png. The image size was reduced to 848x429 and saved as SplashScreen.Scale-140.png. Then, it was further reduced to a size of 620x300 and saved as SplashScreen.Scale-100.png. The file's image size and/or canvas size was modified and scaled appropriately to create all the other logo files at the three different resolutions.

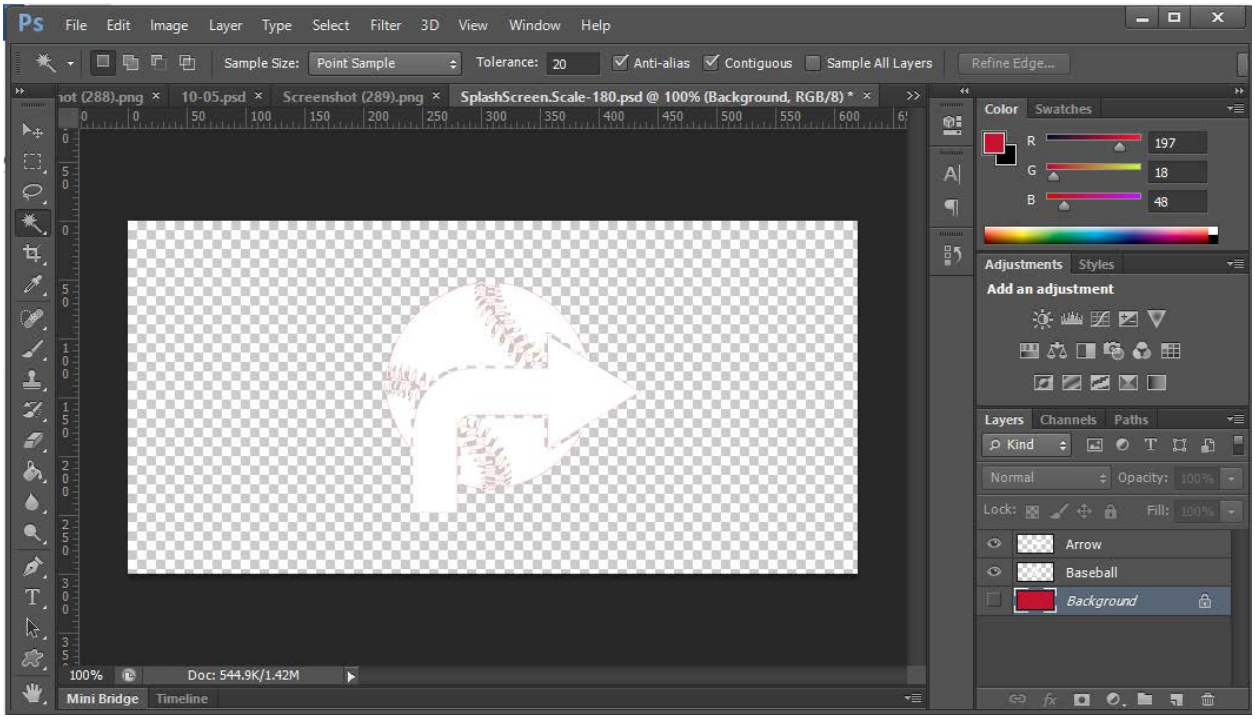


Figure 6 *– Photoshop with its flexibility of layers is an ideal tool for creating the transparent PNG logo images.*

The images were then all imported into the project's Asset's folder by right-clicking the folder in the Solution Explorer and choosing "Existing Item" from the Add submenu.

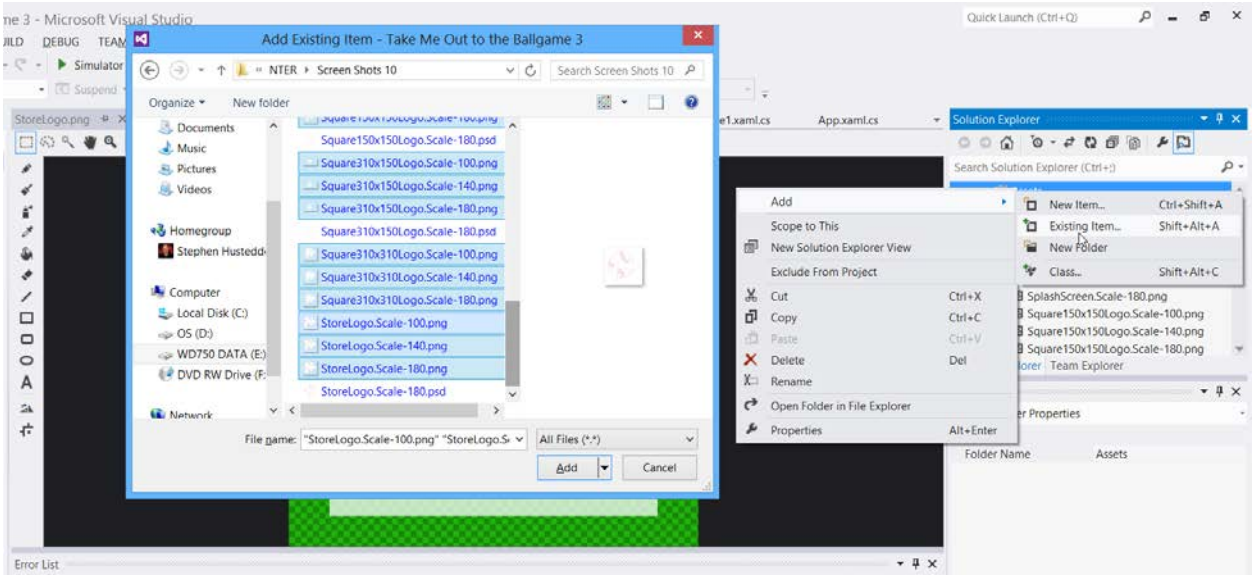


Figure 7 *– The logo and splash screen images are imported into the Assets folder.*

Setting Up the Manifest

The next step is to specify the name, background color, and graphics in the manifest document. Open the *package.appxmanifest*. Provide a name, default language if other than U.S. English, a brief description, and choose the supported rotations. Remember, while the name can be up to 256 characters, it is preferable to keep it short.

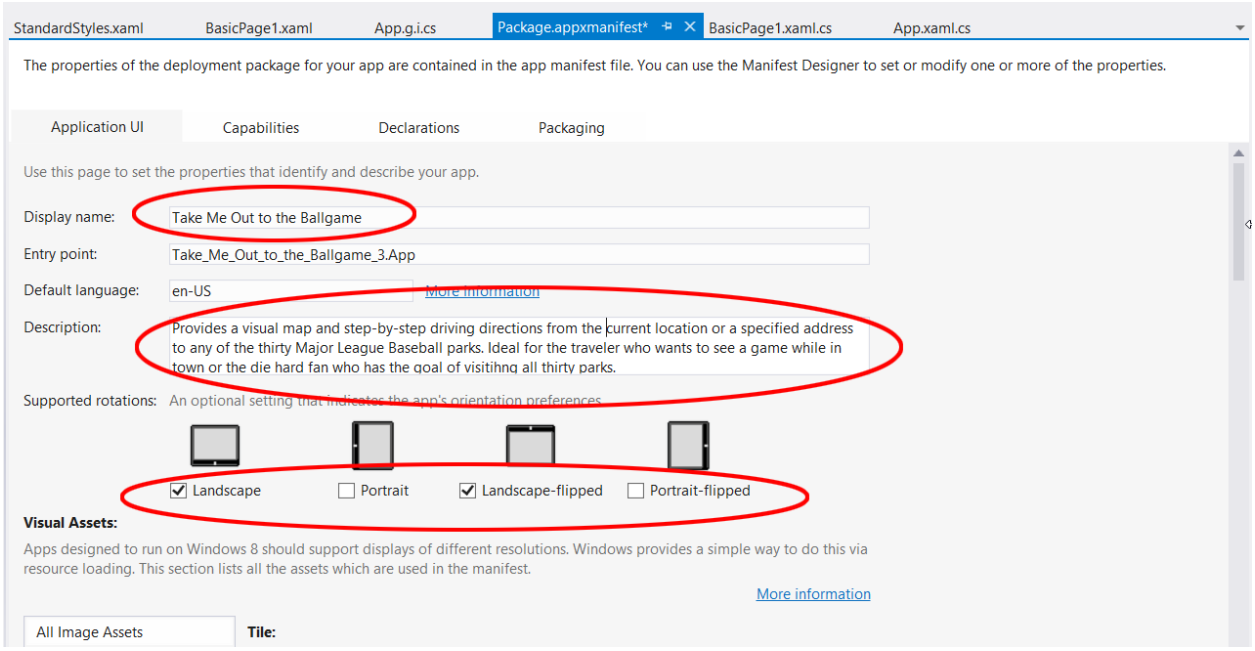


Figure 8 *– Provide a display name, description, and specify the supported rotations.*

Scroll down in the manifest and enter a short name (maximum of 40 characters). Specify a background color as a 24-bit hexadecimal value for the tile background color (on the Start screen) and the Splash screen. It is recommended they are the same, but it is not a requirement.

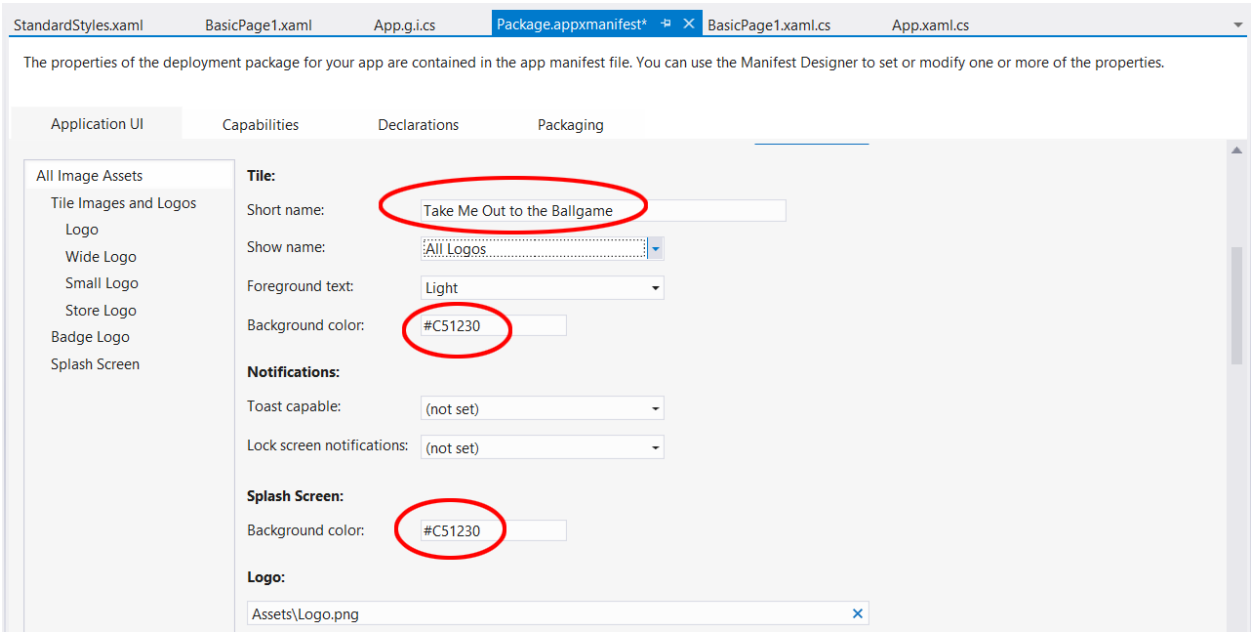


Figure 9 *– Provide a short name and background colors for the Start Screen tile and the Splash Screen in the manifest.*

Scroll down some more and specify the images for the Logo, Wide Logo, Small Logo, Target images, Store Logo, and Splash Screen, as well as the Badge Logo if pertinent. While only the 100% views of the Logo, Small Logo, Store Logo, and Splash Screen are required, it is recommended that you provide images for all sizes and for the Wide Logo as well.

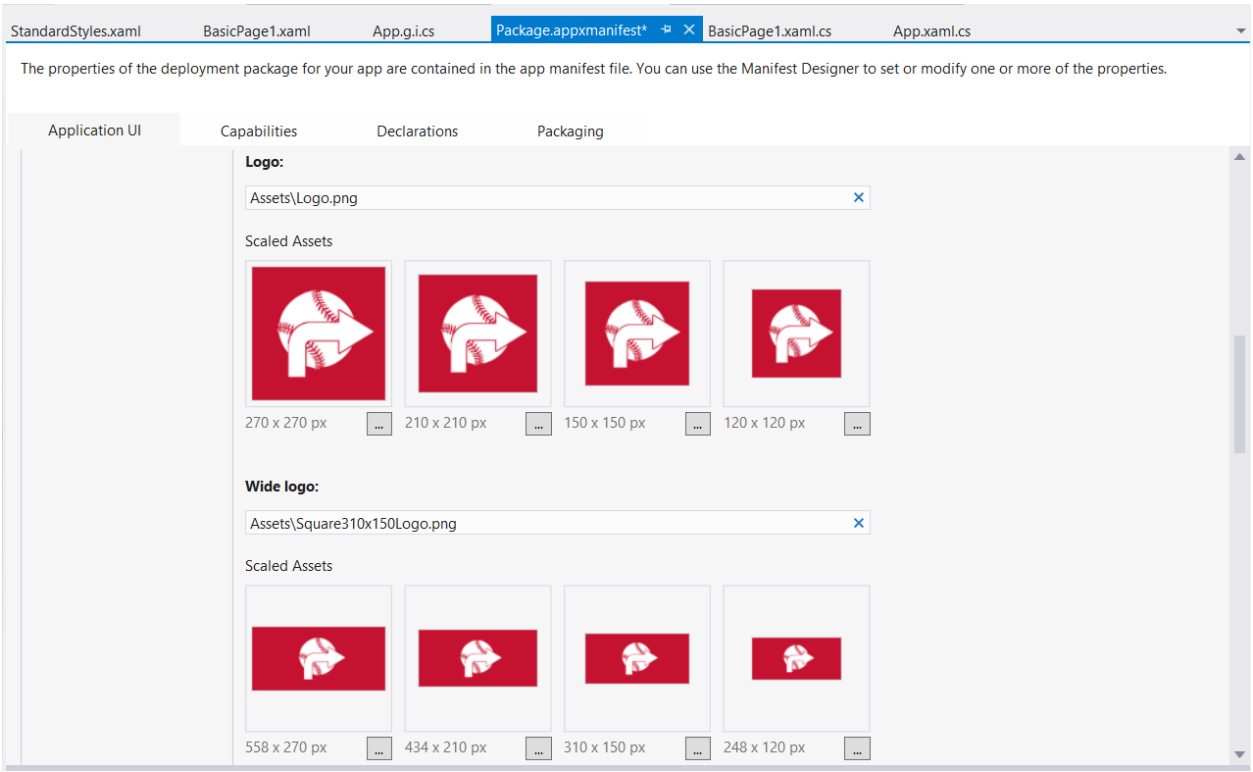


Figure 10 *– The various sized Logo and Splash Screen images are set up in the manifest.*

In the Packaging tab of the manifest, verify the name, provide a version, and provide the Publisher display name as you would like it to appear. The version number will be used to provide automated update notices to the user as the app is modified. Thus, if submitting an update to an app that is already in the Store, it is critical that the version number be increased.

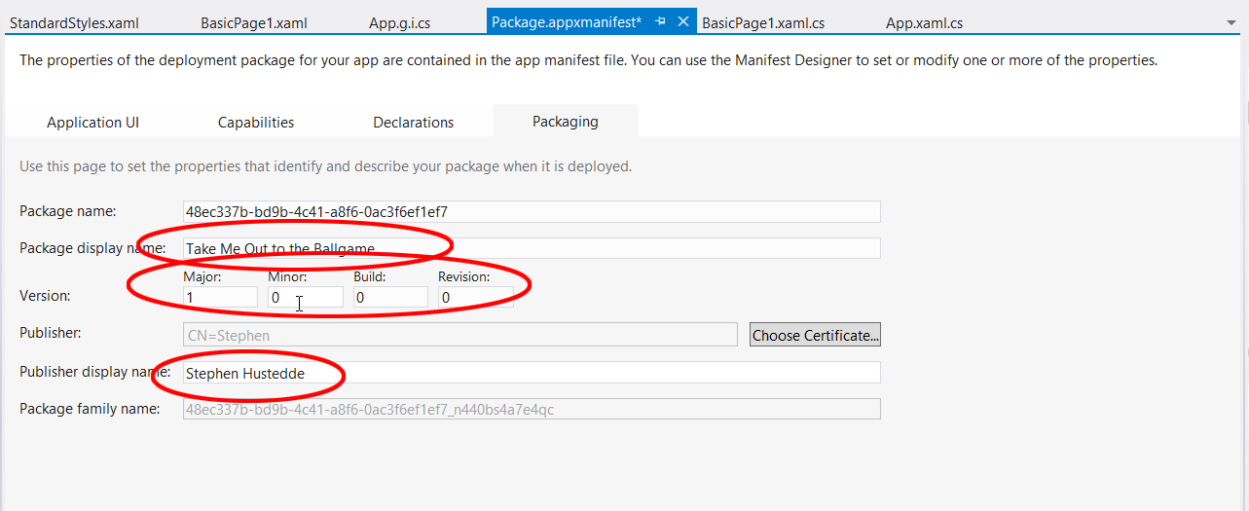


Figure 11 *– Check the Package name, version number, and Publisher display name in the Packaging tab of the manifest.*

Save the project after making these changes.

Testing With the Windows App Certification Kit

The Windows App Certification Kit (WACK) is available free of charge from Microsoft. This tool is used to analyze your app after submitting it to the store to verify that it meets the technical qualifications. WACK examines the manifest to ensure that it is correctly setup and that all necessary capabilities and declarations have been established. It confirms that the image resources for the logos are present in the package. The WACK also checks the stability of the app—testing for crashes and errors. It also verifies that the app can be suspended and re-launched without issue.

To test with WACK, complete the following steps:

Step 1: The app needs to be reconfigured for a Release Build rather than a Debug Build. Open the Configuration Manager from the Build menu. Set the configuration to Release rather than Debug.

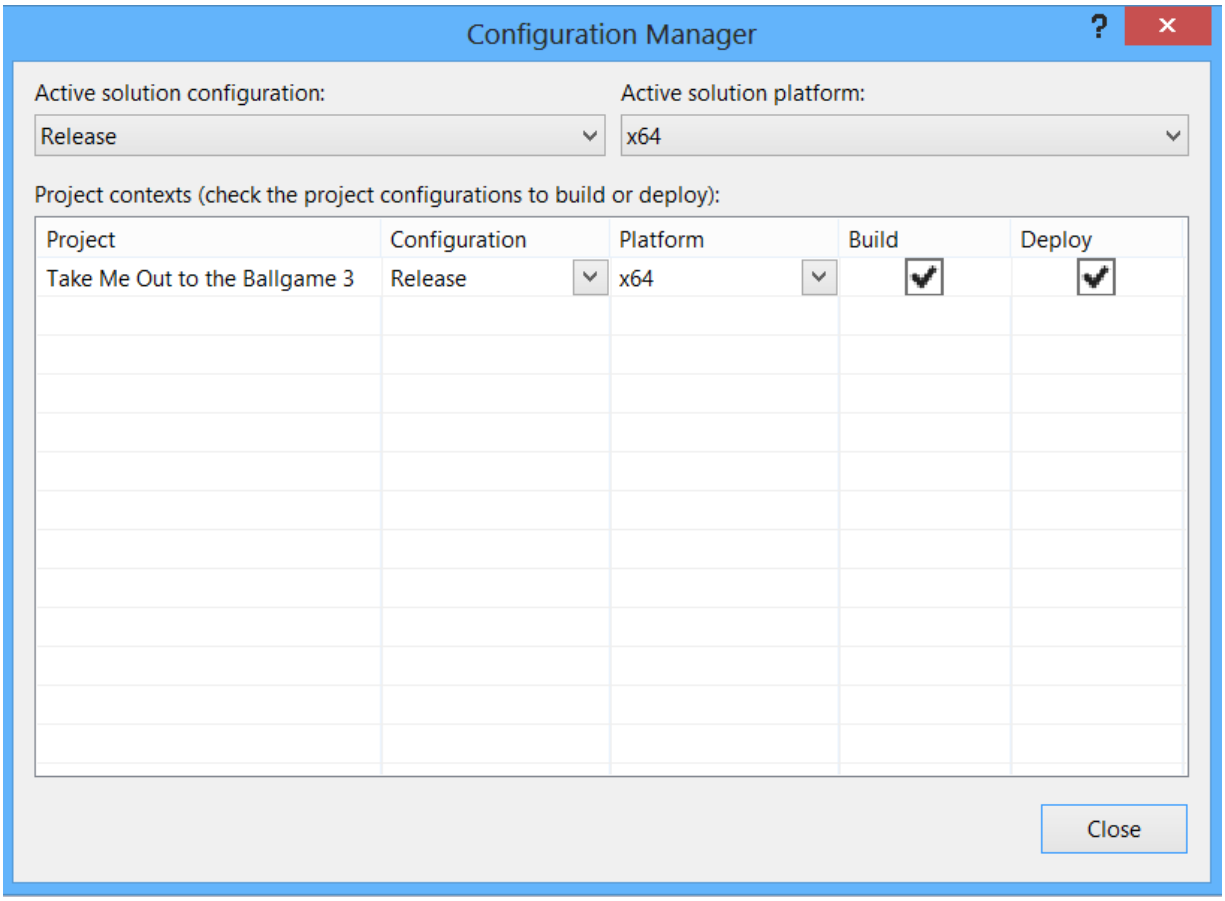


Figure 12 – The Build configuration must be set to Release rather than Debug and the project then rebuilt.

TIP: Because the inclusion of Bing Maps requires that you target a specific processor, the option of Any CPU will cause errors. In this case, you must create three different packages, each one targeted to the different platforms of x64, x86, and ARM processors. In submitting the app for inclusion at the Windows Store, all three packages will be uploaded. The appropriate package will be used to install the app to the client's device based on their device's processor. Normally, you can create one package that handles all three CPUs by selecting "Any CPU" in the Platform dropdown.

Step 2: Download and install the Windows App Certification Kit (WACK) from the Windows Dev Center – Windows Store Apps website. Follow the online instructions for downloading and installing the certification kit.

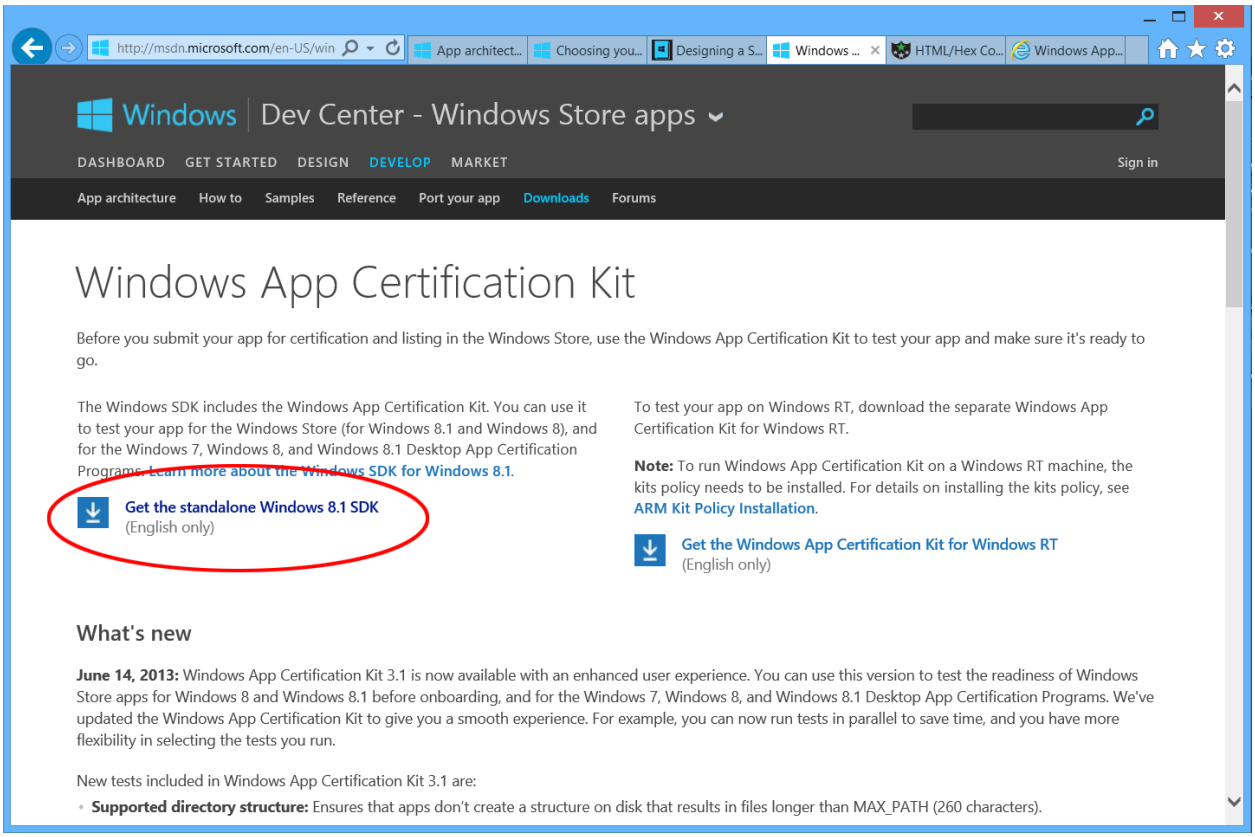


Figure 13 *– Click the 'Get the standalone Windows 8.1 SDK' link to download and install the Windows App Certification Kit.*

Step 3: Open the WACK. The easiest way to do this is to search for "appcertui" in the All Apps screen. Click the tile for the "Windows App Cert Kit".

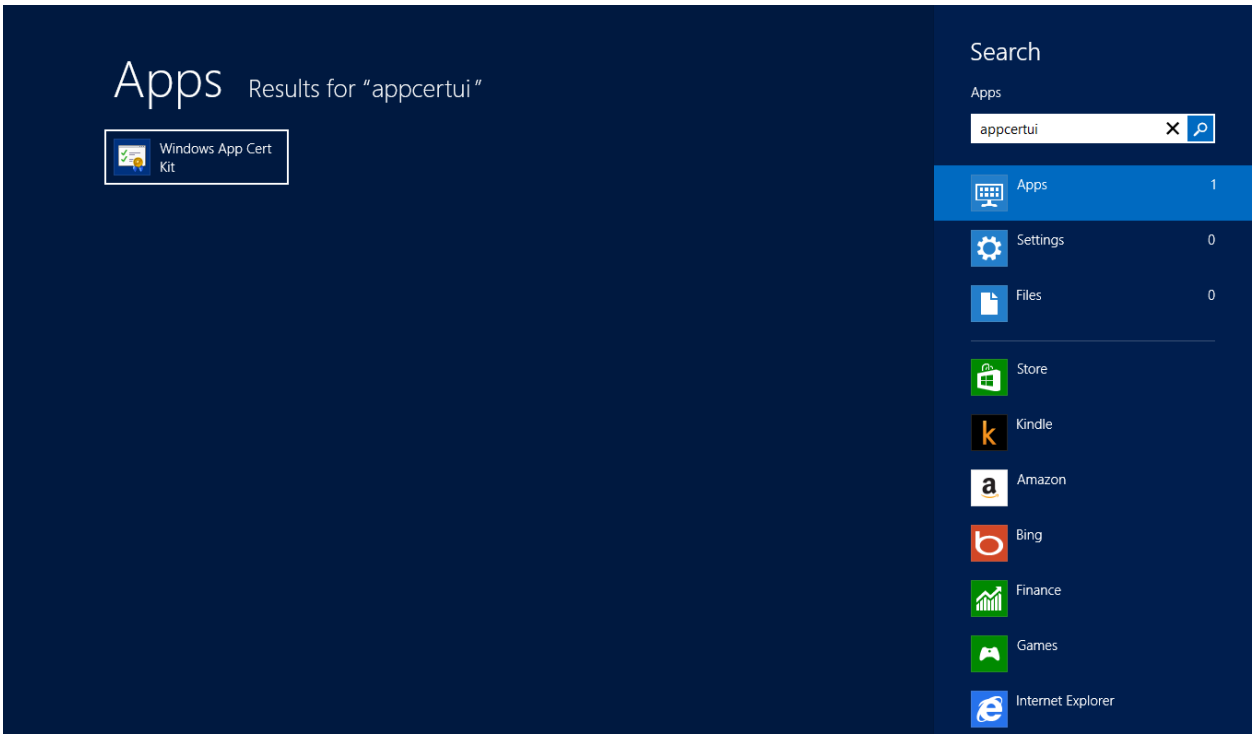


Figure 14 *– To find and execute the Windows App Certification Kit, one can search for "appcertui" in the App Apps screen.*

Step 4: When the WACK launches, you will have the opportunity to validate a Windows Store App, a Desktop App, or a Desktop Device App. Choose the first option.

Step 5: Locate and select the app to be validated in the resulting screen.

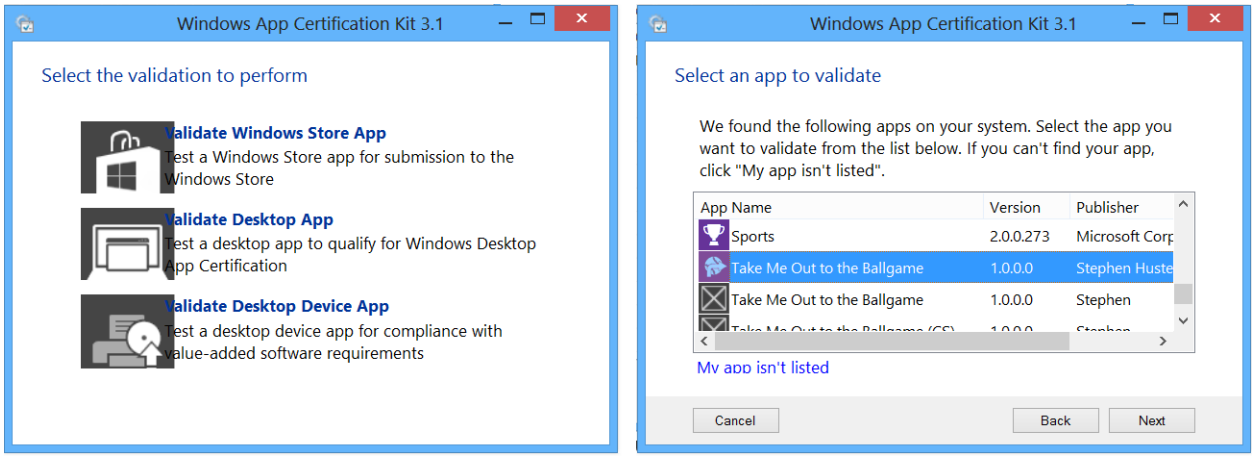


Figure 15 *– Select the top option of "Validate Windows Store App" when the Windows App Certification Kit launches (left). In the subsequent screen (right) find and select the app you want to test.*

Step 6: Click the Next button in the lower right. The test will commence, and you will likely see your app start and close several times. The test will take 10 to 15 minutes. When completed, a feedback screen will show whether the app passed or failed the test.

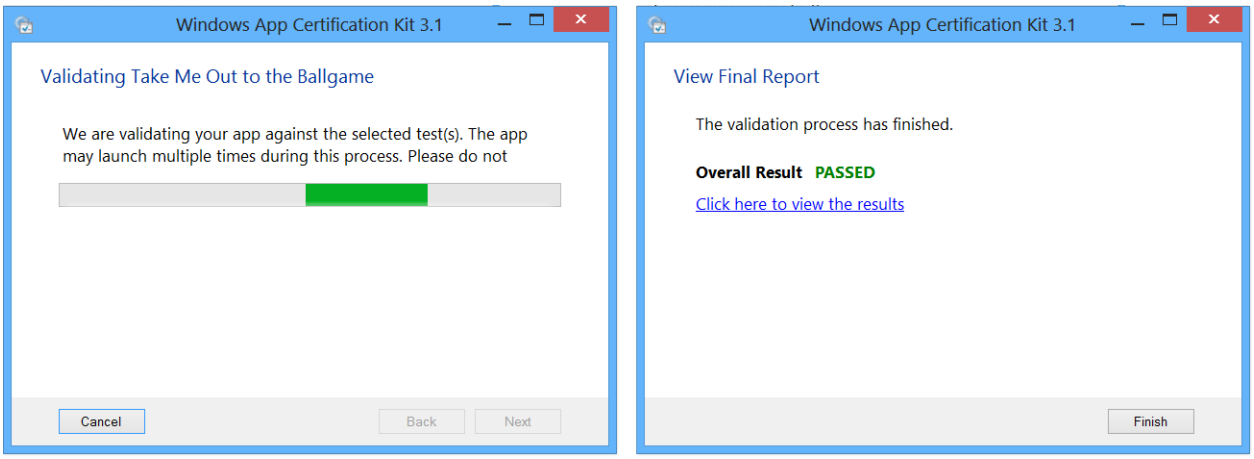


Figure 16 *– The test takes about 10 to 15 minutes and will likely launch and close the app several times (left). When complete, the overall result of "PASSED" or "FAILED" will be shown with an option to view the detailed results.*

If you passed the test, congratulations! You have overcome the technical hurdle of getting your app accepted to the Windows Store. If you failed, examine the detailed report, make the necessary changes, and then retest. (You may examine the detailed report if you passed as well to further relish in the result!)

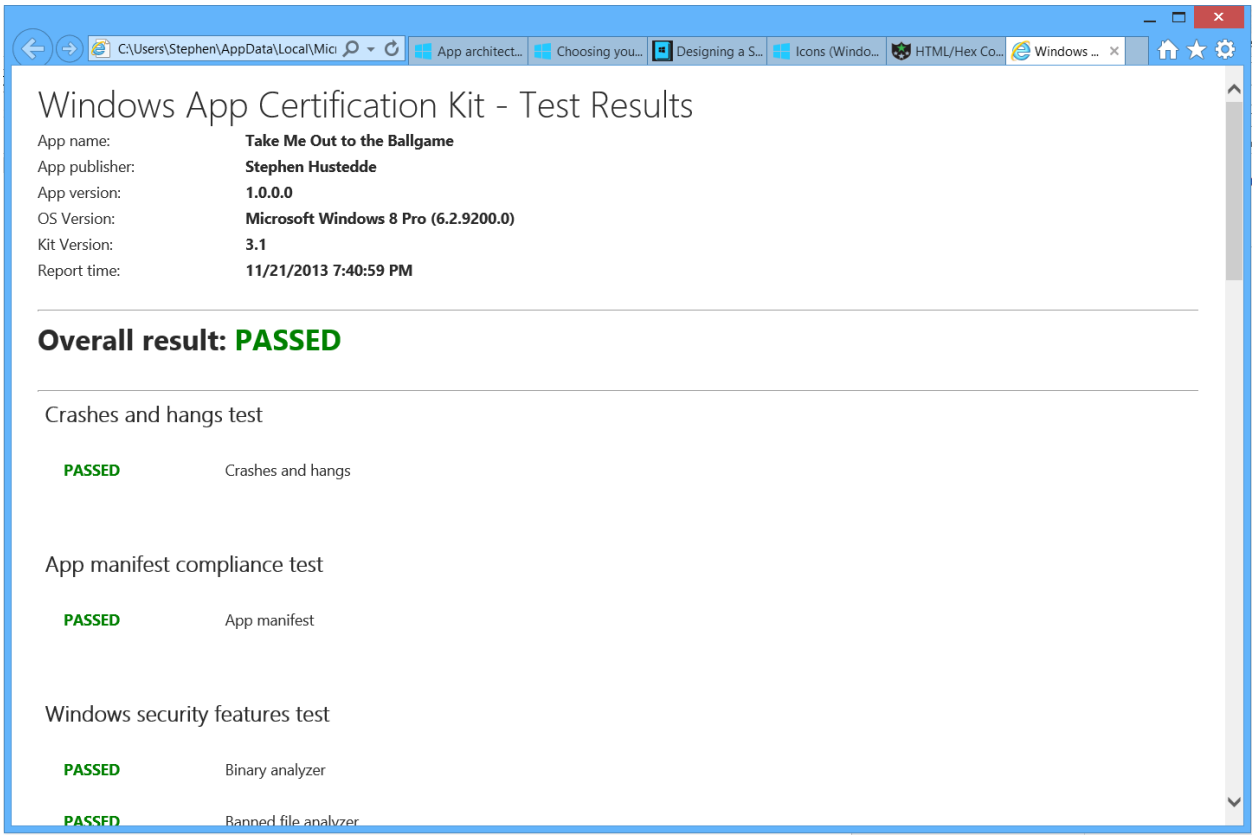


Figure 17 *- The detailed report of the WACK test results can be saved as an XML file and opened in a browser. The report will show the various areas tested. Use it to make appropriate corrections if the app did not pass.*

Submitting the App to the Windows Store

You will need to open a Developer account on the Windows Store to upload your project to the Store. This carries a cost of $49 for an individual account or $299 for a corporate account. Students of DreamSpark institutions may obtain a free license through the DreamSpark site.

To submit an app to the Windows Store, complete the following steps:

Step 1: From the Project menu, choose “Store” and “Open Developer Account”. It will launch the Windows Developer website. Sign in by clicking the link in the upper right and use your Microsoft Account login credentials.

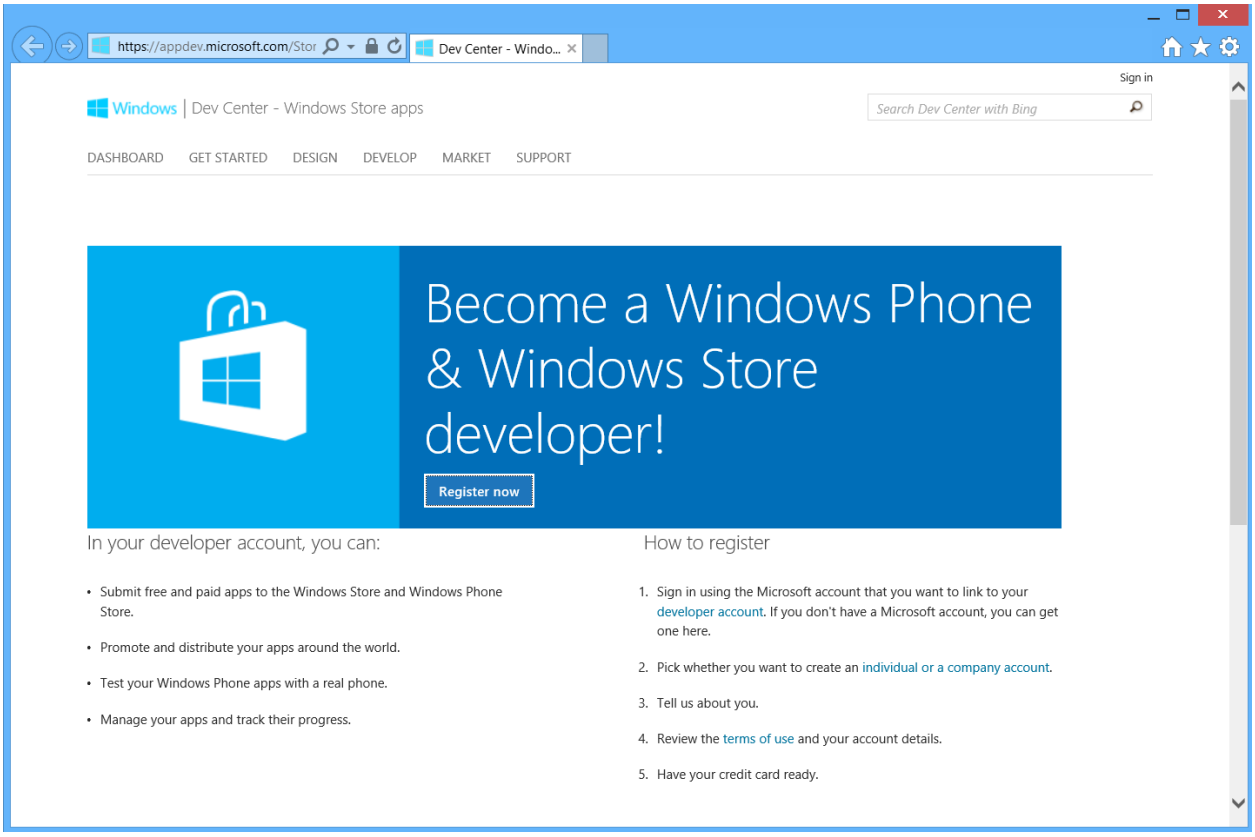


Figure 18 – To submit an app to the Store, you must open a Developer Account.

Step 2: Click the “Join Now” button in the resulting screen.

Step 3: Follow the online instructions to setup your account, providing the required contact information. You will need a credit card to purchase the account. If you plan to sell apps, you will need to provide tax and bank account information. Otherwise you can proceed to the step of reserving your app name in the Store.

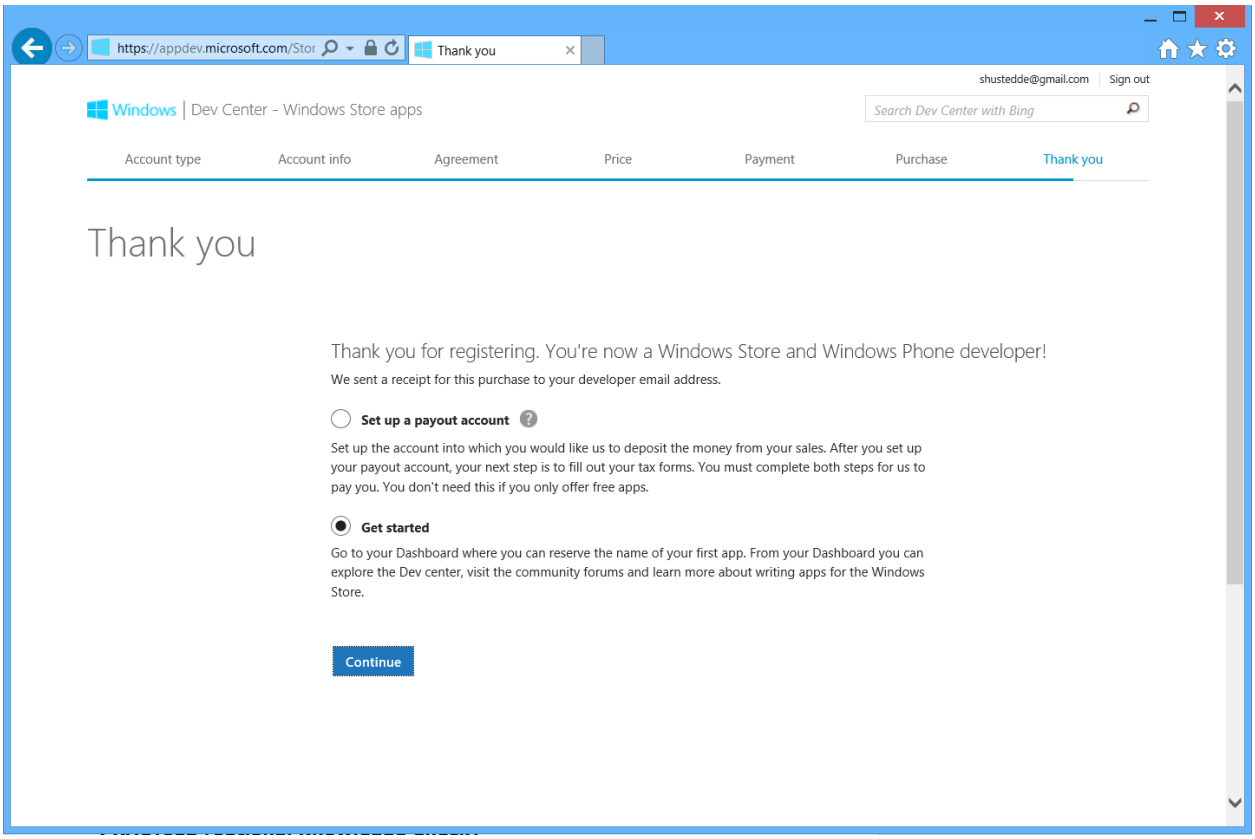


Figure 19 *–If you plan to sell apps, you will need to set up a payment account. If you are going to only offer free apps, you can proceed to the "Get Started" screen.*

Step 4: In Visual Studio, associate the app with the Windows Store. From the Project menu, choose 'Store' and then 'Associate App with Store'. Follow the instructions provided. You will be asked to sign in to your Developer Account.

Step 5: Take screen captures or your project. You must provide at least one screen capture for the Store. To accomplish this, go to the Project > Store > Capture Screenshots… menu item. This will build your app and launch the simulator. In the simulator, click the "Copy screenshot" button on the right to take the snapshot(s). You may upload up to nine images.

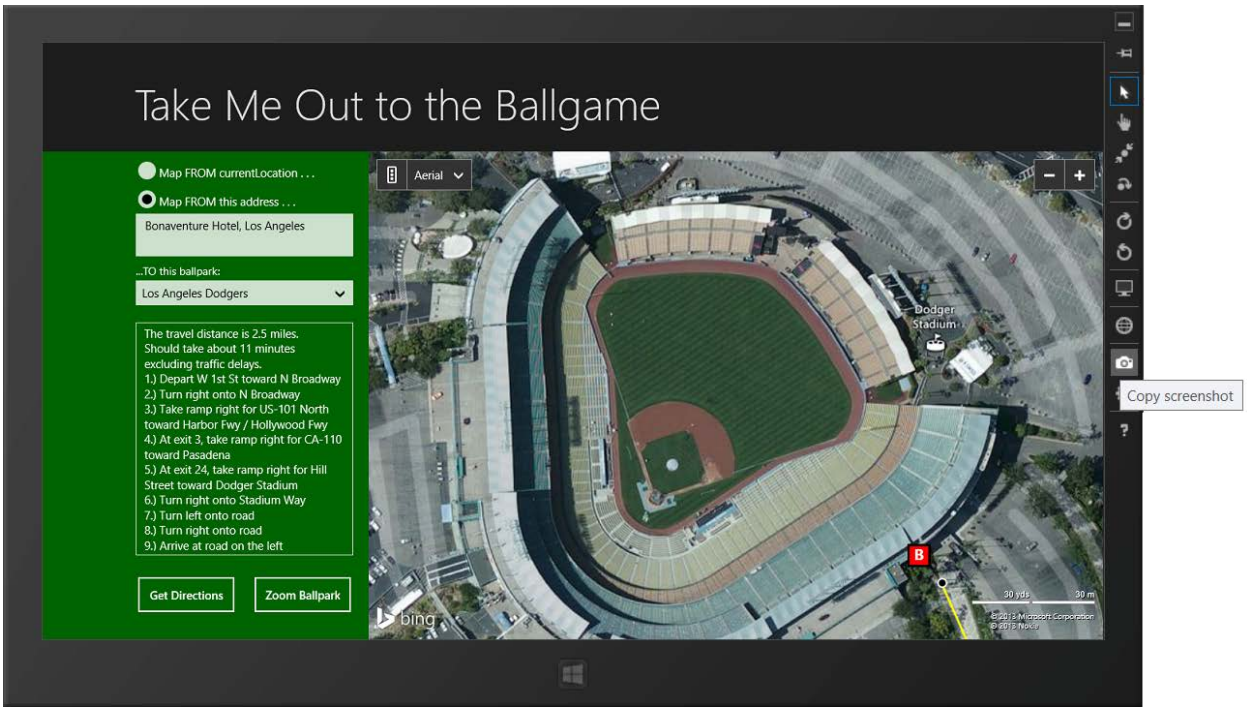


Figure 20 – Screenshots are taken form the simulator for submission to the Store.

Step 6: Create the App Package(s). Chose Project > Store > Create App Packages. Sign in to your Developer account and select the app and click 'Next'. In the resulting screen, choose the processors for which you desire to create target packages. For the "Take Me Out to the Ballgame" package, it was necessary to create three separate packages because of an issue with the Bing Maps. In most cases, the Neutral option can be selected to cover all three processors.

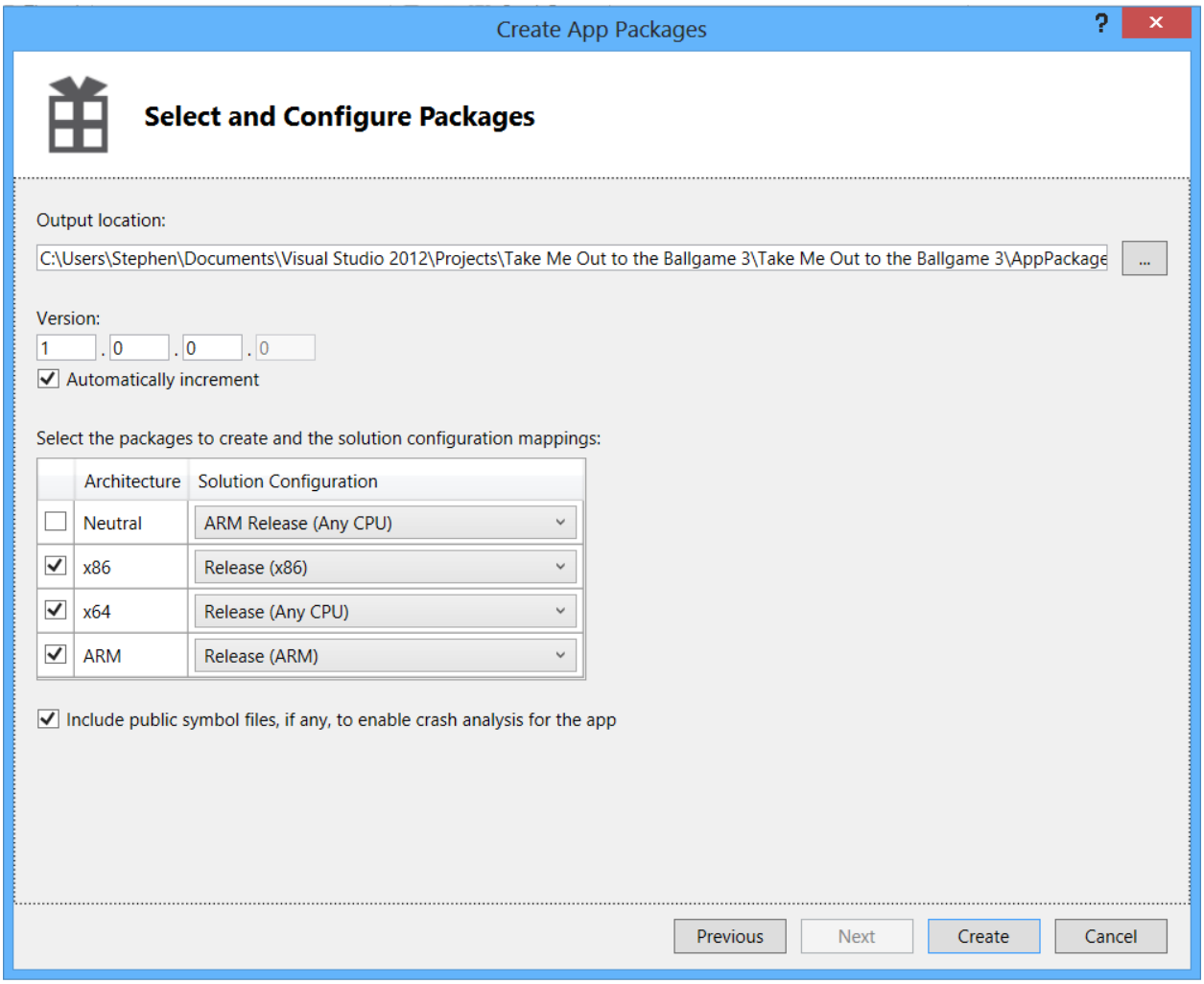


Figure 21 *– In building the packages, an option is provided to target specific processors.*

When completed, you may also choose to launch the WACK again. This is a good idea. Launching the WACK earlier was suggested to address any potential problems before you began the submission process.

Step 7: In your Developer Account, which you can access on the "Become a Windows Phone & Windows Store developer!" page from the Windows Dev Center – Windows Store Apps website, sign in and access the Dashboard. Open the link to the associated app.

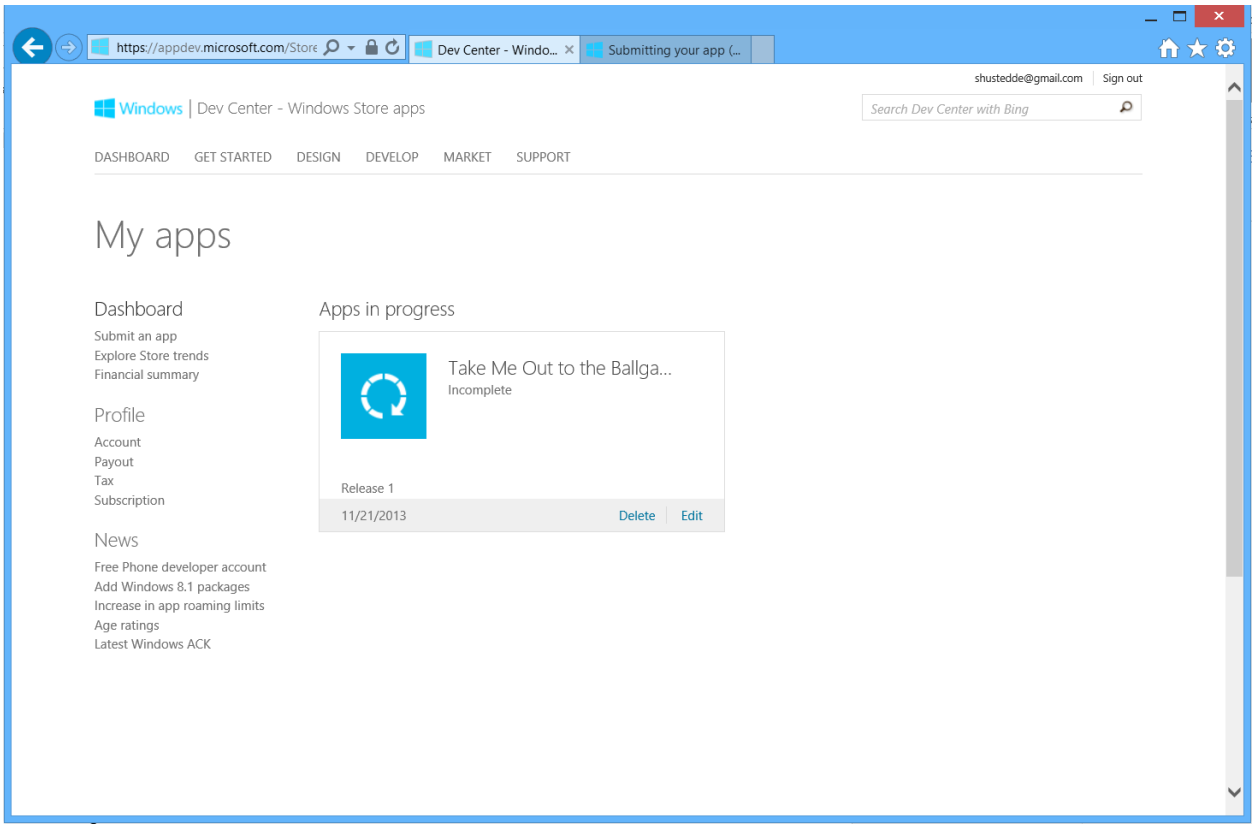


Figure 22 *– The Dashboard of the Windows Store Developer Account is where information on associated apps is edited and where the screen shots and packages are uploaded.*

Step 8: Edit the info on the app by clicking the Edit link in the lower right of the app reference. You will be asked to provide pricing information, declare services, provide an age rating, declare if any cryptography is utilized in the app, upload the packages, provide a description of the app for your customers, and add any notes you would like to give the testers.

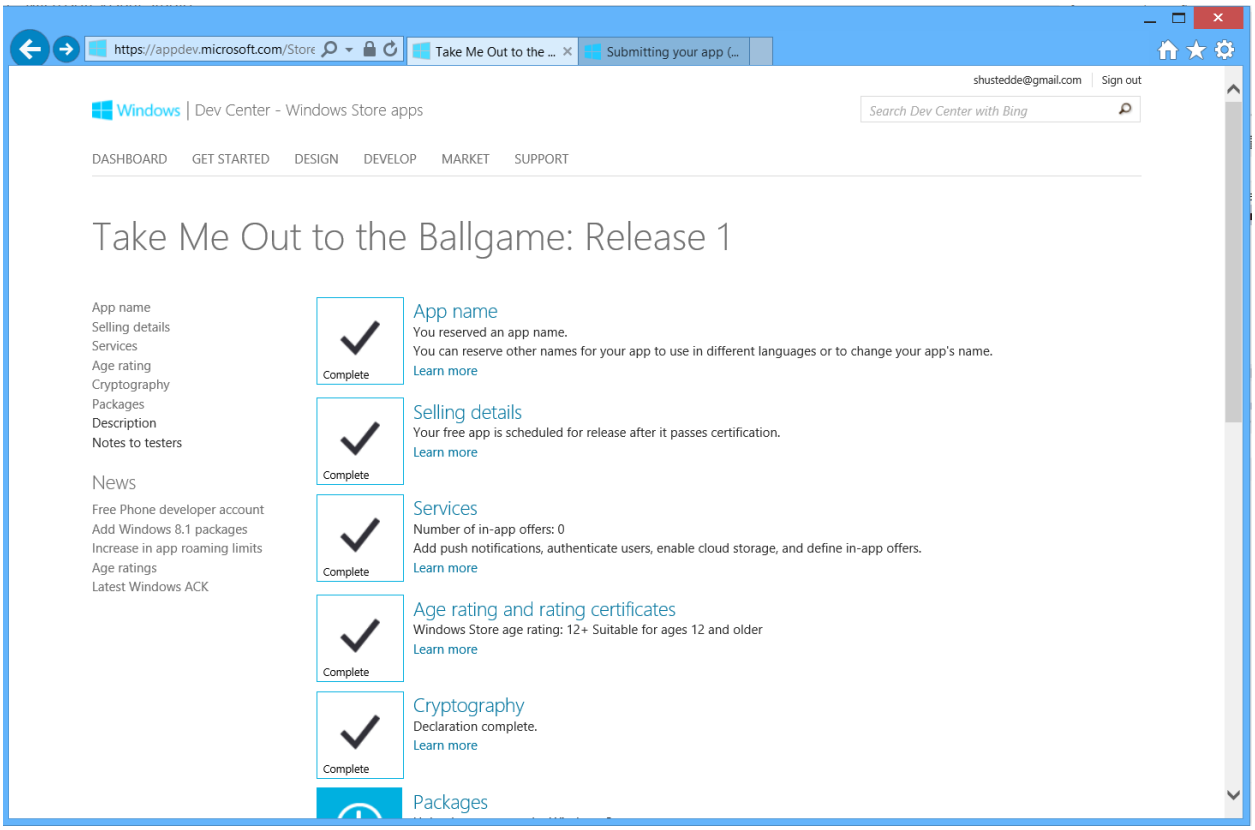


Figure 23 *– A checklist in the Dashboard for the app provides easy-to-follow steps in preparing your app for submission.*

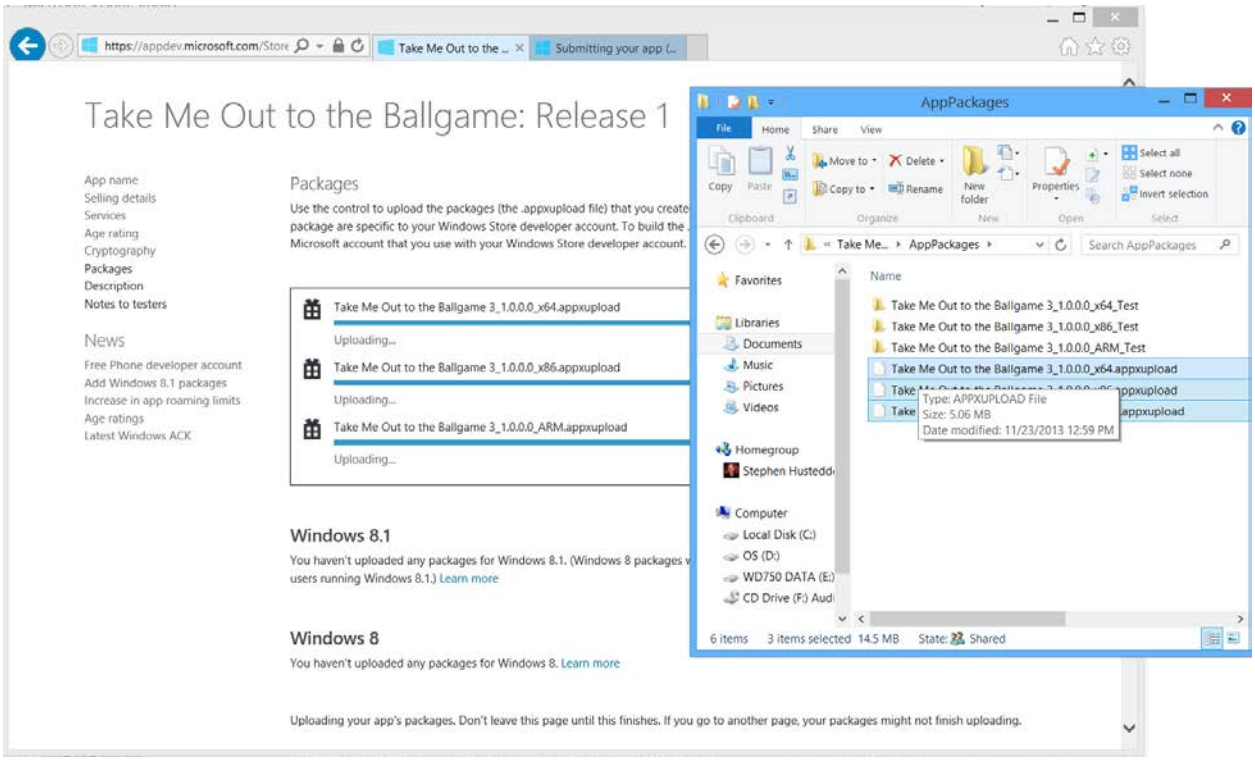
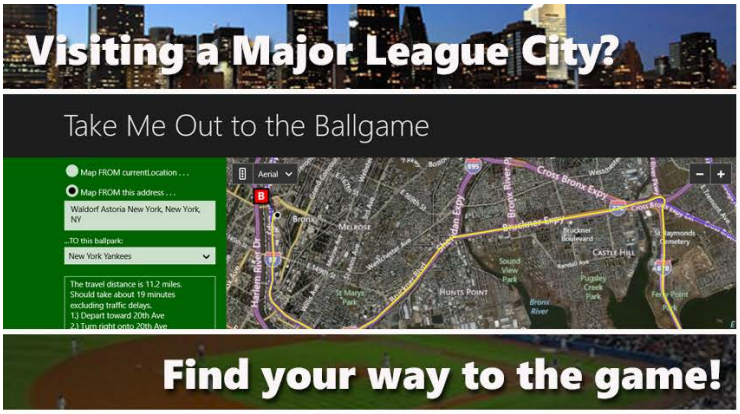
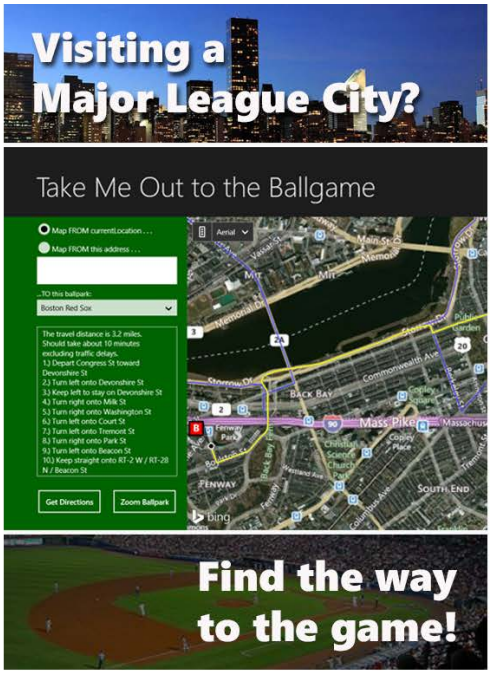


Figure 24 *– The package(s (extension appxupload) can be dragged from the projects AppPackages folder to the Developer account page.*

After uploading the packages, provide a description of the app, list any features you would like to point out, and upload the images created in step 5 of this section ("Submitting the App to the Windows Store"). The screen shots will appear in the User's Pictures Library in a Windows Simulator folder. The images must be 1366x768 in size and should not be modified before uploading; do not enhance them in Photoshop or add additional titling and so on. For each image, provide a description of up to 200 characters. You can also upload up to four promotional PNG graphics of sizes 846x468, 558x756, 414x468, 414x180. These may be used by the Microsoft Store staff for promotion of the app or if the app is featured in the Store. These are optional. Keywords for the search engine are also supplied, and it is a good idea to provide a privacy statement.



846x468



558x756

414x468

414x180

Figure 25 *– Provide up to four promotional images.*

Step 9: When all the above steps are in order, click the Submit button on the Dashboard. You can track the progress through the steps of pre-processing, security tests, technical compliance, content compliance, release, and signing and publishing. The process can take 5 or 6 days.

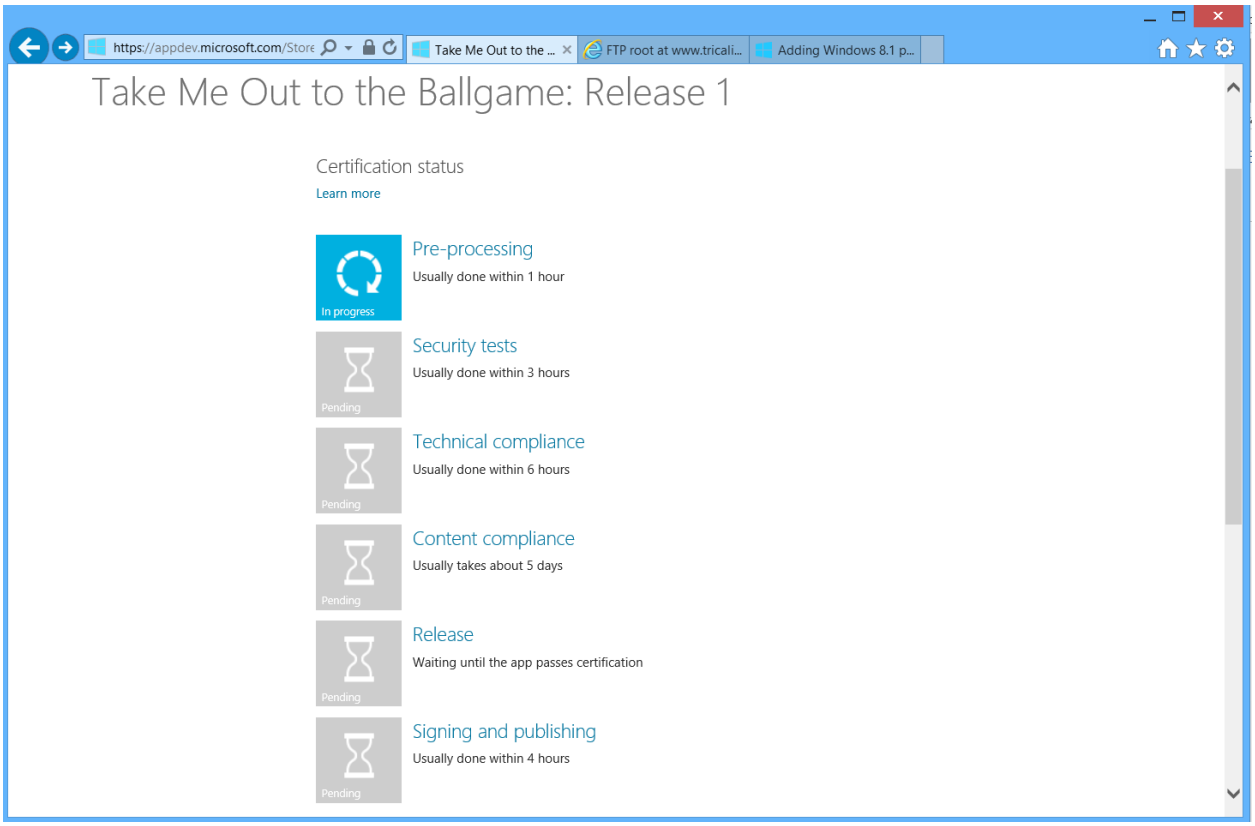


Figure 26 *– The approval process may take 5 or 6 days after the app is submitted. The progress can be tracked in the developer's Dashboard.*