

# Hands-on lab

---

## Lab: Hosted Web Apps

October 2015

## CONTENTS

<b>OVERVIEW .....</b>	<b>3</b>
<b>EXERCISE 1: CREATE A HOSTED WEB APP .....</b>	<b>5</b>
Task 1 – Create a blank Universal Windows JavaScript app .....	5
Task 2 – Host Codepen in a web app .....	7
Task 3 – Enable camera capture .....	11
Task 4 – Default and Live Tiles .....	12
Task 4 – Additional features .....	17
<b>EXERCISE 2: SUPPORT ADDITIONAL PLATFORMS AND DEVICES WITH MANIFOLDJS</b>	
<b>(OPTIONAL) .....</b>	<b>18</b>
Task 1 – Install ManifoldJS and create a manifest .....	18
Task 2 – Generate hosted web apps .....	18

# Overview

---

Following the release of Windows 10, the Windows Bridge toolkits and Project Westminister are opening up the UWP platform to developers with a broad range of development skills, such as iOS, Classic Windows, and web developers. The Windows Bridge for web apps allows you to easily transition your code for the web into the app space by publishing your responsive website to the Windows Store. Web apps and hosted web apps have the ability to call UWP APIs directly from JavaScript to integrate with features such as lives tiles, active notifications, contacts, Cortana voice commands, and Windows Store in-app purchases.

Hosted web apps immediately reflect changes made in your web codebase, making it easy to keep your content up to date.

After creating hosted web apps for Windows 10, you may be interested in expanding to other platforms. ManifoldJS is a new open source framework that generates hosted web apps for major platforms.

## Objectives

This lab will show you how to:

- Created a hosted web app
  - Send a toast notification from a hosted web app
  - Access the camera from the hosted app
  - Define custom image assets for default tiles and the splash screen
  - Update live tiles from the hosted app
  - Generate hosted web apps for major platforms with ManifoldJS
- 

## System requirements

You must have the following to complete this lab:

- Microsoft Windows 10

- Microsoft Visual Studio 2015
- 

## Optional add-ons

If you wish to complete the optional tasks in this lab, you will need:

- The Node Package Manager (npm)
  - ManifoldJS
- 

## Setup

You must perform the following steps to prepare your computer for this lab:

1. Install Microsoft Windows 10.
2. Install Microsoft Visual Studio 2015. Choose a custom install and ensure that the Universal Windows App Development Tools are selected from the optional features list.
3. Optional: Install npm.
4. Optional: Install ManifoldJS

*Instructions and links to install npm and ManifoldJS can be found in Exercise 3: Task 1.*

---

## Exercises

This Hands-on lab includes the following exercises:

1. Create a Hosted Web App
  2. Support Additional Platforms and Devices with ManifoldJS (Optional)
- 

Estimated time to complete this lab: **30 to 45 minutes.**

# Exercise 1: Create a Hosted Web App

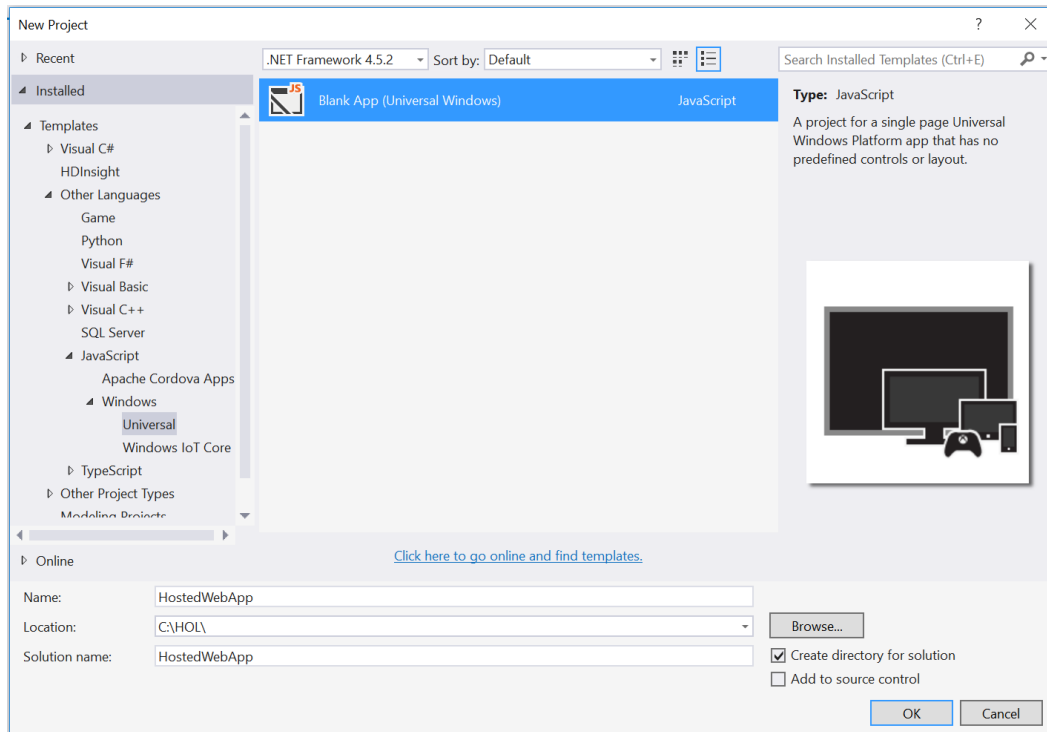
---

With a responsive site already online, you can create a hosted web app for the Windows Store in minutes. In this exercise, you will create a hosted web app using Codepen.io as an example. Codepen allows you to enter and execute custom JavaScript, CSS, and HTML. Once you allow Windows Runtime access in your hosted app, Codepen is a great way to test platform API integration without hosting the code yourself on a server. We will trigger a toast from the hosted app and add camera capture functionality.

## Task 1 – Create a blank Universal Windows JavaScript app

We will begin by creating a project from the UWP Blank App JavaScript template.

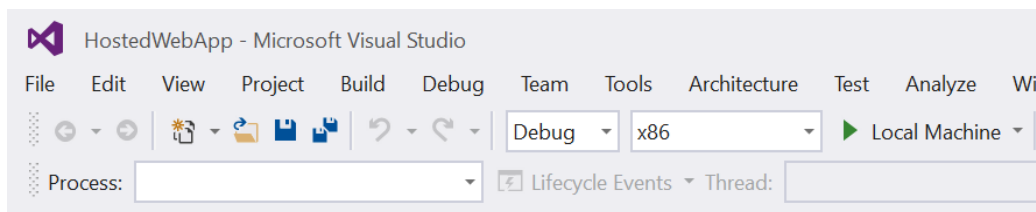
1. In a new instance of Visual Studio 2015, use **File > New> Project** to open the New Project dialog. Navigate to **Installed > Templates > JavaScript** and select the **Blank App (Universal Windows)** template.
2. Name your project **HostedWebApp** and select the file system location where you save your Hands-on Lab solutions. We have created a folder in our **C:** directory called **HOL** that you will see referenced in screenshots throughout the labs.
3. Leave the option selected to **Create directory for solution**. You may deselect **Add to source control** if you don't wish to version your work. Click **OK** to create the project.



**Figure 1**

*Create a new Blank App project in Visual Studio 2015.*

4. Set your **Solution Configuration** to **Debug** and your **Solution Platform** to **x86**. Select **Local Machine** from the Debug Target dropdown next to the Start Debugging Button.

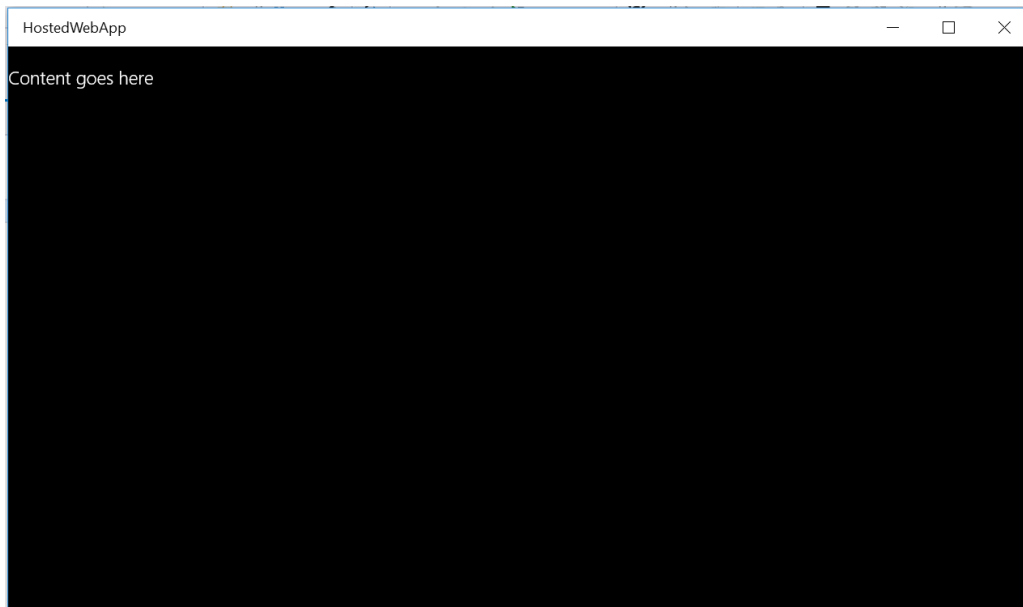


**Figure 2**

*Configure your app to run on the Local Machine.*

**Note:** ► is the Start Debugging button.

5. Use the Start Debugging button to build and run your app. You will see a black app background with the text “Content goes here.”



**Figure 3**

*The blank universal JavaScript app running in Desktop mode.*

6. Stop debugging and return to Visual Studio.

---

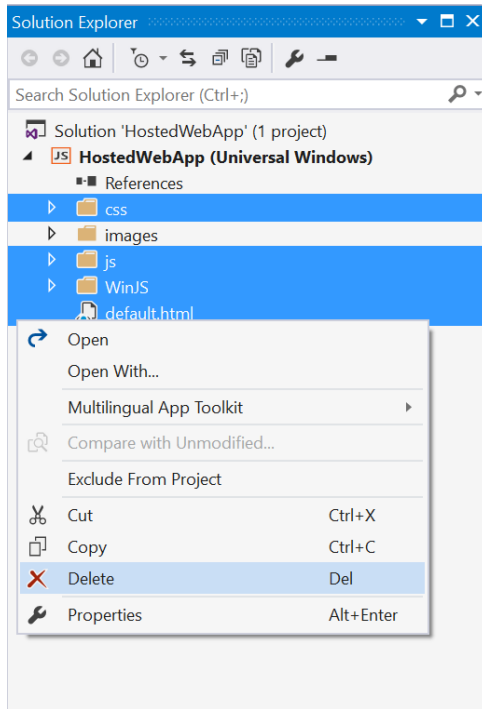
## Task 2 – Host Codepen in a web app

A hosted web app provides a wrapper for a website that serves up its content using the Edge rendering engine. You can limit the scope of your users' browsing to a specific site or sites and force other links to open externally in the default browser. In this task, you will display the Codepen site as a hosted web app and trigger a toast notification from the app.

1. Delete the **css**, **js**, and **WinJS** folders and the **default.html** file from the **HostedWebApp** project.

**Note:** When creating a web app that contains only hosted content, you may delete the **css**, **js**, and **WinJS** folders as well as the **default.html** file. We've deleted these files, but you might choose to keep them and use them to implement an offline landing page for your app.

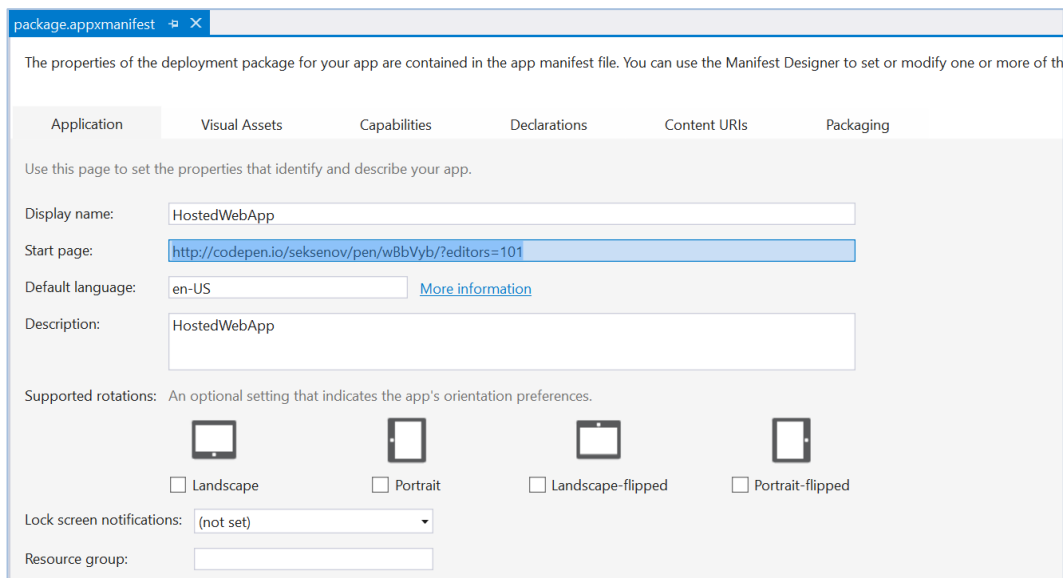
We will leave the **images** folder in place, because it provides app images like the splash screen and store logo that are still relevant to a hosted app.



**Figure 4**

*Delete files and folders that aren't needed by a hosted web app.*

2. Open **package.appxmanifest** in your **HostedWebApp** project with the manifest editor.
3. On the **Application** tab, change the **Start page** to **<http://codepen.io/seksenov/pen/wBbVyb/?editors=101>**.



**Figure 5**



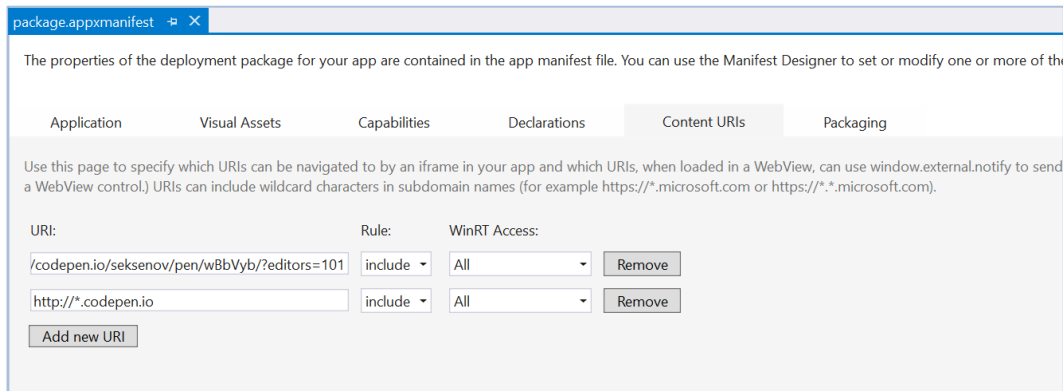
*Set Codepen as the Start page.*

4. Browse to the **Content URIs** tab and add **`http://codepen.io/seksenov/pen/wBbVyb/?editors=101`** to the URI field. Leave the **Rule** set to **include**, and set **WinRT Access** to **All**.

**Note:** The Application Content URI Rules (ACURS) for your app dictate the pages that are hosted or allowed by the app. For instance, you may wish your users to be able to browse Codepen within the app but force external links to open in a browser. These inclusions and exclusions allow you to control the boundaries of your app and prevent it from behaving like a standard web browser. Content URIs also give you the ability to turn Windows Runtime access on or off for different parts of the app and to decide if that access should be given for **None**, **All**, or **Allow for web only**.

To specify a remote URI, use the **`http://`** protocol. To specify a local URI, use the **`ms-appx-web:///`** protocol.

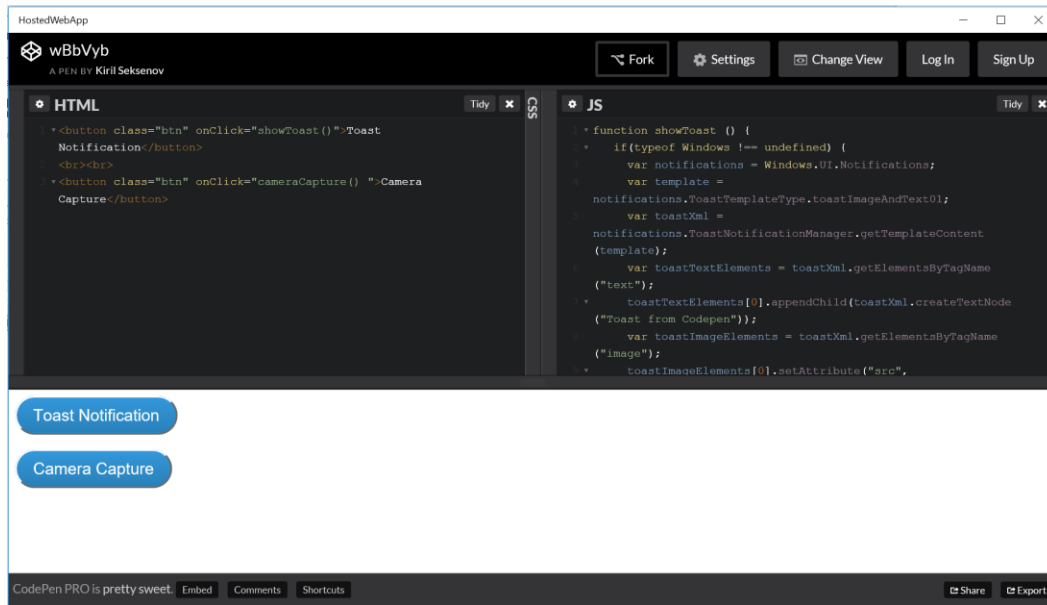
5. Add **`http://*.codepen.io/`** as an additional Content URI with the same **Rule** and **WinRT Access** settings as the first URI. The asterisk is a wildcard that allows you to target all potential subdomains.



**Figure 6**

*Add the Codepen Content URIs.*

6. Build and run your app. You will see Codepen appear in the app window with HTML and JavaScript content and code. The code to generate a toast has been provided for you.

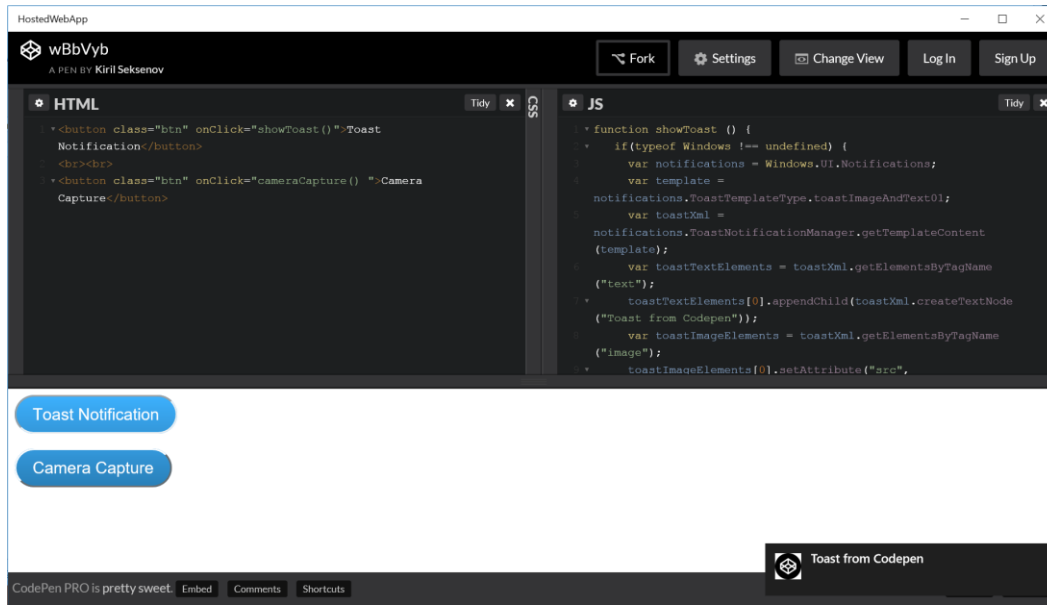


**Figure 7**

*Codepen as a hosted web app.*

7. Use the Toast Notification button to send a toast to your system. We set **WinRT Access** to **All** for **Codepen**, so the code in the JavaScript pane has permission to access Windows APIs, including those needed for notifications.

**Note:** We are using Codepen as a convenient tool to enter custom JavaScript on a remote server that we can immediately host and test in our app. In a real-world scenario, the **showToast()** function would run as a script inside a web project, which you would deploy to a server to make it live. You could then create a separate hosted web app project and use ACURS to host the live content within an app.



**Figure 8**  
A toast notification generated by the hosted web app.

### Task 3 – Enable camera capture

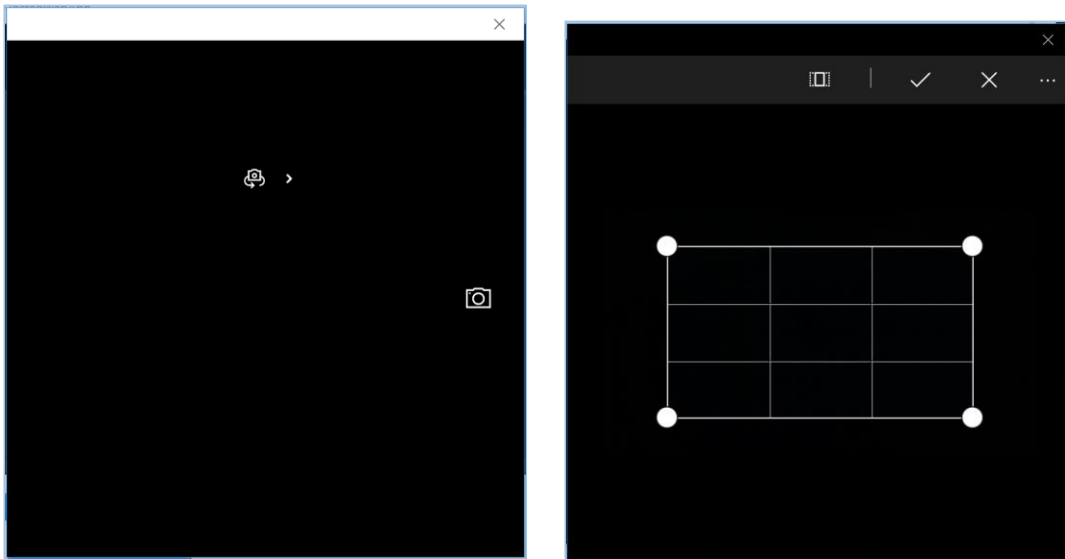
Although there is a Camera Capture button defined in the HTML for this pen, camera capture has not been implemented. We will add the necessary code to enable it.

1. With your app still running, add a function below the `systemAlertCommandInvokedHandler()` in the **JS** pane to handle camera capture.

**JavaScript**

```
function cameraCapture() {
    if (typeof Windows !== 'undefined')
    {
        var captureUI = new Windows.Media.Capture.CameraCaptureUI();
        //Set the format of the picture to be captured (.png, .jpg, ...)
        captureUI.photoSettings.format =
            Windows.Media.Capture.CameraCaptureUIPhotoFormat.png;
        //Pop up the camera UI to take a picture
        captureUI.captureFileAsync(
            Windows.Media.Capture.CameraCaptureUIMode.photo).then(
            function(capturedItem) {
                // Do something with the picture
            });
    }
}
```

2. Use the **Camera Capture** button to launch the camera interface. If prompted to allow access to the camera, choose **Yes**. When the camera interface opens, you have the option to take a picture with the camera button on the right side of the window.



**Figure 9**

*Taking a picture with the camera opens the photo editing dialog.*

**Note:** Save behavior isn't implemented in the **then** statement, so any picture you take will be temporary.

3. Stop debugging and return to Visual Studio.

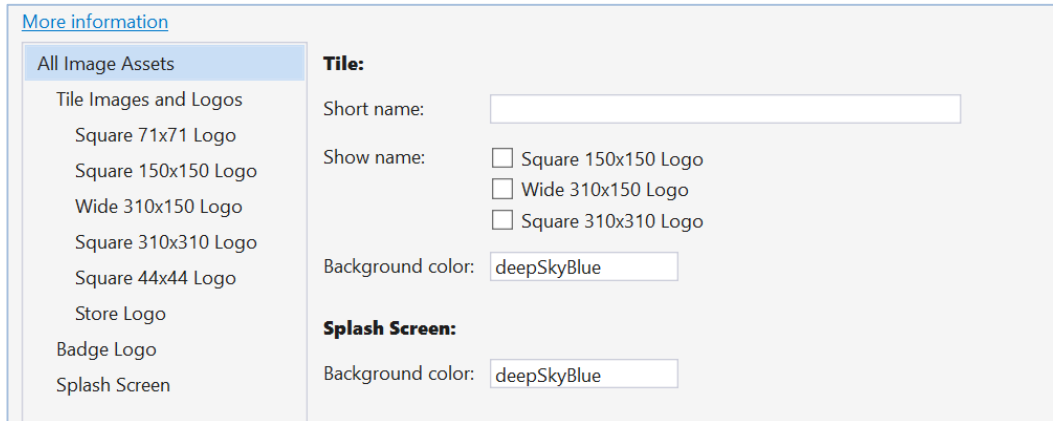
#### Task 4 – Default and Live Tiles

As with other UWP apps, you can define app images for hosted web apps. In this task, we will add visual assets for the splash screen, the Medium default tile, and the Start menu. After customizing the default tile, we will create and update a Live Tile from the app.

1. Open the **package.appxmanifest** in the manifest editor and select the **Visual Assets** tab.
2. Use the ... ellipsis under the **Square 71x71 logo** at **Scale 200** to open the image picker. Browse to the **Lab Assets** folder in your Hands-on Labs directory and select the **Square71x71Logo.scale-200.png** file. Choose **Open** to replace the default logo with the new selected logo. If prompted, agree to overwrite the existing files.
3. Repeat Step 1 with the **Square150x150Logo** at **Scale 200**, the **Square44x44Logo** at **Scale 200**, and the **Splash screen logo** at **Scale 200**, respective to each appropriate asset in the manifest.

**Note:** We are only adding selected logo assets for the purposes of this demo. For a more in-depth look at Lives Tiles and best practices, check out the **Live Tiles and Notifications** lab.

4. With **All Image Assets** selected, change the **Background color** fields for both the tile and the start screen to **deepSkyBlue**. Scroll down to verify that your new images appear in the visual assets in the manifest.

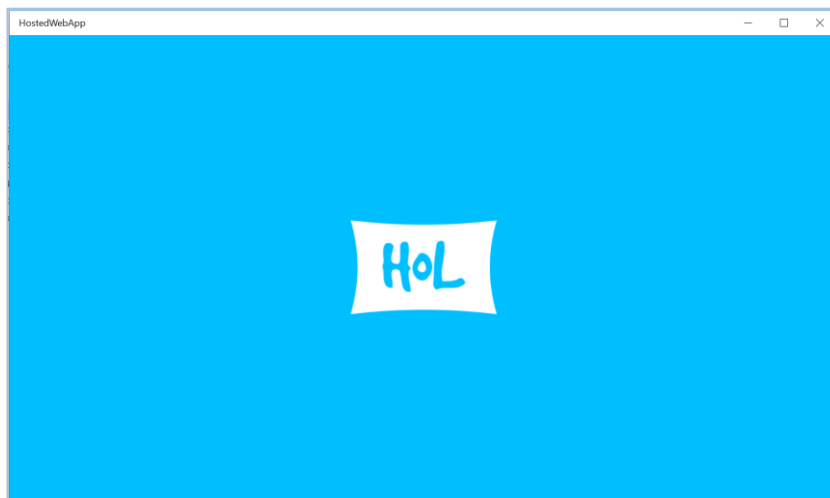


The screenshot shows the 'More information' tab in the Visual Studio manifest editor. On the left, a sidebar lists 'All Image Assets' with sub-items: 'Tile Images and Logos', 'Square 71x71 Logo', 'Square 150x150 Logo', 'Wide 310x150 Logo', 'Square 310x310 Logo', 'Square 44x44 Logo', 'Store Logo', 'Badge Logo', and 'Splash Screen'. The main area is divided into two sections: 'Tile:' and 'Splash Screen:'. Under 'Tile:', there is a 'Short name:' text box, a 'Show name:' section with three checkboxes (all unchecked) for 'Square 150x150 Logo', 'Wide 310x150 Logo', and 'Square 310x310 Logo', and a 'Background color:' text box containing 'deepSkyBlue'. Under 'Splash Screen:', there is a 'Background color:' text box also containing 'deepSkyBlue'.

**Figure 10**

*Set background colors for tiles and the splash screen.*

5. Build and run your app. While your app is loading, you will see the new splash screen with a blue background and the white HoL logo in the foreground.

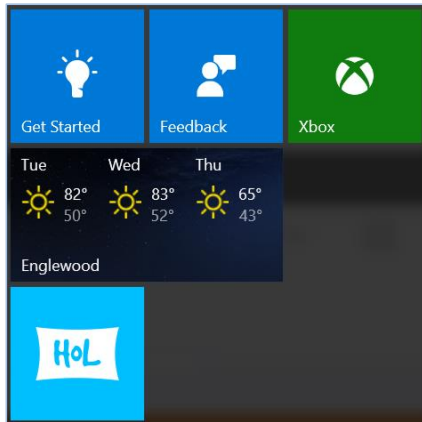


**Figure 11**

*The new, custom splash screen in the hosted web app.*

6. With your app still running, find **HostedWebApp** in the list of **All apps** in the Start menu. Right-click on the app name and choose **Pin to Start**. If your tile defaults to any size other than

medium, resize it to Medium. You will see the DeepSkyBlue background with the HoL logo in white.



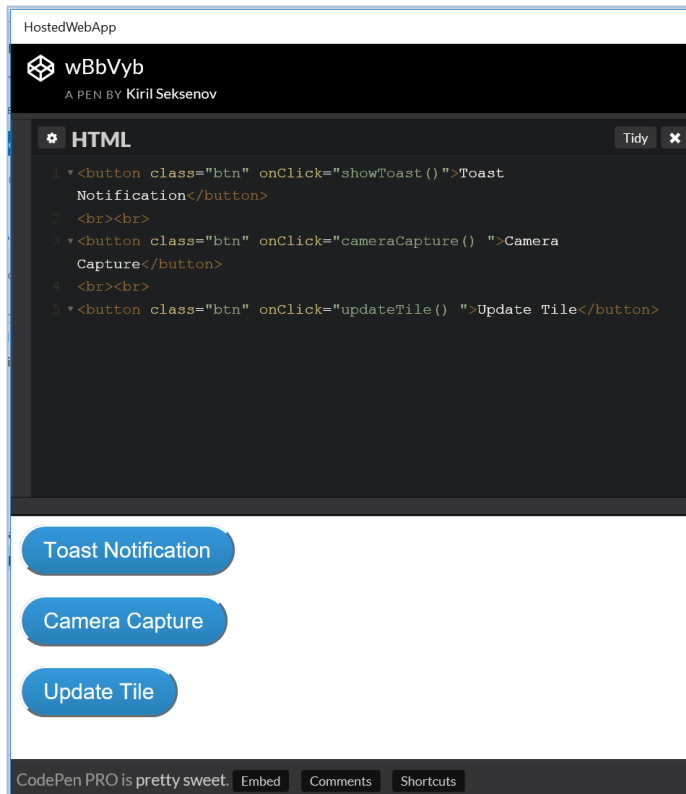
**Figure 12**

*The Medium default tile.*

7. Return to your running app. In the HTML pane, add a button that calls a method named **updateTile()** on the click event.

#### HTML

```
<button class="btn" onClick="cameraCapture() ">Camera Capture</button>
<br><br>
<button class="btn" onClick="updateTile() ">Update Tile</button>
```



**Figure 13**

*The Update Tile button in the HTML pane.*

8. Add the **updateTile()** function in the **JS** pane below the cameraCapture() function.

```

JavaScript

function updateTile() {
    if (typeof Windows !== 'undefined' && typeof Windows.UI !== 'undefined' &&
        typeof Windows.UI.Notifications !== 'undefined')
    {
        console.log('Attempting to update the tile');
        var notifications = Windows.UI.Notifications,
            tile =
notifications.TileTemplateType.tileSquare150x150PeekImageAndText01,
            tileContent = notifications.TileUpdateManager.getTemplateContent(
                tile),
            tileText = tileContent.getElementsByTagName('text'),
            tileImage = tileContent.getElementsByTagName('image');

        tileText[0].appendChild(tileContent.createTextNode('Demo Message'));
        tileImage[0].setAttribute('src',
            'http://unsplash.it/150/150/?random');
        tileImage[0].setAttribute('alt', 'Random demo image');
    }
}

```

```

var tileNotification = new
    notifications.TileNotification(tileContent);
var currentTime = new Date();
tileNotification.expirationTime = new Date(currentTime.getTime() + 20
    * 1000);

notifications.TileUpdateManager.createTileUpdaterForApplication(
    ).update( tileNotification);

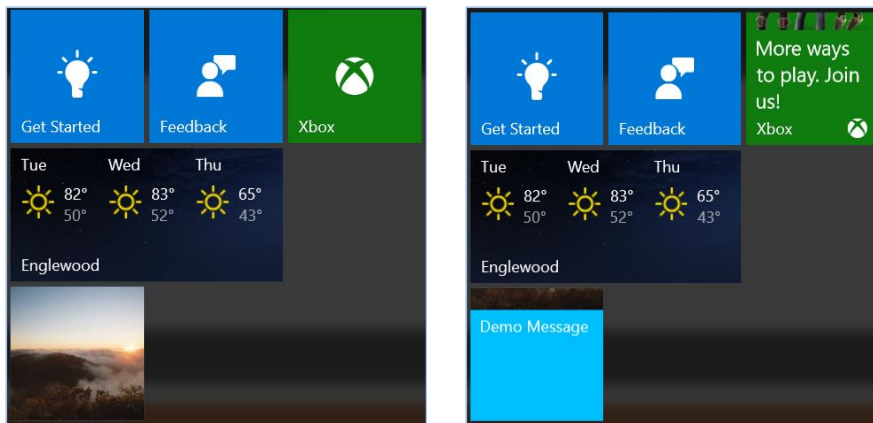
}
else {
    //alternate behavior
}
}

```

**Note:** Every time we add a method to hosted content that will access platform APIs, we also add a conditional to check if **typeof Windows** is defined. This condition evaluates to true if the site is running as a hosted web app on a Windows 10 device. Additionally, in the `updateTile()` method, we will also check if **Windows.UI** and **Windows.UI.Notifications** are defined before proceeding.

For simplicity, we are using a legacy template to quickly create the tile. To learn about customizing tiles with adaptive tile templates and tile XML, visit the **Live Tiles and Notifications** lab.

9. Use the **Update Tile** button to trigger the **Live Tile** update. When you open your Start menu, you will see an image slide over your tile as it goes live. After a few seconds, the demo message text will slide over the image.



**Figure 14**

*Live Tile content.*

10. Wait 20 seconds while watching your tile. When the notification expires, your tile will revert to the default tile.



**Note:** There is a console.log message inside the updateTile() method. If you are having trouble getting the tile to update, check your JavaScript Console window in Visual Studio to see if the message appears. If your console prints **Attempting to update the tile**, you may have a syntax error within the method.

11. Stop debugging and return to Visual Studio.

---

#### Task 4 – Additional features

In addition to the scenarios we've covered in this lab, hosted web apps can integrate with many other features to provide a comprehensive Windows 10 user experience. Although we won't cover them in depth in this lab, here are some features that may interest you. For more information, visit <http://microsoftedge.github.io/WebAppsDocs/en-US/win10/HWAfeatures.htm>.

##### Cortana Voice Commands

You can integrate a hosted web app with Cortana by specifying a Voice Command Definition (VCD) file in your html using a **meta** tag. Once registered, you can use the VCD to launch your app via voice command, interact with background tasks, and perform other Cortana interactions.

##### Hybrid Apps

If you would like your users to have access to your hosted web app even when offline, you may wish to create a Hybrid app. A hybrid app can serve up hosted content when available and default to local content within the app package when offline.

##### Web Authentication Broker

Your hosted app can take advantage of the web authentication broker to handle the login flow for your users if you provide access with internet protocols like **OpenID** and **OAuth**. The webauth URIs can be defined in a **meta** tag on any html page in your app.

##### App Capability Declarations

To provide access to the microphone or other devices or resources, you must declare the appropriate capabilities in the app manifest. You can specify capabilities through the manifest editor interface or manually in the XML manifest. For more information, visit <https://msdn.microsoft.com/en-us/library/windows/apps/br211477.aspx>.

---

## Exercise 2: Support Additional Platforms and Devices with ManifoldJS (optional)

---

Hosted web apps are a great way to quickly bring your existing responsive web projects to new platforms. ManifoldJS is a tool that uses existing metadata from your website to generate native hosted apps for a variety of platforms, including iOS, Android, Windows 10, Chrome OS, and Firefox OS. For platforms that don't support hosted web apps natively, ManifoldJS uses Cordova.

The manifest generated by ManifoldJS follows the W3C standard for web app manifests and includes metadata such as the start page for the site, URL whitelist, site name, theme color, and app images.

**Note:** For the latest on ManifoldJS, visit <http://www.manifoldjs.com/>. You can read more about the W3C manifest for web apps at <https://w3c.github.io/manifest/>.

### Task 1 – Install ManifoldJS and create a manifest

Install ManifoldJS and generate a manifest.

1. Open a command prompt as Administrator. With the node package manager installed, use the command **npm install -g manifoldjs** to install ManifoldJS globally on your development machine.

#### Command Prompt

```
> npm install -g manifoldjs
```

**Note:** Visit <https://nodejs.org/> to download and install the node package manager (npm).

2. Generate a manifest for your site at <http://www.manifoldjs.com/generator>. You may also upload a manifest, and the generator tool will fix and alert you to any gaps it may have.

**Note:** If your site doesn't have a manifest, ManifoldJS will generate one for you. However, you may still wish to create your own to take advantage of your site's branding and provide app images.

3. Upload the manifest to the root directory of your site on the server. The manifest typically lives in the same location as your index.html file.

---

### Task 2 – Generate hosted web apps

In this task, you will generate hosted web apps from your website for a variety of platforms.

4. Return to your local machine. Create a directory to hold your hosted web apps and navigate to it in the command prompt. Pass your live website URL into manifoldjs to generate the manifest.

We will use Bing for the sake of this example. You may optionally use the **-l debug** options for more verbose output.

#### Command Prompt

```
> manifoldjs http://www.bing.com/
```

5. Explore the code generated by ManifoldJS in your hosted apps folder.
6. To install and run the generated Windows10 app, run the following command in the app folder created by ManifoldJS:

#### Command Prompt

```
> manifoldjs run windows
```

7. Your generated app will install and launch.

---

## Summary

---

Hosted web apps provide a powerful option to integrate your existing web projects with the Windows Store and platform APIs. In this lab, we created a hosted web app with custom visual assets that can trigger notifications, update tiles, and launch the camera on your device. We also explored options for generating hosted web apps for a variety of other platforms.