

Hand­s-on lab

Lab: WebViews and Hosted Web Apps

September 2015

Contents

[Overview 3](#_Toc429552071)

[Exercise 1: Create a Simple Hosted Web App 5](#_Toc429552072)

[Task 1 – Create a blank Universal Windows JavaScript app 5](#_Toc429552073)

[Task 2 – Host Bing in a web app 7](#_Toc429552074)

[Exercise 2: Interact with Platform APIs 9](#_Toc429552075)

[Task 1 – Create the native project 9](#_Toc429552076)

[Task 2 – Add a webview to the web project 13](#_Toc429552077)

[Task 3 – Add a new HTML page and a script to generate toast 14](#_Toc429552078)

[Exercise 3: Support Additional Platforms and Devices with ManifoldJS (optional) 18](#_Toc429552079)

[Task 1 – Install ManifoldJS and create a manifest 18](#_Toc429552080)

[Task 2 – Generate hosted web apps 18](#_Toc429552081)

[Summary 19](#_Toc429552082)

Overview

Following the release of Windows 10, the Windows Bridge toolkits are opening up the UWP platform to Android, 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. 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. You may mix local and remote content to provide offline backups for the site.

After creating web apps for Windows, 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

* 1. This lab will show you how to:
  + Created a hosted web app
  + Serve up remote and local content
  + Mix online and offline content
  + Call platform APIs from the web app
  + Generate hosted web apps with ManifoldJS

# System requirements

* 1. You must have the following to complete this lab:
  + Microsoft Windows 10
  + Microsoft Visual Studio 2015

# Optional add-ons

* 1. If you wish to complete the optional tasks in this lab, you will need:
  + The Node Package Manager (npm)
  + ManifoldJS

# Setup

* 1. You must perform the following steps to prepare your computer for this lab:
  2. Install Microsoft Windows 10.
  3. 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.
  4. Optional: Install npm.
  5. Optional: Install ManifoldJS

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

# Exercises

* 1. This Hands-on lab includes the following exercises:
  2. Add a Simple Hosted Web App
  3. Integrate with Platform APIs
  4. Support Additional Platforms and Devices with ManifoldJS (Optional)
  5. Estimated time to complete this lab:  **45 to 60 minutes**.

Exercise 1: Interact with Platform APIs

* 1. In Windows 10, you have the ability to create a seamless user experience for web apps by interacting with platform APIs. In this exercise, you will trigger a toast notification from a web app that includes content displayed through a WebView. Although this content is local rather than hosted, it demonstrates how JavaScript running in the WebView can call C# code in your native project.

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 **WebApp** 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 options selected to **Create new solution** and **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.

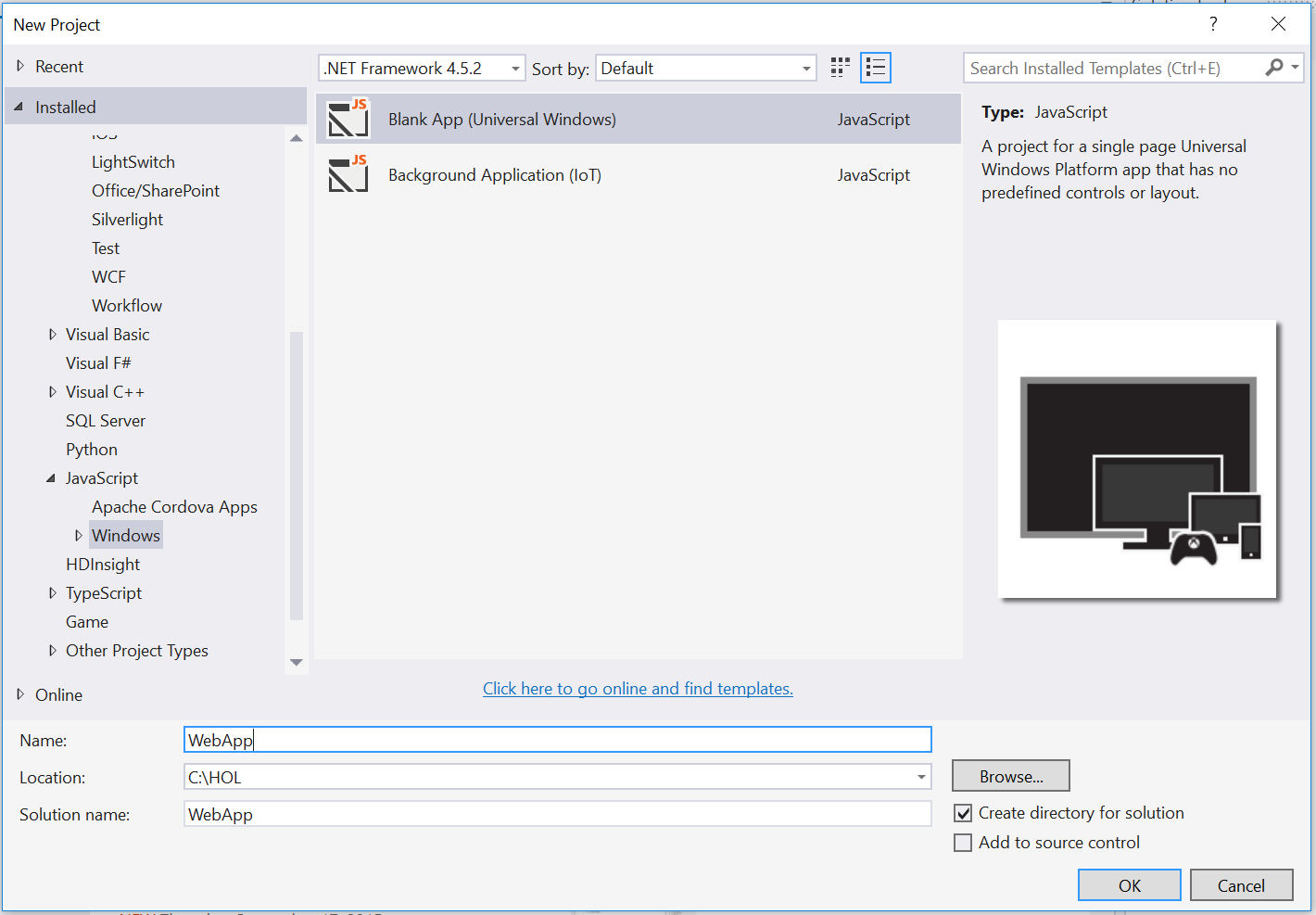


Figure 2

Create a new Blank App project in Visual Studio 2015.

1. 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 3

* + 1. Configure your app to run on the Local Machine.
  1. **Note:**  is the Start Debugging button.

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

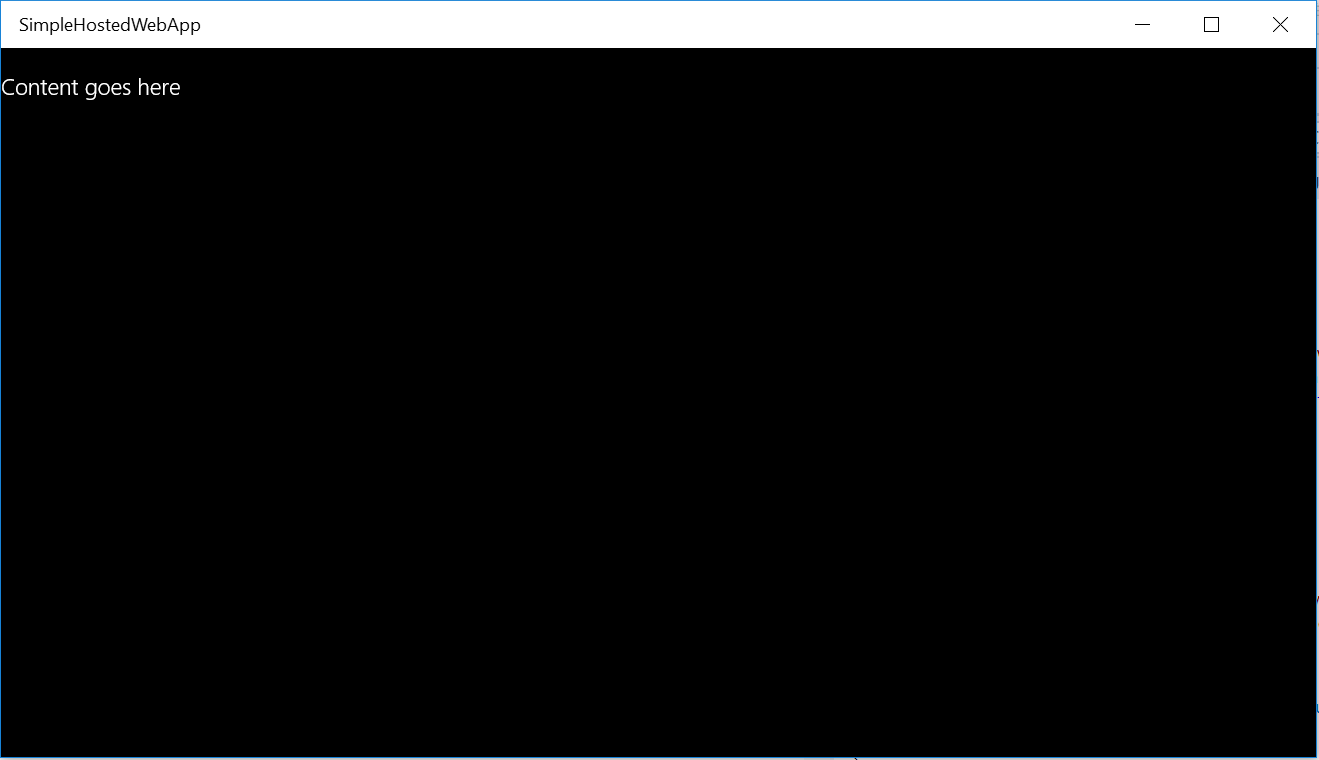


Figure 4

The blank universal JavaScript app running in Desktop mode.

1. Stop debugging and return to Visual Studio.

Task 2 – Create the native project

Our overall goal in this exercise is to trigger a toast notification from a webview. In this task, you will create a Windows Runtime Component to handle the toast notification through Platform APIs.

1. In your WebApp project, right-click on the solution name in the Solution Explorer and choose **Add > New Project**.
2. Select the project type **Visual C# > Windows > Windows Runtime Component (Universal Windows)** and give it the name **CommunicationWinRT**. Click **OK** to create the project.

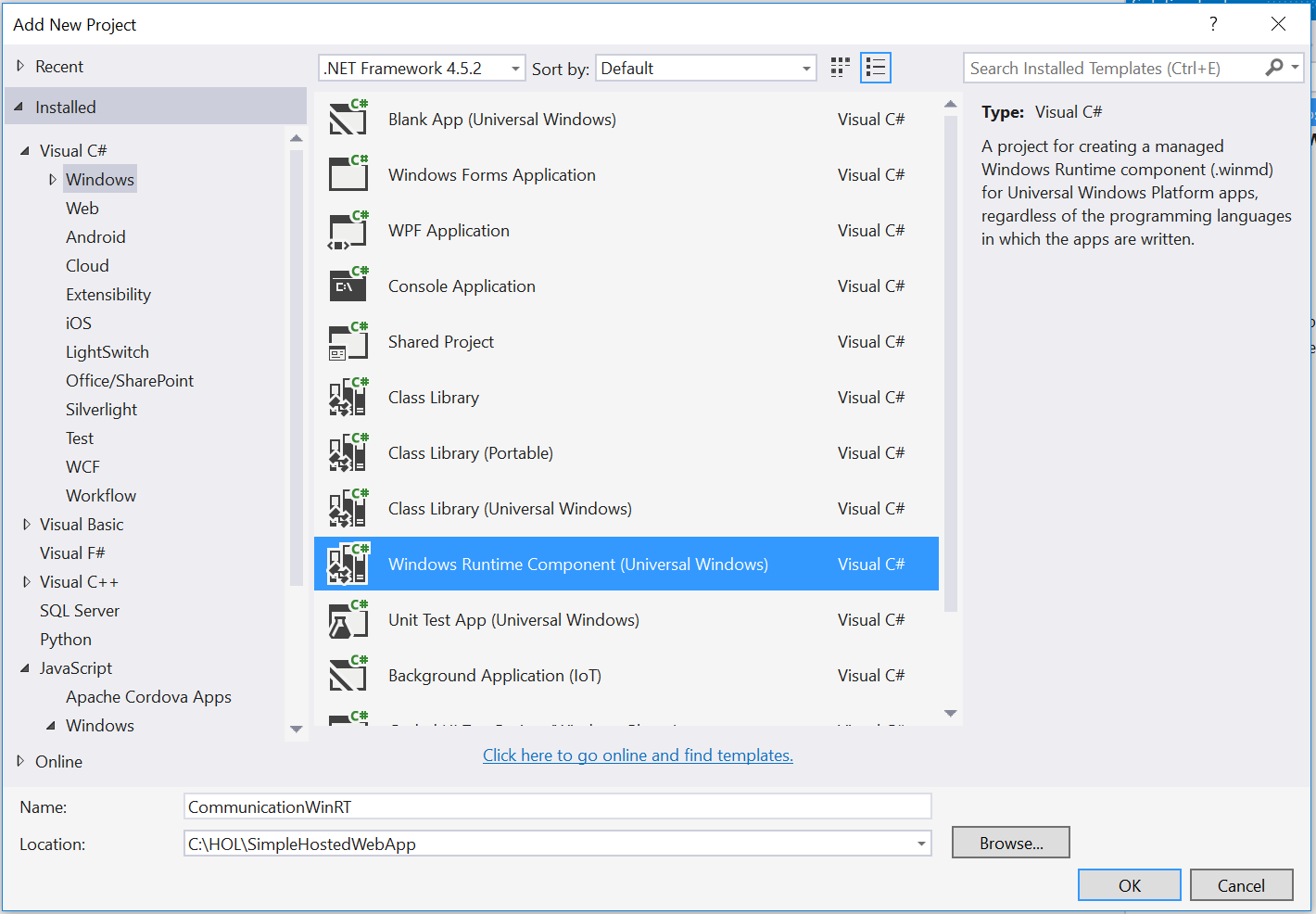


Figure 1

Add the CommunicationWinRT project.

1. Right-click on the **Class1.cs** file in the Solution Explorer and choose **Rename** to rename it to **CommunicationWinRT**. If prompted to perform a rename in the project of all references to Class1, choose **Yes**.

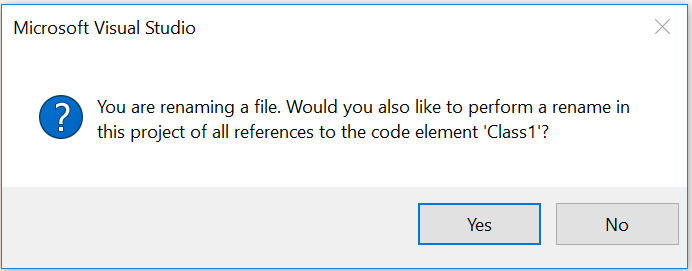


Figure 2

Rename Class1.cs to CommunicationWinRT.cs.

1. Open **CommunicationWinRT.cs**. Add the Windows.UI.Notifications and Windows.Data.Xml.Dom namespaces to the class.
   * 1. C#
   1. using Windows.UI.Notifications;
   2. using Windows.Data.Xml.Dom;
2. Add the **AllowForWeb** meta content and a constructor to the sealed class **CommunicationWinRt**.
   * 1. C#
   1. namespace CommunicationWinRT
   2. {
   3. [Windows.Foundation.Metadata.AllowForWeb]
   4. public sealed class CommunicationWinRT
   5. {
   6. public CommunicationWinRT()
   7. {
   8. }
   9. **Note:** A sealed class cannot be inherited from or expanded with properties from JavaScript. WinRT objects are sealed from JavaScript so that JavaScript can’t interfere with their behavior.
   10. For a Windows Runtime object to be projected and available to a WebView, it must have the AllowForWeb attribute.
3. Add an **async** method below the constructor that accepts a string and delay value which will be used to create a toast notification.
   * 1. C#
   1. public CommunicationWinRT()
   2. {
   3. }

public async void toastMessage(String message, int delay)  
{  
 ToastTemplateType toastTemplate = ToastTemplateType.ToastText01;  
 XmlDocument toastXml = ToastNotificationManager.GetTemplateContent(toastTemplate);

* 1. XmlNodeList toastTextElements = toastXml.GetElementsByTagName("text");  
      toastTextElements[0].AppendChild(toastXml.CreateTextNode(message));
  2. ToastNotification toast = new ToastNotification(toastXml);
  3. await Task.Delay(delay);  
      ToastNotificationManager.CreateToastNotifier().Show(toast);  
     }

1. Save **CommunicationWinRT.cs** and return to the web project.
   1. **Note:** You may add additional sealed public classes to the **CommunicationWinRT** namespace to handle other API integrations.
   2. For a deeper look at notifications, check out the **Live Tiles and Notifications Lab**.
2. Right-click on the **References** directory in the **WebApp** project and choose **Add > Reference**. Add the **CommunicationWinRT** project as a reference.

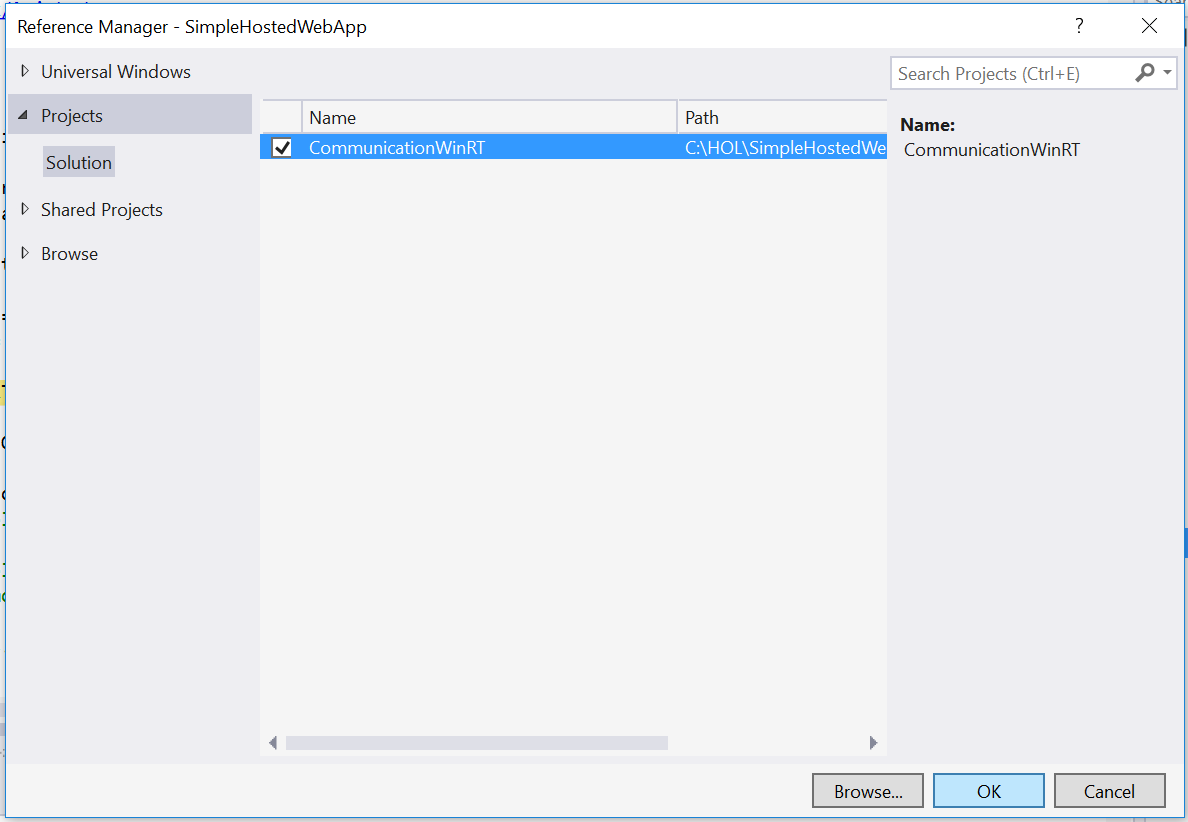


Figure 3

Add the native project as a reference to the web project.

Task 2 – Add a webview to the web project

In Windows 10, your JavaScript app can directly host navigation to a remote URI. However, a WebView may still be necessary when you need more granular control. In this task, you will add a webview element to default.html and prepare your project to display local content.

* 1. **Note:** A WebView can run locally and point to content on the web server and vice versa. In Windows 10, the WebView is now separated off the UI thread to improve performance and responsiveness. The WebView enables Windows Runtime access and uses the current Edge rendering engine to serve up content.

1. In **default.html**, replace the content of the **<body>** tag with a **WebView** element. Set its id to **toastView**. The id will make it easier to target the webview from your JavaScript.
   * 1. HTML
   1. <body class="win-type-body">

<x-ms-webview id="toastView"></x-ms-webview>

</body>

1. Open **default.js**. Give your webview a height and width in the **ActivationKind.Launch** conditional.
   * 1. JavaScript
   1. if (args.detail.kind === activation.ActivationKind.launch) {

var webview = document.getElementById('toastView');

* 1. webview.style.height = "100%";
  2. webview.style.width = "100%";

1. Instantiate the **CommunicationWinRT** class as a JavaScript object and use **addWebAllowedObject** to expose and name it within the webview. We have chosen to give it the name **toastApi** within the context of the webview.
   * 1. JavaScript
   1. if (args.detail.kind === activation.ActivationKind.launch) {

var webview = document.getElementById('toastView');

* 1. webview.style.height = "100%";
  2. webview.style.width = "100%";

var communicationWinRT = new CommunicationWinRT.CommunicationWinRT();

webview.addWebAllowedObject("toastApi", communicationWinRT);

1. Save and close **default.js**.
2. Open the app manifest and navigate to the **Application** tab. Change your start page back to **default.html** and use the **Remove** button to delete the Bing URI from the **Content URIs** tab. Save and close the manifest.

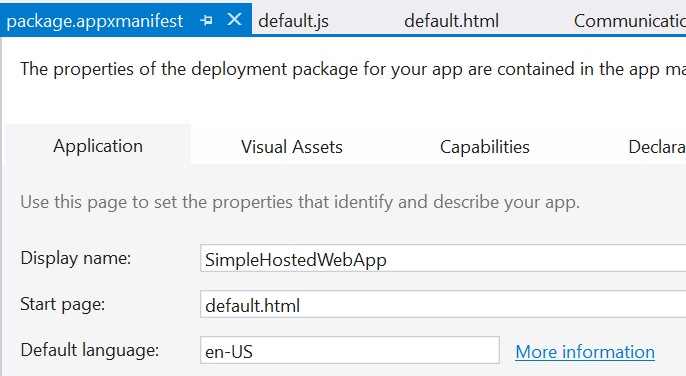


Figure 4

Set the start page to default.html.

* 1. **Note:** The webview has access to Windows APIs, so you don’t need to specify a Content URI to achieve WinRT access in this case. If you host remote content without a webview, you will need to specify Content URIs and access levels for that content in the app manifest.

Task 3 – Add a new HTML page and a script to generate toast

Your app is now set up to host content in your webview. In this task, you will add an HTML page that will display in the webview and run a script to generate a toast notification.

1. Right-click on the SimpleHostedWebApp project and select **Add > New Item**. Choose the **HTML Page** item type and give it the name **toastGenerator.html**. Add the page to the project.

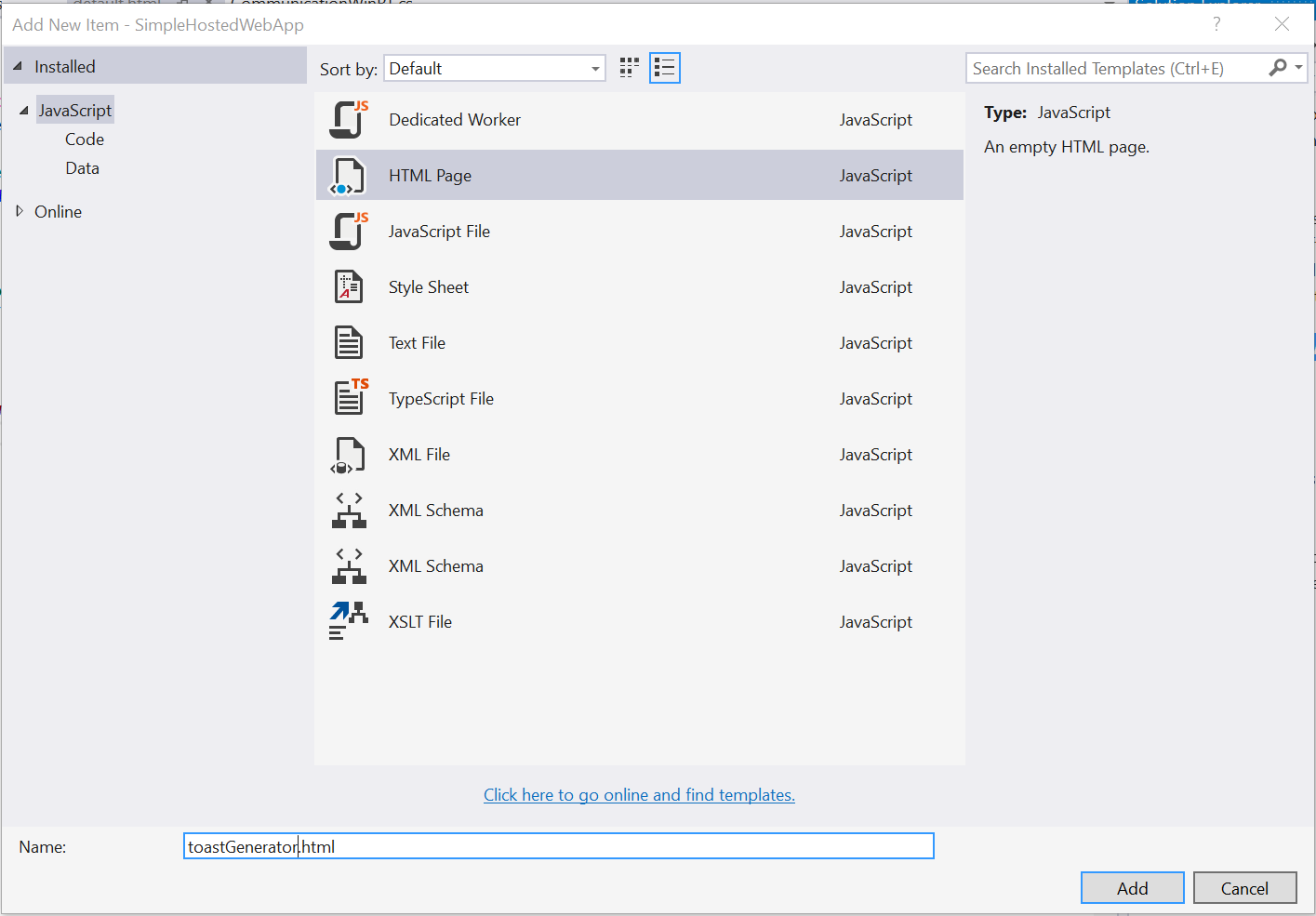


Figure 5

Add a new HTML page to the web project.

1. In **toastGenerator.html**, add a page title to the **<title>** element. Create a button in the **<body>** whose **onclick** event calls the **makeToast()** function. You will create the makeToast() handler in the next step.
   * 1. HTML
   1. <!DOCTYPE html>  
      <html>  
      <head>  
          <title>Toast Integration</title>  
          <script type="text/javascript">  
              function makeToast() {  
                  var object = window.toastApi;
   2. if (object) {  
                      console.log('found');  
                      object.toastMessage('Hello Toast', 0);  
                  }  
                  else {  
                      console.log("Error!")  
                  }  
              }  
          </script>  
      </head>  
      <body>  
          <button onclick="makeToast()">Notify</button>  
      </body>  
      </html>
2. Add the **makeToast()** handler in a **<script>** tag. This function looks for the **toastApi** object and calls its **toastMessage()** method if found. If the method toastApi object is not found, your console log will show an error message.
   * 1. HTML
   1. <!DOCTYPE html>  
      <html>  
      <head>  
          <title>Toast Integration</title>  
          <script type="text/javascript">  
              function makeToast() {  
                  var object = window.toastApi;

            if (object) {  
                console.log('found');  
                object.toastMessage('Hello Toast', 0);  
            }  
            else {  
                console.log("Error! No toast for you.")  
            }  
        }  
    </script>  
</head>  
<body>  
    <button onclick="makeToast()">Notify</button>  
</body>  
</html>

1. Return to **default.js** to navigate your webview to the **toastGenerator.html** page.
   * 1. JavaScript
   1. webview.addWebAllowedObject("toastApi", communicationWinRT);

webview.navigate("ms-appx-web:///toastGenerator.html");

1. Set your Solution Configuration to **Debug** and your Solution Platform to **x86**. Build and run your app on the **Local** Machine.



Figure 6

The web app displays the Notify button.

1. Use the **Notify** button to generate a toast notification. Your **Hello Toast** message will pop up near the Action Center icon in the task bar.

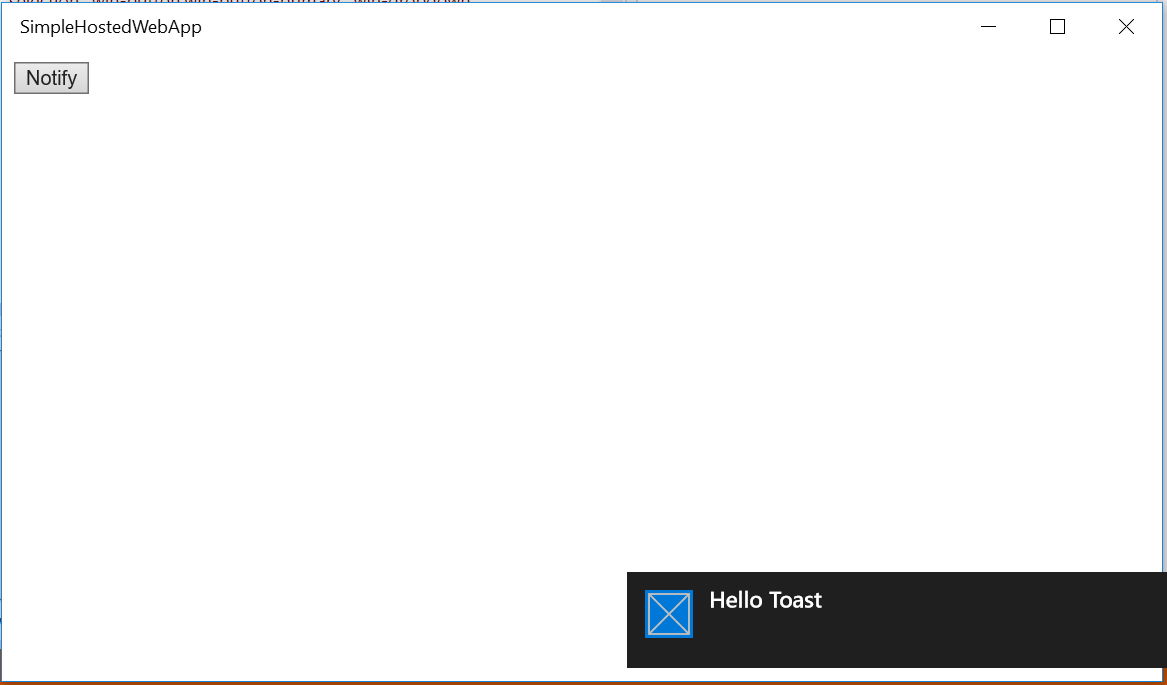


Figure 7

The web project has successfully integrated with the native project to generate toast.

1. Stop debugging and return to Visual Studio.

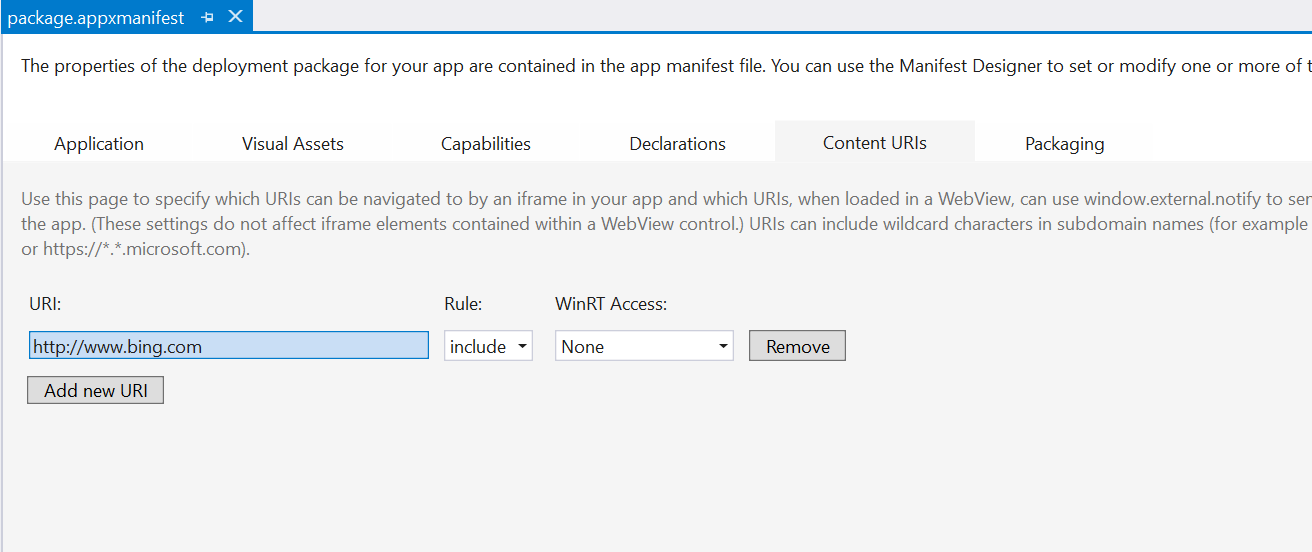
Exercise 2: Create a Hosted Web App

* 1. 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 simple hosted web app using Bing.com as an example. You may also substitute your own site for Bing.

Task 1 – Host Bing in a web app

A hosted web app essentially functions as a browser using the Edge rendering engine. In this task, you will display the Bing search engine as a hosted web app.

1. Open **package.appxmanifest** in your **SimpleHostedWebApp** project using the **Project editor**.
2. Navigate to the Content URIs tab. Add **http://www.bing.com/** to the URI field and leave the other fields as default values.

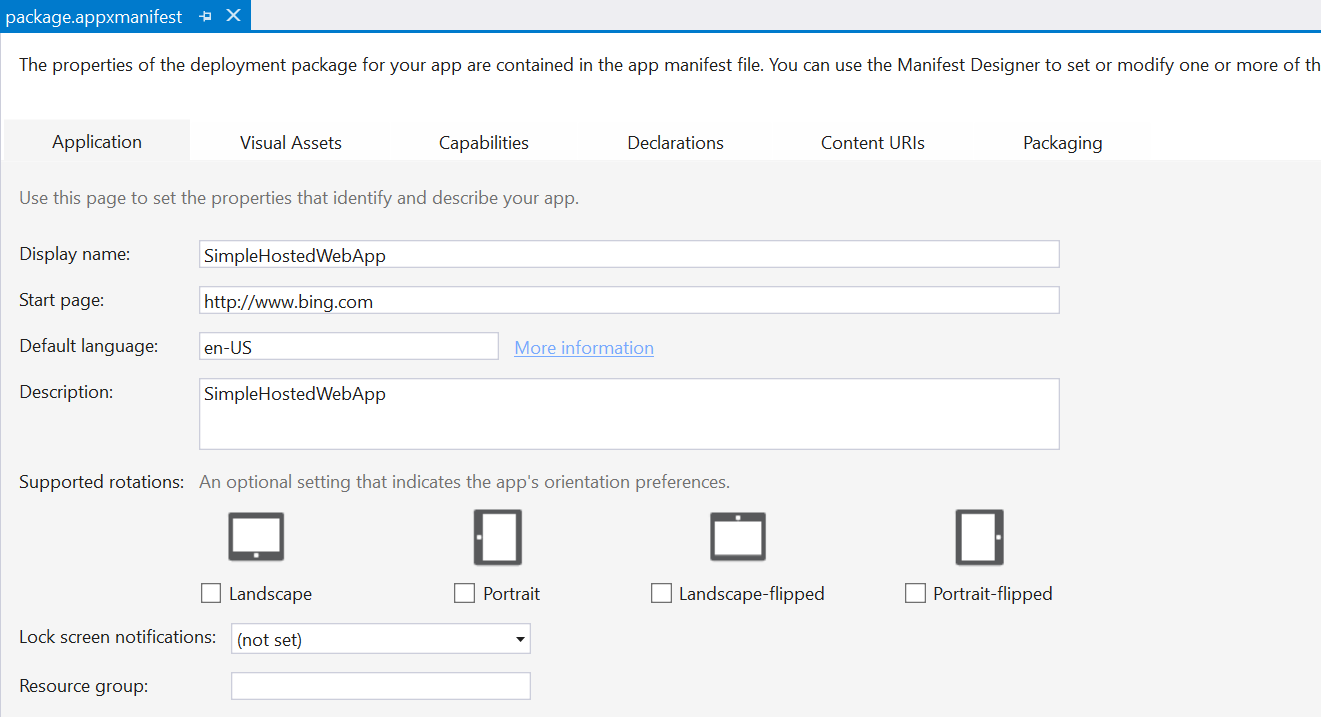


Figure

Add the Bing content URI.

* 1. **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 Bing.com within the app but force links external to Bing 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**.
  2. To specify a remote URI, use the **http://** protocol. To specify a local URI, use the **ms-appx-web:///** protocol.

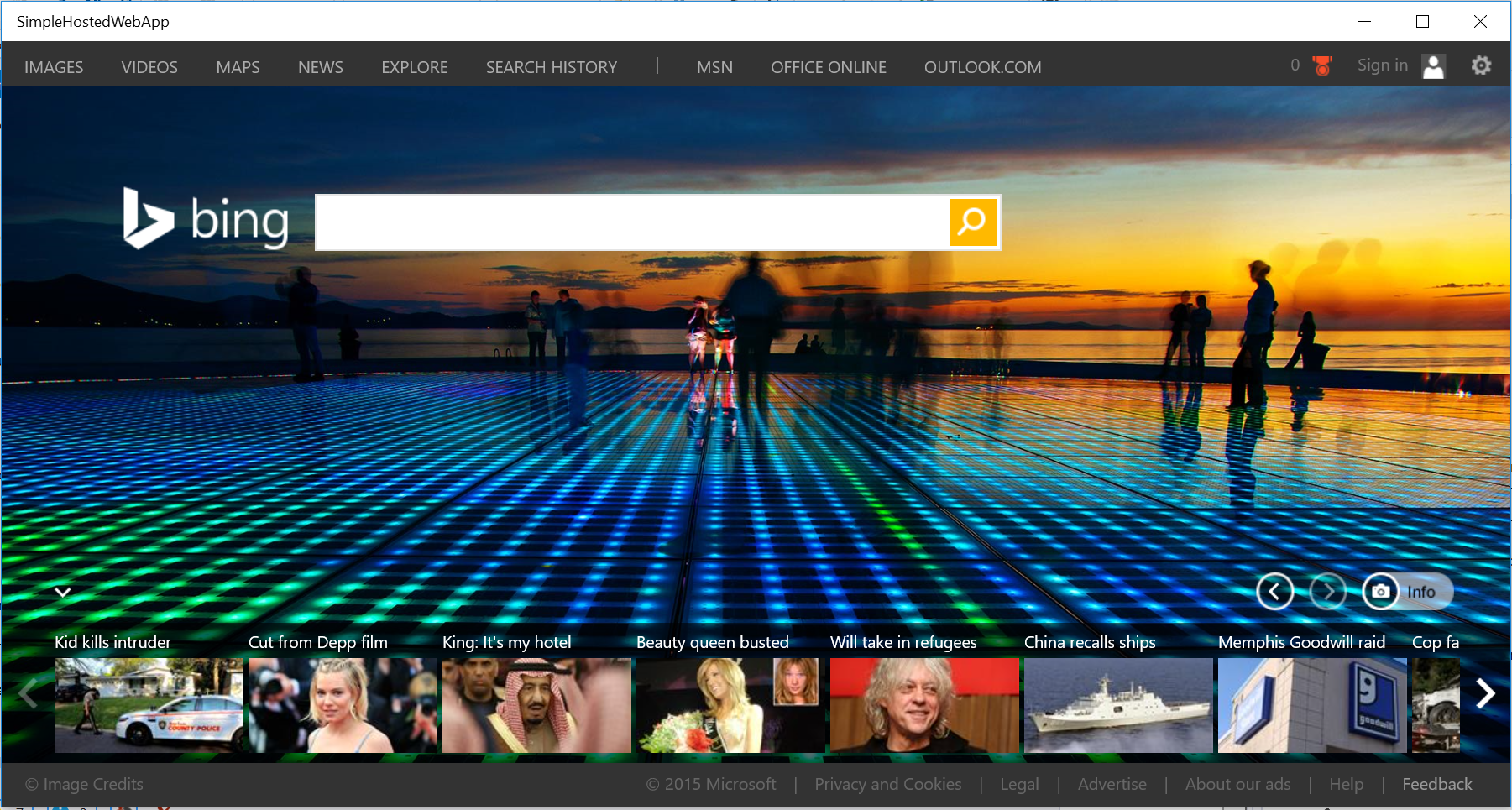
1. Navigate to the **Application** tab. Change the **Start page** to [**http://www.bing.com**](http://www.bing.com)**.**



Figure

Set Bing as the Start-up page.

1. Build and run your app. You will see Bing.com appear in the app window.



Figure

Bing as a hosted web app.

1. Enter a search in Bing. You will notice that navigation on the Bing site stays within the app, but outside links will open in Edge.
2. Stop debugging and return to Visual Studio.
   1. **Note:** When creating an app that contains only hosted content, you may delete the **css**, **js**, and **WinJS** folders as well as the **default.html** file. However, you may wish to use these files to create an offline backup of your site in case your server is not available. It is good practice to create a local error page in the project at the very least. You will create an error page in Task () of this lab.

Exercise 3: Support Additional Platforms and Devices with ManifoldJS (optional)

* 1. 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 generated 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.
  2. 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 to 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.
   * 1. Command Prompt
   1. > npm install –g manifoldjs
   2. **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.
   1. **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.

1. 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.
   * 1. Command Prompt
   1. > manifoldjs http://www.bing.com/
2. Explore the code generated by ManifoldJS in your hosted apps folder.
3. To install and run the generated Windows10 app, run the following command in the app folder created by ManifoldJS:
   * 1. Command Prompt
   1. > manifoldjs run windows
4. Your generated app will install and launch.

Summary

* 1. Hosted web apps provide powerful options to integrate your existing web projects with the Windows Store and platform APIs. In this lab, we created a web app to host remote content and learned how to integrate native Windows APIs with JavaScript. We also explored options for generating hosted web apps for a variety of platforms.