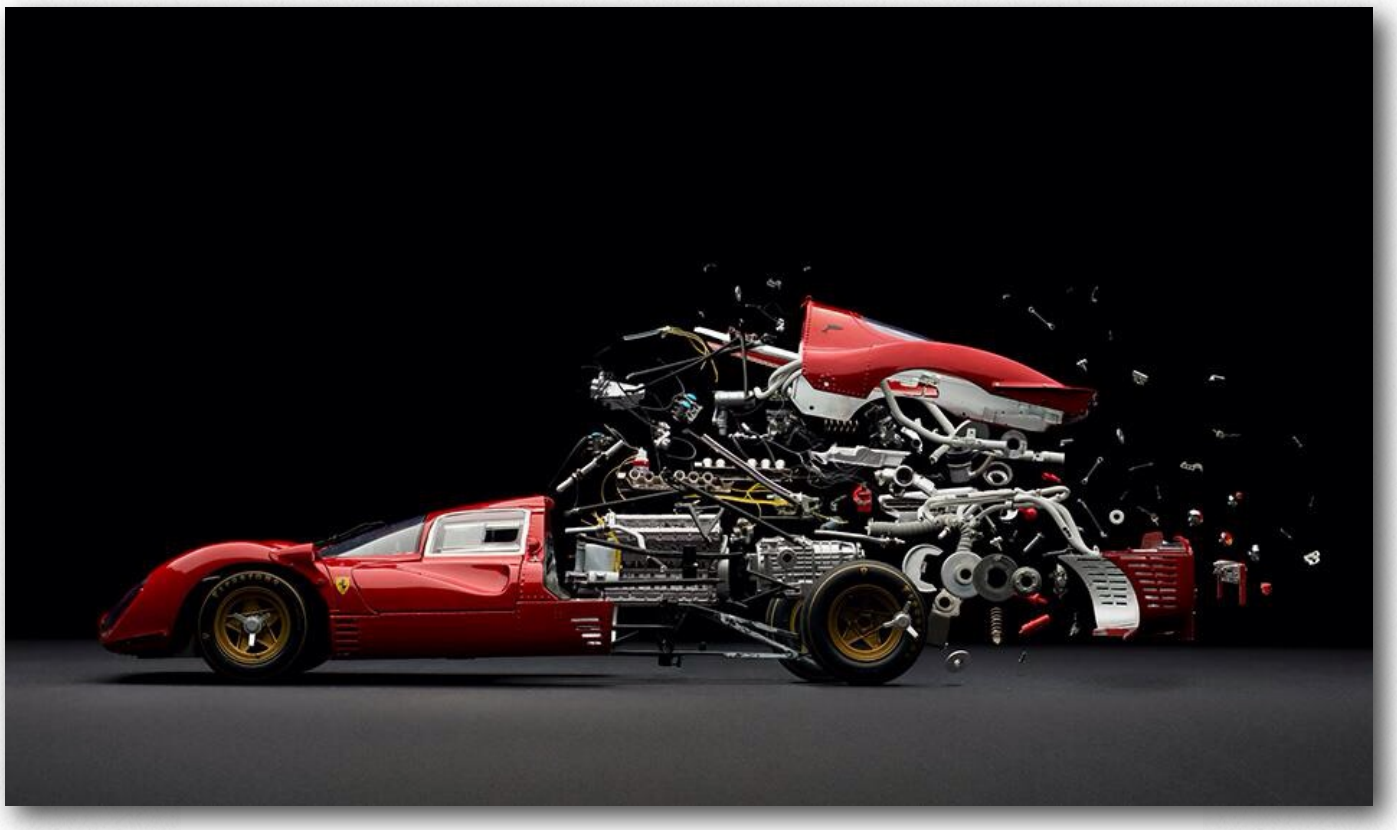
Getting Started with

PhoneGap



alpha Anywhere™
Powered by Alpha Five V12

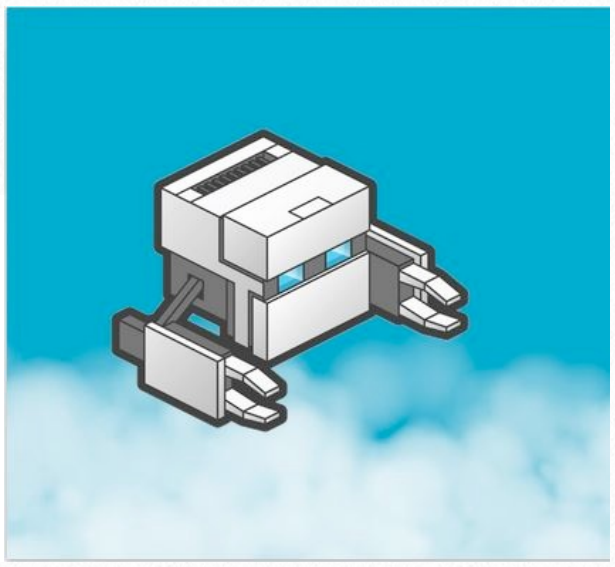
Last revised: 24 - November - 2014

By: R.E. Moore Jr.
VP Mobile Development
Alpha Software, Inc.
@remoorejr

1

Overview

Building native mobile applications with Alpha Anywhere and PhoneGap



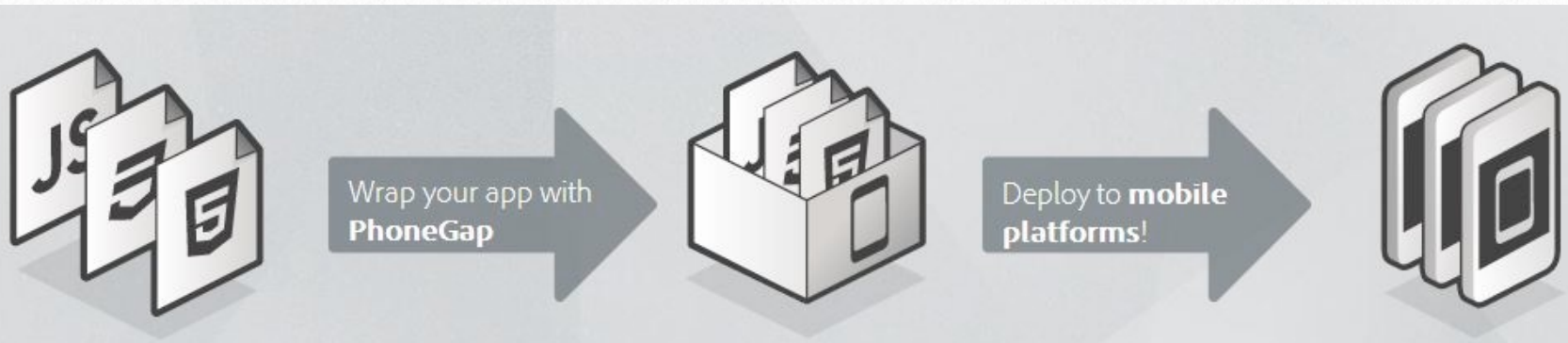
Alpha Anywhere is a comprehensive HTML5 development framework that on it's own can build powerful mobile HTML5 web applications that look and feel like native apps.

Once created, a web application on a mobile device is launched within the web browser that is supplied with the device.

iOS devices typically use Mobile Safari, while Android devices can use a variety of browsers with Chrome or some derivative of Chrome typically being the native browser. Since the app is running within the browser, it is limited to the capabilities that the browser exposes.

But suppose you want to access some of the native features of the phone or tablet.

For example, let's say you want to allow your users to access a bar code scanner, which incorporates the devices camera and some special image processing software, that sits locally on the device, that can decipher a QR Code or a barcode and send that encoded information back to your web application for a



possible database lookup or submission to a web service to lookup specifics on a book or an inventory part. You can do that with PhoneGap and that is just the tip of the iceberg.

PhoneGap allows you to access ALL of the functionality of the mobile device.
You are really only limited by your imagination and skills.

PhoneGap includes a large set of core and third party plugins that expose all sorts of native functionality. A variety of the plugins support iOS, Android and Windows Phone so that you can target a wide range of mobile phones and tablets.

And best of all, PhoneGap allows you to build true native apps for iOS, Android and Windows Phone 8 devices, apps that you can easily distribute through the numerous app stores.

An Overview Of The Development Process with PhoneGap and Alpha Anywhere

You will use the Alpha Anywhere IDE to develop a base UX component that may contain all of the code and data that is required for your PhoneGap application, or you may choose to create a base UX component that loads data and additional components from a remote server (assuming the phone or tablet has Internet access).

When you are ready to convert your base UX component into a PhoneGap app, the integrated PhoneGap App Builder is used from within the Alpha Anywhere IDE to:

- Generate the PhoneGap app
- Generate all required PhoneGap configuration options, including the addition of required PhoneGap plugins
- Convert the UX Component into a set of files that contain all of the standalone HTML, CSS, JavaScript and additional assets (images, etc.) required to build a single page PhoneGap app that is installed and loaded locally from the device. This ensures that the initial page can provide a fast, workable and pleasant user experience, even when the device is offline.
- Package and upload the content to the PhoneGap Build Cloud service, which can create native apps for iOS, Android and Windows Phone 8 devices.

Once the app is built with PhoneGap Build, you can:

- Install the native app for testing on iOS and Android devices
- Debug your code wirelessly through the use of a local or remotely hosted debug server
- Push updates to all of your test devices without the need to reinstall the app from a file or from PhoneGap Build

App testing can be done directly on a phone or tablet by scanning a QRCode that is displayed within the integrated PhoneGap App Manager, once the build is complete.

You can also share the PhoneGap Build QRCode with a customer or other app testers for feedback, etc.

PhoneGap Build eliminates the need for you to install all of the individual SDK's for each platform that you are targeting on your development machine. It also eliminates the need to compile and build for each targeted platform. PhoneGap Build does all of this for you as a web service.

Let's get started.

2

Setup A PhoneGap Build Account

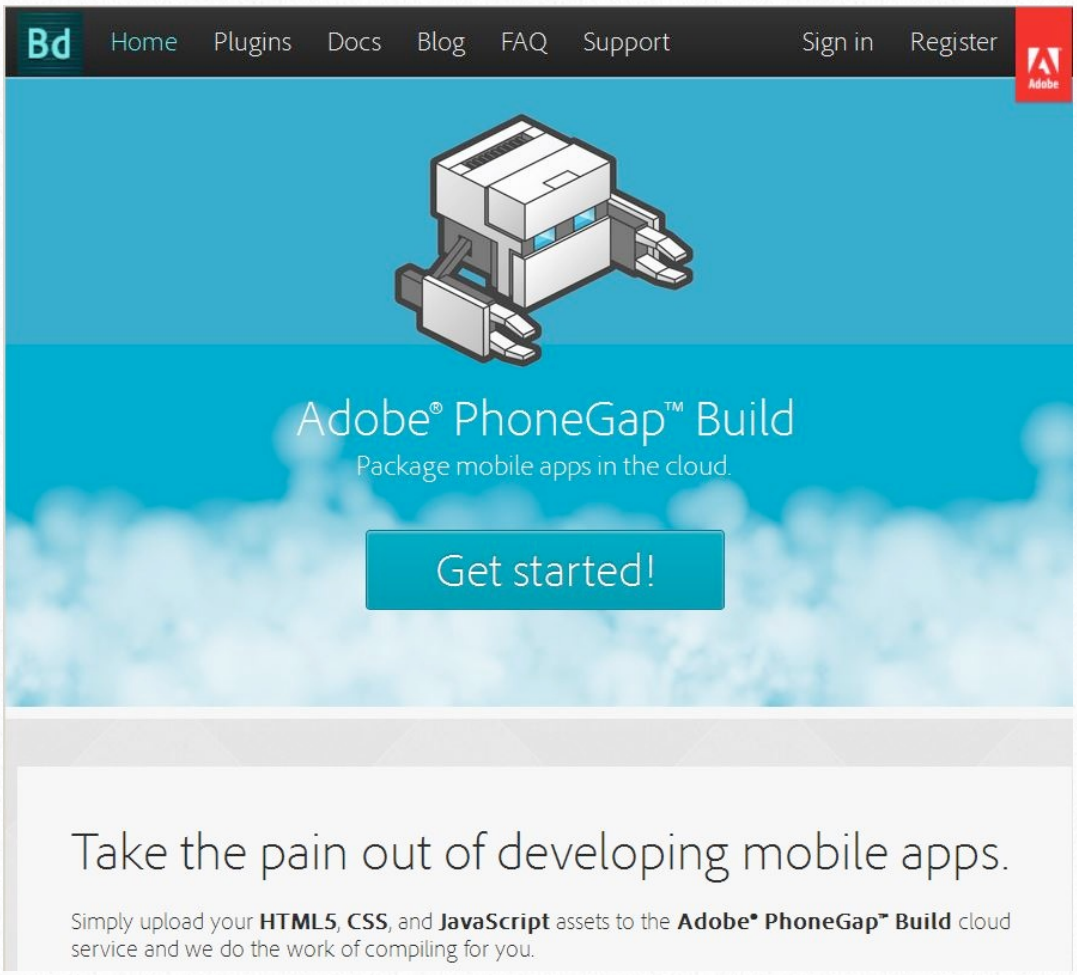
PhoneGap Build is a cloud service offered by Adobe, that allows the creation of native PhoneGap applications without the requirement of installing the native SDK's (Android Developer Tools or iOS Xcode) on your development machine.

By using PhoneGap Build, it is easy to generate applications that run on iOS devices from a Windows computer running Alpha Anywhere. We have built the ability to call the PhoneGap Build Service directly into the Alpha Anywhere PhoneGap App Builder, making it very simple to generate or update a native PhoneGap app from an Alpha Anywhere designed UX component.

Signup For a Free PhoneGap Build Account

You will need a PhoneGap Build account to build applications with PhoneGap Build.

Adobe allows a free account that can build one private app for multiple platforms. If you need to keep more than one app available within PhoneGap Build you can signup for a paid account. For as little as $9.95 per month (with no annual commitment, you can have up to 25 apps available on PhoneGap Build.



To get started, ***go to http://build.phonegap.com and setup your account***.

You will need to log in to this account from the PhoneGap App Builder prior to submitting an app to PhoneGap Build, so make sure to keep track of your user name and password.

3

Building your first app with Alpha Anywhere and PhoneGap 3.x

It is assumed that you are familiar with Alpha Anywhere and know how to build a UX component for a mobile device. You should already be familiar with the concepts of a Panel Navigator, Panels, Buttons controls, etc. If you are not familiar with these terms, it would be best for you to work through a number of the tutorials available prior to attempting to build PhoneGap apps with Alpha Anywhere.

Building a Simple Test App

To verify you've got everything setup and working properly, we're going to build a very simple PhoneGap app that contains a Panel and a button that when tapped, initiates a native beep tone. This will work on both Android and iOS devices. We will build the test UX component, generate a PhoneGap app from that component, upload the app to the PhoneGap Build service and finally, install the app on an Android device (or Android emulator).

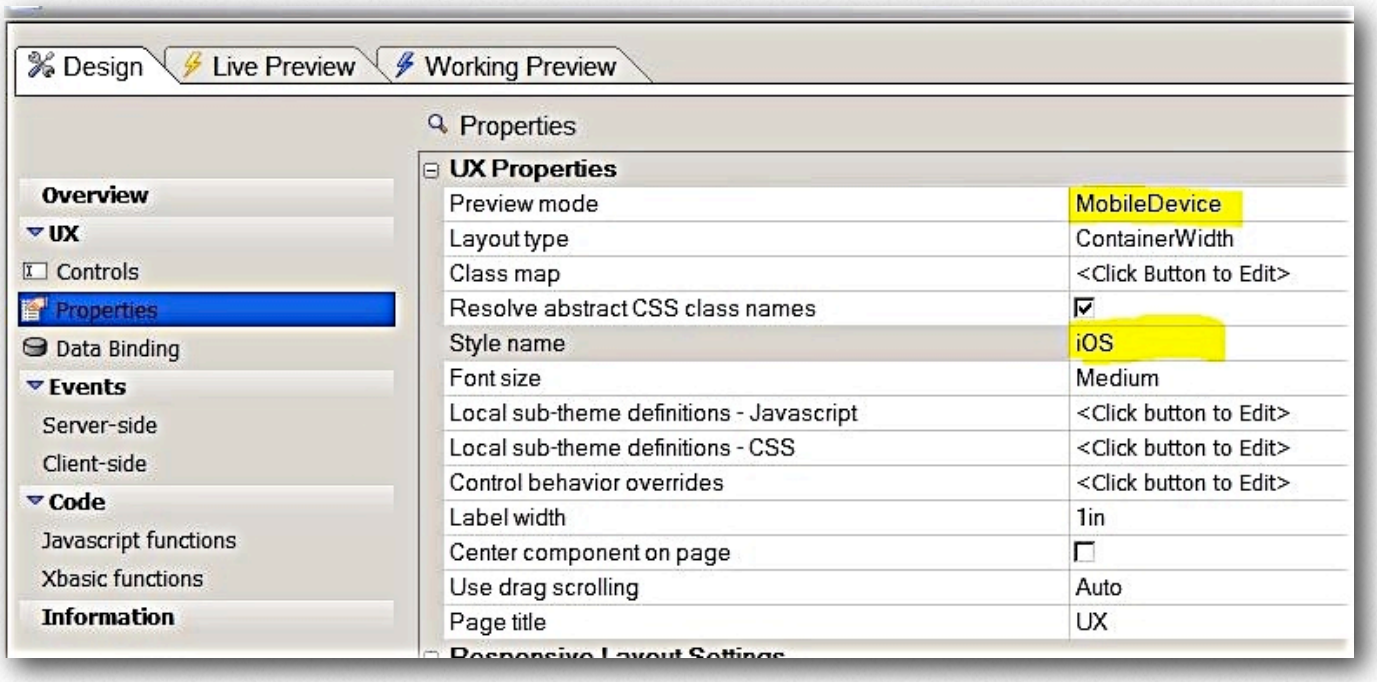
Note: *Installation on an iOS device requires that you setup an Apple iOS Developer Account and that you provide your Apple Developer credentials and provisioning profile to PhoneGap Build. This will be covered in a later chapter*.

Building the Test UX Component

In Alpha Anywhere, from the Web Projects Control Panel, ***click***:

New | Web Component | UX | Global Templates | Panel Card With Header, Title & Buttons

Switch to the UX Properties Panel and ***set Preview mode to Mobile Device and Style name to iOS***.



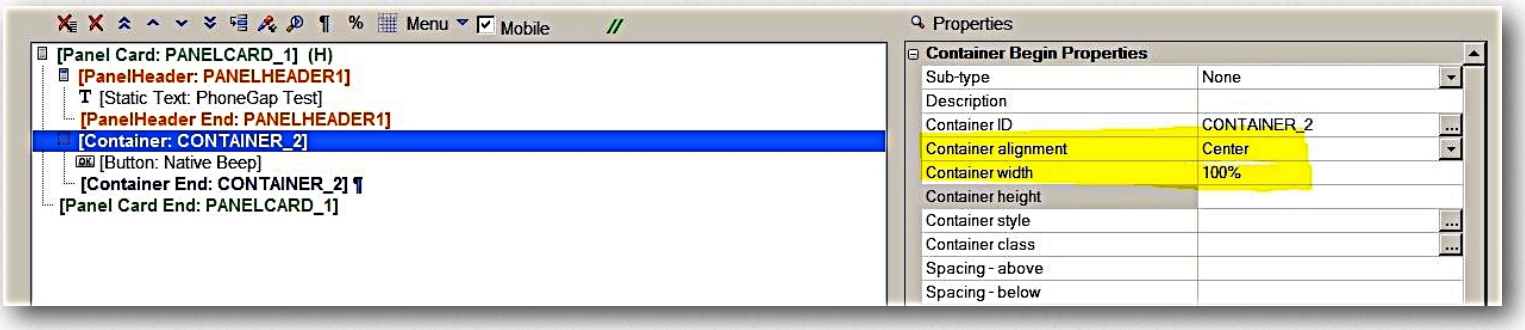
Set UX Properties Preview Mode and Style name

Switch back to Controls and ***Delete the Panel Header Buttons and Tab Stop Controls***, leave the Static Text control and change the static text property to PhoneGap Test.

Delete Panel Header Buttons and Tab Stops

Set the Panel Header Alignment to Center.

Add a Container below the Panel Header.

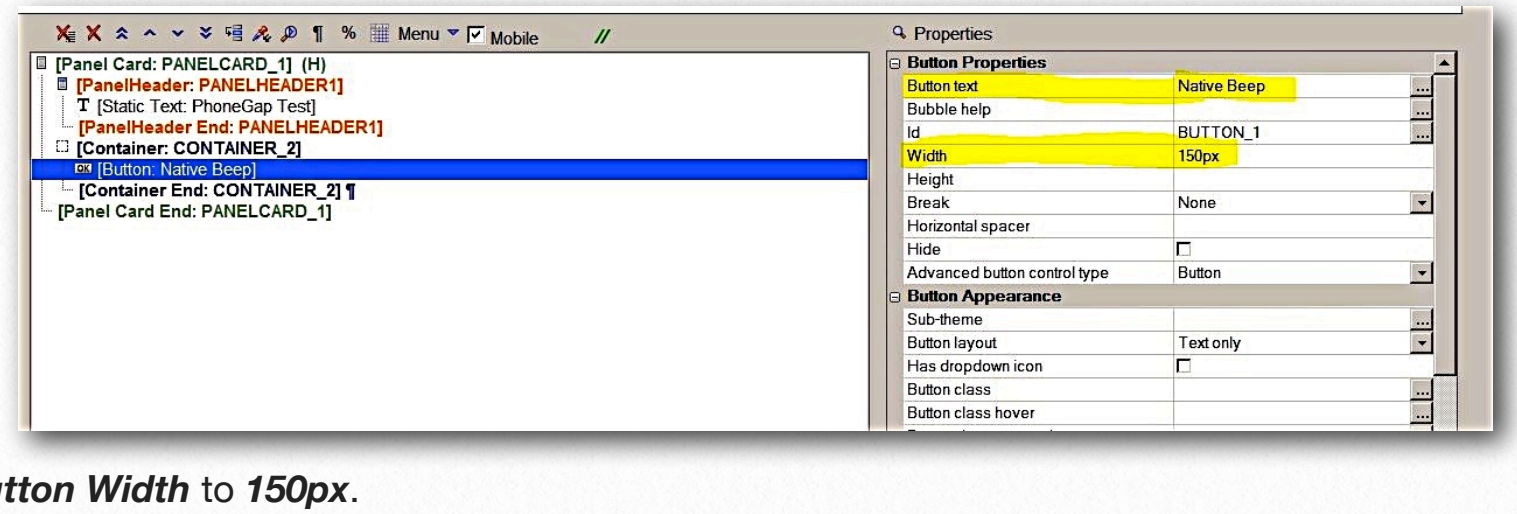


Set the Container alignment to ***Center.***

Set the Container width to ***100%.***

From **Other Controls**, ***select a Button*** and ***insert the button*** into the Container.

Set the Button Text property to Native Beep**.**

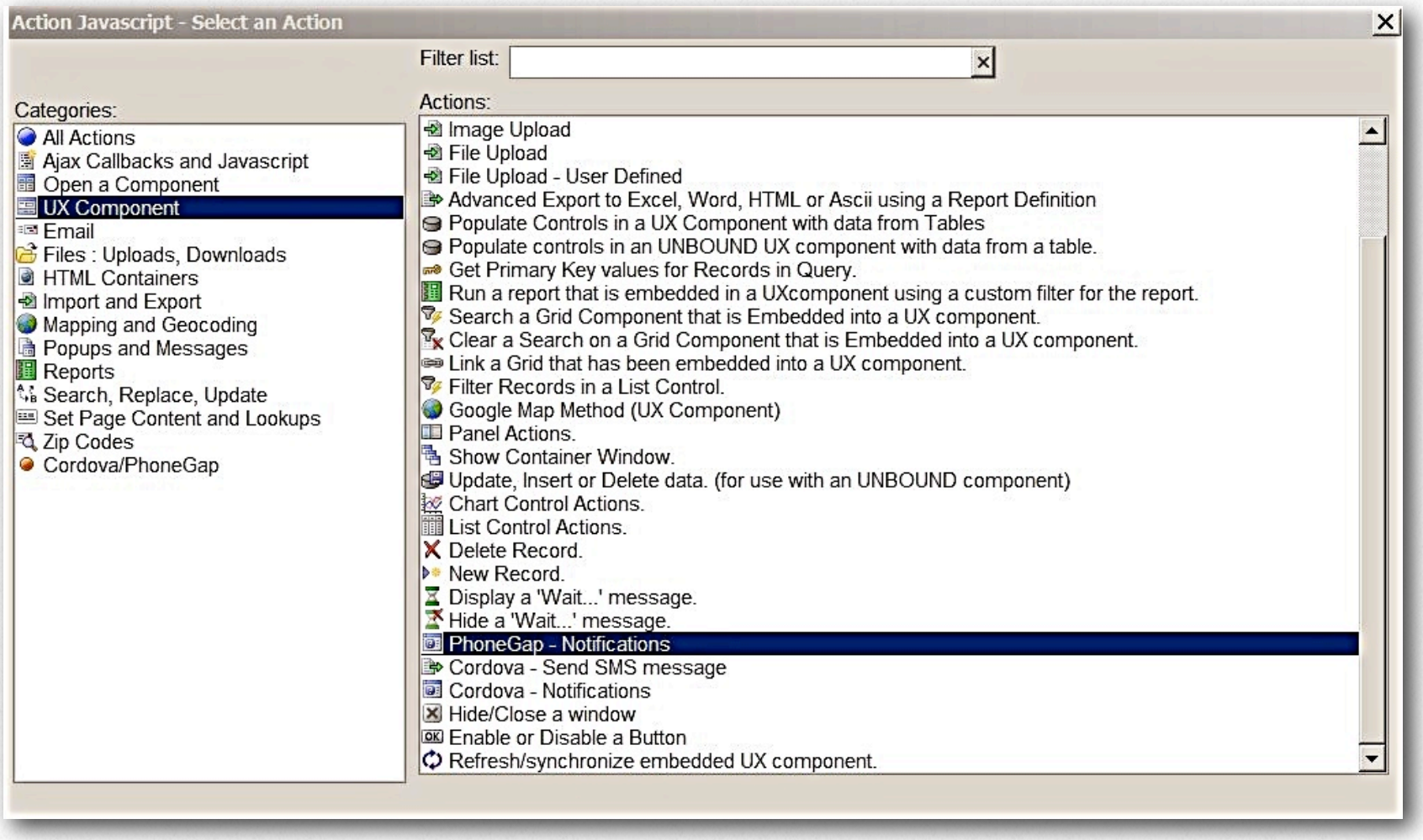


Set the Button Width to ***150px***.

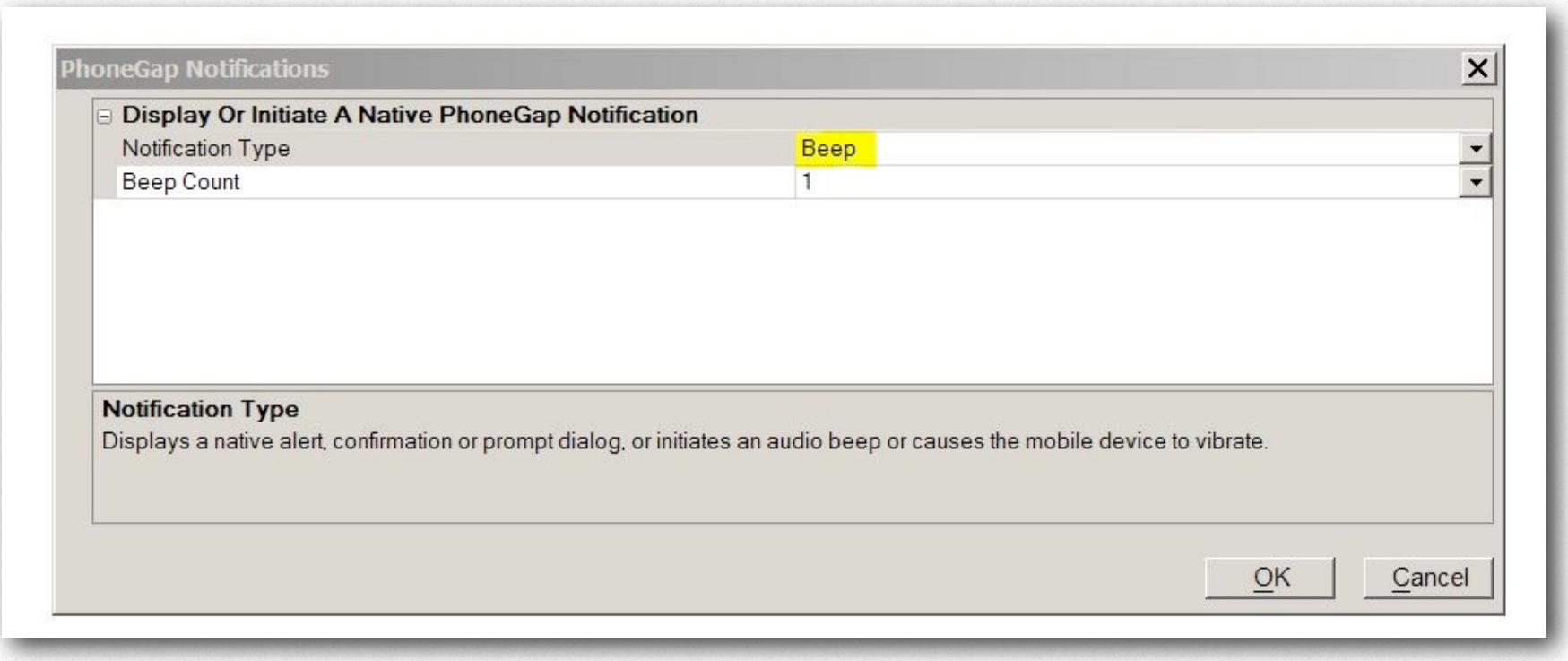
Click on **Working Preview,** your UX component should look like the image below:

Switch back to design mode and ***select the button control***. In the property grid, find the JavaScript Touch event for click and ***select***. Next ***add new action*** to assign an Action JavaScript to the button.

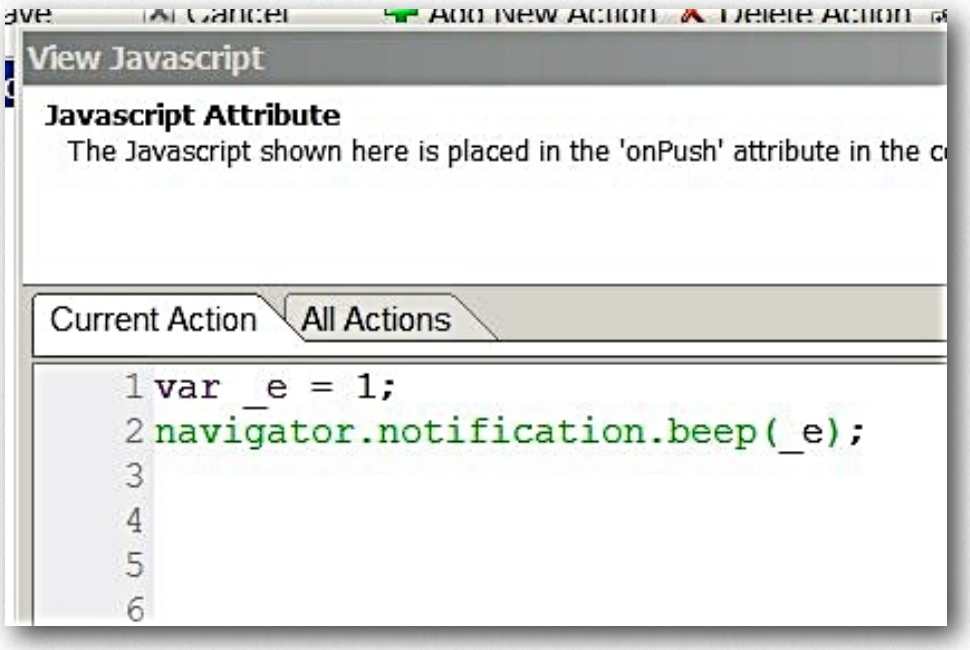
Pick the UX Component Category on the left panel and the PhoneGap - Notification Action on the right panel.



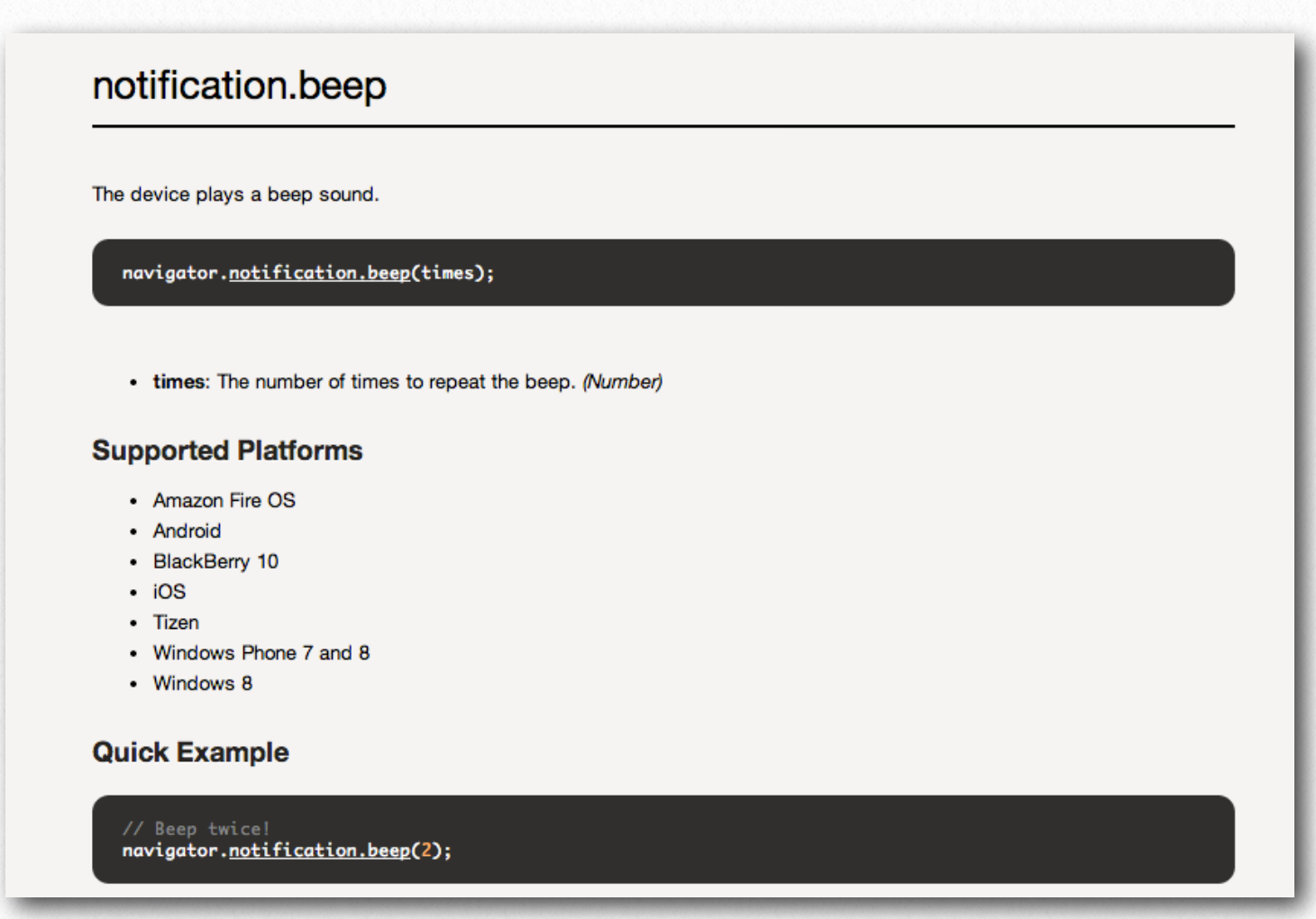
Select the Beep Notification Type from the combo box. Leave the beep count parameter at 1. (Note: this parameter is ignored on iOS devices.)



Once you ***click OK*** you can view the JavaScript generated by the Action JavaScript builder.



If you open the PhoneGap documentation site at http://docs.phonegap.com under the API Reference, Notification.beep you will see the PhoneGap sample code is quite similar to the Action JavaScript generated code. If you are comfortable with JavaScript, you can look through the PhoneGap documentation and use the



notification.beep

The device plays a beep sound.

```
navigator.notification.beep(times);
```

- **times**: The number of times to repeat the beep. *(Number)*

Supported Platforms

- Amazon Fire OS
- Android
- BlackBerry 10
- iOS
- Tizen
- Windows Phone 7 and 8
- Windows 8

Quick Example

```
// Beep twice!
navigator.notification.beep(2);
```

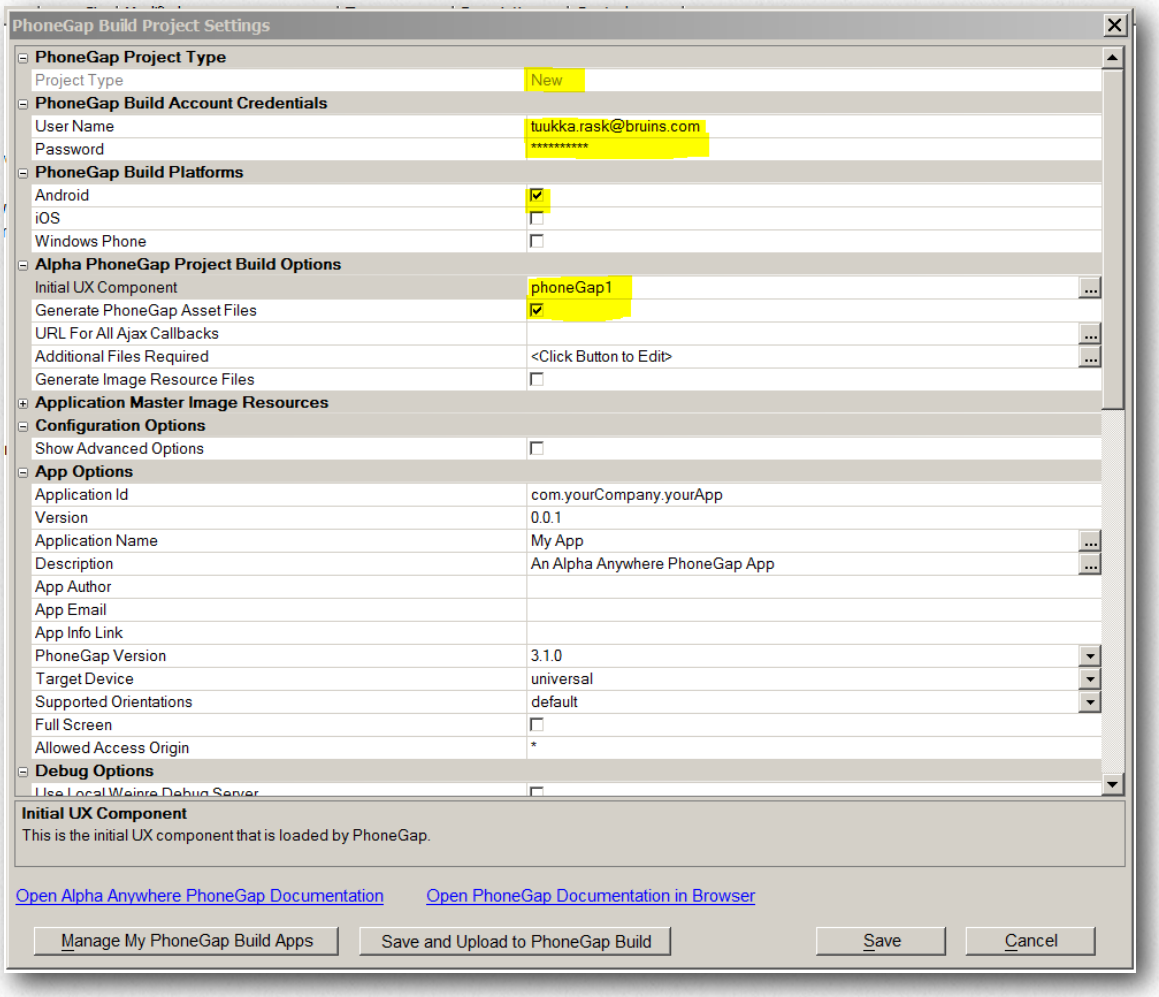
samples to add your own JavaScript to any button or event in your own UX components.

Save the component and name it PhoneGap1. We will use this component within the Alpha Anywhere integrated PhoneGap App Builder as the basis for the PhoneGap App which will be installed on the phone or tablet.

You can now ***close the UX component builder*** and we will move on to generate the PhoneGap application within your web projects folder.

Using The PhoneGap App Builder

On the Web Projects Toolbar, ***click the PhoneGap button***, this will launch the **PhoneGap App Builder.**



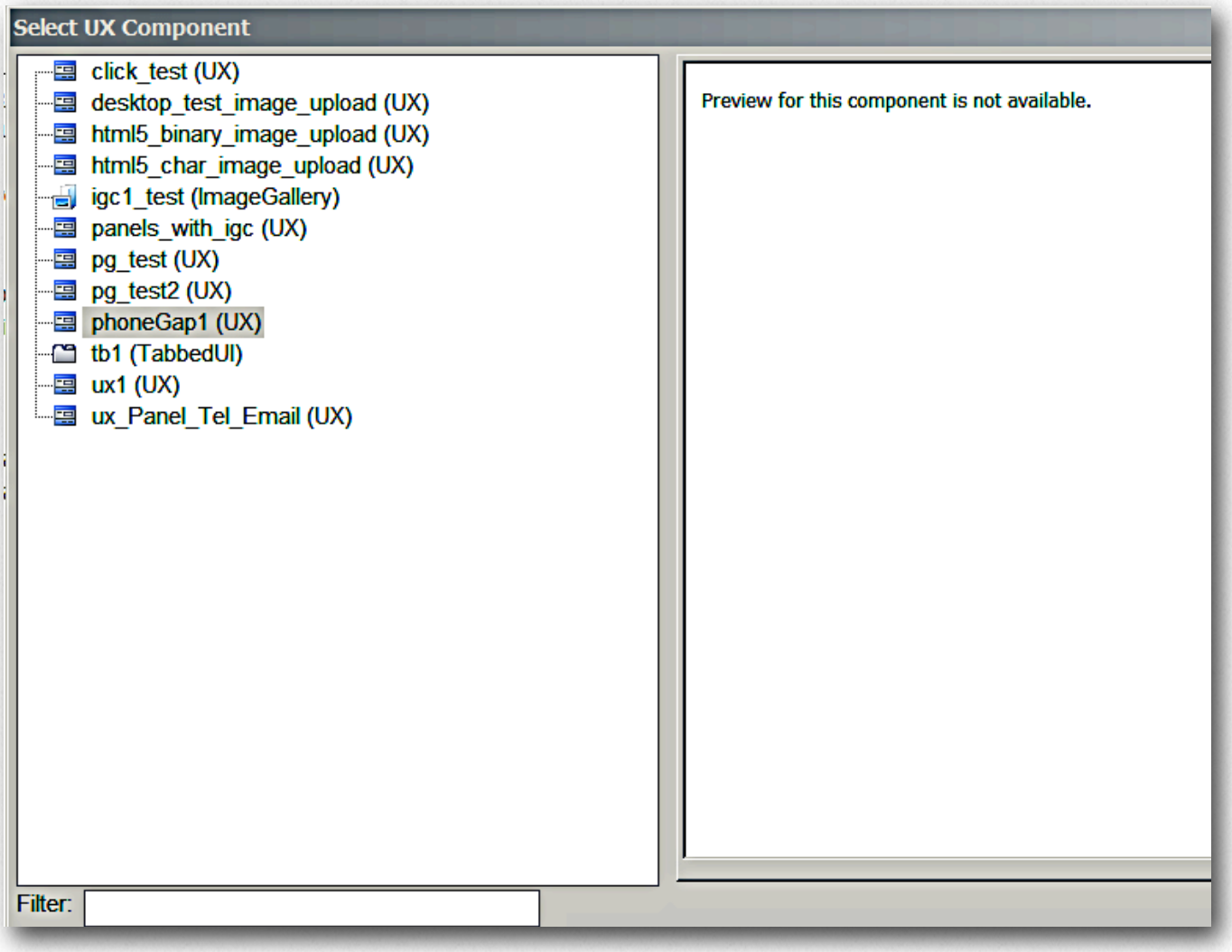
The **PhoneGap App Builder** contains a large number of options and properties for PhoneGap app customization. Within the PhoneGap App Builder, you select the target platform(s) to build, numerous properties for the app name, app description, app version, author, etc. You also select the PhoneGap plugins to include in the project and many other options. In this simple application, we are only going to be using a few of the many options available.

Notice the Project Type property is **New**, indicating that there are no pre-existing PhoneGap assets in the Web Projects Directory. If a previous project existed, the Project Type would indicate **Existing** and the builder would skip the step of generating the PhoneGap scaffold application, which is created in a PhoneGap folder in the Web Project Directory.

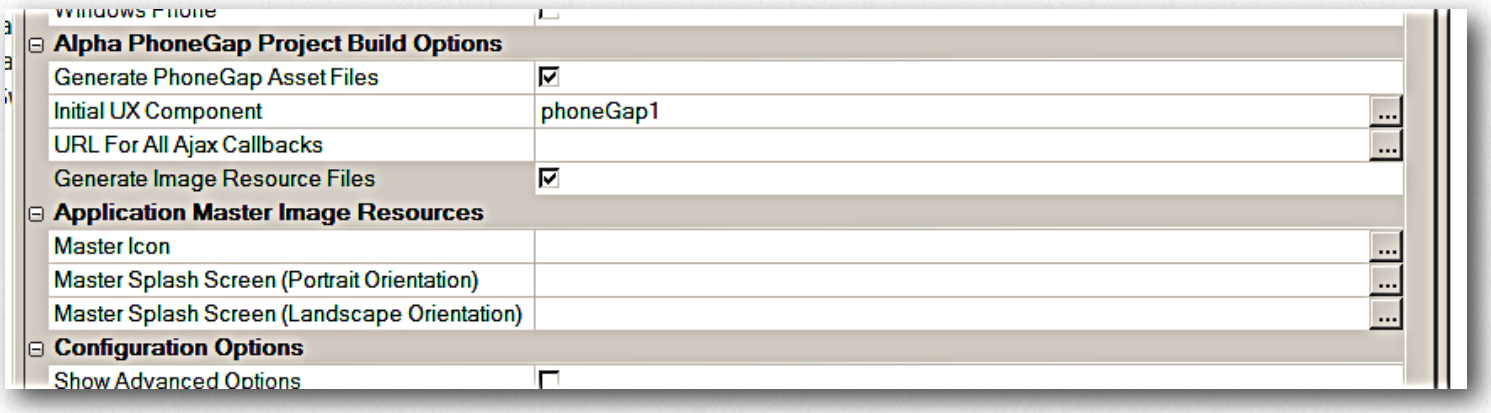
In the **PhoneGap Build Account Credentials**, ***enter the user name and password*** for your PhoneGap Build account.

In the **PhoneGap Build Platforms** section, ***check the Android box***. This triggers an Android project build in PhoneGap Build. It also enables a set of Android specific options within the PhoneGap App Builder. Since an Android project build does not require the submission of developer credentials to PhoneGap Build, we will stick to Android for the time being.

Under the Alpha PhoneGap Project Build Options, ***select the Initial UX Component*** that was previously built, **PhoneGap1(UX)**. Once selected, the **Generate PhoneGap Asset Files checkbox** will be ***automatically checked***. Leave it checked because we want to generate the PhoneGap app from the PhoneGap1 UX component.

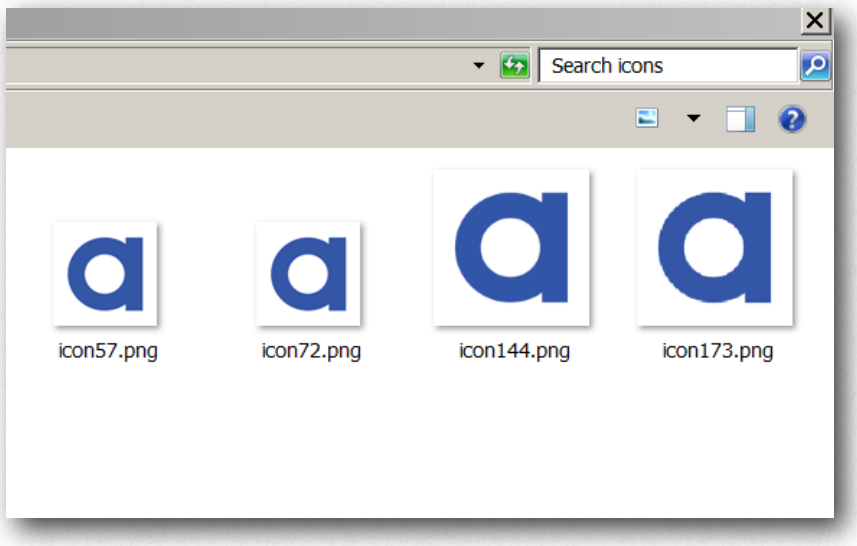


App Icon and Splash Screens



Check Generate Image Resource Files and ***select*** the Alpha icon144.png for the Master Icon

from the PhoneGap/appIcons folder, included in the Alpha Anywhere executable directory.



The PhoneGap App Builder will generate all of the required app icons for the targeted platforms at all required sizes and resolutions for each. The app icon is the icon that is installed and displayed on the devices home screen.

When building your own app icons, you should build the largest size that is required by all of the targeted devices. A 144px x 144px icon is the largest required for iOS and Android devices. If you were building for Windows Phone 8, a 173px x 173px icon would be best.

Since Alpha Anywhere is generating all of the required icons from the one icon selected, it's best to generate images that are smaller in size from a larger icon as this preserves the image resolution, eliminating pixillation of the smaller image. The same is true when generating the app splash screens, which you can leave blank for this test application, (The default PhoneGap splash screens will be used).

In the App Options section, ***set the Application ID*** to the reverse domain name of your app.

Show Advanced Options	☐
App Options	
Application Id	com.alphasoftware.myApp
Version	0.0.1
Application Name	My App ...
Description	An Alpha Anywhere PhoneGap App ...
App Author	
App Email	
App Info Link	
PhoneGap Version	3.1.0 ▾
Target Device	universal ▾
Supported Orientations	default ▾
Full Screen	☐
Allowed Access Origin	*
Android Only	
Minimum SDK Version	7 ▾

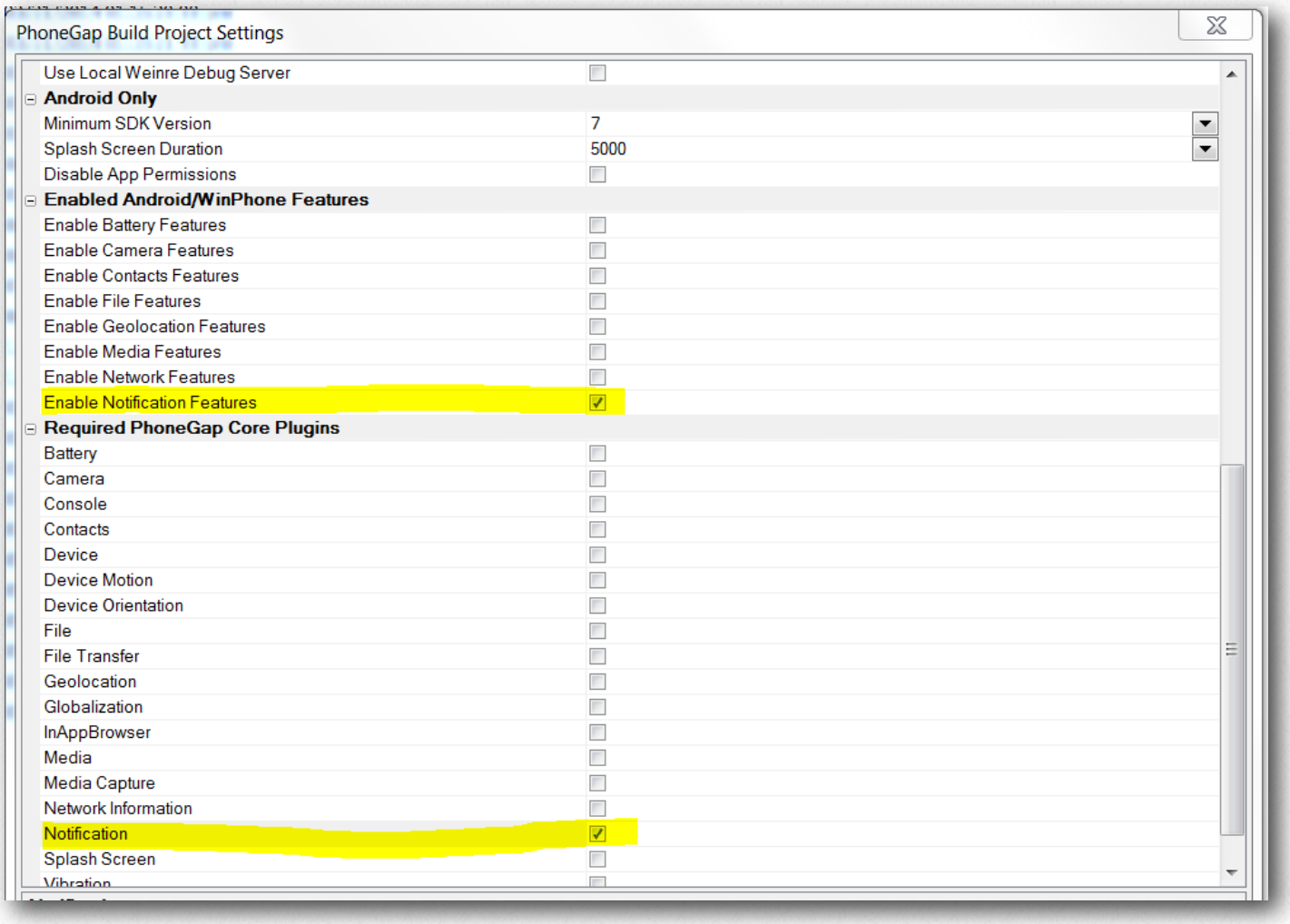
In this test app we've used **com.alphasoftware.myApp** but you can use whatever you would like. If you plan to distribute your app through an app store, the App Id must match the bundle identifier of your application.

Verify or set the **PhoneGap version property** to 3.1.0, or greater.

I have set the **Allow Access Origin property** to * which will allow your app to access all other URL's on the Internet. Our app doesn't actually require this setting this however it is a common requirement to load data from other web services, or an Alpha App Server or possibly to load a web page in the inApp Web Browser and if the URL is not specified in the Allowed Access Origin Property, the request will be blocked by PhoneGap. This is a common point of failure so you need to be aware of this setting from the first app you publish.

Android Only Section

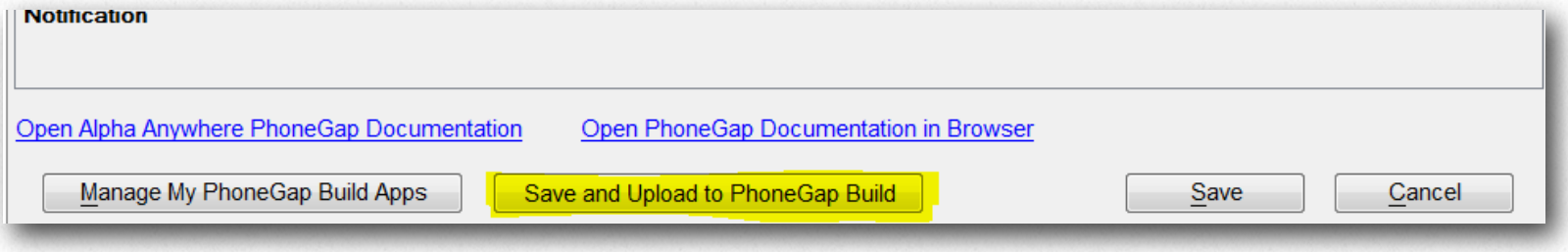
In the Android Only section, ***check the Enable Notification Features***. When the App is installed on an Android device, the user will be made aware of the features the app requires and they may choose to allow or abort the app installation.



Required PhoneGap Core Plugins Section

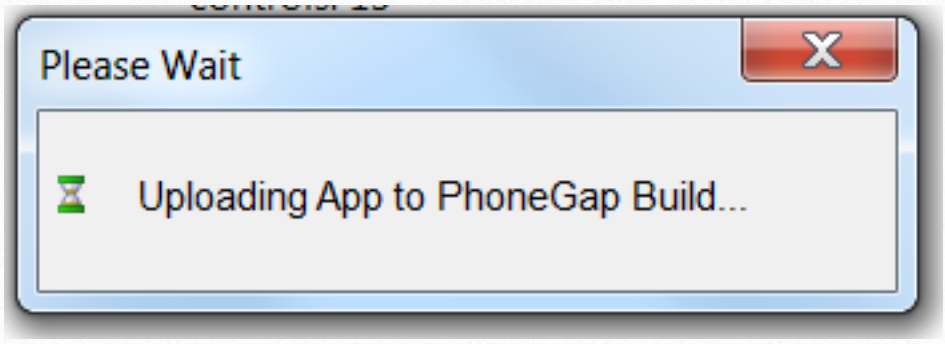
In version 3.x of PhoneGap, all native functionality is handled through plugins, even the core PhoneGap functions. In previous versions, all of the core features/functions were always included in the project. This made the project files much larger than necessary so rather than including everything in version 3.x, a change was made to exclude ALL features/functions and allow the programmer to specify exactly what features/ functions are required by their app.

Our app only uses the Notification plugin (it beeps) ***so make sure the Notification plugin is checked***.



At this point enough information has been provided to build the PhoneGap application. ***Click Save and Upload to PhoneGap Build*** and the PhoneGap App Builder will create the application (if it did not previously exist), all of the required app icons will be created, the app will be packaged up, and a new app will be uploaded to PhoneGap Build.

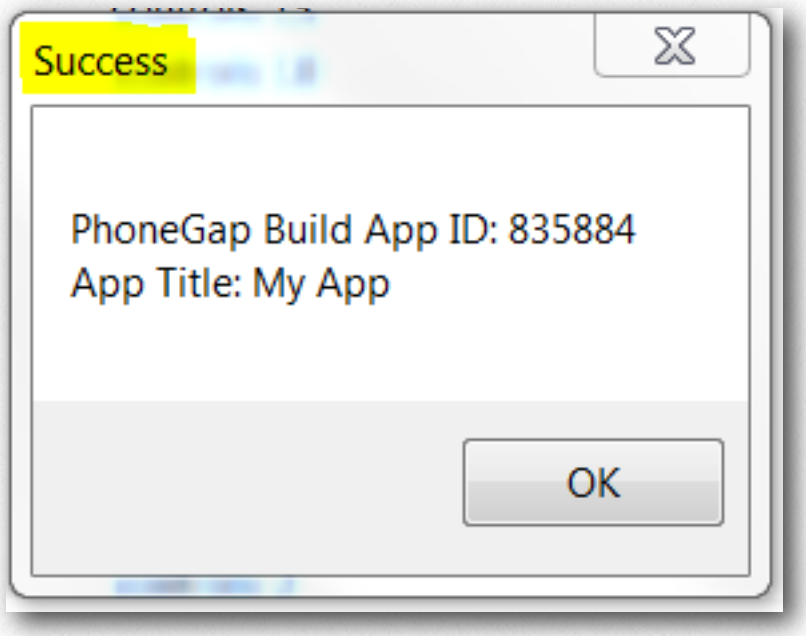
You will see numerous progress pop up windows, similar to the one below.



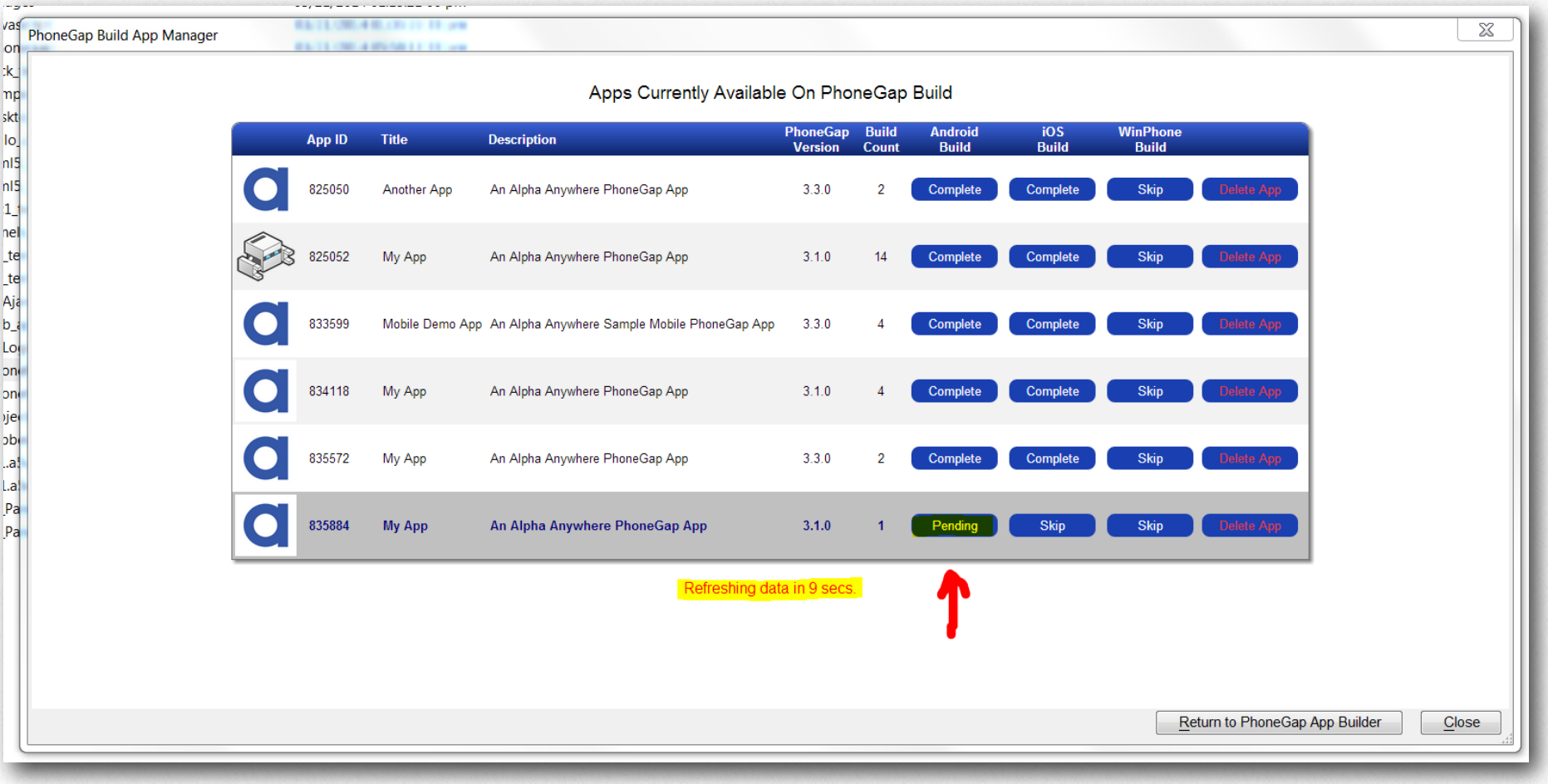
First the app files are generated.

Next the builder will try to login to the PhoneGap Build service, If this fails, a popup window will let you know and you will be returned to the PhoneGap App Builder, where you can correct your login credentials and try again.

Upon successful upload and app creation on PhoneGap Build, you will see a success popup window.



Click OK and the **PhoneGap App Manager** will appear, with a listing of all of your available apps on PhoneGap Build. Since this is your one and only app, you will only see one row however, I've included an image that shows numerous apps. The active app is always displayed in a darker gray row.



When the app is initially uploaded, the numerous builds you've selected take place on PhoneGap Build as an asynchronous process. The time it takes to finish a build is dependent upon the load on the PhoneGap Build servers at that time and it seems to vary by the time of the day.

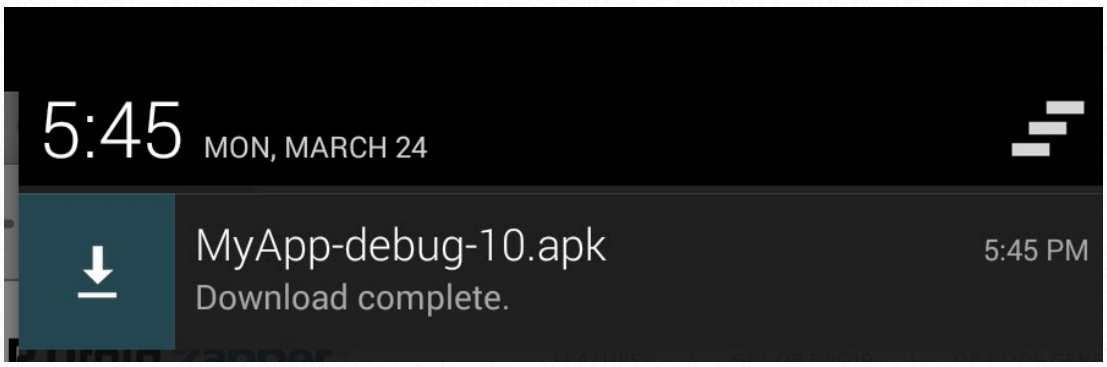
The PhoneGap App Manager window will automatically refresh the contents every 15 seconds until all builds either complete or fail. You do not have to leave this window visible for the build process to complete.

Once the build process is completed, click on the Complete button and a QRCode will be displayed, that will allow you to install the app on an Android phone or tablet.

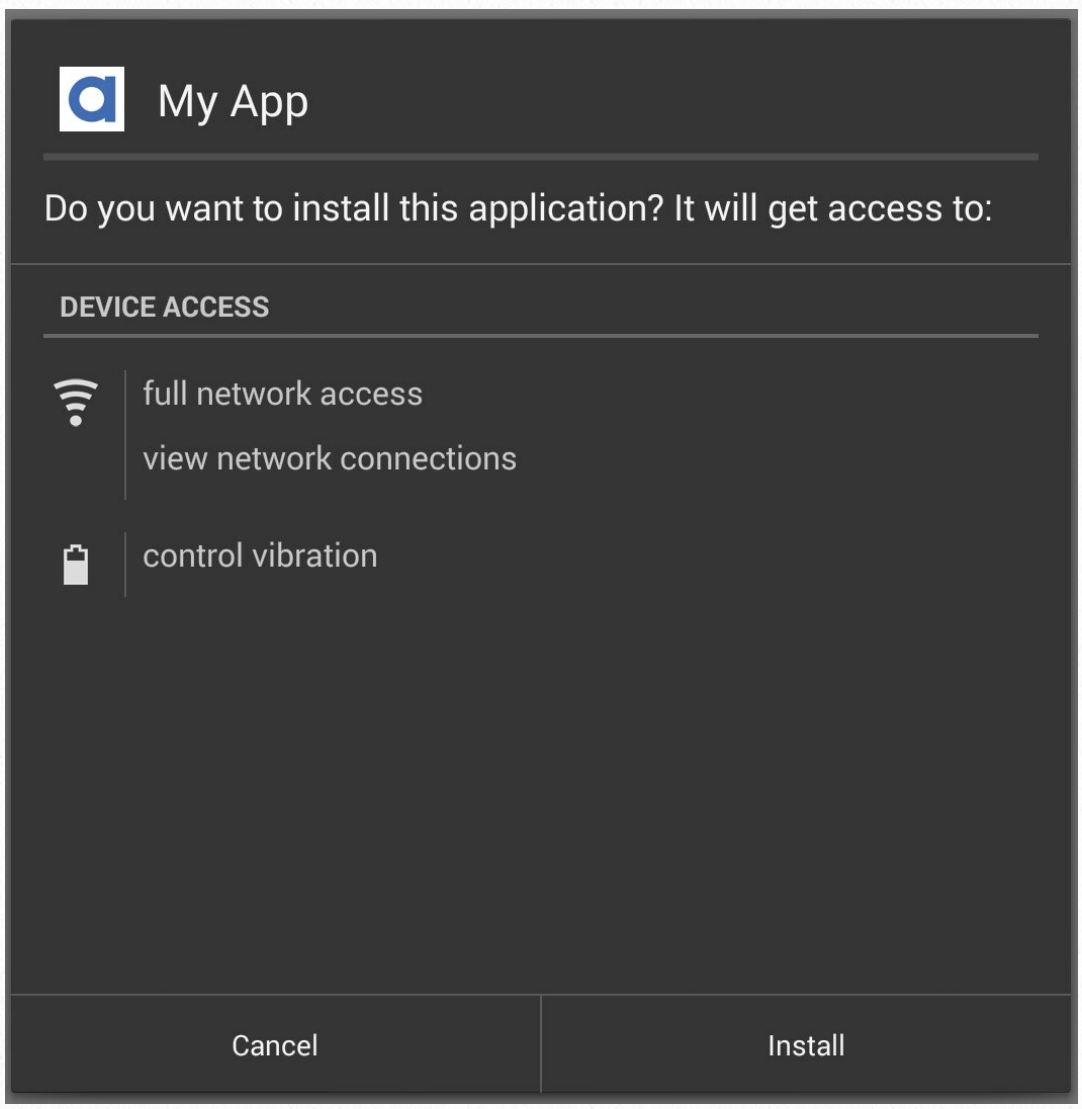
There are a number of free third party Android scanner apps that will work. I seem to use QR Droid most often.

Once the file is downloaded, tap and pull down from the top of the screen to see the .apk file.

Tap on the .apk file to load the app onto the device.



Click the Install button.



Click the Open button to launch the app.

Success!

Here we see the test app successfully loaded on a Samsung Galaxy Note 8.

Tap the Native Beep button and you should hear a beep tone.

Congratulations, you've just successfully built and installed a native mobile app!

Summary

You have now:

- Built an Alpha Anywhere UX component that included PhoneGap functionality.
- Use the integrated PhoneGap App Builder to configure and build a PhoneGap App on your development machine.
- Generated a full series of app icons at all device and OS specific sizes and resolutions for your app.
- Uploaded and created a native app using PhoneGap Build
- Installed the app on an Android device for testing.

4

iOS Provisioning

If you want to build apps for the iPhone or iPad, you will need an Apple Developer Certificate and a Provisioning Profile to install the signed IPA file on your iOS device.

To submit your credentials to PhoneGap Build, you need 3 things:

- A Developer Certificate with private key (*.p12)
- A provisioning profile (*.mobileprovision)
- The password for the private key

The **Developer Certificate** is added to your Mac's keychain using Keychain Access. It is used to digitally sign your code.

The **Provisioning Profile** is a file that will exist on your development device(s) and your Mac. It will reference a Developer Certificate, an App ID (a development profile can use a wildcard) and a list of device ID's that the app can be installed on for testing.

It is *assumed you have a Mac for iOS development running OSX and that you have an iOS developer account*.

If you do not have a Mac, you can setup a virtual Mac at http://www.macincloud.com.

If you need to setup an Apple iOS Developer Account, visit http://developer.apple.com.
Cost is $ 99.00 per year.

To install an app on your development device you need to get a valid developer certificate from Apple.

Apple has extensive documentation on setting up all of the resources required to get a valid developer certificate at http://developer.apple.com.

The interface to the developer site is constantly changing so it is pointless to try to document it in detail here.

You'll also need to register your development devices for testing. You can register up to 100 devices for ad-hoc distribution. See Apple's Developer web site for more info.

The guidelines below should be helpful for uploading your credentials to PhoneGap Build.

Step 1: Get The Certificate

Login into your Apple Developer Account and go to the iOS Provisioning Portal. Next, go to the Certificates section.

If you have a valid certificate already, you can download it now and skip to step 2. If not, create it as documented below and download afterwards.

Step 1.1: Requesting A New Certificate

Open the keychain on your computer. (Use spotlight shortcut command/space, and enter "Key"). Then launch Keychain Access.

From the menu, select "***Certificate assistant***" and choose "**Request a certificate from a Certificate Authority (...)**".

Enter your Apple ID email address and click on "Save to disk". Also check the checkbox below the email address box.

You should be prompted for a password for your keys. Enter any string that you would like to use, just make sure to write it down for safekeeping. You will need the password for PhoneGap Build.

Upload your Certificate Request file (created above) and wait until the certificate is issued (may take a few minutes, refresh the site a few times).

Download your certificate.

Step 2: The Provisioning Profile

Click on the "Provisioning" section.

You can either create a new provisioning profile or edit an existing one. Make sure to have your certificate enabled in the profile you are going to use.

Download the desired provisioning profile.

Now you have a .mobileprovision file and a certificate (.cer) - there is still one missing: The certificate/key bundle (*.p12) - we will create that now.

Step 3: Bundle the certificate and your key

Open the keychain tool on your Mac.

Import the certificate by selecting "File" > "Import object" from the menu. The certificate should now show up below the key you used to request the certificate.

Right-click on your key and select "Export".

This is your *.p12 file for use with build.phonegap.com

Add a new signing key and upload your *.mobileprovision and *.p12 file.

You will need to enter the password to unlock the key on PhoneGap Build prior to creating an IPA file with PhoneGap Build.

5

More Functionality, Using 3rd Party

This example app incorporates native notifications, device information, inAppBrowser, bar code scanner and the flashlight plugin. It also demonstrates a technique to call Apple's Search API web service, parse the JSON results and format the results in a HTML template within a Panel Card. No remote Alpha server access is required.

Included in the PhoneGap folder (located within the Alpha exe directory) is an **examples\1** folder that contains an example component and a background png file that we will use to build a more advanced PhoneGap app.

Copy pgExample1.a5wcb and background.png into an existing or new Web Project within Alpha Anywhere.

Open the pgExample1 UX component and ***click on Working Preview***.

pgExample1 UX Component, Working Preview

This application includes seven (7) buttons on the first Panel Card that call native PhoneGap functions. Since these buttons are calling native PhoneGap functions, they will not work in Working or Live Preview on a desktop browser. The app needs to be installed on an actual device (or one of the emulators) to work.

Beep calls the native PhoneGap **navigator.notification.beep** function, which will cause your device to generate a beep sound.

Vibrate calls the native PhoneGap **navigator.notification.vibrate** function. Not all devices support vibrate. You will need to make sure your phone/tablet is set to vibrate on ring in order for this function to work on your device.

Native Alert calls the PhoneGap **navigator.notification.alert** function, which displays a native alert dialog as opposed to a browser “alert” dialog. On iOS devices you will notice the dialog title does not contain the referencing page (which in this case will be index.html). A callback function is supported and is called when the user closes the alert dialog. In this case, the callback function displays a standard javaScript alert dialog. You will most certainly notice a difference between the native alert dialog and the JavaScript alert dialog on an iOS device. On an Android device, there is little difference.

Native Confirm calls the Phonegap **navigator.notification.confirm** function, which displays a native confirm dialog. Similar to an Alert in many ways, the confirm dialog adds the ability to display two (2) buttons and passes the one based index number (1 or 2, instead of a typical zero based array, which passes 0 and 1) in the callback when a button is pressed.

Native Prompt calls the native PhoneGap **navigator.notification.prompt** function, which displays a native prompt dialog. The native prompt dialog can be used to prompt the user to enter any information required and can capture a user entered string. Two (2) buttons can be used to display

Flashlight calls the PhoneGap third party plugin **flashlight** function. This function will turn on the camera light on the back of a phone for three seconds. If the device does not include a camera light, a callback will display a native alert dialog to indicate that a light is not available on the device.

Scan and Search iTunes calls the PhoneGap **barcodescanner** third party plugin which calls the **cordova.plugins.barcodescanner.Scan** function. This function will bring up a barcode scanner which uses the devices camera and some very clever image processing software to allow you to scan in a barcode from the back of most any book and return the results to a success (or fail) callback function.

In the example component, the success callback function makes a client side call to Apple’s iTunes Search API where the text result of the barcode scan is submitted to iTunes Search for an ISBN lookup. If the ISBN from the book is available within iTunes, the descriptive and pricing information will be displayed on a second PanelCard that will perform a slide transition into view.

Panel Footer

iTunes Info Button will navigate to the second PanelCard that contains a freeform HTML control that will display iTunes related information after a books ISBN has been scanned.

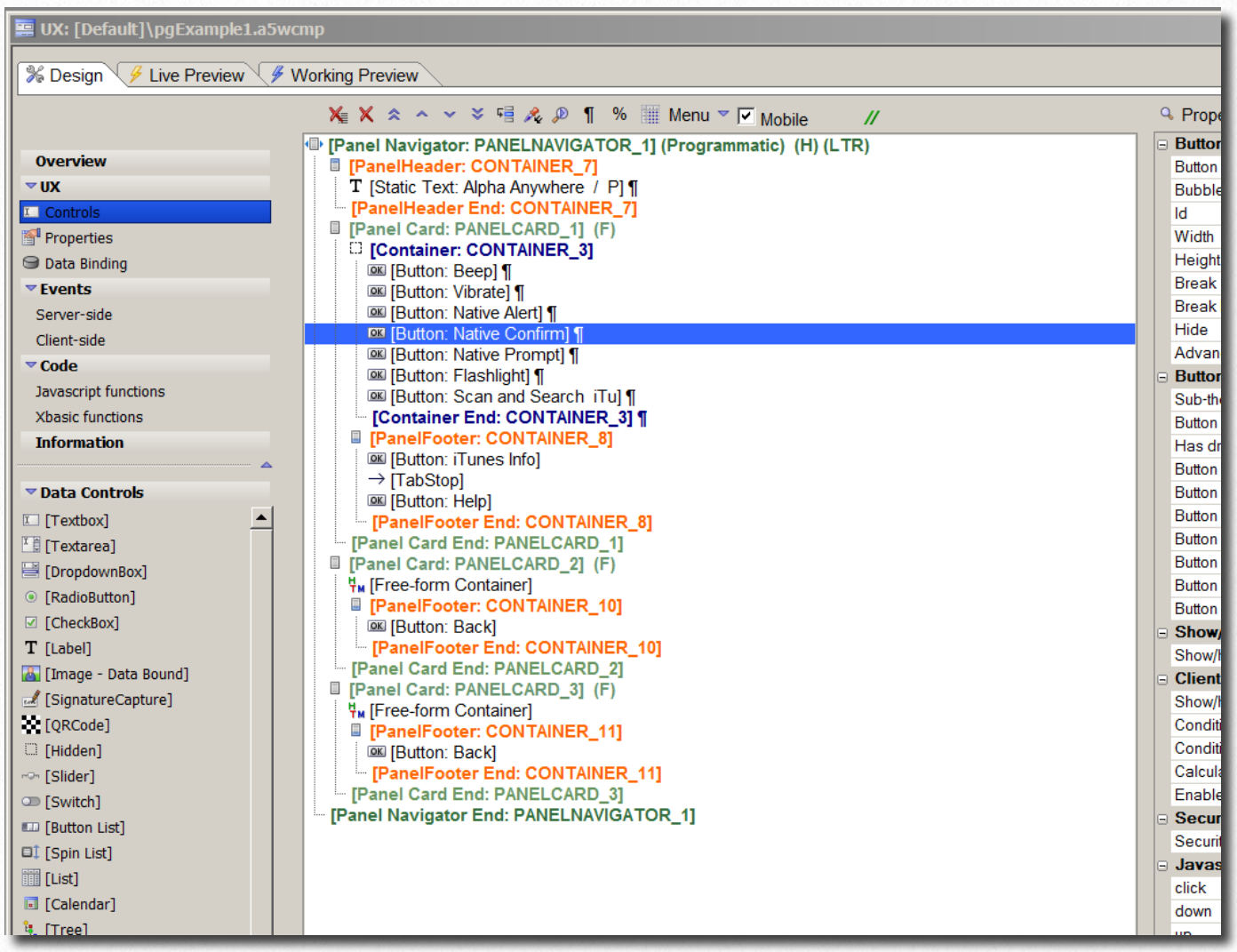
Help Button navigates to a third panel that will display application help. The help content is contained within a freeform HTML control, that initially contains an empty div. When the app is fired up, the client side onitialize complete event is used to populate the div with the help HTML, which is defined in the JavaScript Functions of the component. This technique is one I prefer because it allows the definition of the components HTML content to be centralized and not tied to a specific control.

There's a lot going on here!

When I created this component I used a lot of different techniques that I've developed over the years. As with most everything in programming, there are a lot of different ways to accomplish something. I hope you learn a few tricks from tearing this component apart and seeing how I approach component development with Alpha Anywhere. The techniques I've used are what I consider best practice.

Let's take a look at how this component was built.

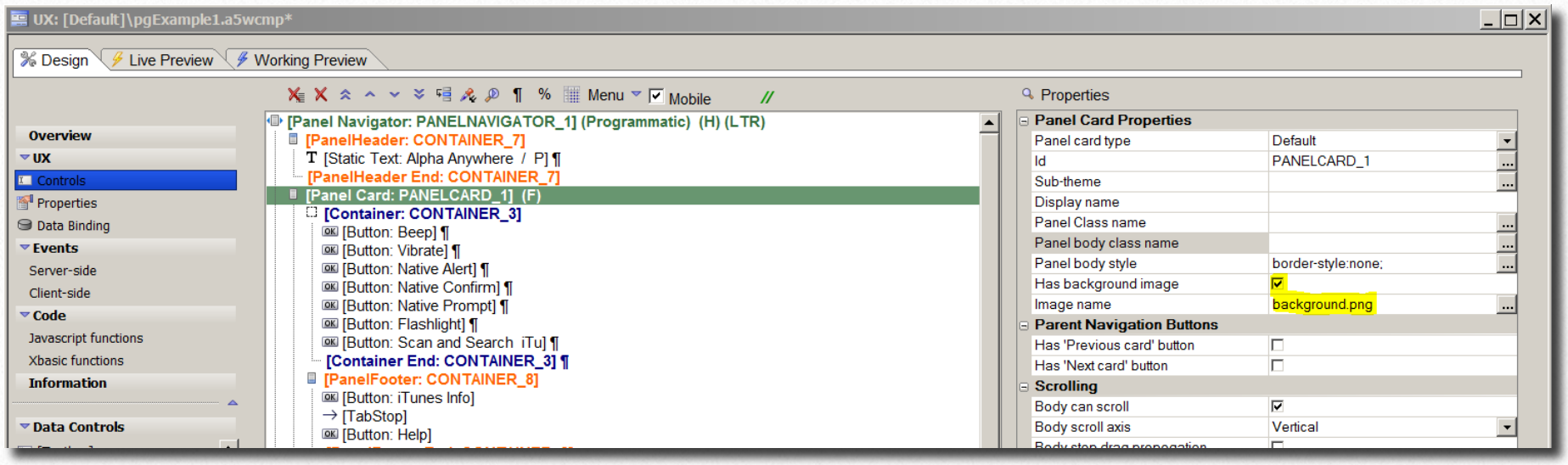
Click on the Design tab.



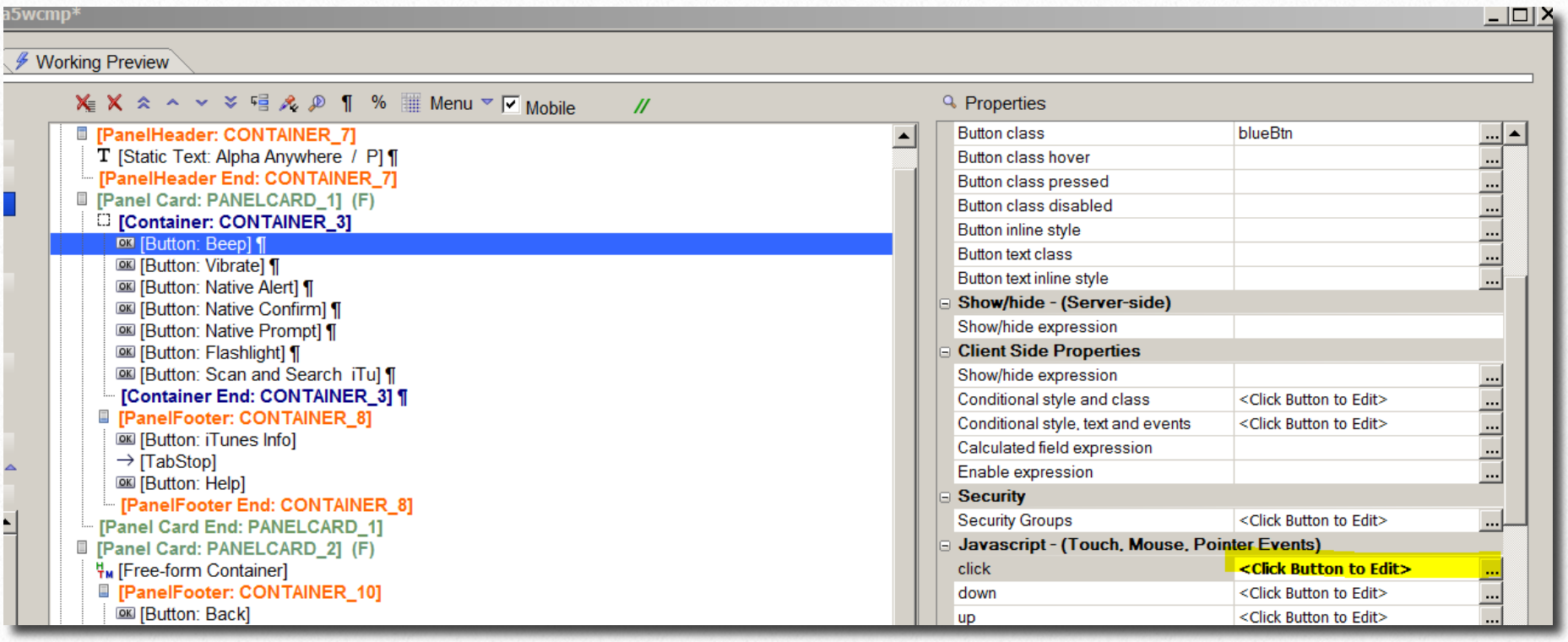
pgExample1 UX Component, Design

The component includes a programmatic Panel Navigator that includes a static Panel Header. The Panel Header container alignment is set to centered and it's width by convention is 100%. That will center the text.

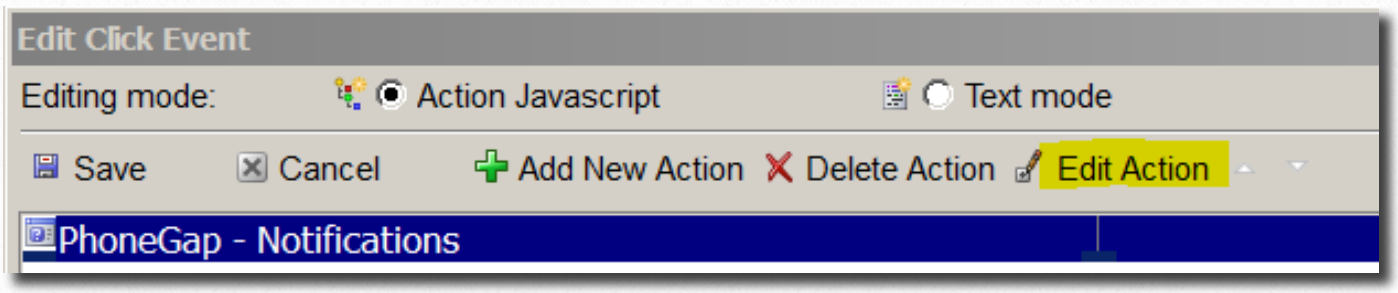
The first Panel Card includes a Container (for centering the buttons) and a series of buttons (with a width defined of 150px) and a Panel Footer with two (2) buttons that navigate to the other panels. It also includes a a background image, which I've tailored to an iPhone 5 in portrait orientation.



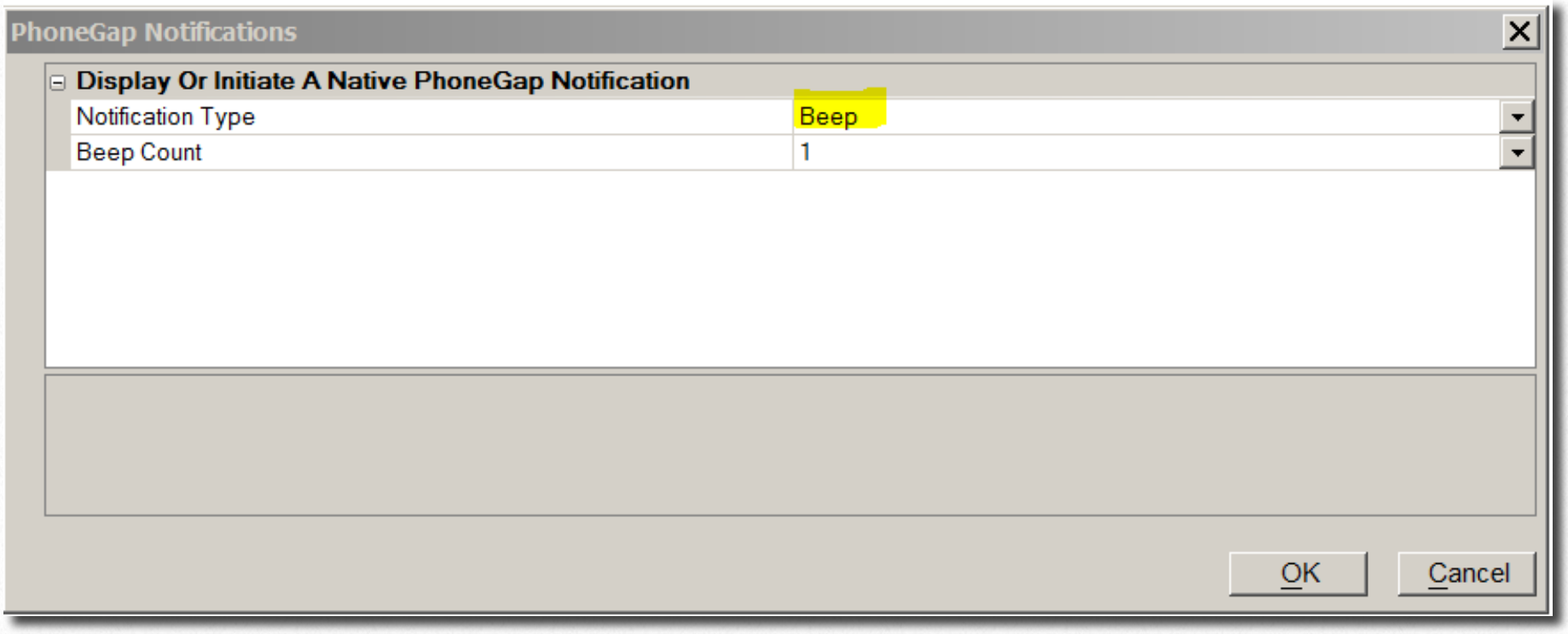
Let's take a look at the abstract click event handler on the Beep button.



A PhoneGap Action JavaScript Notifications function has been assigned.



Click on Edit Action

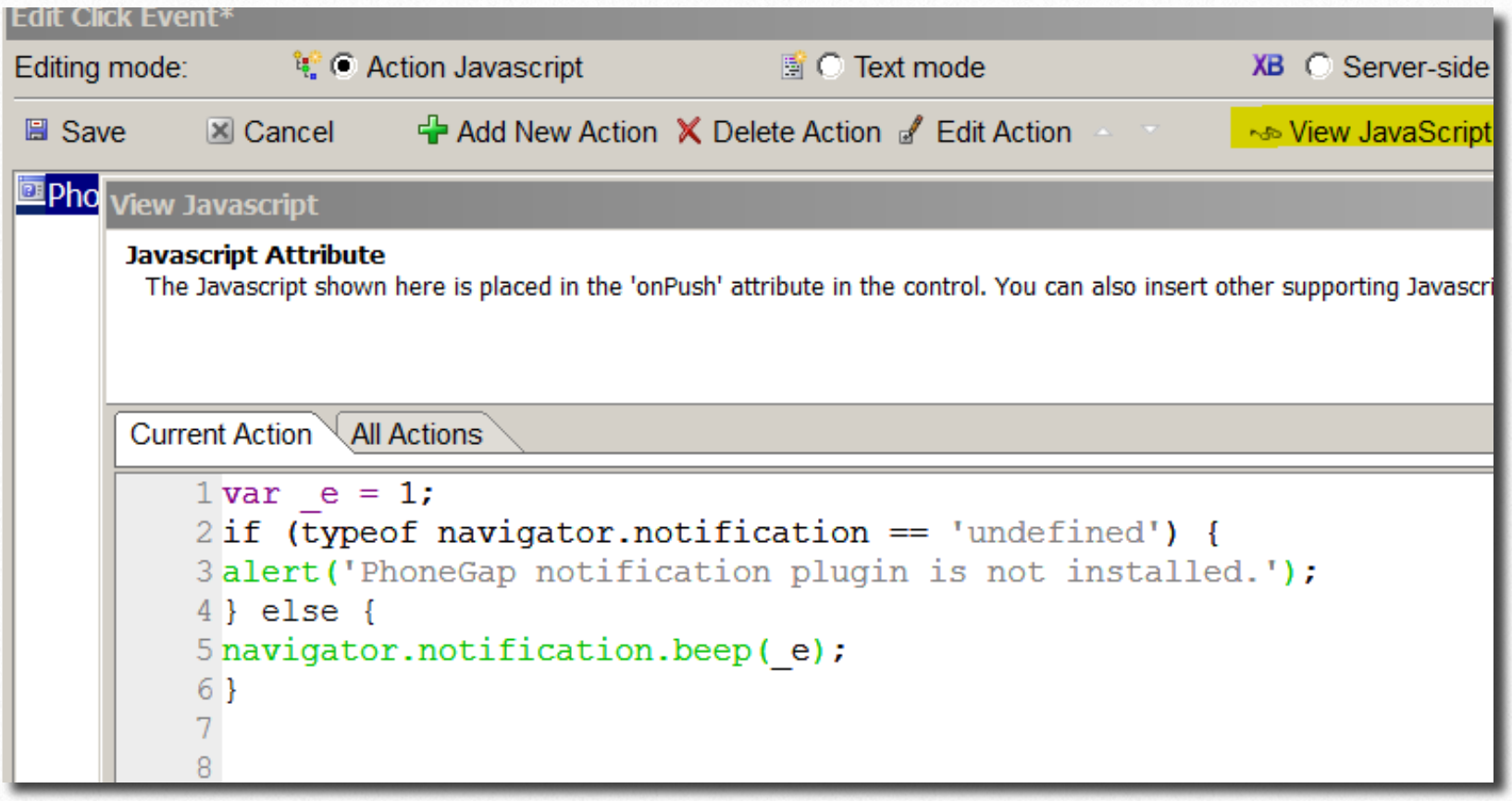


Notice the notification type selected is Beep and the Beep Count is 1.

Per the PhoneGap documentation, the beep count is ignored on iOS devices.

Click OK and let's take a look at the JavaScript that the Action JavaScript builder generated.

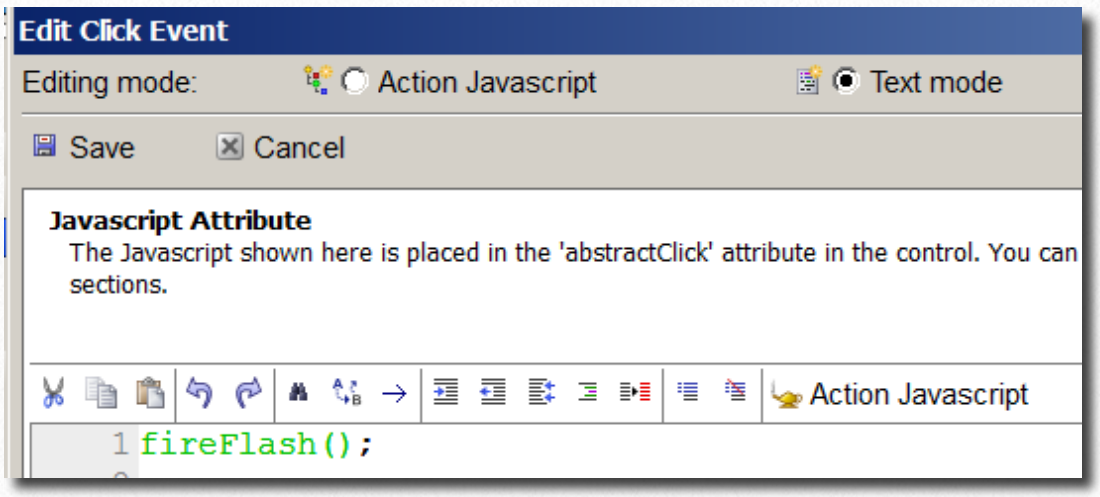
This is the JavaScript that runs when the abstract click event on the button fires.



At this point, only a handful of Action JavaScript functions have been defined to generate the JavaScript for PhoneGap 3. (when this documentation was written, it is very early in the product rollout cycle), however this will be expanded on a continuing basis to make it easy for you to use all of PhoneGap's capabilities.

As you will see later on, I've written the JavaScript to interact with the third party plugins for the Flashlight and the Bar Code Scanner used by this component. Keep in mind that by referring to the PhoneGap documentation, and writing a little bit of JavaScript, you can do anything with Alpha Anywhere and PhoneGap today.

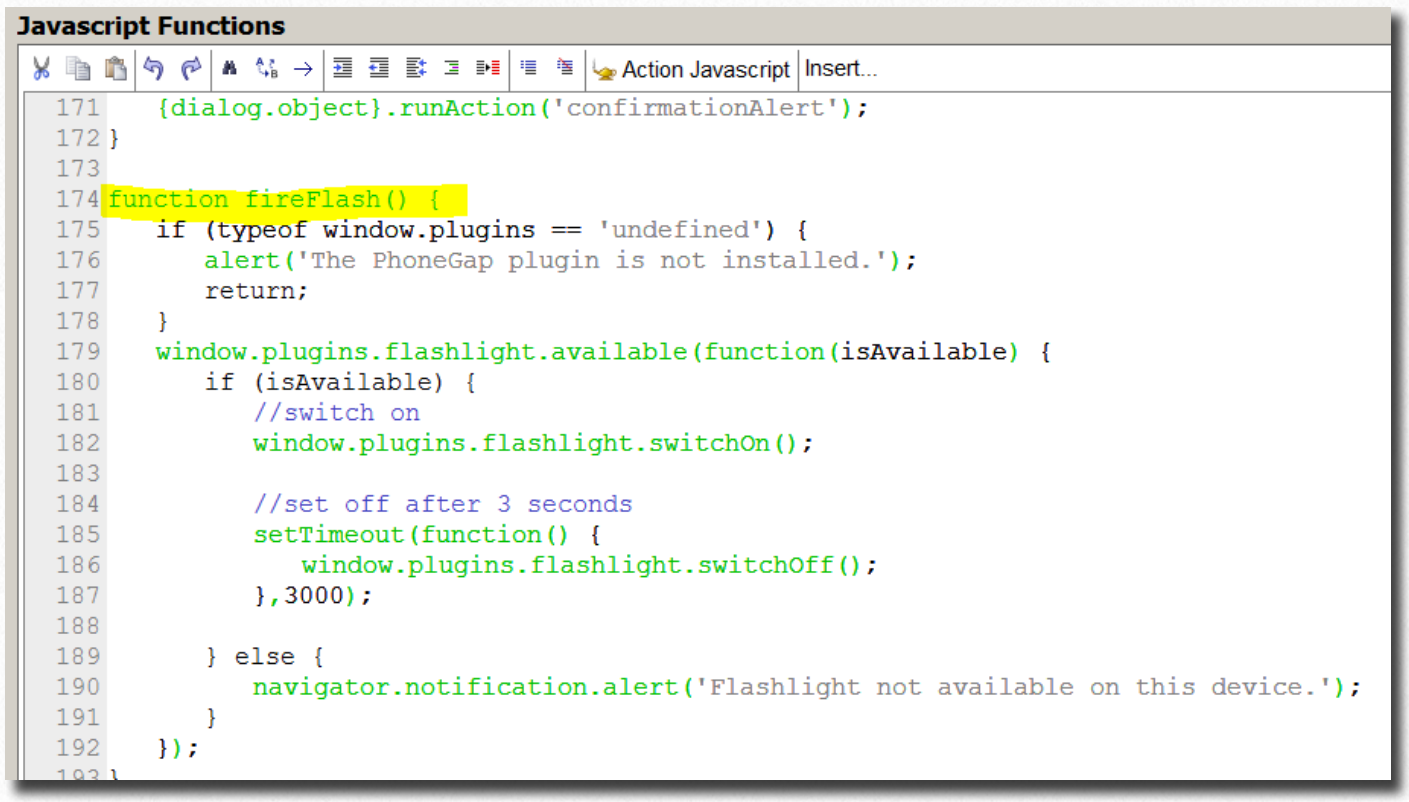
Close the JavaScript Preview Dialog and take a look at the abstract click handler on the Flashlight button.



Notice the dialog is set to Text Mode, which allows you to enter the JavaScript directly. The **fireFlash()** function is called will be called when the abstract onclick event fires. The fireFlash function is defined in the component's JavaScript functions.

Cancel out of Edit Click Event Dialog and open the JavaScript Functions panel.

Scroll down to the fireFlash function definition.



The fireFlash function is called when the Flashlight button is clicked

First a check is made for the existence of the window.plugins object (which the Flashlight plugin JavaScript files create) and if the object is not defined, an alert is displayed and the function returns. This will prevent an error in the app and allows you to test the component in Working Preview. This technique is used throughout the button event handlers.

Next a test is done to determine if the mobile device physically includes a photo flash and if so, the flashlight is turned on for 3 seconds.

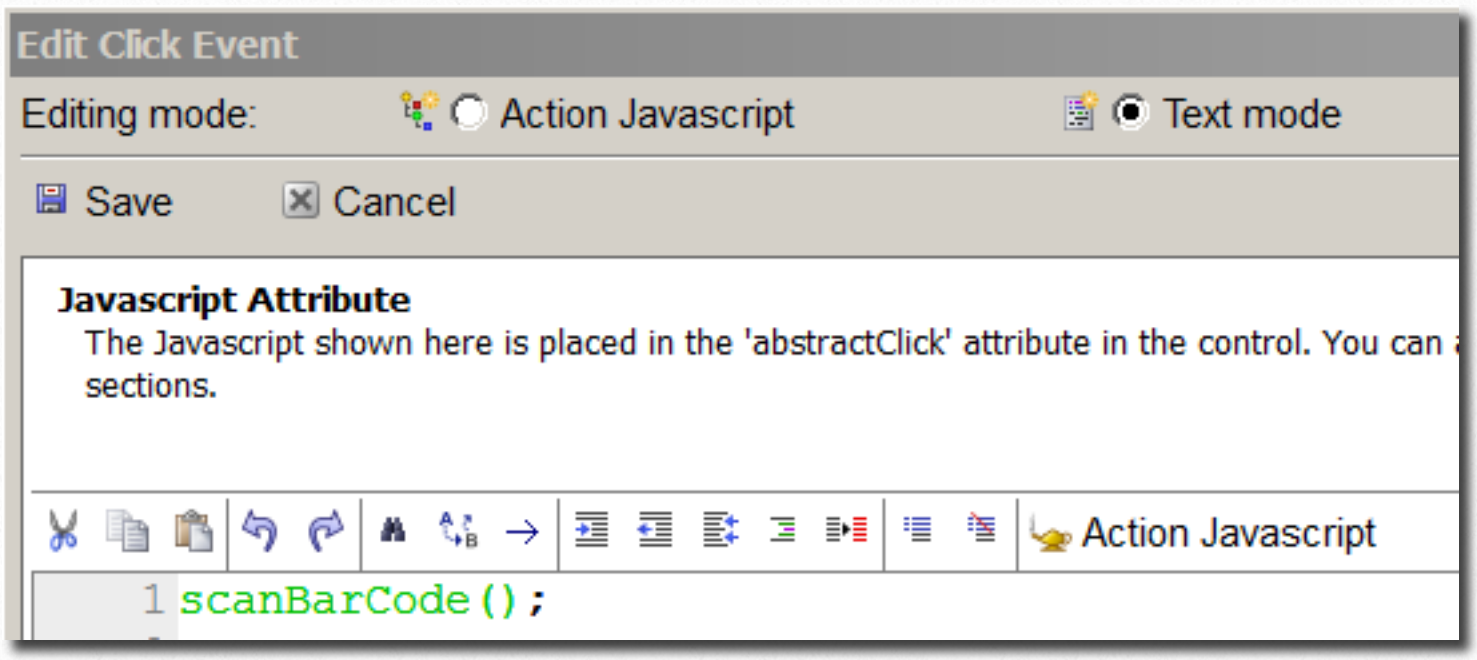
> ***Remember, the PhoneGap functions are not going to work on a Desktop Browser. They are ONLY designed for use on a mobile device.***

If the device does not include a photo flash, a native notification alert dialog is displayed to let the user know the plugin is working, but a flash is not available on the device.

If you take a look at the PhoneGap Build, third party plugins documentation for the Flashlight plugin, you will see the sample code which looks quite similar.

See: https://build.phonegap.com/plugins/294 and click on the link under Docs.

Next, take a look at the abstract onclick event handler for the Scanner button. Here the **scanBarCode** function is called. This function is also defined in the component's JavaScript Functions.



The scanBarCode function does a quick check for the cordova object. If this object does not exist then either we are not running within PhoneGap on a mobile device or the bar code scanner third party plugin is not installed. Since either condition will cause a JavaScript error, an alert dialog is displayed and the function exits gracefully.

Javascript Functions

Action Javascript | Insert...

```
}

function scanBarCode(){
    // this is a minimal check to verify plugin installation
    if (typeof (cordova) == 'undefined') {
        alert('PhoneGap plugin is not installed.');
        return;
    }

    cordova.plugins.barcodeScanner.scan(function(result) {
        //save to global for use in callback if ISBN not found
        {dialog.object}.appVars.scannerResult = result;
        if (result.cancelled) {
            navigator.notification.alert('Scan cancelled.');
        } else {
            var _lookupUrl = 'https://itunes.apple.com/lookup?isbn='+result.text;
            {dialog.object}.ajaxCallbackCrossDomain(_lookupUrl,'iTunesCallback');
        }
    },
    function(error) {
        navigator.notification.alert('Scanner failed: '+error);
    });
}
```

The scanBarCode Function

Next the **cordova.plugins.barcodeScanner.scan** function is called. Below is a snippet from the Bar Code Scanner plugins documentation page on GitHub. As is very typical with PhoneGap, the call to the scan function is asynchronous and requires two callback functions, one for success and one for fail.

iOS

- QR_CODE
- DATA_MATRIX
- UPC_E
- UPC_A
- EAN_8
- EAN_13
- CODE_128
- CODE_39
- ITF

`success` and `fail` are callback functions. Success is passed an object with data, type and cancelled properties. Data is the text representation of the barcode data, type is the type of barcode detected and cancelled is whether or not the user cancelled the scan.

A full example could be:

```
cordova.plugins.barcodeScanner.scan(
   function (result) {
       alert("We got a barcode\n" +
             "Result: " + result.text + "\n" +
             "Format: " + result.format + "\n" +
             "Cancelled: " + result.cancelled);
   },
   function (error) {
       alert("Scanning failed: " + error);
   }
);
```

As in the example code, the success and fail functions are anonymously defined within the scanBarCode function. This technique is very common for asynchronous JavaScript functions and is used throughout the PhoneGap example code. You can choose to define independent external functions and simply call those functions if you wish.

The Scan Success Function

Our anonymous success function includes a result parameter which will contain a result object if the bar code scanner is able to read a bar code. From the PhoneGap example code it is easy to see the result object contains:

result.text : contains the string encoded within the barcode or the QRCode
result.format : contains a string with the barcode format
result.cancelled : contains true if barcode scan is cancelled

A Note on JavaScript Style

In this component, I've used a style that I prefer for JavaScript.

Here are the basic rules I've followed:

1. All globals are assigned to the {dialog.object}.appVars object.
2. All function parameters are defined in lowercase, no leading characters.
3. All variables that are local to a function are defined with a leading underscore example: _result.

There is no right or wrong technique, just guidelines.

Google has a decent JavaScript Style Guidelines available at:

http://google-styleguide.googlecode.com/svn/trunk/javascriptguide.xml

Whatever style you (or your team) develop, try to be consistent.

Notice the result object is stored to an instance variable (previously defined) called **{dialog.object}.appVars.scannerResult**. This is required because the iTunesCallback function requires access to this information after a call is made to the Apple Search API. If the ISBN is NOT found in the iTunes

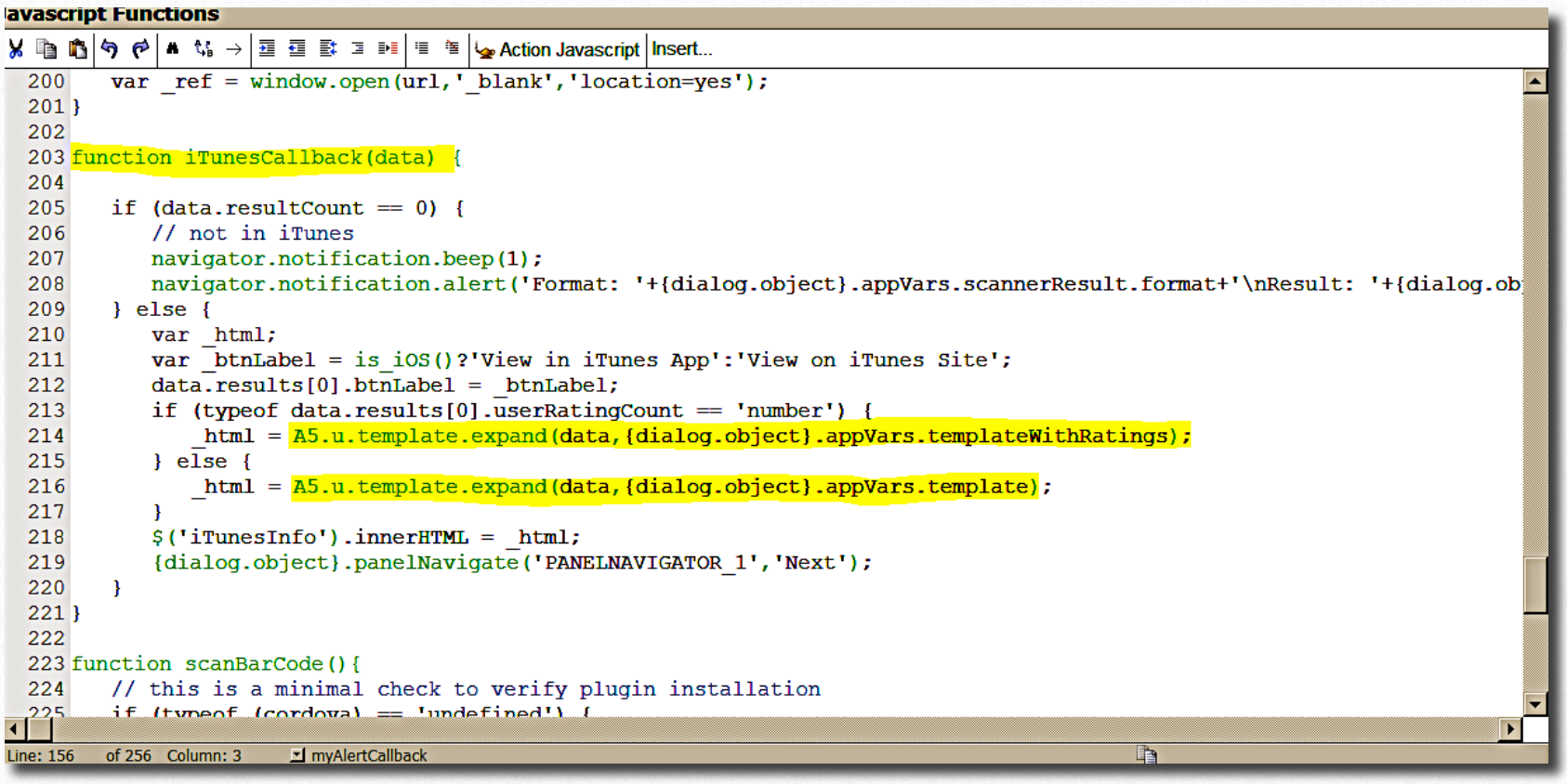
store, the callback will display an alert that will reference the scanned ISBN and the barcode type. This information is in the result object.

> In this example component, **all of the globals are defined as part of the {dialog.object}.appVars object**. This is referred to as an **instance variable** since the variables are assigned to a child object (appVars) of the UX component instance {dialog.object}. They are in fact global variables to the object instance.
>
> This technique keeps the components global variables contained within the object designated by the template placeholder {dialog.object} and keeps the global namespace clean, an important consideration in JavaScript programming.
>
> **Note:** *You may wonder why the placeholder {dialog.object} is used instead of a placeholder named {ux.object}. {dialog.object} comes from a legacy naming convention, used in previous versions of Alpha Five. The UX was was derived from the legacy dialog component.*

At the top of the function, if result.cancelled is true the user cancelled the scan and a PhoneGap native alert dialog is presented.

If a result is returned, the scanned text is submitted to Apple's Search API for a lookup in the iTunes store.

There are a number of ways to do this but in this case the new **{dialog.object}.ajaxCallbackCrossDomain** method is used to make the request.



The iTunesCallback function using the new template method of Alpha Anywhere

The callback function that will process the returned result is called **iTunesCallback.**

Since the result from iTunes may or may not contain user ratings, a test is made to see if the data returned contains a variable named **data.results[0].userRatingCount**. If the data.results[0].userRatingCount variable exists, a template that includes ratings is used, otherwise, a template that excludes ratings is used.

Javascript Functions

Action Javascript | Insert...

```
49 // This recommended technique doesn't pollute the global namespace
50
51 {dialog.object}.appVars={};
52 {dialog.object}.appVars.msg = '';
53 {dialog.object}.appVars.scannerResult = {};
54
55 {dialog.object}.appVars.template = [
56       '<center><img src = "{results[0].artworkUrl100}"></center><br><br>',
57       '<b>{results[0].trackName}</b><br><br>',
58       '<b>Author: </b>{results[0].artistName}<br><br>',
59       '<b>Type:</b> {results[0].kind} <b>Price:</b> {results[0].formattedPrice}<br><br>',
60       '<center><button class="blueBtn" ontouchend="launchInAppBrowser(\'{results[0].trackViewUrl}\');">{results[0]
61       '{results[0].description}'].join('');
62
63 {dialog.object}.appVars.templateWithRatings = [
64       '<center><img src = "{results[0].artworkUrl100}"></center><br><br>',
65       '<b>{results[0].trackName}</b><br><br>',
66       '<b>Author: </b>{results[0].artistName}<br><br>',
67       '<b>Type:</b> {results[0].kind} <b>Price:</b> {results[0].formattedPrice}<br>',
68       '<b>Rating:</b> {results[0].averageUserRating} based on <b>{results[0].userRatingCount}</b> reviews.<br><br>
69       '<center><button class="blueBtn" ontouchend="launchInAppBrowser(\'{results[0].trackViewUrl}\');">{results[0]
70       '{results[0].description}'].join('');
```

The templates are defined as instance variables and are part of the **{dialog.object}.appVars** object. This is done for efficiency. You could define the templates within the iTunesCallback function but it's more efficient to define the templates once as a global and reference them when needed.

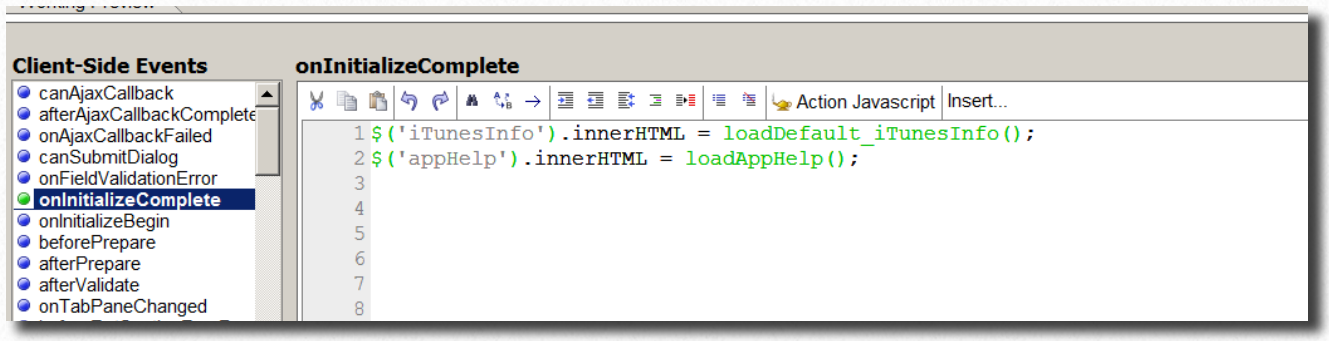
Performance and efficient memory use on mobile devices is important.

Panel Card 2 has a freeform HTML container that contains a single div, ** **. Once the HTML is derived from the template, the iTunesInfo div is populated with this HTML and Panel Card 2 is transitioned into view.

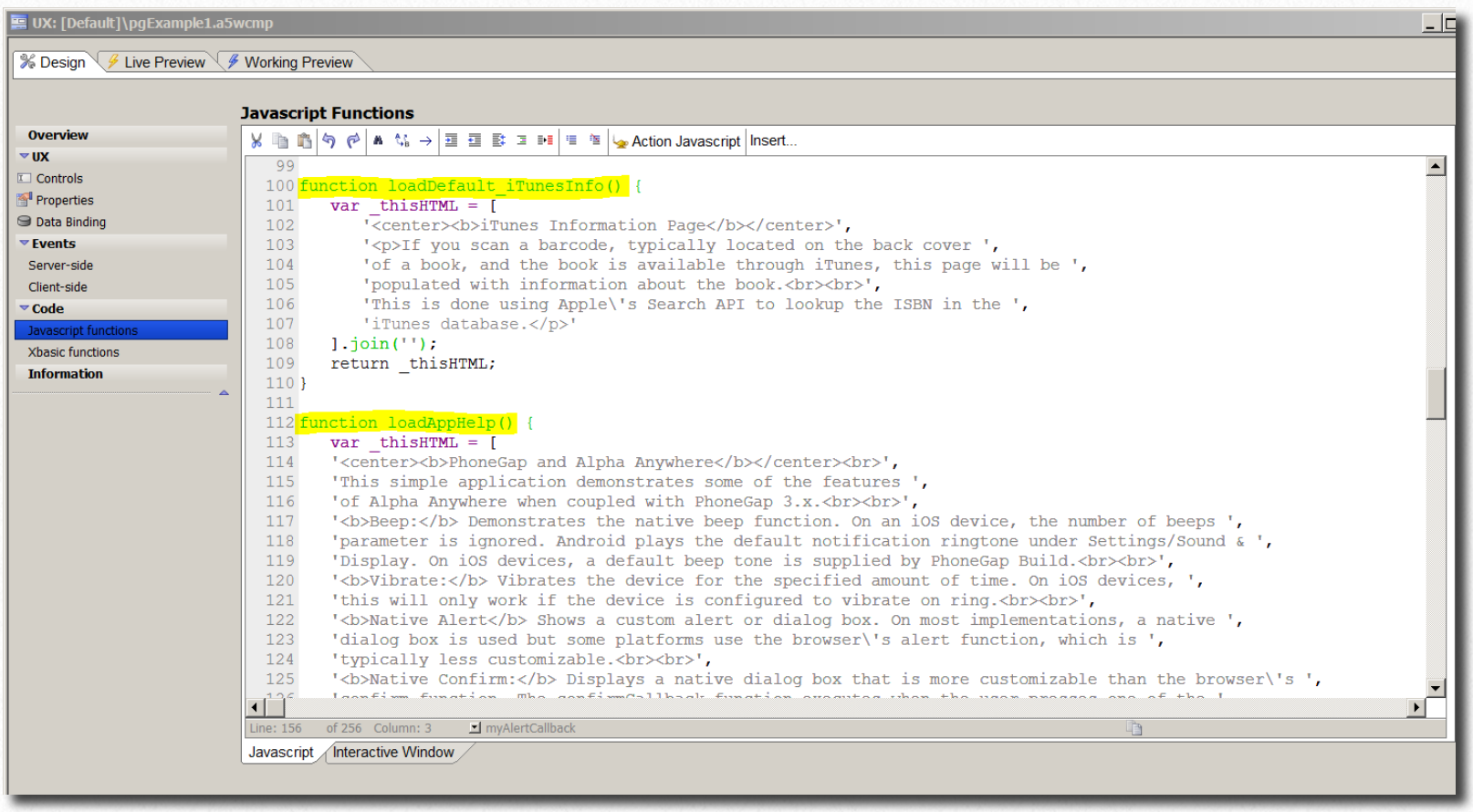
The Scan Fail Function

The scan fail function is another anonymous function that accepts an error parameter. If the scan fails for any reason, this function will display a PhoneGap native alert dialog, indicating the error.

The UX Component onInitializeComplete Event



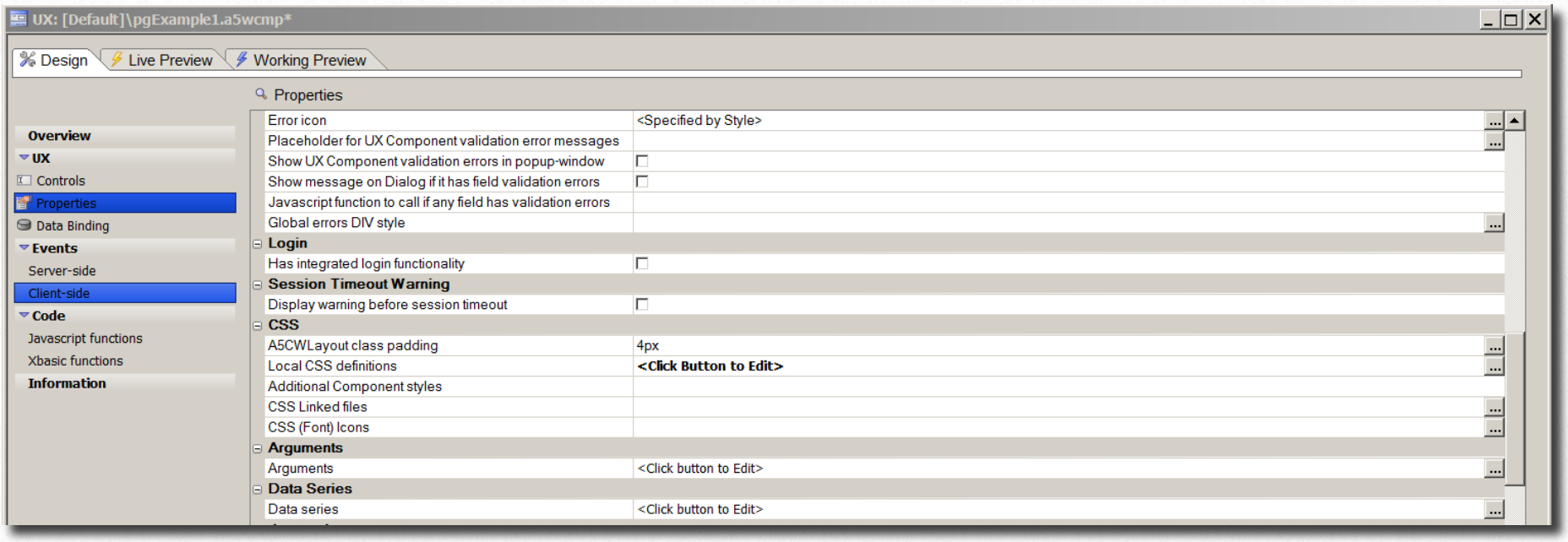
Looking at the Client Side onInitializeComplete event, the initial html is loaded into the iTunesInfo div from a call to the **loadDefault_iTunesInfo()** function. The same technique is used to load the appHelp div which is



defined in a free form HTML container on Panel Card 3 by calling the **loadAppHelp()** function.

This is a technique that allows you to define all of the HTML within the JavaScript functions. It provides a central location for all of the HTML, rather than defining the HTML within each container, which makes it harder to manage and edit the HTML used within your app. All you need to define within the HTML freeform container control is the div with an id that you will later populate with what may be dynamic content as in the case of the HTML we derive from the templates used within the barcode scan function.

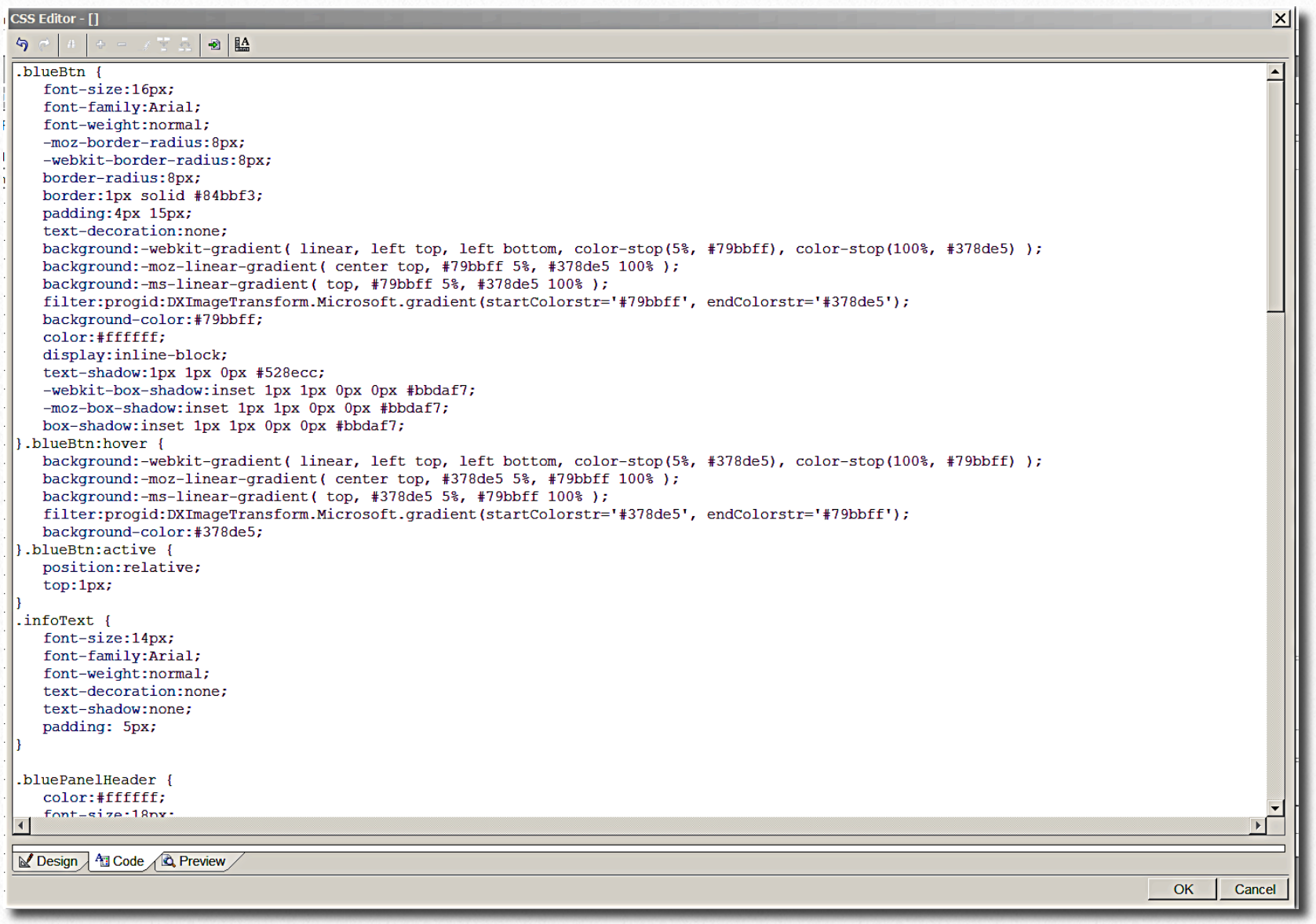
Local CSS Styles



This component's styling was derived from the standard iOS style, although it looks quite different.

A few local style CSS classes have been defined to change the look quite significantly.

From ***Properties***, scroll down to the CSS section and ***click on the button to edit the local CSS definitions***.



5 custom CSS classes have been defined. These classes are assigned to the specific elements in the CSS Class property of numerous controls within the UX component property grid.

These include:

> **.blueBtn** : defines the CSS style for all of the buttons used in the component
> **.infoText:** defines the CSS style for the information in PanelCard 2
> **.bluePanelHeader:** defines the CSS style for all of the Panel Card Headers
> **.bluePanelBody:** defines the CSS style for the Panel Card body in Panel Card 2 and Panel Card

The .blueBtn CSS was built using one of the many CSS button generators that are freely available on the web. Just Google “CSS Button Generator” and you will see many options.

The CSS Button Generator below is available at http://html-generator.weebly.com/css-button-generator.html

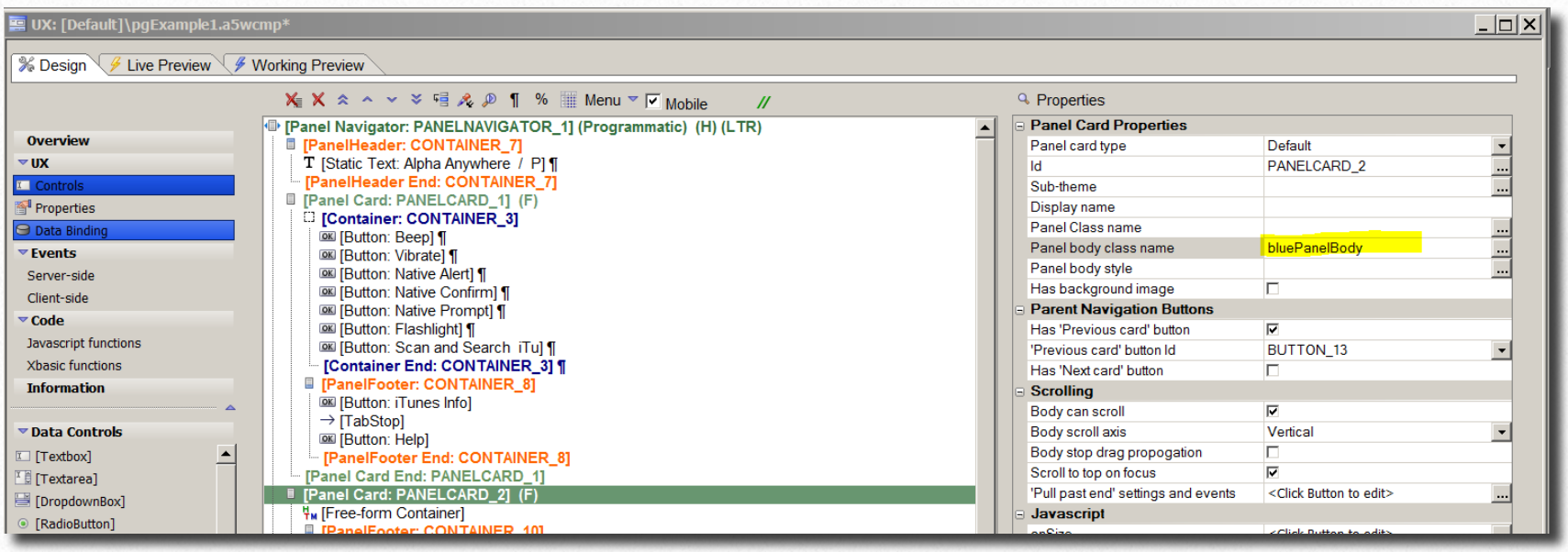
CSS Button Generator

Once you’ve defined the style, copy and paste the generated CSS into the CSS Editor Code Panel.

CSS Code: Class name: blueBtn Style tag:

```
.blueBtn {
	font-size:16px;
	font-family:Arial;
	font-weight:normal;
	-moz-border-radius:8px;
	-webkit-border-radius:8px;
	border-radius:8px;
	border:8px solid #3866a3;
	padding:30px 18px;
	text-decoration:none;
	background:-webkit-gradient( linear, left top, left bottom, color-stop(NaN%, #63b8ee),
color-stop(NaN%, #468ccf) );
	background:-moz-linear-gradient( center top, #63b8ee NaN%, #468ccf NaN% );
	background:-ms-linear-gradient( top, #63b8ee NaN%, #468ccf NaN% );
	filter:progid:DXImageTransform.Microsoft.gradient(startColorstr='#63b8ee',
endColorstr='#468ccf');
	background-color:#63b8ee;
	color:#ffffff;
	display:inline-block;
	text-shadow:1px 1px 0px #7cacde;
	-webkit-box-shadow:inset 1px 1px 0px 0px #bee2f9;
	-moz-box-shadow:inset 1px 1px 0px 0px #bee2f9;
	box-shadow:inset 1px 1px 0px 0px #bee2f9;
}.blueBtn:hover {
	background:-webkit-gradient( linear, left top, left bottom, color-stop(NaN%, #468ccf),
color-stop(NaN%, #63b8ee) );
	background:-moz-linear-gradient( center top, #468ccf NaN%, #63b8ee NaN% );
	background:-ms-linear-gradient( top, #468ccf NaN%, #63b8ee NaN% );
	filter:progid:DXImageTransform.Microsoft.gradient(startColorstr='#468ccf',
endColorstr='#63b8ee');
	background-color:#468ccf;
}.blueBtn:active {
	position:relative;
	top:1px;
}
/* This css button was generated by http://html-generator.weebly.com */
```

Sample generated CSS, copy and paste into the CSS Editor



bluePanelBody class assigned to PANELCARD_2

As you can see, customizing the look of the UX component is quite simple.

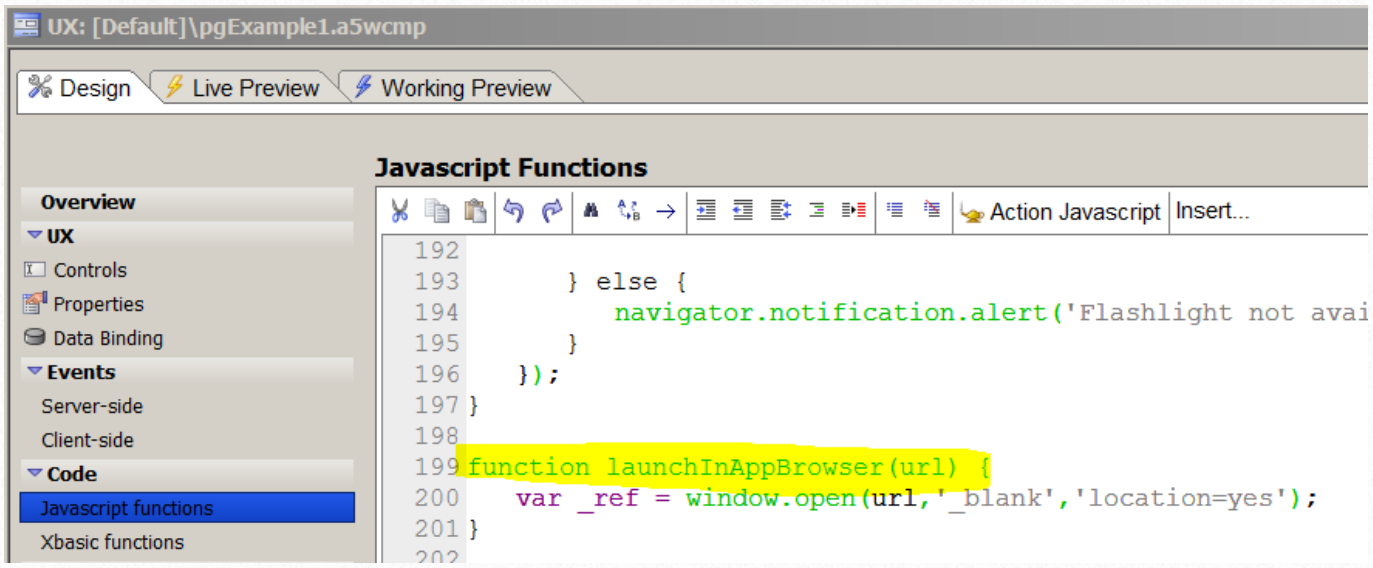
Launching the inAppBrowser

The **PhoneGap inAppBrowser plugin** allows an app to load an external web page within the app without having to exit the app and load a mobile browser. Think of it as a browser within your app. Typically an Action Sheet appears from the bottom of the screen that contains the browser.

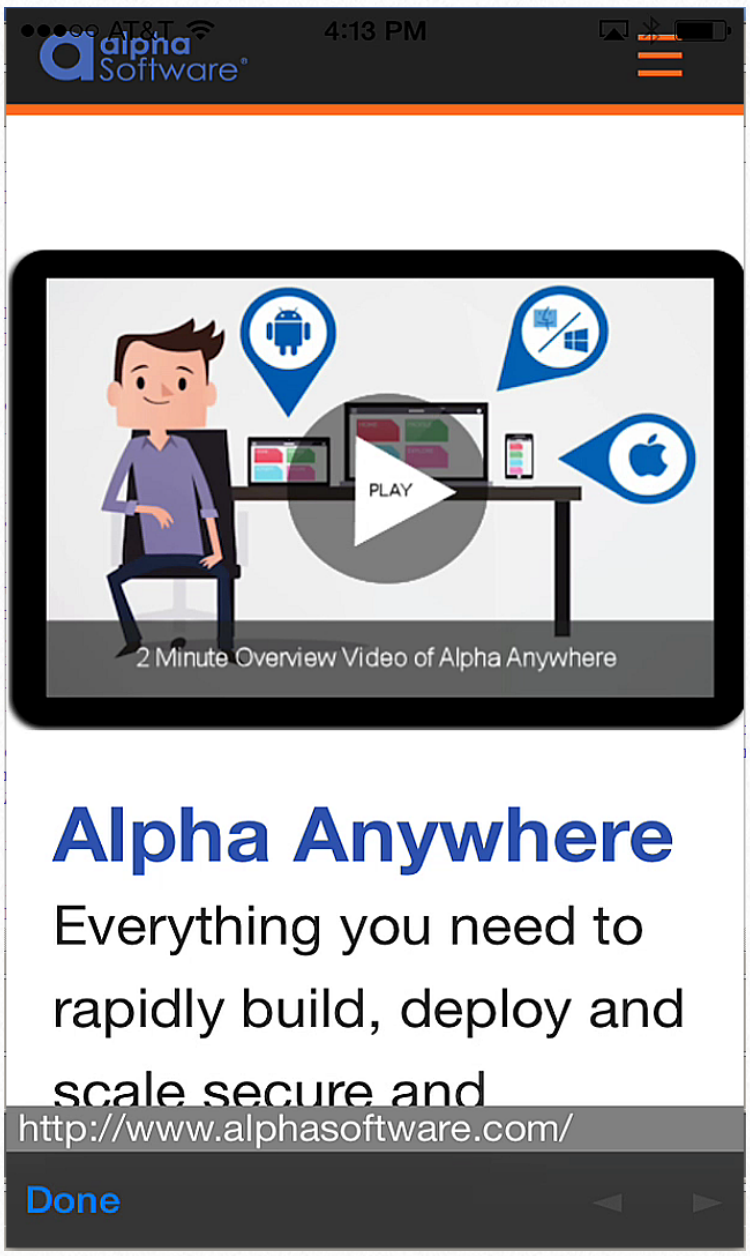
The help panel HTML includes a button with a link to http://alphasoftware.com. The ontouchend event is used on mobile devices to eliminate a 300ms delay that mobile browsers add to the onclick event.

```
130     'prompt function. A results object is passed to the callback function which executes when the ',
131     'user presses one of the buttons in the prompt dialog box. The results object includes ',
132     'buttonIndex, which is the one based button index of the button pressed and input1 which ',
133     'is the text entered into the prompt dialog box.<br><br>',
134     '<b>Flashlight:</b> Using a third party PhoneGap plugin, the device is queried for a camera light ',
135     'and if available, the light is turned on for 3 seconds.<br><br>',
136     '<b>Scan and Search iTunes:</b> This demonstrates a number of plugins: barCodeScanner, ',
137     'alert.notification, beep and inAppBrowser. Once a valid bar code is scanned a client side XHR2 ',
138     'request of a web service (in this case Apple\'s Search API), is made to look up information ',
139     'in the iTunes database from the scanned ISBN code. If the book is contained within iTunes, ',
140     'the information is displayed in a panel, along with a button to view the listing directly in iTunes. ',
141     'On an iOS device, this will call the native iTunes app if it is installed in the device.<br><br>',
142     '<b>inAppBrowser:</b> The button below links to Alpha\'s website and demonstrates ',
143     'the use of the inAppBrowser plugin.<br><br>',
144     '<center><button class="blueBtn" ontouchend="launchInAppBrowser(\'http://www.alphasoftware.com\');">',
145     'www.alphasoftware.com</button></center>'
146     ].join('');
147     return _thisHTML;
148 }
```

The ontouchend event calls the launchInAppBrowser function defined in JavaScript functions



launchInAppBrowser(url) function



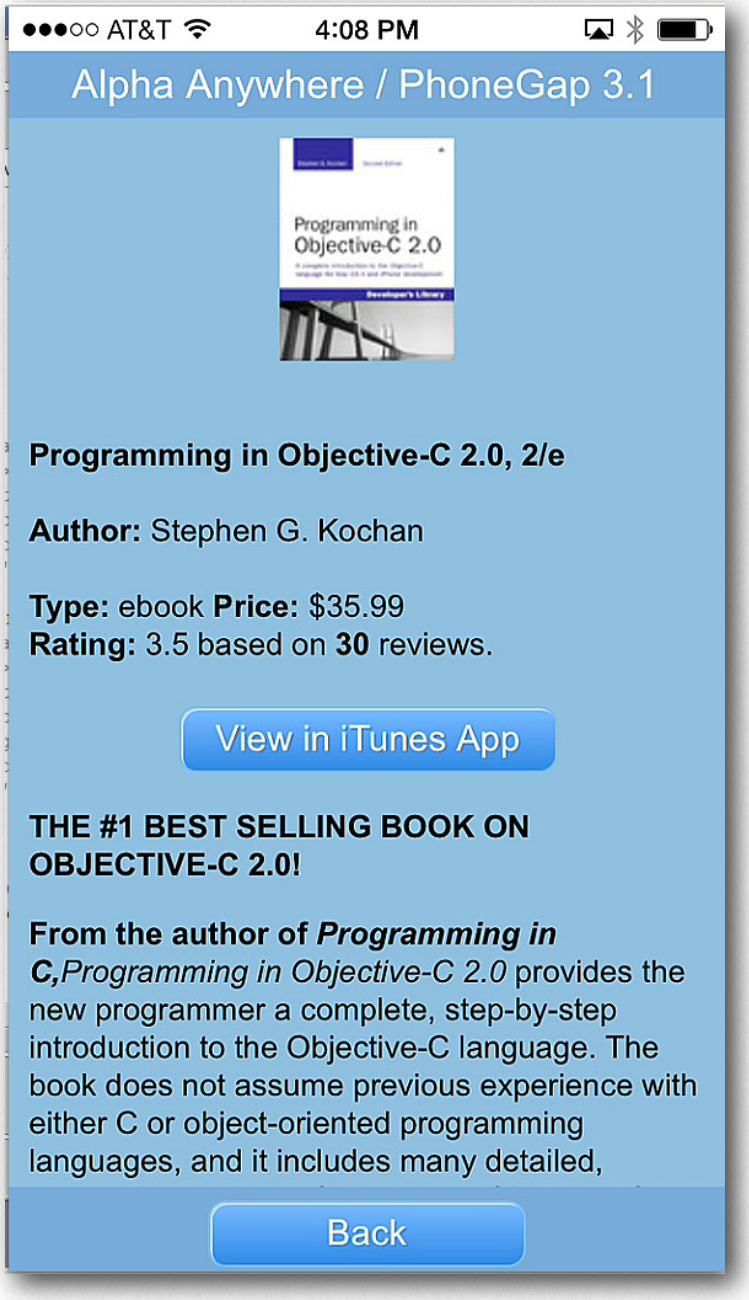
The inAppBrowser displaying www.alphasoftware.com within the app.
*The **Done button** in the lower left corner will close the inAppBrowser.*

The HTML templates that are used to present the data returned from the Apple Search API also contain a button that links to the Apple iTunes Info

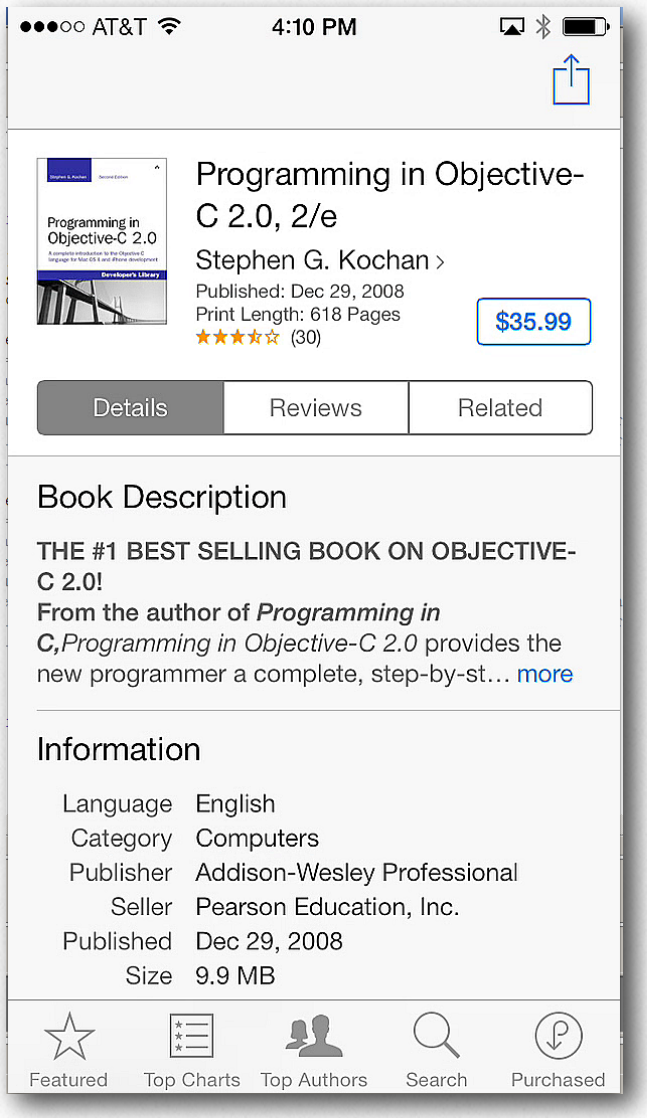
Javascript Functions

Action Javascript | Insert...

```
46
47 // App Globals
48 // are referenced through the {dialog.object}.appVars object
49 // This recommended technique doesn't pollute the global namespace
50
51 {dialog.object}.appVars={};
52 {dialog.object}.appVars.msg = '';
53 {dialog.object}.appVars.scannerResult = {};
54
55 {dialog.object}.appVars.template = [
56       '<center><img src = "{results[0].artworkUrl100}"></center><br><br>',
57       '<b>{results[0].trackName}</b><br><br>',
58       '<b>Author: </b>{results[0].artistName}<br><br>',
59       '<b>Type:</b> {results[0].kind} <b>Price:</b> {results[0].formattedPrice}<br><br>',
60       '<center><button class="blueBtn" ontouchend="launchInAppBrowser(\'{results[0].trackViewUrl}\');">{results[0]
61       '{results[0].description}'].join('');
62
63 {dialog.object}.appVars.templateWithRatings = [
64       '<center><img src = "{results[0].artworkUrl100}"></center><br><br>',
65       '<b>{results[0].trackName}</b><br><br>',
66       '<b>Author: </b>{results[0].artistName}<br><br>',
67       '<b>Type:</b> {results[0].kind} <b>Price:</b> {results[0].formattedPrice}<br>',
68       '<b>Rating:</b> {results[0].averageUserRating} based on <b>{results[0].userRatingCount}</b> reviews.<br><br>
69       '<center><button class="blueBtn" ontouchend="launchInAppBrowser(\'{results[0].trackViewUrl}\');">{results[0]
70       '{results[0].description}'].join('');
71
```



While the button is configured to launch the inAppBrowser, the iTunes App on iOS devices has registered a **Custom URL Scheme** which launches the iTunes App (if it is not already running) and displays the link in iTunes, instead of launching the inAppBrowser.



On Android devices, the inAppBrowser is used. As such, the label on the button needs to be dynamically assigned. This is handled in the **iTunesCallback** function.

```
function iTunesCallback(data) {

    if (data.resultCount == 0) {
        // not in iTunes
        navigator.notification.beep(1);
        navigator.notification.alert('Format: '+{dialog.object}.appVars.scannerResult.form
    } else {
        var _html;
        var _btnLabel = is_iOS()?'View in iTunes App':'View on iTunes Site';
        data.results[0].btnLabel = _btnLabel;
        if (typeof data.results[0].userRatingCount == 'number') {
            _html = A5.u.template.expand(data,{dialog.object}.appVars.templateWithRatings);
        } else {
            _html = A5.u.template.expand(data,{dialog.object}.appVars.template);
        }
        $('iTunesInfo').innerHTML = _html;
        {dialog.object}.panelNavigate('PANELNAVIGATOR_1','Next');
    }
}
```

If the app is running on an iOS device, the button label is assigned “View in iTunes App”.
If the App is running on an Android device, the button label is assigned “View on iTunes Site”.

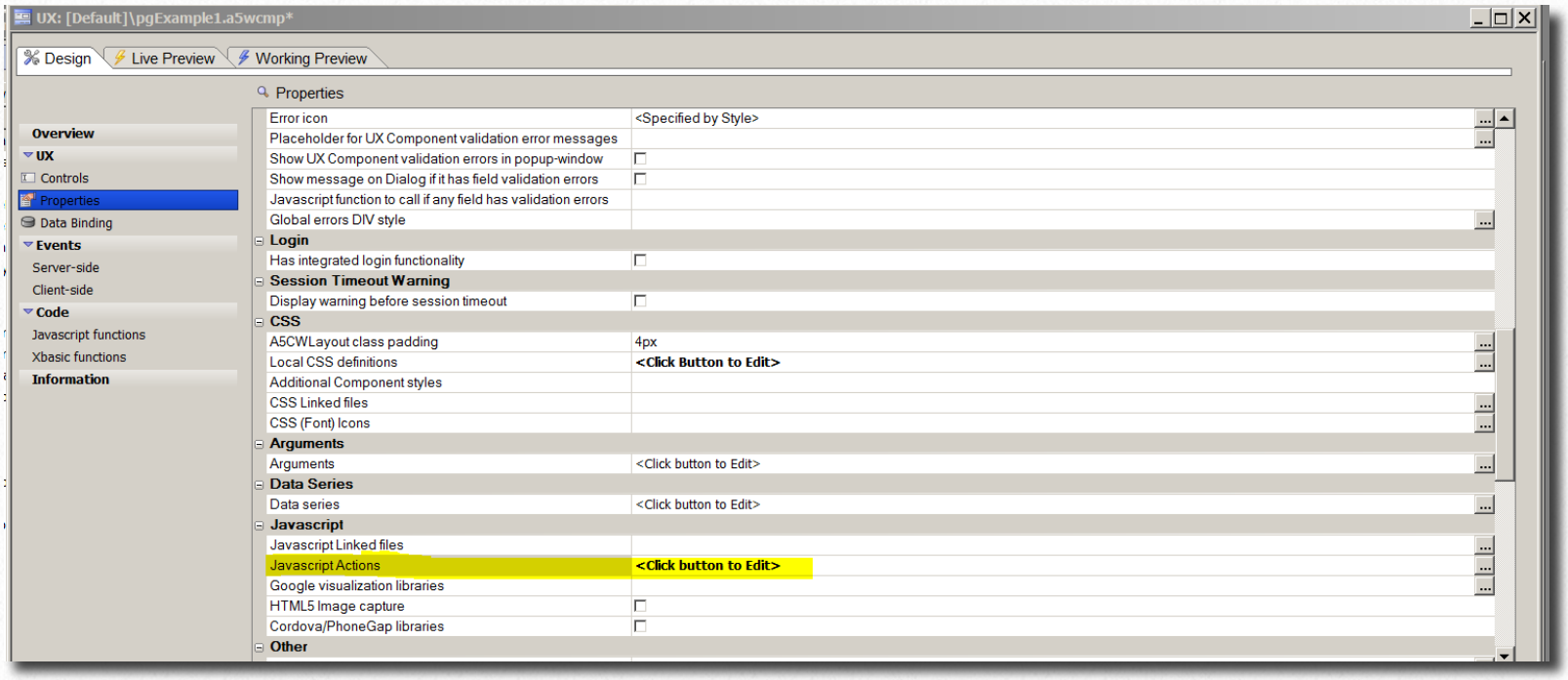
Notice the addition of a new variable, .btnLabel to the data.results[0] object. You will see this variable in use within the templates that process the JSON results from the Apple Search API.

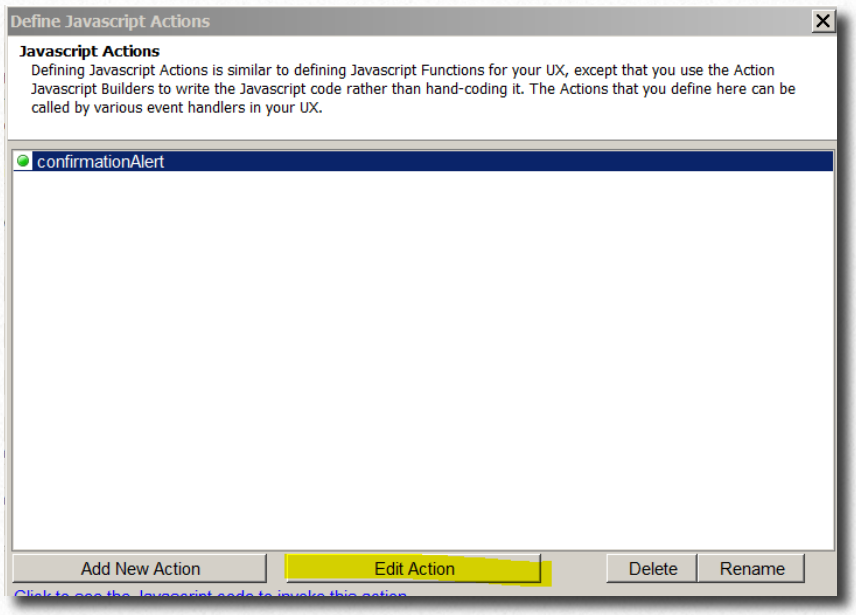
It is this level of detail that helps your app stand out.

For more info on iOS Custom URL Schemes see:
https://developer.apple.com/library/ios/documentation/iPhone/Conceptual/iPhoneOSProgrammingGuide/AdvancedAppTricks/AdvancedAppTricks.html#//apple_ref/doc/uid/TP40007072-CH7-SW50

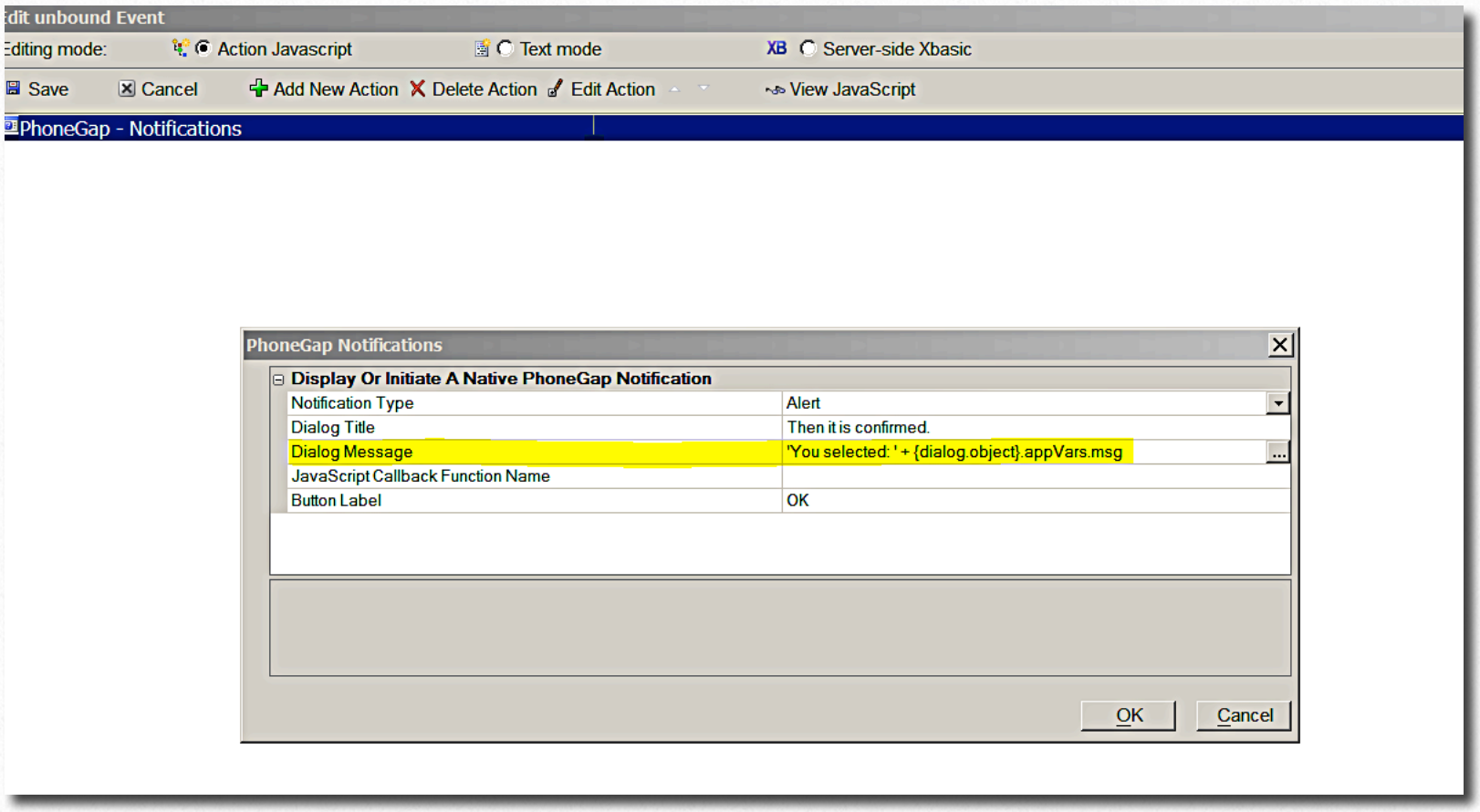
Programmatically Calling JavaScript Actions and Using a Function to Generate a Dynamic Message

The component contains one JavaScript Action that is called from the **myConfirmCallback** function. This callback is fired when the ‘Yes’ or ‘No’ button is selected from the **native.notification.confirm** function which is tied to the Native Confirm button.



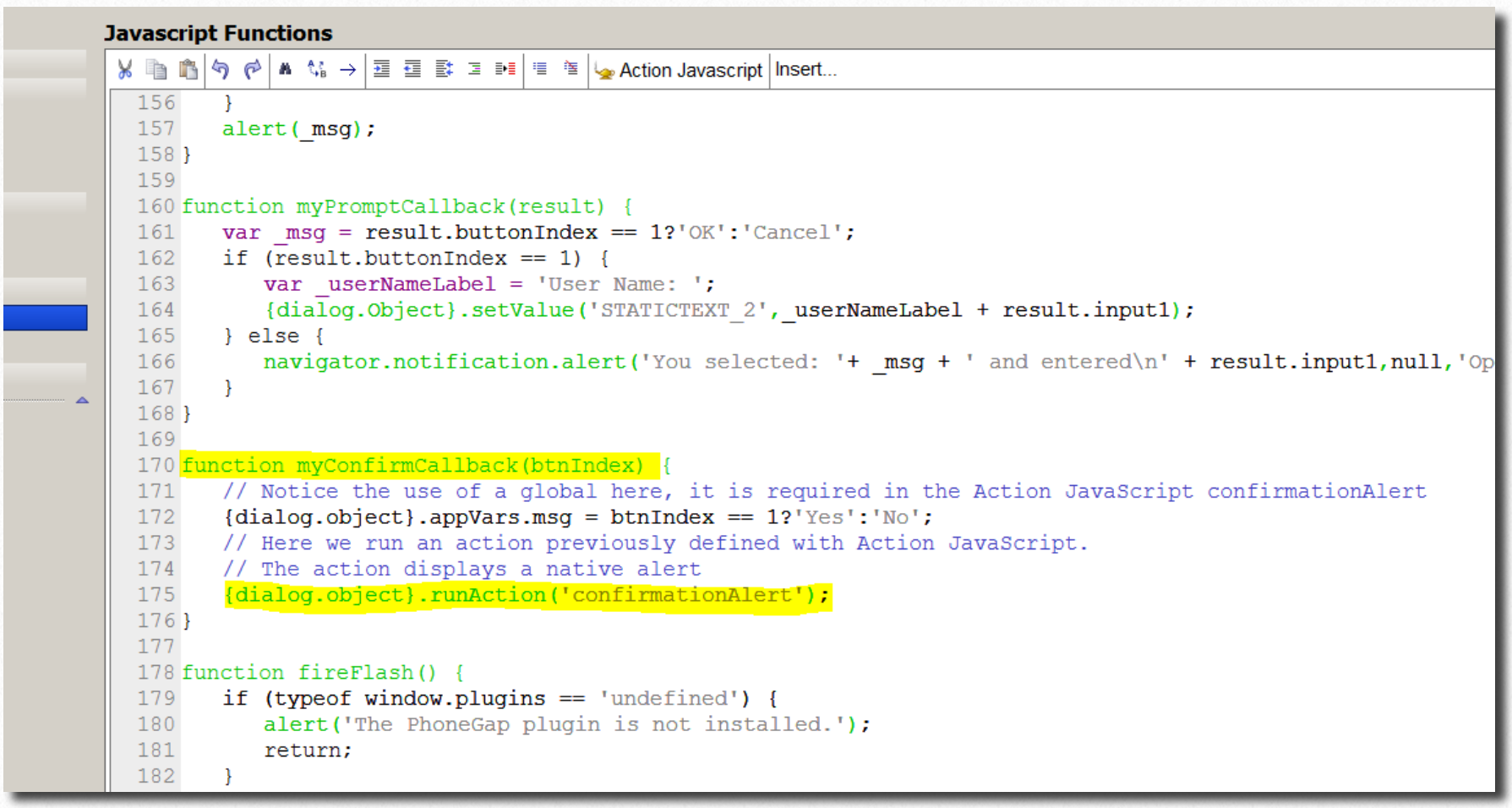


Open JavaScript Actions and ***edit*** the confirmationAlert action.



The confirmationAlert Action calls the Action JavaScript PhoneGap Notifications and displays a native alert. The Dialog Message is populated from an instance variable.

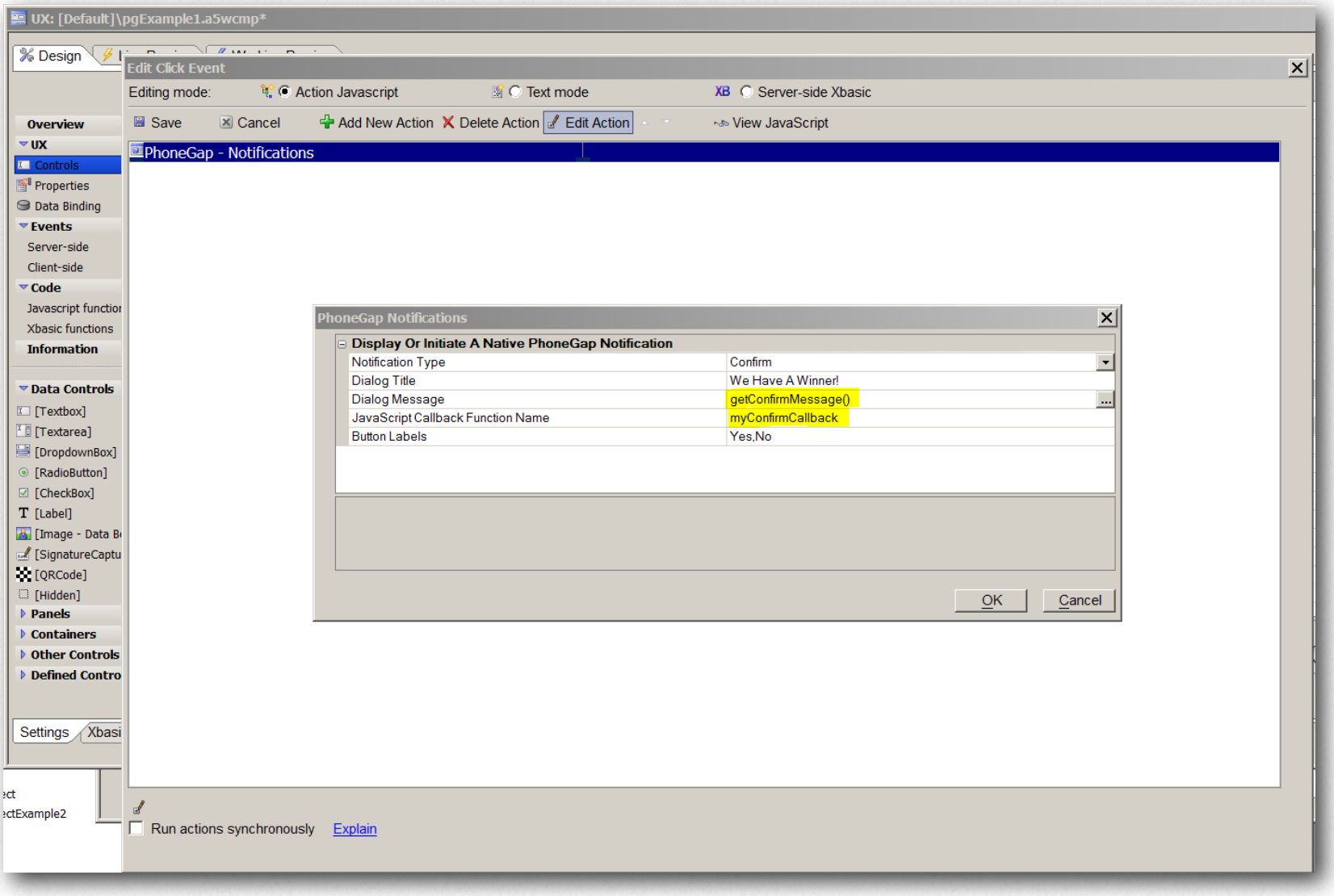
Since this JavaScript Action is going to be called programmatically from a callback function, the message to display needs to be stored in an instance variable. This should be clear from the code below which is tied to the confirmationCallback. Notice how **{dialog.object}.appVars.msg** is set to either Yes or No, depending upon which button the user clicks in response to the PhoneGap navigator.notification.confirm dialog. Also notice the use of the **{dialog.object}.runAction** method to run the confirmationAlert JavaScript action.



Let's take a look at the **Native Confirm button abstract onclick event**.

This Action JavaScript uses a number of advanced techniques that are worth understanding.

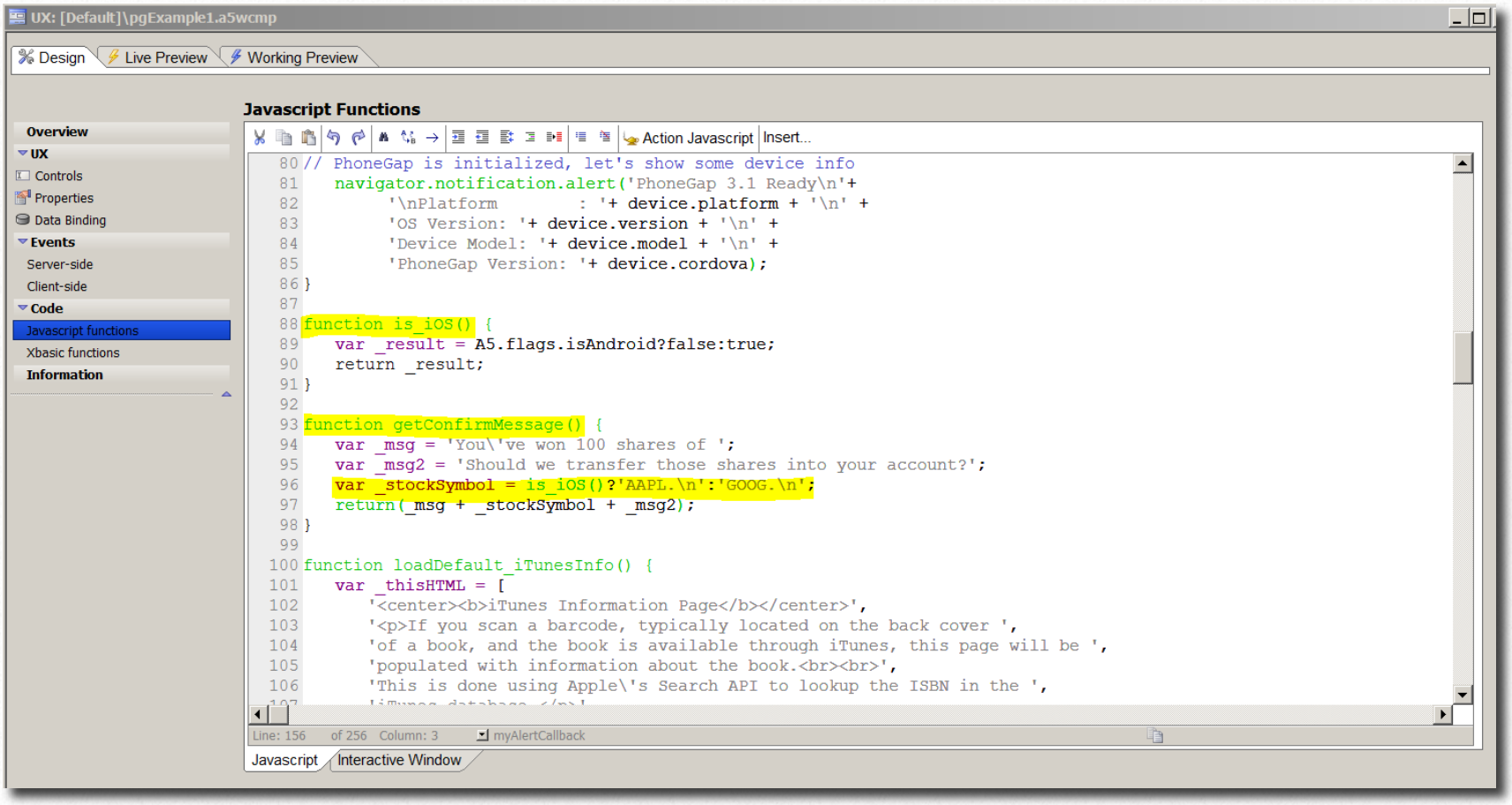
The JavaScript callback function name is myConfirmCallback. That is the function we just looked at. It is passed a btnIndex which indicates the button the user clicked. Since the myConfirmCallback **calls another** JavaScript Action (**confirmationAlert**), the result of the button selected is stored in an instance variable allowing reference to it from the confirmationAlert JavaScript.



Using a JavaScript function to return a dynamic message

The Dialog Message is populated from a JavaScript function called **getConfirmMessage()**. This allows the message to be tailored to the mobile platform that the app is running on.

As you can see in the **getConfirmMessage()** function, the message content will vary for iOS and Android devices.



The getConfirmMessage function returns a dynamic message, based on the mobile platform at runtime

The PhoneGap deviceready event

In Javascript functions you will see an event listener added for the **deviceready** event. This event is fired after PhoneGap initialization is complete. It can be helpful to use this event to enable PhoneGap related controls

```
/* Event Handlers */

// onDeviceReady: this fires after PhoneGap has been fully initialized.
document.addEventListener('deviceready',onDeviceReady,false);

/* Support Functions */

function onDeviceReady() {
// PhoneGap is initialized, let's show some device info
   navigator.notification.alert('PhoneGap 3.1 Ready\n'+
         '\nPlatform        : '+ device.platform + '\n' +
         'OS Version: '+ device.version + '\n' +
         'Device Model: '+ device.model + '\n' +
         'PhoneGap Version: '+ device.cordova);
}
```

within your app. In this case, the **onDeviceReady** function uses a PhoneGap native.notification.alert to display device related information in a native dialog.

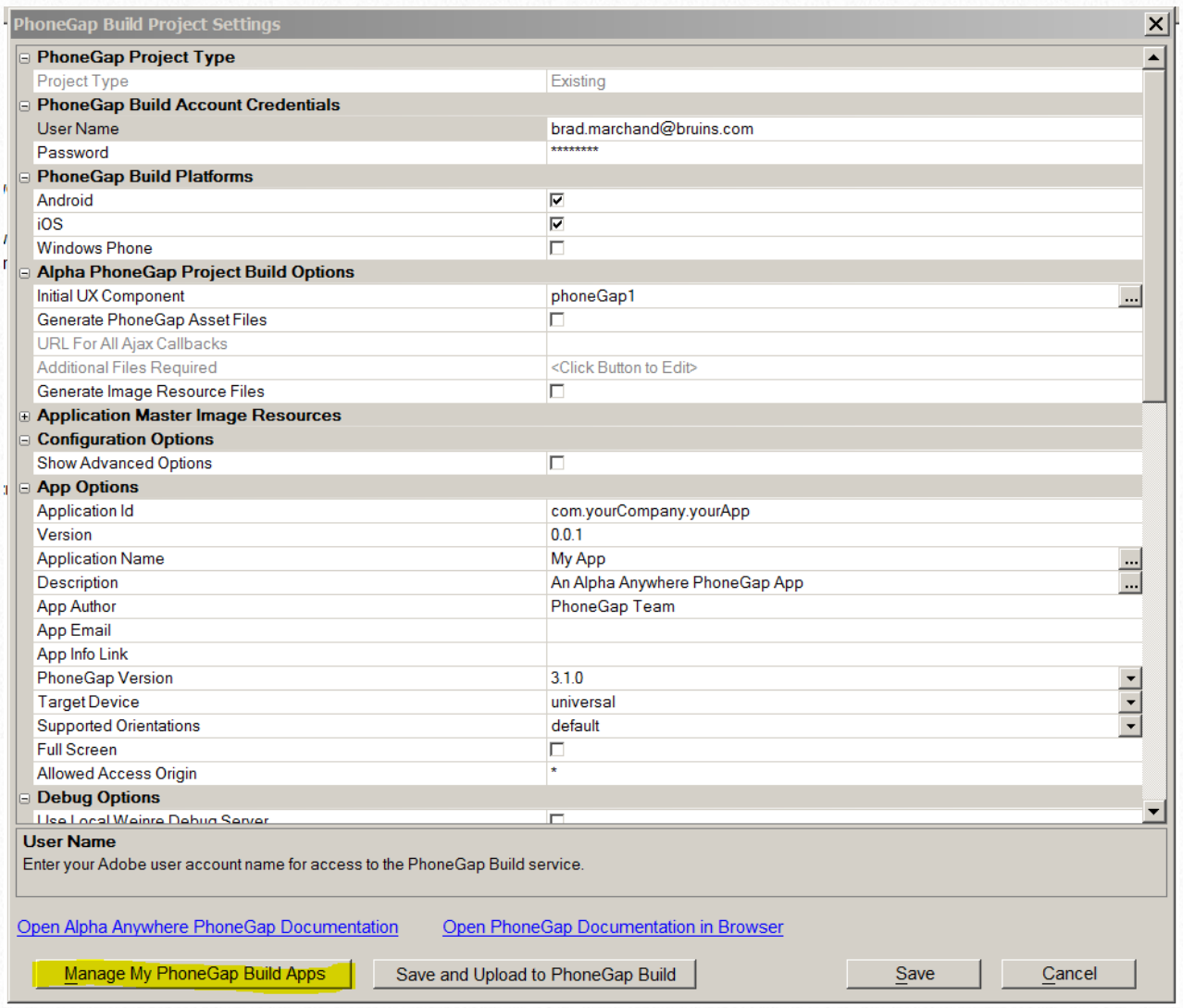
Using the Integrated PhoneGap App Builder To Build and Submit The App To PhoneGap Build

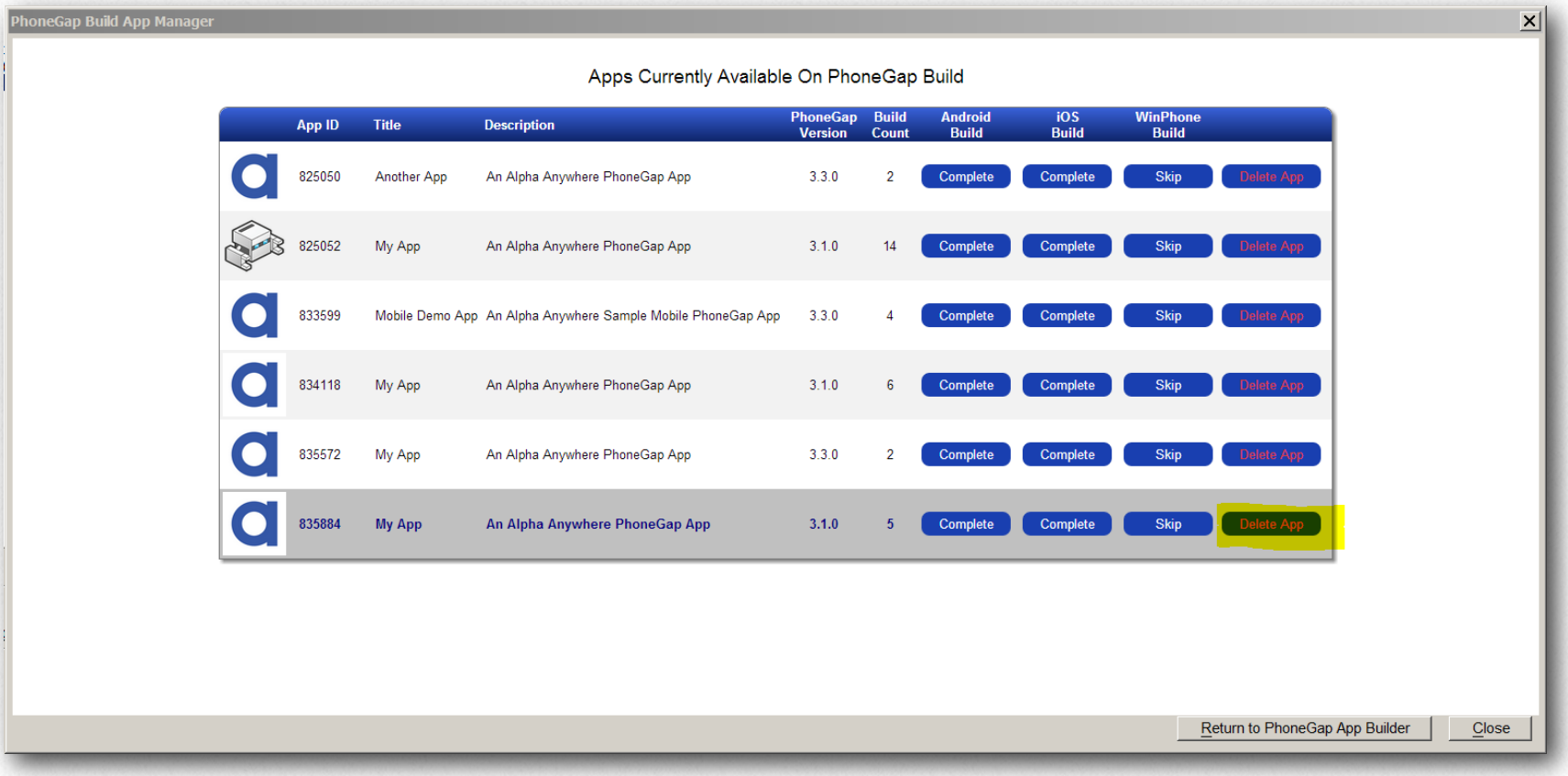
Delete The Previous Project From PhoneGap Build

If you have built the previous simple app in PhoneGap Build and have a free PhoneGap Build account, you may need to delete the existing app from PhoneGap Build before you can submit another app to PhoneGap Build. The free PhoneGap Build account only accommodates one (1) private app. If you would like to keep more than one (1) project active on PhoneGap Build you will need to setup a paid account.

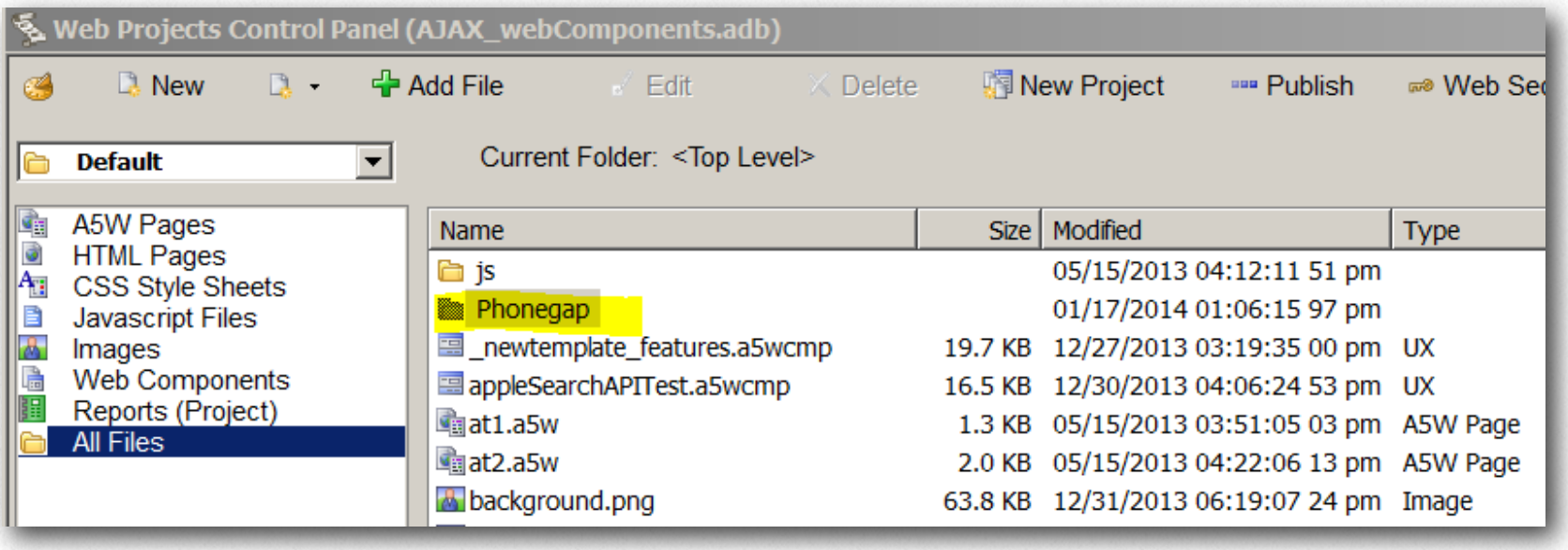
If you do not plan on changing the App ID or the App Name for your new app, (which may be likely in a test environment) you can overwrite the files by basing the build (done in the integrated Alpha Anywhere PhoneGap Builder) on the new component. In this case, you do not have to delete the existing PhoneGap Build App.

To delete the previous project on PhoneGap Build, bring up the PhoneGap App Manager by clicking the PhoneGap button in the Web Projects Directory and next ***click Manage My PhoneGap Build Apps.***





Click the Delete App button to delete the app from PhoneGap Build. ***Warning ... this is destructive*** and while you may upload the app again, which will create a new app with a new app id on PhoneGap Build, the PhoneGap app referenced by this app id is permanently deleted.



Web Projects Folder contains a previously defined PhoneGap Project

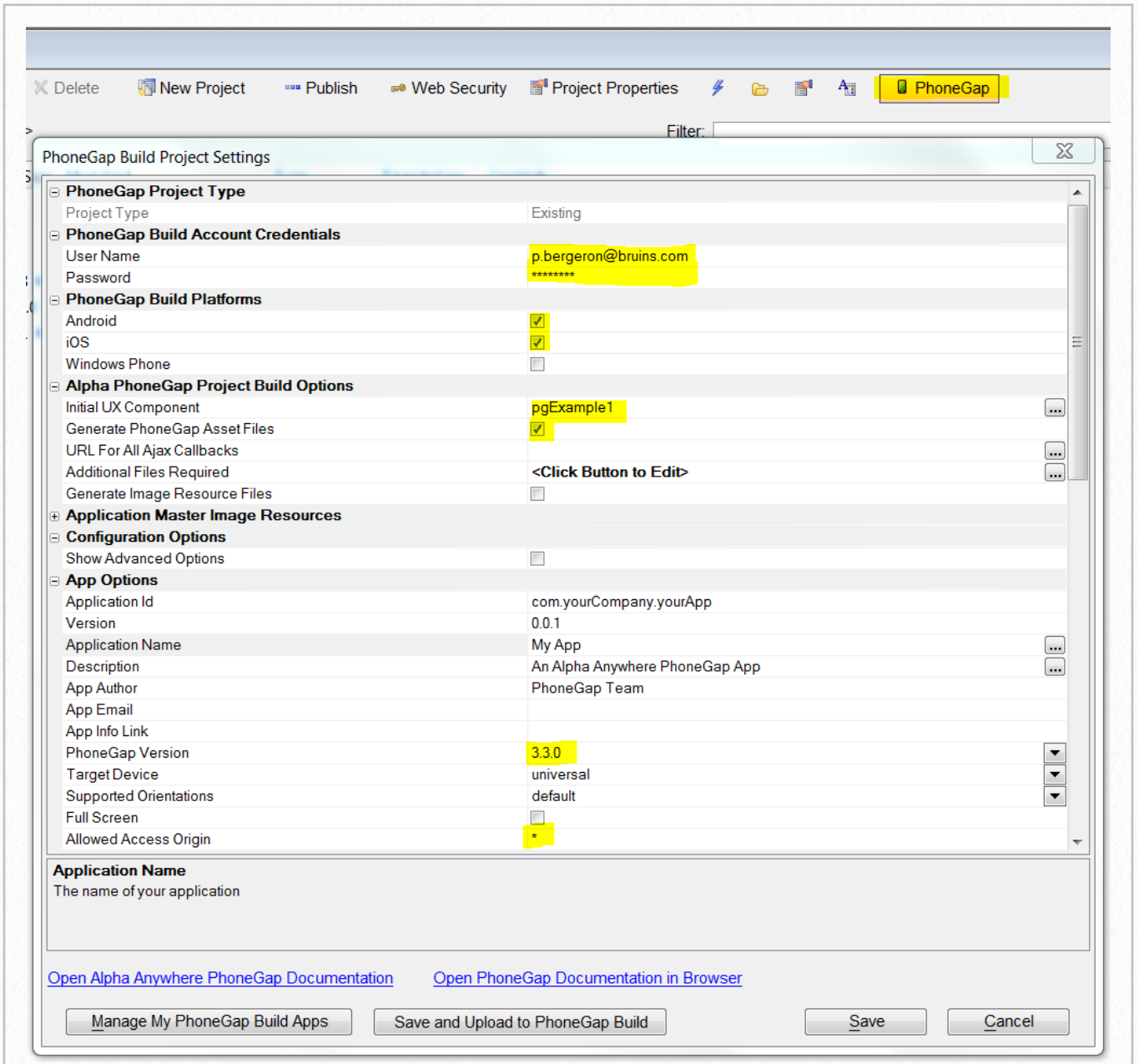
Delete An Existing Project In The Web Projects Directory

If the existing Web Projects folder contains a PhoneGap folder, you may want to delete it in order to build a new PhoneGap project however, this is not necessary.

You only need to delete the project if you want to delete all of the files referenced within the folder. If not, you can overwrite the files in the previous project. You can always select a new component to base the PhoneGap Project build on and the integrated PhoneGap App Builder will overwrite the previous components assets and files.

if you want to delete the previous PhoneGap project, right click on the folder and delete it.

Building The PhoneGap Project



Click the PhoneGap Button *to bring up the PhoneGap App Builder*

The project type may be new or existing. This depends on the existence of the PhoneGap folder in the Web Projects directory.

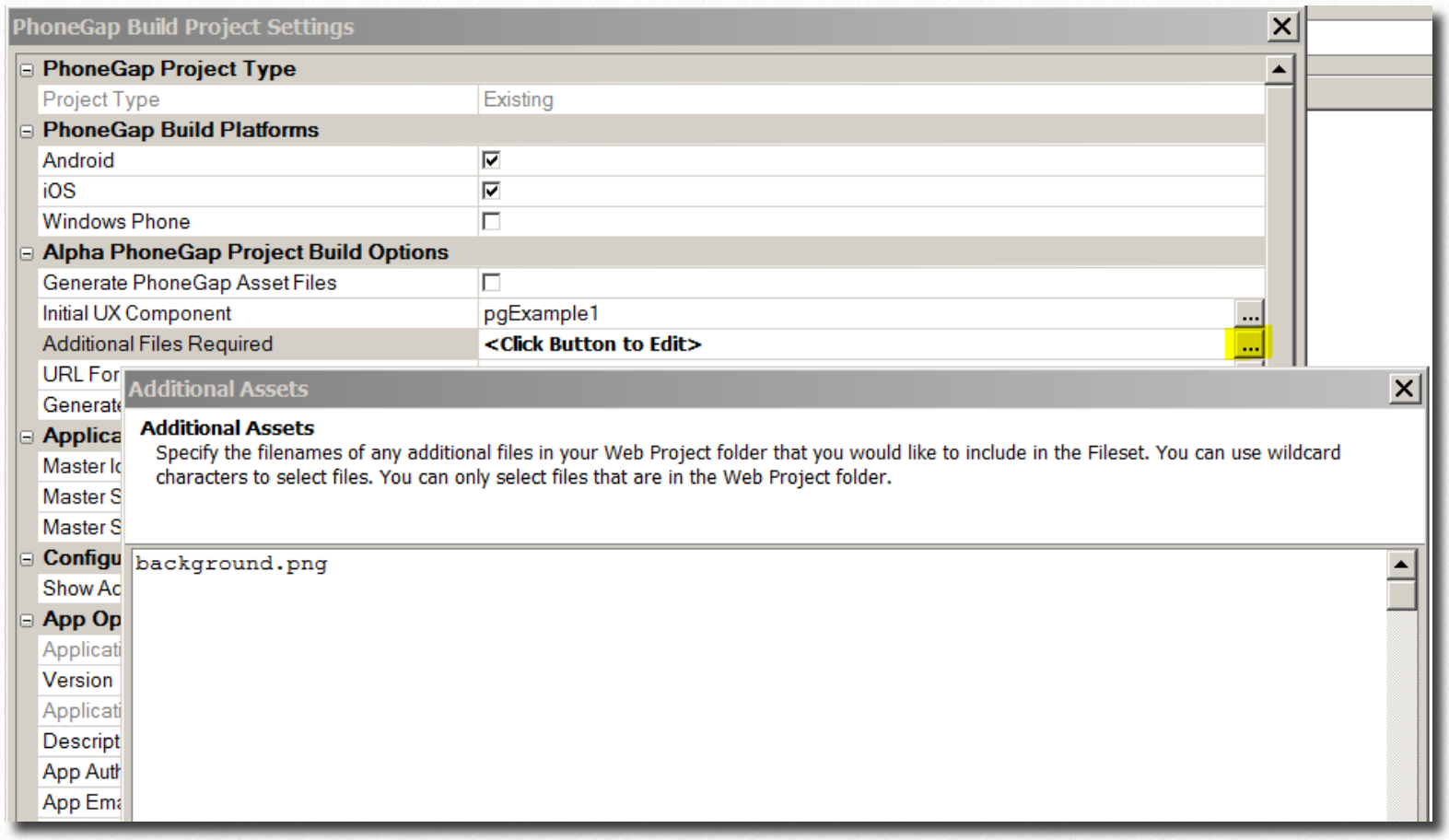
Enter your PhoneGap Build credentials, if they are not present. The credentials are encrypted and stored within the PhoneGap folder, so that you do not have to enter them each time you bring up the builder.

Select the platforms that you would like to build for. In the image above, Android and iOS are selected.

Select the pgExample1 component and make sure the Generate PhoneGap Asset files option is checked.

Select the PhoneGap Version that you would like to use. In the image above I've selected PhoneGap 3.3.0.

Make sure the Allowed Access Origin is set to *. This allows us to communicate with any URL on the web. This this app requires access to Apple's site and to Alpha's site.



This project requires an additional background.png file

Click on the Additional Files Required button and select the background.png file. Anytime you need to include additional image assets, JavaScript files, etc. this is where you will specify the files to include in your project.

Select the PhoneGap plugins required for this project.

PhoneGap Plugins Required

Core Plugins

- device
- inAppBrowser
- notification
- vibration

Third Party Plugins

- barcodescanner
- flashlight

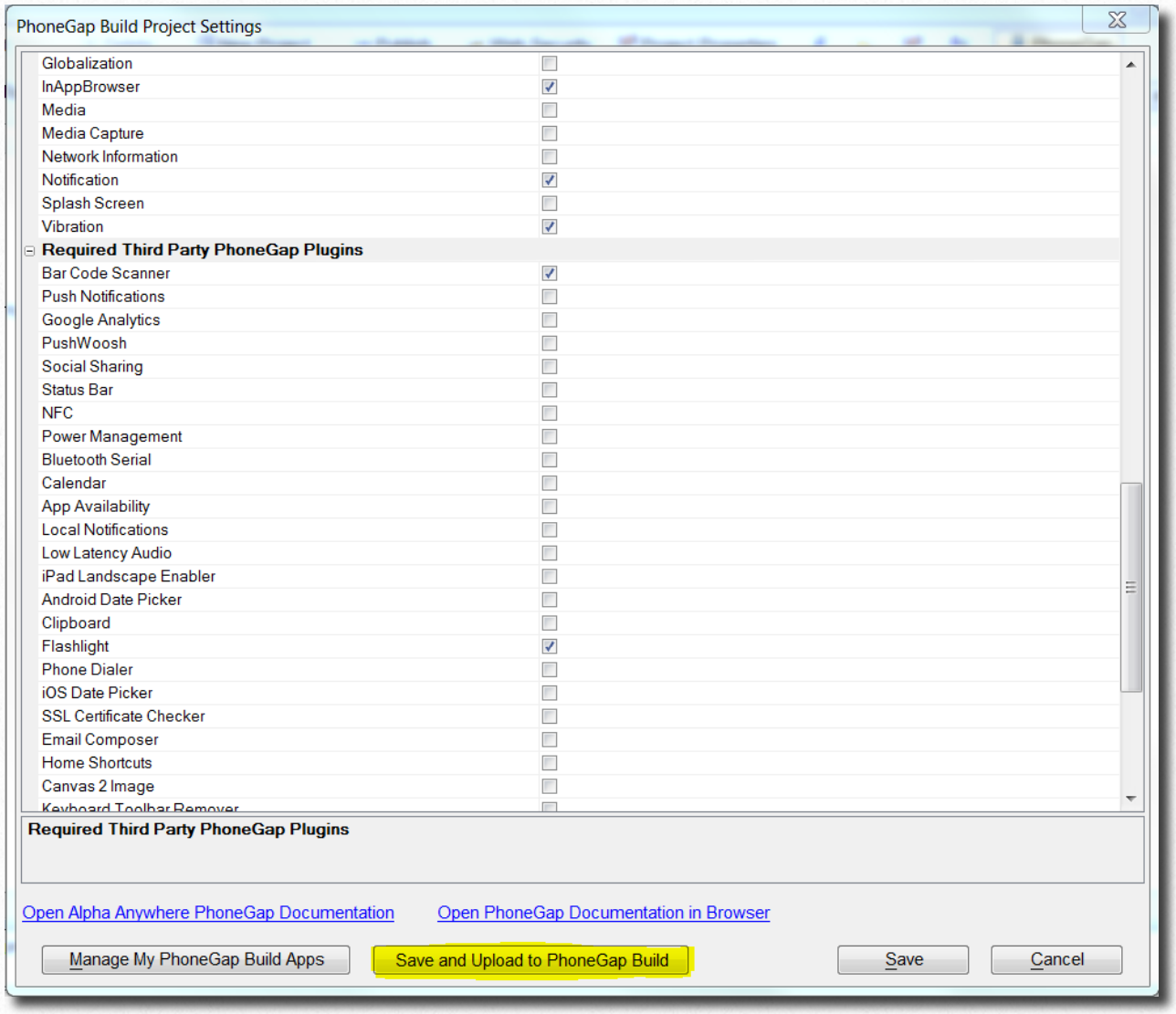
Required PhoneGap Core Plugins	
Battery	☐
Camera	☐
Console	☐
Contacts	☐
Device	☑
Device Motion	☐
Device Orientation	☐
File	☐
File Transfer	☐
Geolocation	☐
Globalization	☐
Hydration	☐
InAppBrowser	☑
Media	☐
Media Capture	☐
Network Information	☐
Notification	☑
Splash Screen	☐
Vibration	☑
Required Third Party PhoneGap Plugins	

Core PhoneGap Plugins Required

Required Third Party PhoneGap Plugins	
Bar Code Scanner	☑
Push Notifications	☐
Google Analytics	☐
PushWoosh	☐
Social Sharing	☐
Status Bar	☐
Power Management	☐
Bluetooth Serial	☐
Calendar	☐
App Availability	☐
Local Notifications	☐
Low Latency Audio	☐
iPad Landscape Enabler	☐
Android Date Picker	☐
Clipboard	☐
Flashlight	☑
Phone Dialer	☐
iOS Date Picker	☐
SSL Certificate Checker	☐
Required Third Party PhoneGap Plugins	

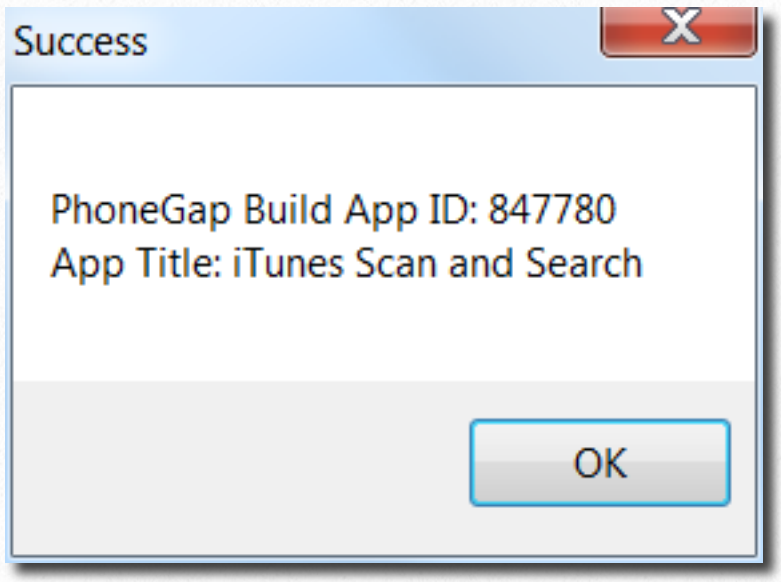
PhoneGap Third Party Plugins Required

Submit the Project To PhoneGap Build

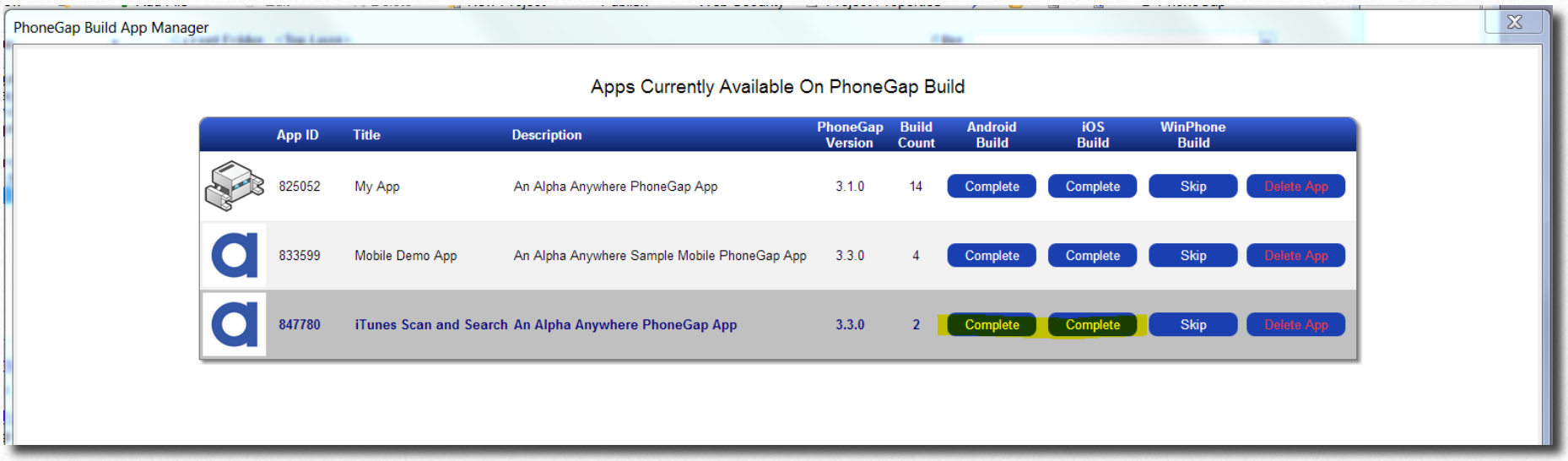


Select the Save and Upload to PhoneGap Build Button.

This will start the upload process. First the login credentials are checked to verify that a valid PhoneGap Build account exists. Next the app is packaged up for upload and finally the app is uploaded to PhoneGap Build. If it is a new app, then a new app is created on PhoneGap Build. If the app already exists on PhoneGap Build, then it is updated.



Once the app is published, the PhoneGap App Manager is displayed which shows you the status of the pending builds.



Once the build buttons indicate the build is completed, you can select the button and a QRCode will be displayed, allowing you to scan in the code and install the app on your phone or tablet.

Summary

In this chapter:

- We looked at the construction of a UX component that utilizes both core and third party PhoneGap plugins.
- We discussed a number of advanced techniques and best practices for creating a PhoneGap App with a UX Component.
- We built a PhoneGap App using the Alpha Anywhere integrated PhoneGap App Builder.
- We submitted the app to the PhoneGap Build Service.
- We used the integrated PhoneGap App Manager to install the app on an iOS or Android device.

6

Debugging

Debugging With PhoneGap Build

PhoneGap Build includes the ability to wirelessly debug your apps by using weinre (**We**b **In**spector **Re**mote, typically pronounced *“winery”*). Weinre is a debugger for web pages, similar to FireBug or Web Inspector except it is designed to work with remote devices, allowing you to debug web apps on a mobile device without tethering the device to the development machine and running the native debugger.

The complete documentation for weinre is available at:
http://people.apache.org/~pmuellr/weinre/docs/latest/Home.html

Debugging with weinre requires

- **A debug server:** The HTTP server running the weinre service, may be local or remote. Acts as a messaging switchboard between the debug target and the debug client.
- **A debug client:** The browser that runs the debugger (Web Inspector) user interface.
- **A debug target:** The web page to debug. In this case, the WebView that is running your app.

The weinre service is run on the debug server. This can be hosted through PhoneGap Build (remote debugging) or through a local weinre install on you own development machine (local debugging). A weinre server installation is quite simple, using NodeJS and NPM to install.

Weinre features include

- **DOM / CSS inspector:** inspect, edit, delete DOM elements and CSS rules
- **localStorage / WebSQL inspector**
- **Event Timeline**

- **Console:** Log debug messages, run arbitrary JavaScript code

Weinre does not support

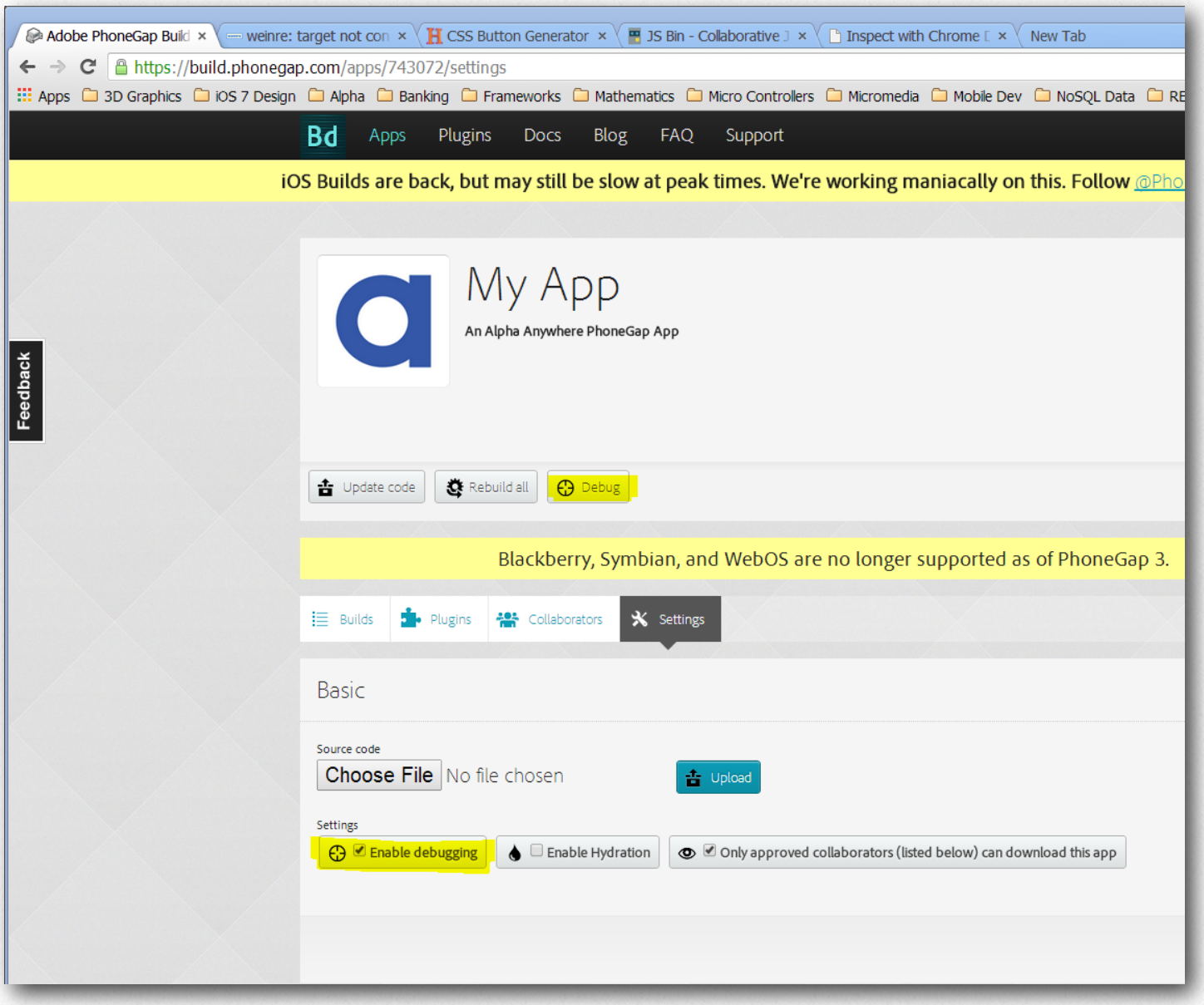
- **JavaScript debugging:** no breakpoints, etc.
- **Network diagnostics:** is limited and because of the debug server, normally useless
- **Resource diagnostics**
- **Profiling**

PhoneGap Build Debug Support

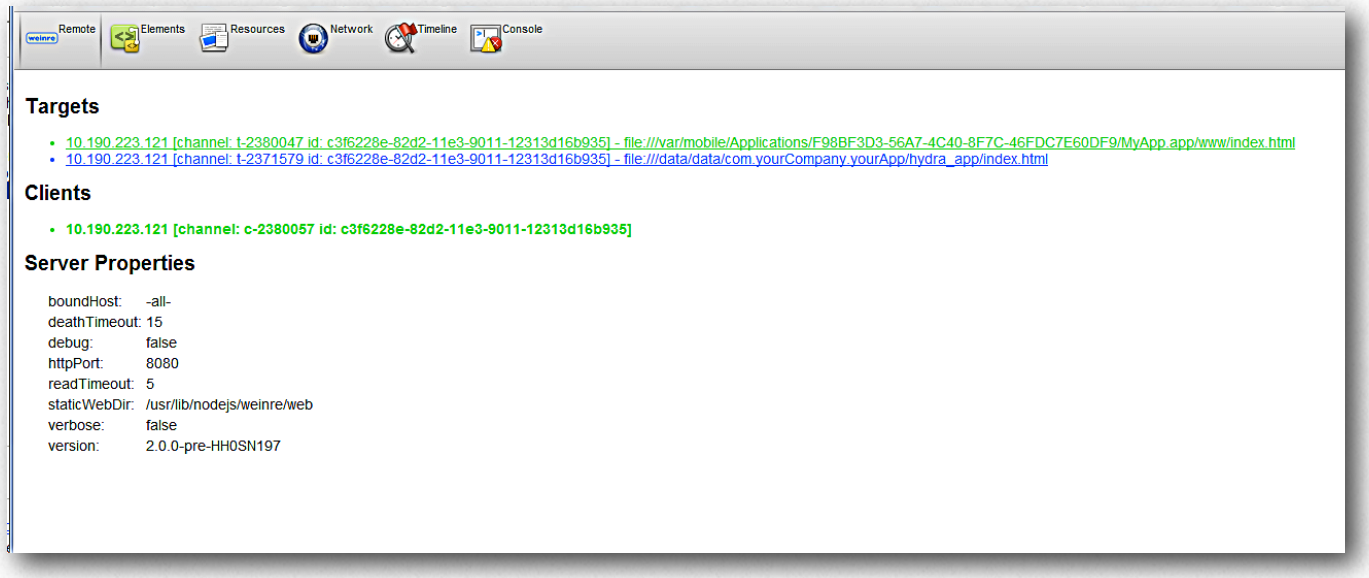
Using A Remotely Hosted Weinre Server Via PhoneGap Build

This is the simplest way to debug your app.

From App Settings, click Enable debugging.

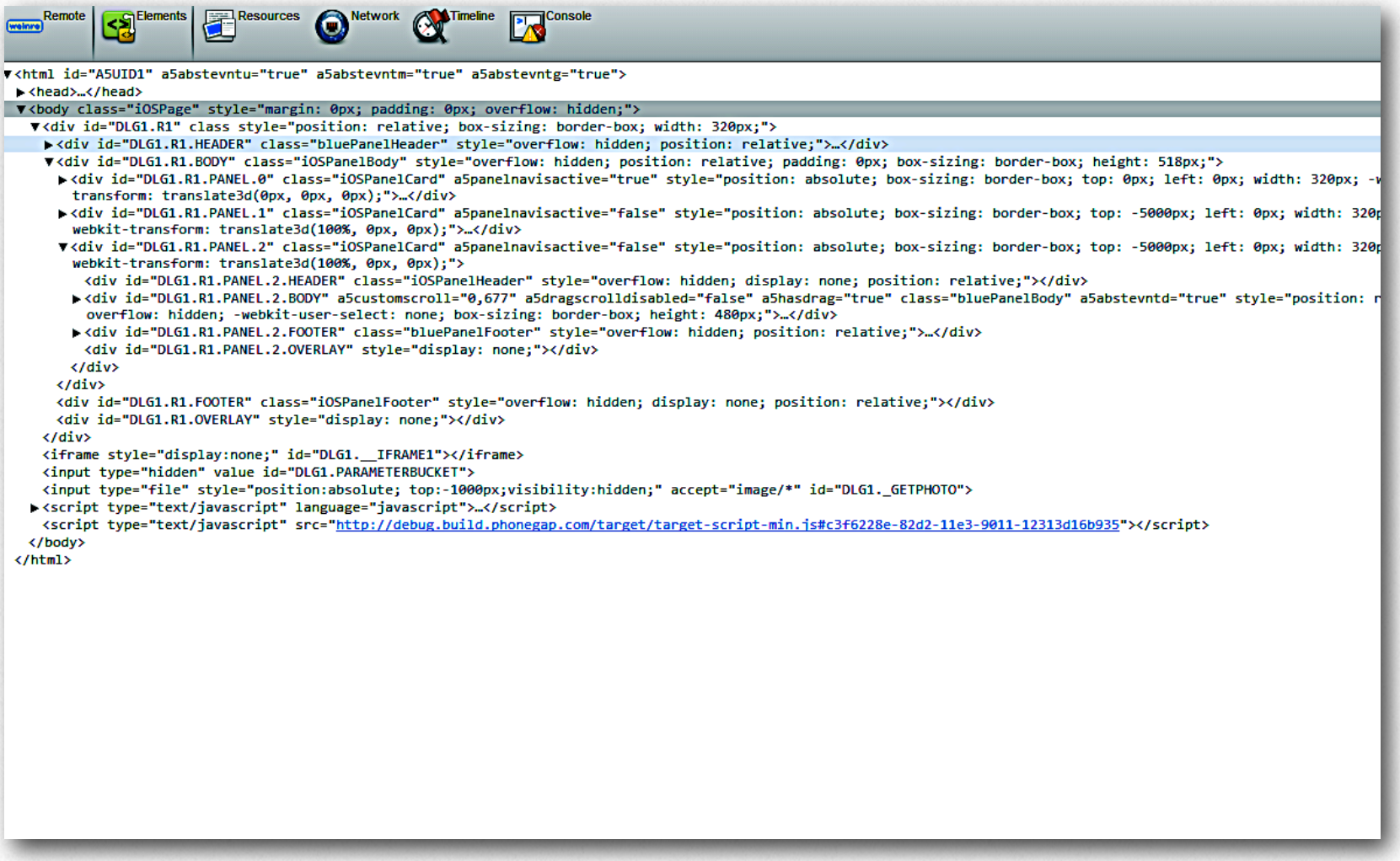


To launch a remote debug session, click the Debug button, next to the Rebuild all button. This should launch the weinre client in another tab within your browser.

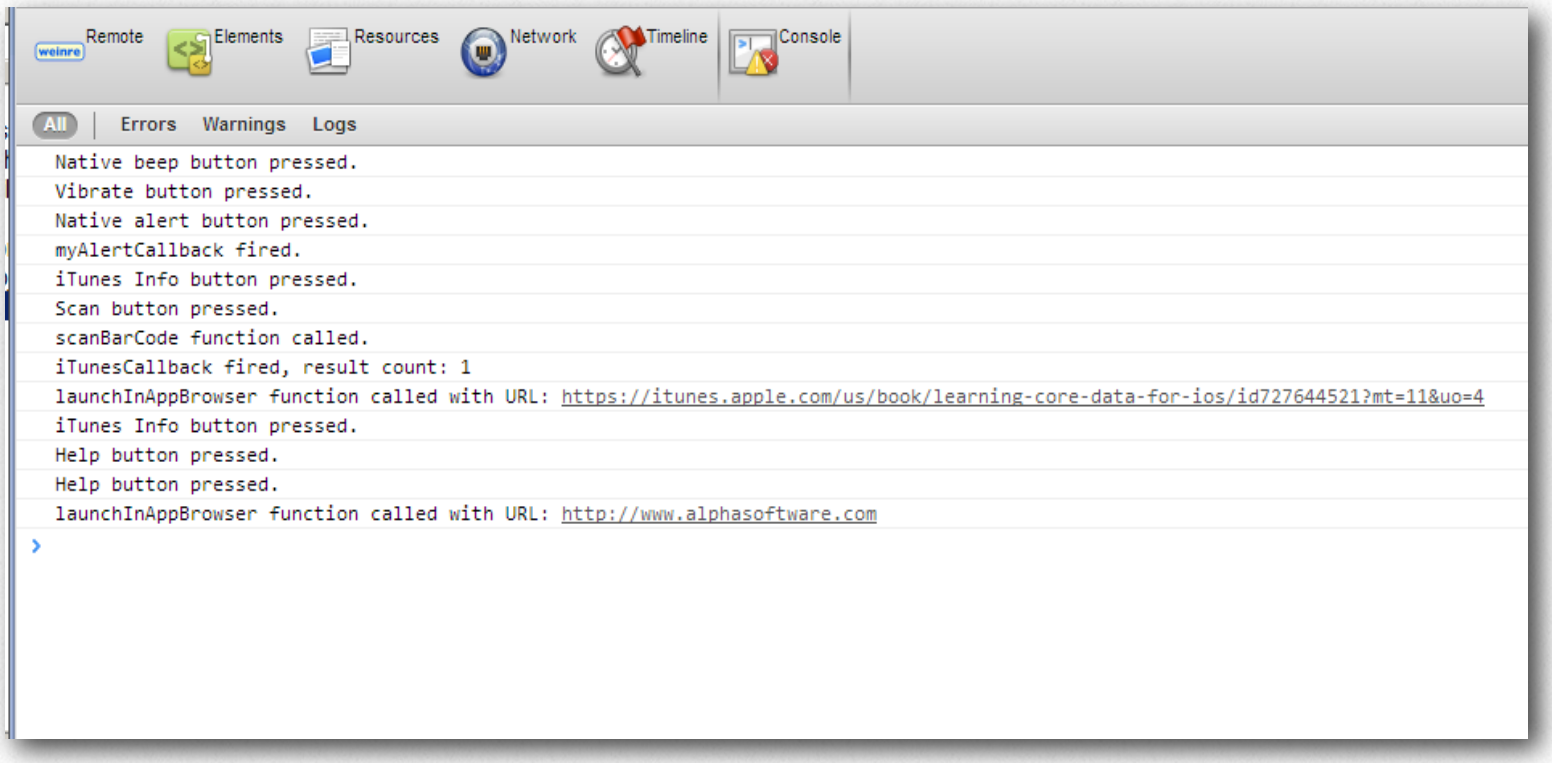


Assuming that your mobile device is connected to a wireless network and the IP address is on the same subnet as your development machine, you should be able to debug with the remote weinre server and the developer machine browser weinre client.

Click on one of the links displayed to start your debug session.



Inspecting the DOM within a PhoneGap App with weinre client



The weinre Console showing console.log messages from the PhoneGap app

You may wish to setup your own weinre server, rather than using the PhoneGap Build remote server because it's typically faster. To do so you need to install the weinre server on your development machine and configure your PhoneGap app to use your local weinre server.

Installing A Weinre Server On Your Development Machine

The weinre server runs under NodeJS so you will need to have NodeJS installed. Since the PhoneGap CLI also requires NodeJS, most likely it is already installed on your development machine.

To install a local weinre server, ***open a Windows command prompt*** and ***enter:***

On a Windows machine : ***npm -g install weinre*** <press return>

On a Mac : ***sudo npm -g install weinre*** <press return>

This should install the weinre server.

By default, weinre uses http://localhost and port 8080. Localhost will not work properly in this case so it is important that you run the service using the IP Address assigned to your wireless network. Remember, the weinre server acts as a messaging relay between the app running on your mobile device and the weinre client running within your browser. They all need to be on the same wireless network in order to function properly.

To detemine the IP address of your development machine, from a Windows command prompt ***enter ipconfig <press return>***. All of the IP addresses assigned to your machine will be displayed. Look at the adapters to identify your wireless network. That is the IP Address that will be used for the weinre server.

Let's assume your development machine wireless IP Address is 192.168.30.17.

T**o launch a weinre server at this address, on port 8080**, from a Windows command prompt ***enter:***

weinre --boundHost 192.168.30.17 *<press enter>*

To install the weinre server on a port other than the default 8080, use the --httpPort parameter.

The command line below will launch a weinre server, using an IP Address of 192.168.30.17 and port 82.

weinre --boundHost 192.168.30.17 --httpPort 82 *<press enter>*

With the weinre server up and running, you will need to modify the configuration file of the PhoneGap app and re-submit the app to PhoneGap Build.

Debugging JavaScript Within Your PhoneGap App

While debugging with weinre is certainly helpful, it cannot catch JavaScript errors (because it is also written in JavaScript and does not have low level access to the browser internals required for JavaScript debugging).

You can use weinre to send console.log messages to verify variable values or to follow the apps event flow (as is done in the example app in the previous chapter). Weinre will not catch a JavaScript error like a call to missing function or a failed call to a function because of a missing parameter, etc.

To debug the running JavaScript, you need to tether the mobile device with a USB cable to the development machine and work with the debugger that is integrated within the development machine's browser.

iOS devices support remote debugging with a Mac running OSX and Safari 6 and higher.

See http://phonegap-tips.com/articles/debugging-ios-phonegap-apps-with-safaris-web-inspector.html for a discussion of remote debugging with Safari and iOS.

Android finally supports remote debugging of native webviews (used by PhoneGap) with the release of Android 4.4, KitKat. If your Android device is running anything less than Android 4.4 you are out of luck.

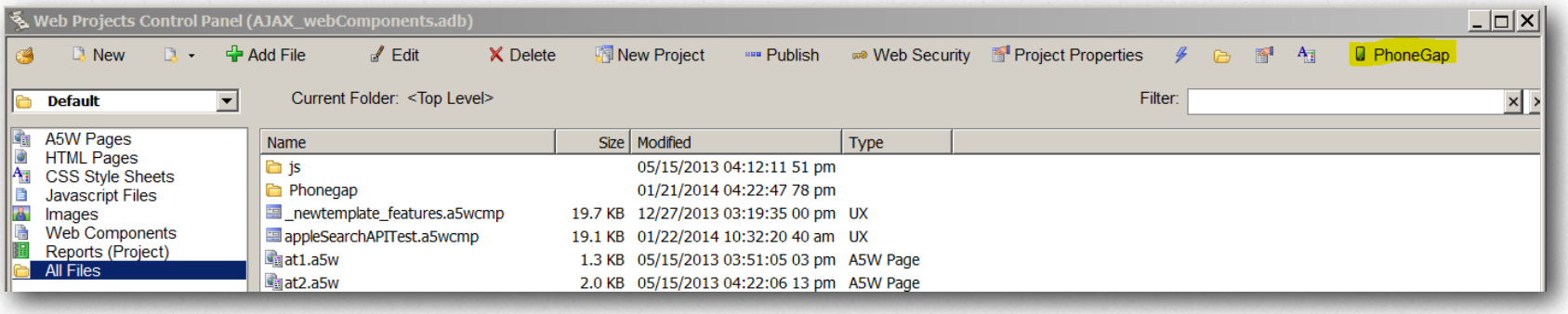
See http://www.mobilexweb.com/blog/android-4-4-kitkat-browser-chrome-webview for an overview of the new Chromium 30 web engine for the WebView in Android.

Don’t Forget About Syntax Checking With Working Preview in Alpha Anywhere

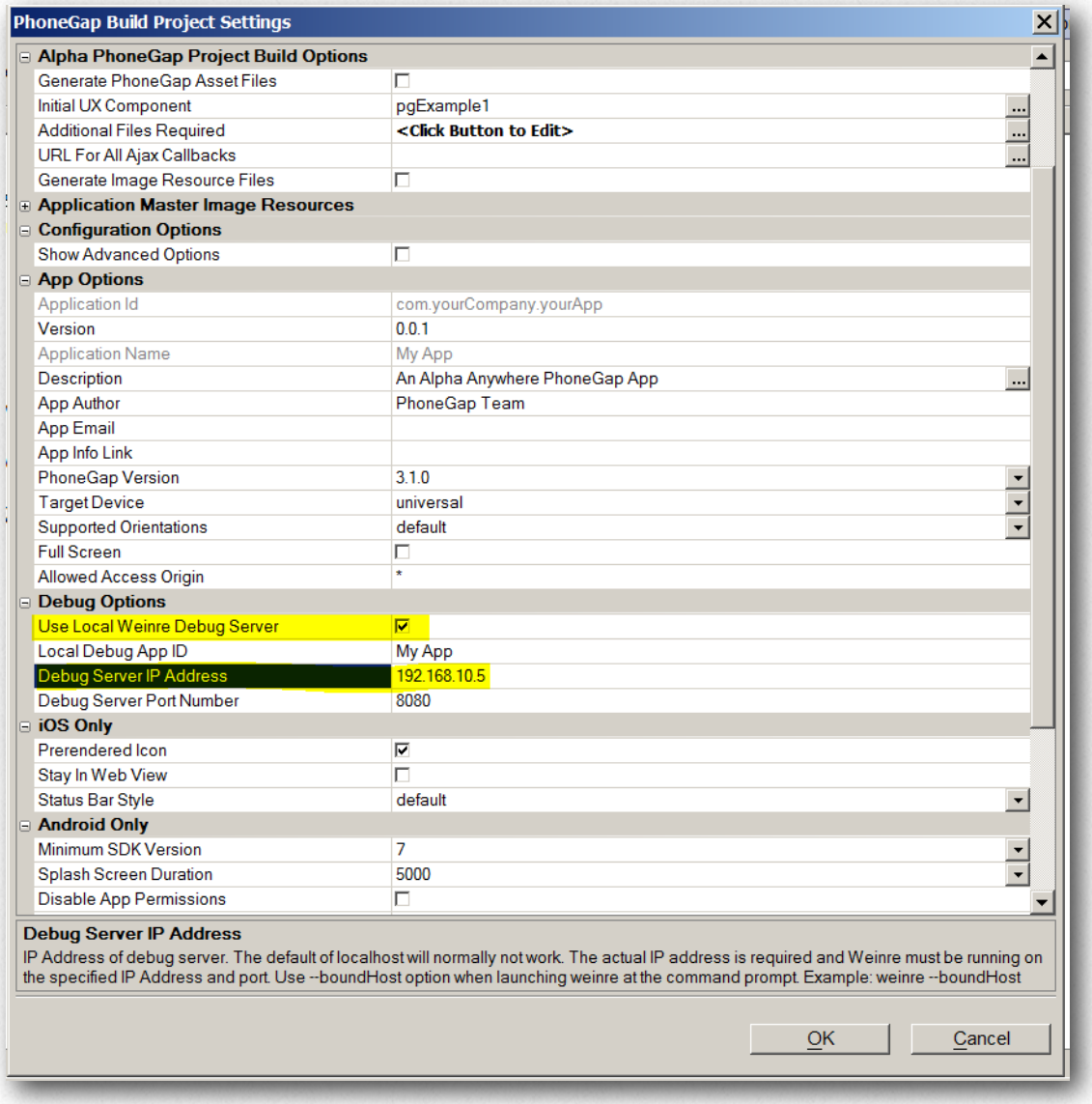
While working preview cannot run native PhoneGap functions/code, it does a good job of checking JavaScript syntax. If you cannot bring the UX component up in Working Preview, most likely you’ve got a JavaScript syntax error somewhere within your component.

Configuring your PhoneGap App To Use A Local A Weinre Debug Server

Launch the PhoneGap App Builder within the Web Projects



Launch the PhoneGap App Builder



Alpha Anywhere Integrated PhoneGap App Builder

Under Debug Options

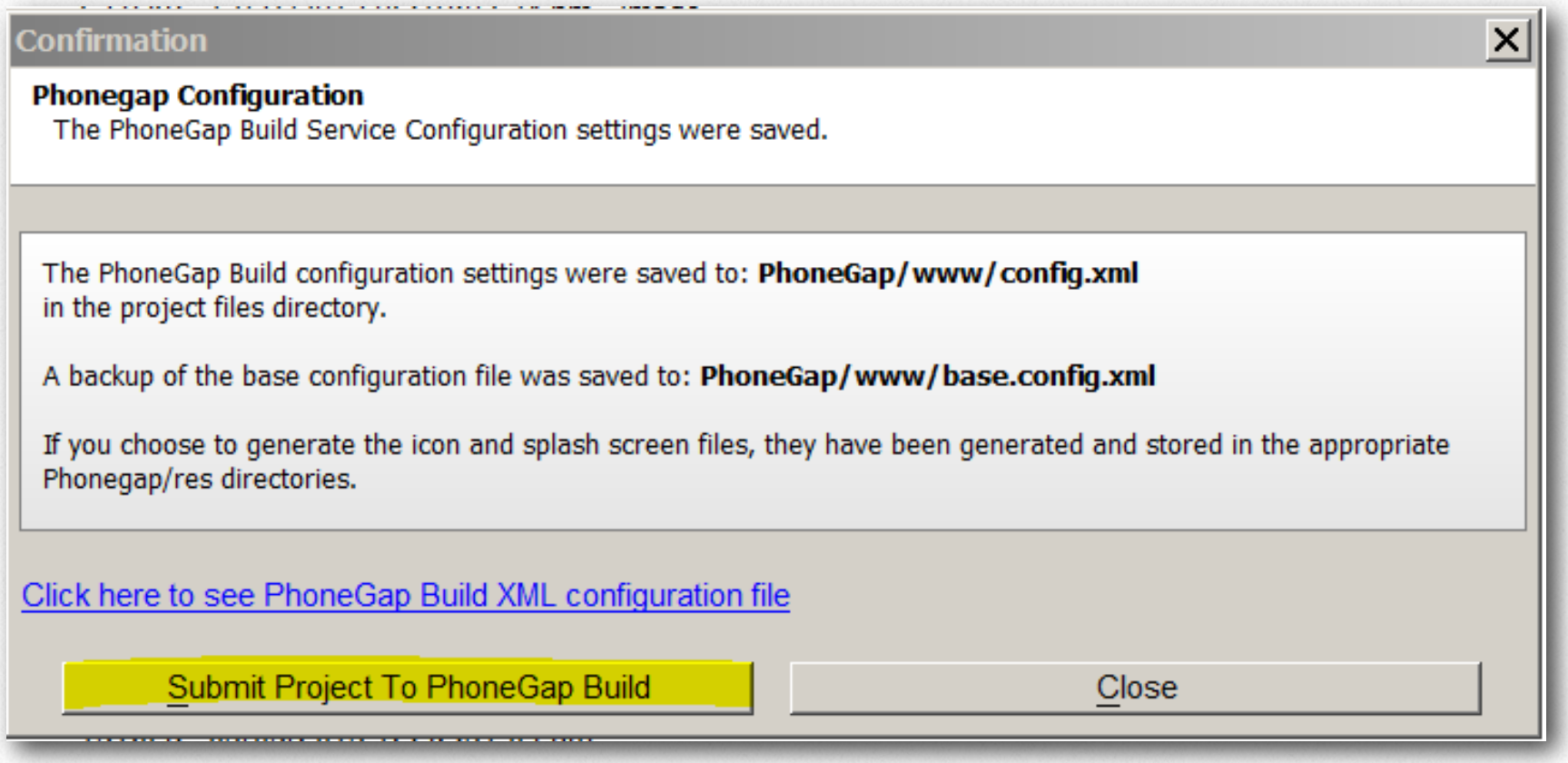
Check Use Local Weinre Server.

The **Local Debug App ID** will be displayed in the client debug window. By default it will be the same as the App name. You can change this to anything that you wish.

The **Debug Server IP Address** should be set to the IP Address of the local weinre server. The default of localhost will not work properly.

The **Debug Server Port Number** should be set to the port of the local weinre server. The default is 8080.

Click the OK button **and submit the project to PhoneGap Build**.



Submit Project To PhoneGap Build to enable debugging with the local weinre server

Once the project build is complete, launch PhoneGap Build in your browser, refresh the page and re-install the app on your mobile device. Next click the debug button on PhoneGap build and the browser will open a new tab on your development machine and you will see debug activity begin within the Windows Command Prompt running the weinre server.

You can now debug your app in a similar fashion as when using the remote weinre server however the performance should be better.

7

Pushing App Updates with Hydration

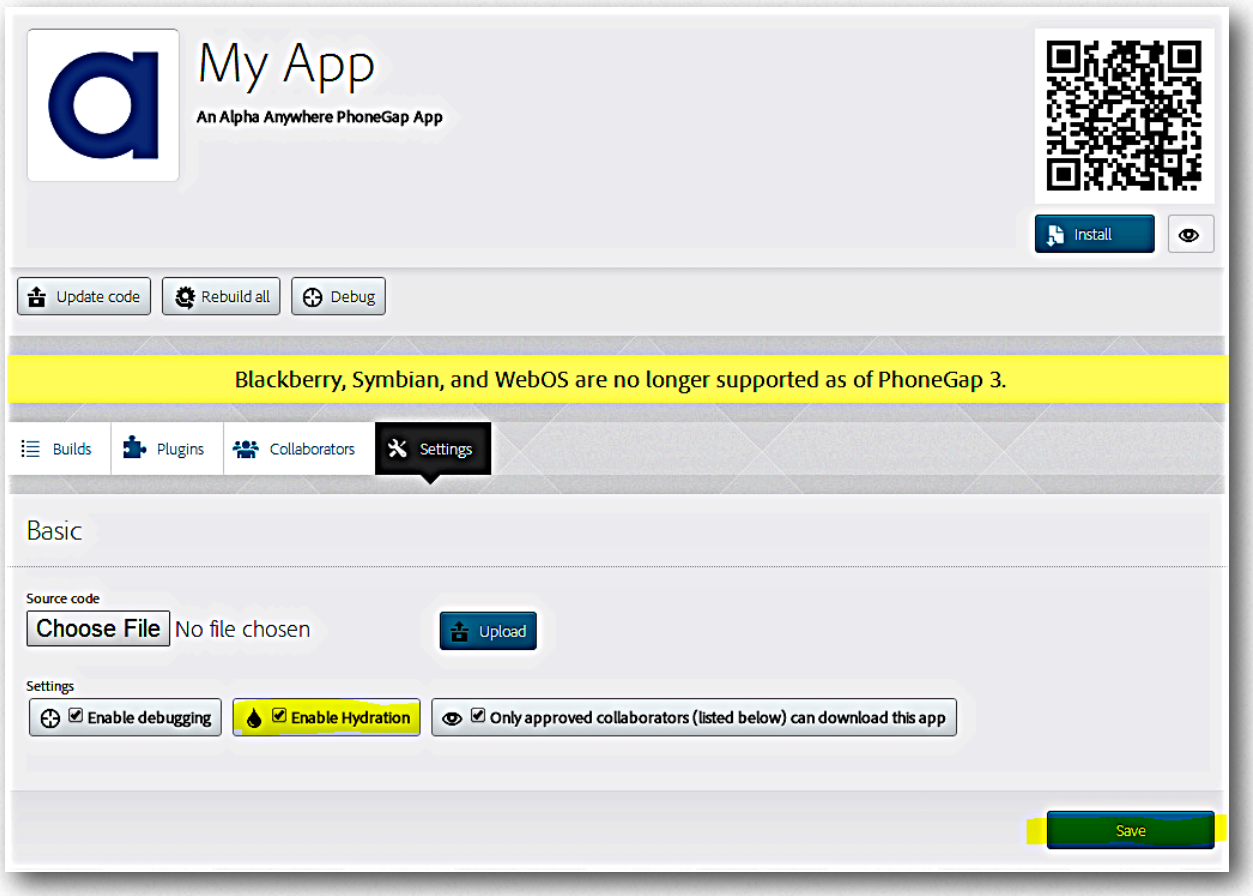
Hydration is a tool that has two main benefits for developers and testers:

- Compilation times are improved significantly.
- Updates are pushed directly to the application installed on a device.

PhoneGap accomplishes this by compiling a native binary that acts as a container for your mobile application. Once a developer uploads a new build the users testing the application will be notified upon restart of the application that a new version is available. If the user decides to run the new code base the hydrated app will automatically be downloaded and the app will run with the latest code base.

Hydrate an Existing Application

After logging in to PhoneGap Build, ***navigate to the apps page***. Next, ***select your app by clicking its title*** or icon. On the app's detail page, ***open the "Settings" panel***. Then, under the "Basic" header, ***enable the checkbox labelled "Enable hydration"***.



Until disabled, every build will produce a hydrated version of this app.

Building an application with Hydration

Building an application with hydration is simple. Once you've enabled Hydration simply hit the rebuild button and you've got a Hydrated build(s).

Build will generate a new binary if PhoneGap Build detects a modification to the project's settings or files.

Installing the hydrated application

The binary provided by build is exactly like its non-hydrated binary equivalent. Simply install the application as you would a non-Hydrating version; this can be via the QR code or the platform's specific tool (eg: iTunes, adb).

Please note that any limitations on native binaries also apply to Hydration builds. For example, Adhoc distribution (IOS) builds can only be provisioned for one-hundred devices.

Updating the application

Once a developer has compiled a new build the hydrated application can be updated by the end user. This update process occurs on every restart of the application.

When the user restarts the application they will be prompted with a dialog requesting them to update the application.

If the user decides to update the application, the new build will be downloaded and executed.

If the user decides to skip the update they will remain frozen at the current build and will be prompted to update on the next restart.

If a non-hydratable build is generated by the developer the user will need to obtain a new version of the compiled binary. The user can obtain the binary by downloading it, or by using one of our other methods.

Disabling Hydration

After logging in Navigate to the apps page, ***select your app by clicking its title or icon***. On the app's detail page, ***open the "Settings" panel***. Under the "Basic" header ***de-select the checkbox labeled "enable hydration", and hit save***.

You've now successfully disabled Hydration.

Users will now need to download / install updates to the app manually since their mobile devices will contain the hydrated app.

Anomalies with Hydrated Apps

Some of the third party plugins do not work in hydrated apps. These seem to be plugins that create objects tied to the root window object in the DOM. Specifically the flashlight plugin used in the example component has proven to have problems.

8

Ajax Callbacks

Ajax callbacks are essential to create, edit and update the data stored within a UX component.
In this chapter, we modify the Demo Mobile App included with Alpha Anywhere to run as a PhoneGap App that communicates with an Alpha Anywhere Server, storing and retrieving photos from a backend database.

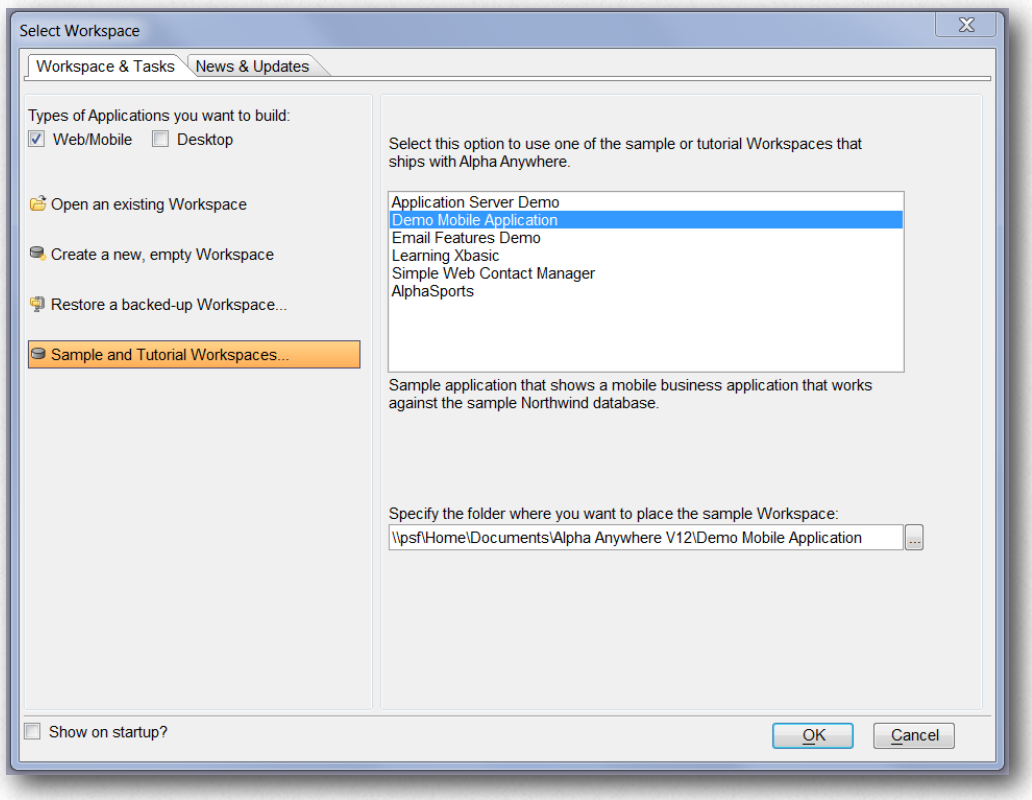
Up until this point, the UX components we have used have been communicating with outside web services, like the Apple Search API used in chapter 5.

Alpha Anywhere makes it easy to tie a UX component to a wide variety of backend databases through the Alpha Anywhere Server. Some popular SQL database engines include MySQL, SQL Server, PostgreSQL, Oracle, DB2, etc.

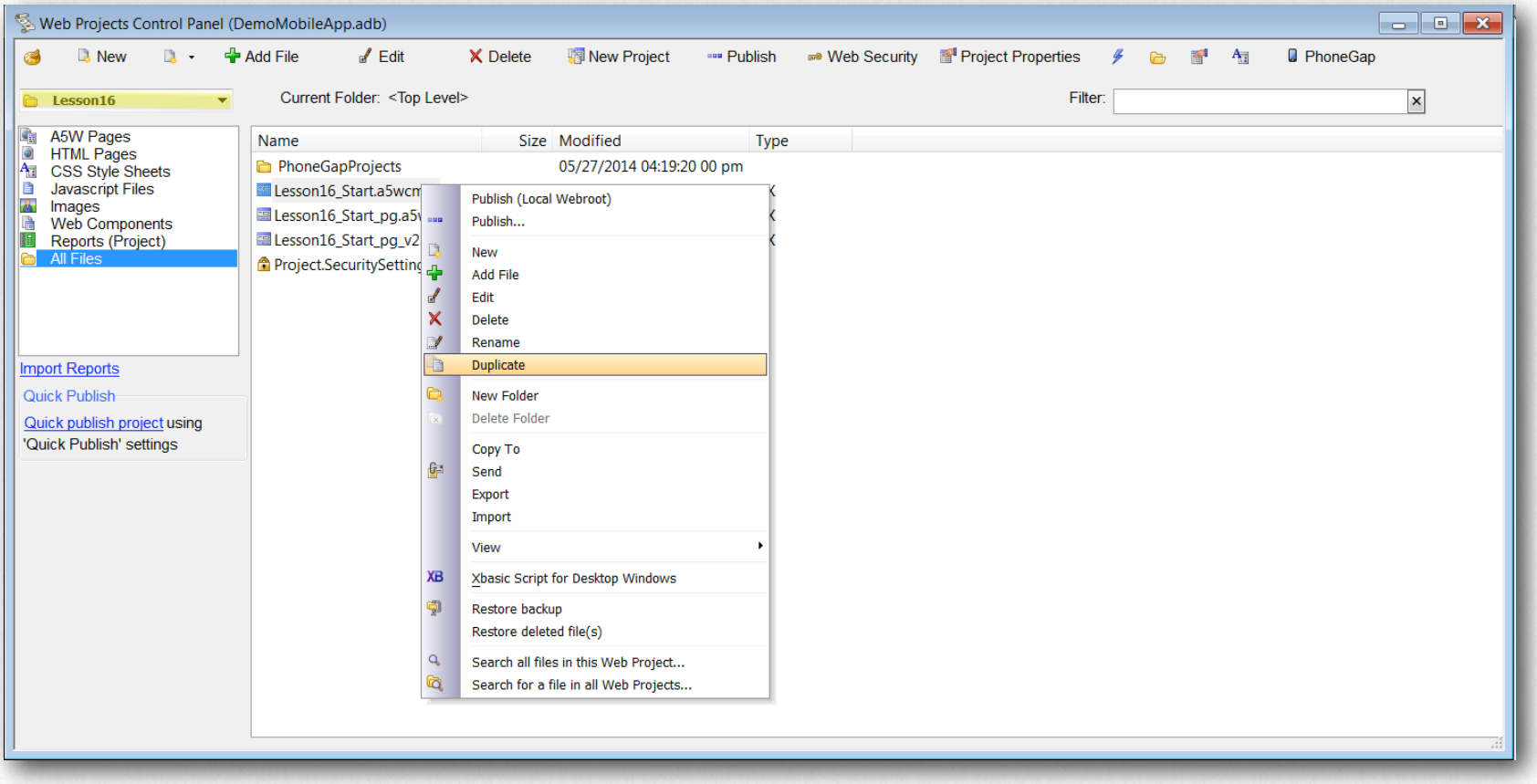
In order to communicate with an Alpha Server, you MUST publish the initial UX component to an Alpha Anywhere Server and you MUST provide a proper URL to locate the server from within the PhoneGap app installed on the mobile device.

In this chapter, we will modify the image upload capability of the sample mobile application supplied with Alpha Anywhere to use a native camera/image control and the image will be uploaded from the device to an Alpha Anywhere Server. This is done with an Ajax callback.

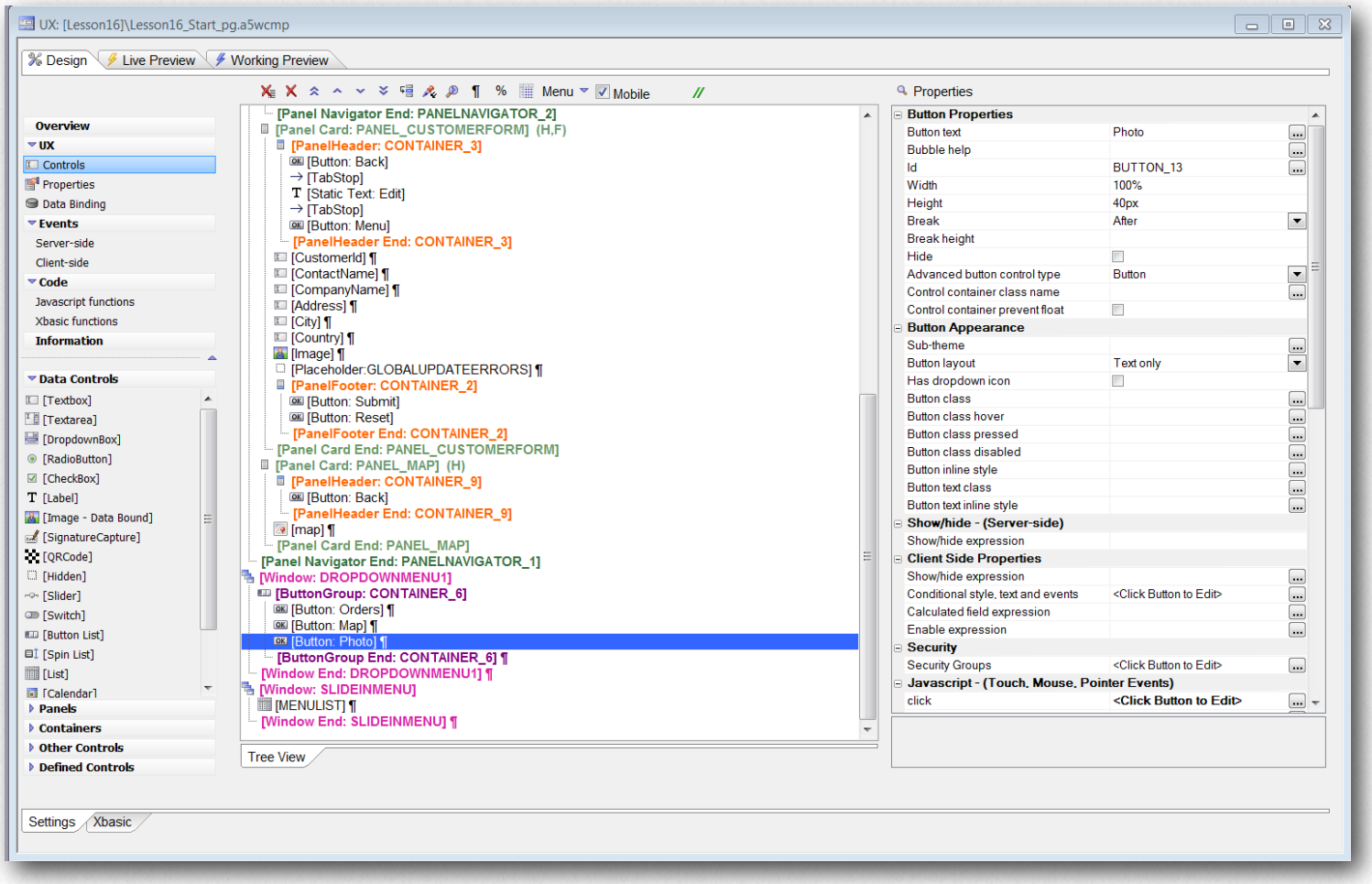
Click on Recent Workspaces | Sample and Tutorial Workspaces and select the Demo Mobile Application



Once the files are copied, ***select Chapter 16*** from the project list and ***right click*** on the component named Lesson16_Start.a5wcmd and ***Select Duplicate. Enter Lesson16_Start_pg.a5wcmd*** to create a duplicate copy of the component that will be used for the PhoneGap modifications.

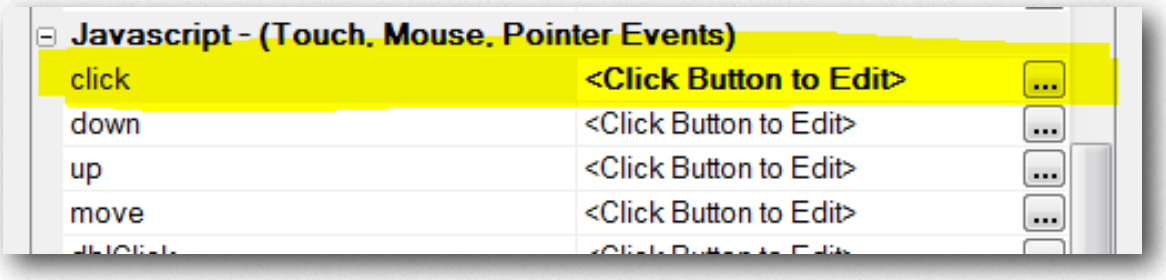


Open the Lesson16_Start_pg UX component and ***select the button labeled Photo*** from within the Window DropDownMenu container.

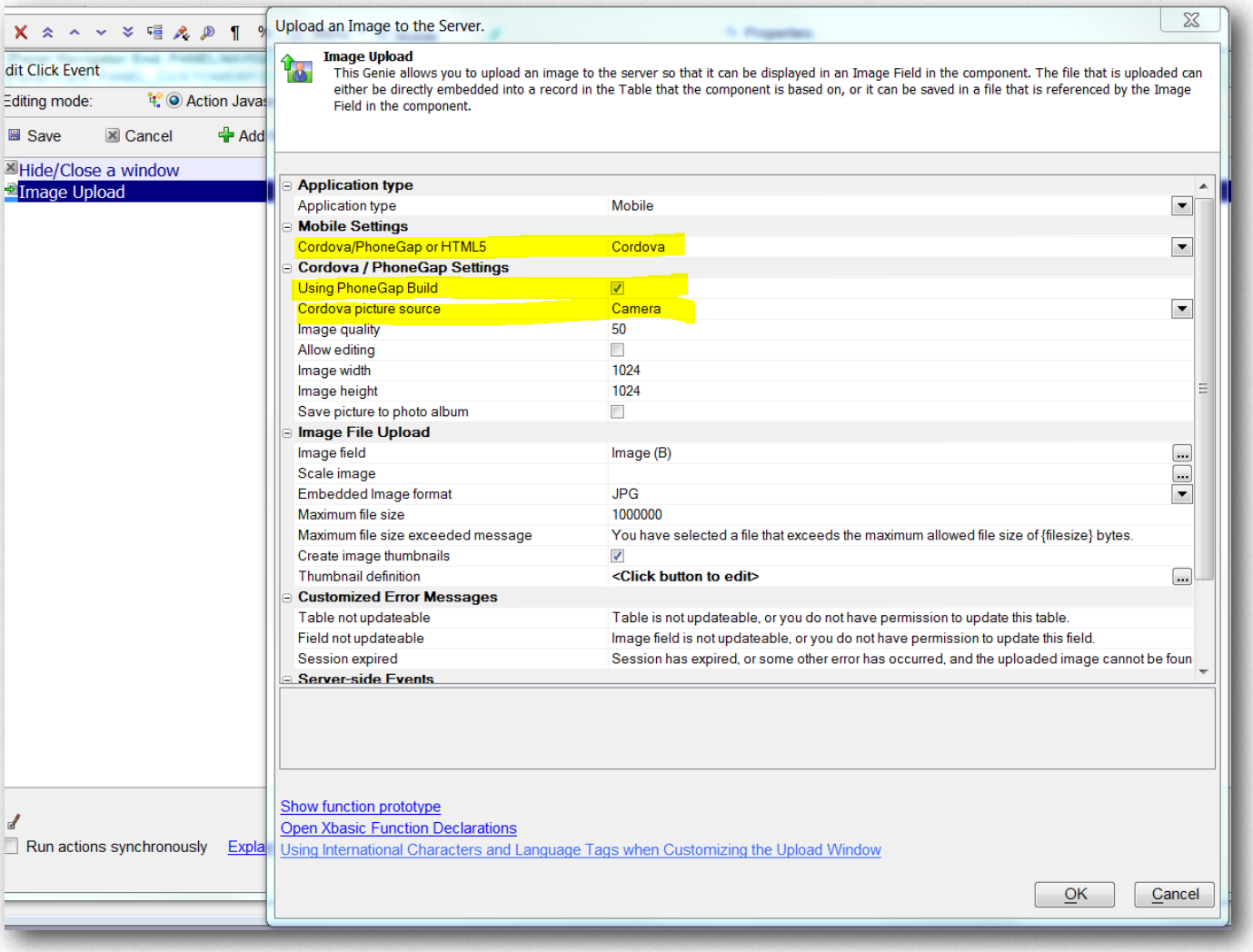


We are going to modify the Action JavaScript associated with the virtual click event on this button.

Select Click Button To Edit



and ***modify the Image Upload Action JavaScript*** to use Cordova/PhoneGap and make sure Using



PhoneGap Build is checked. The Cordova picture source should be Camera. Click OK and save the component. You have just made all of the changes required to this component to use the native camera controls within a PhoneGap app.

In order for a PhoneGap App, installed on a mobile phone or tablet to communicate with an Alpha Anywhere server, there are a number of considerations:

1. What is the IP address of the server?

2. What protocol and port are being used by the server?

3. What is the name of the directory within the A5Weboot (if any) on your Alpha Anywhere Server that the component is being published to?

4. What is the name of the component that the PhoneGap app is based upon?

5. Did you publish the component that the PhoneGap app is based upon to the Alpha Anywhere server?

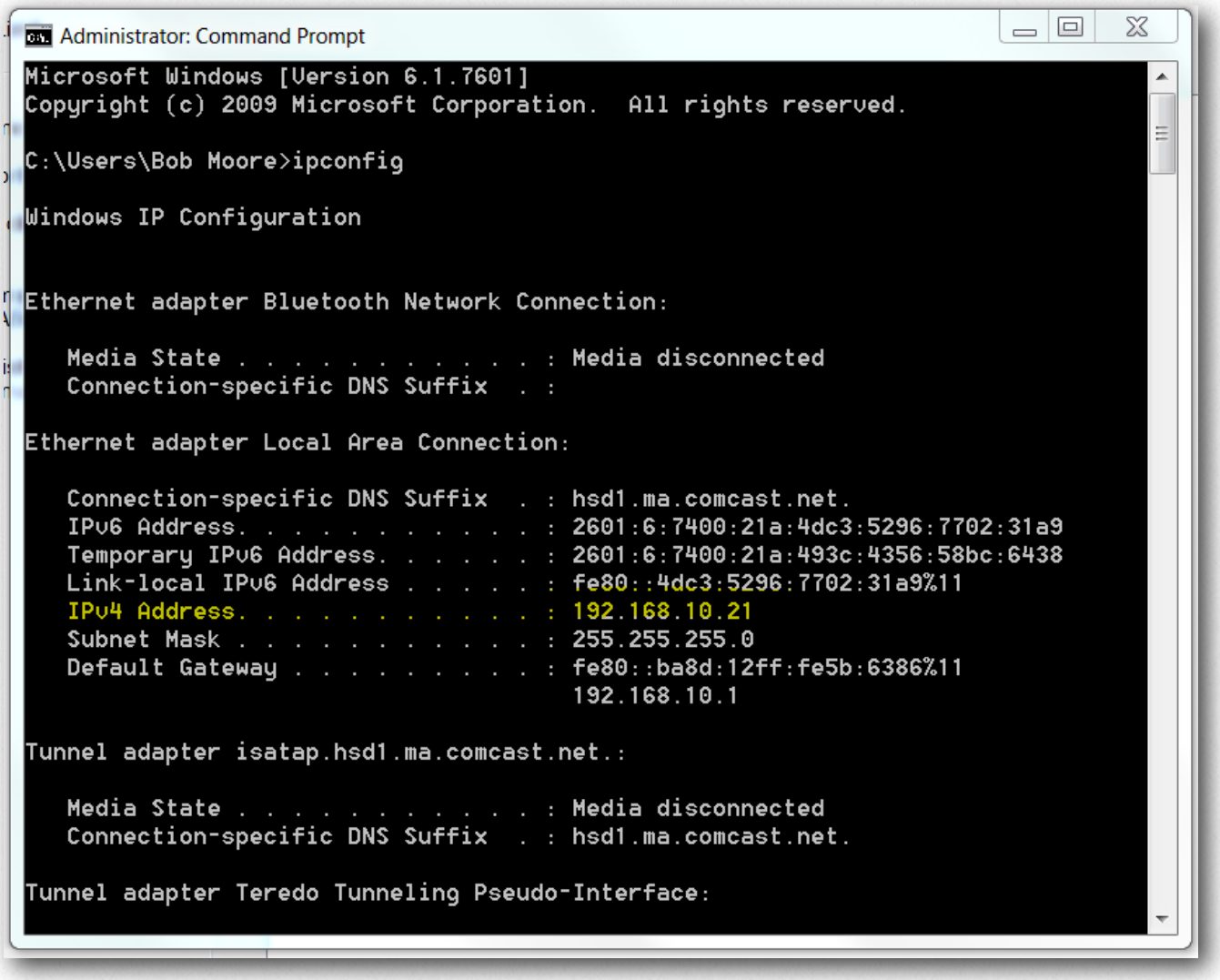
We'll look at each of these questions in detail.

Determining IP address of the server

For PhoneGap apps, it's best to use the IP address of your server rather than relying on DNS name resolution simply because it is faster. Your mobile users will never see this IP address when running your app.

Your testing will initially be done using the Alpha Development Server that is included with Alpha Anywhere. This is a server that runs on your development PC that allows a limited number of sessions (up to 5). It is perfect for testing and debugging PhoneGap apps on your mobile devices.

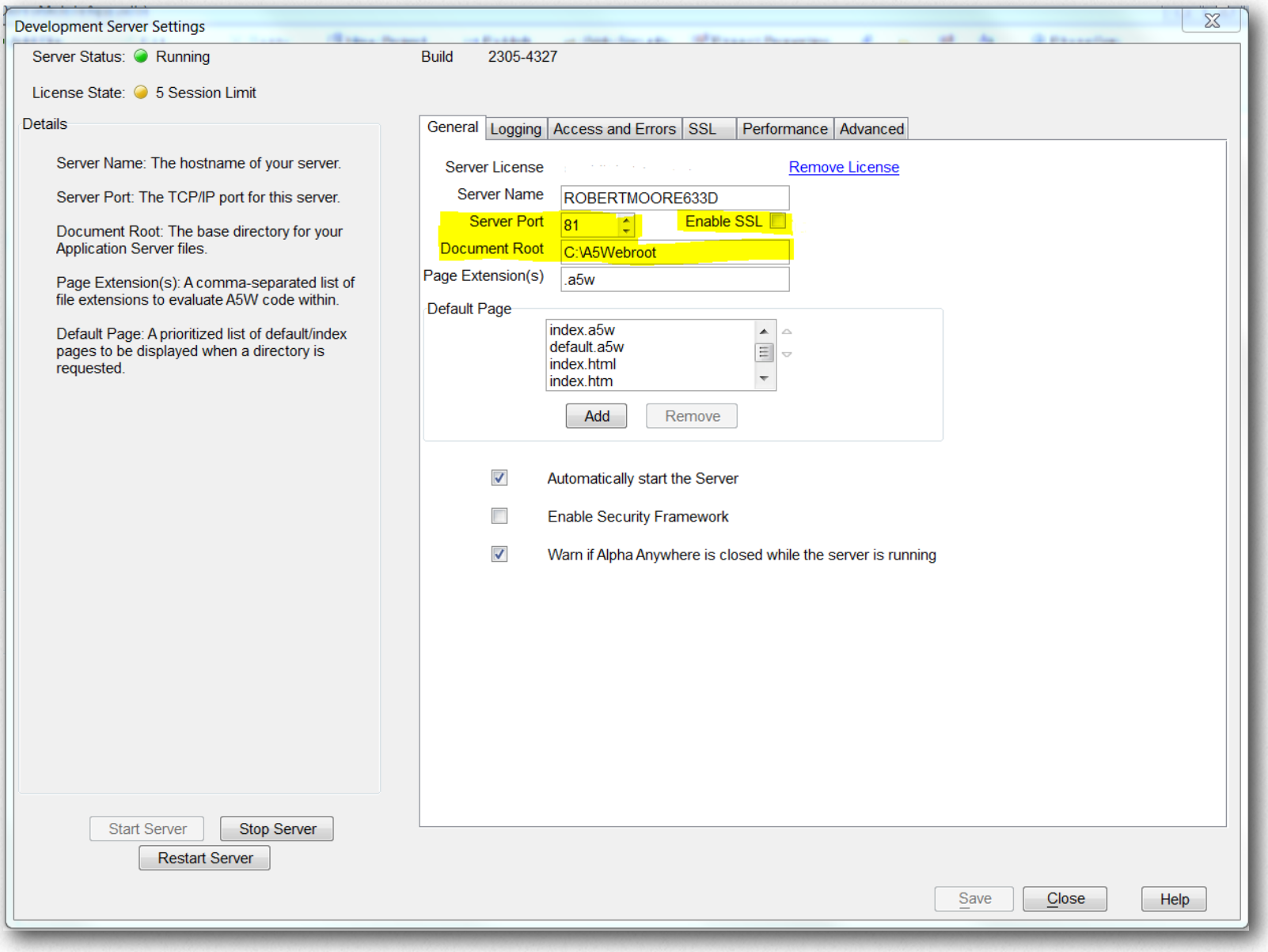
To determine the IP address in use on your machine, open a command prompt at ***enter ipconfig <enter>***. Your screen will look similar to the image below:



Here we can see the IPv4 Address assigned to this machine is 192.168.10.21. Your machine's IP address will vary. ***Make a note of this value.*** In my development setup, this IP Address was issued by a wireless router running a DHCP server. For initial testing on a mobile device, it is important that the mobile device be on this same network/sub-net as the development server. You want to make sure your phone or tablet is tied in to the same wireless router as your development server.

Determining the protocol and port of your development server

Click on Web | Development Server from within the IDE to display the Development Server Settings.

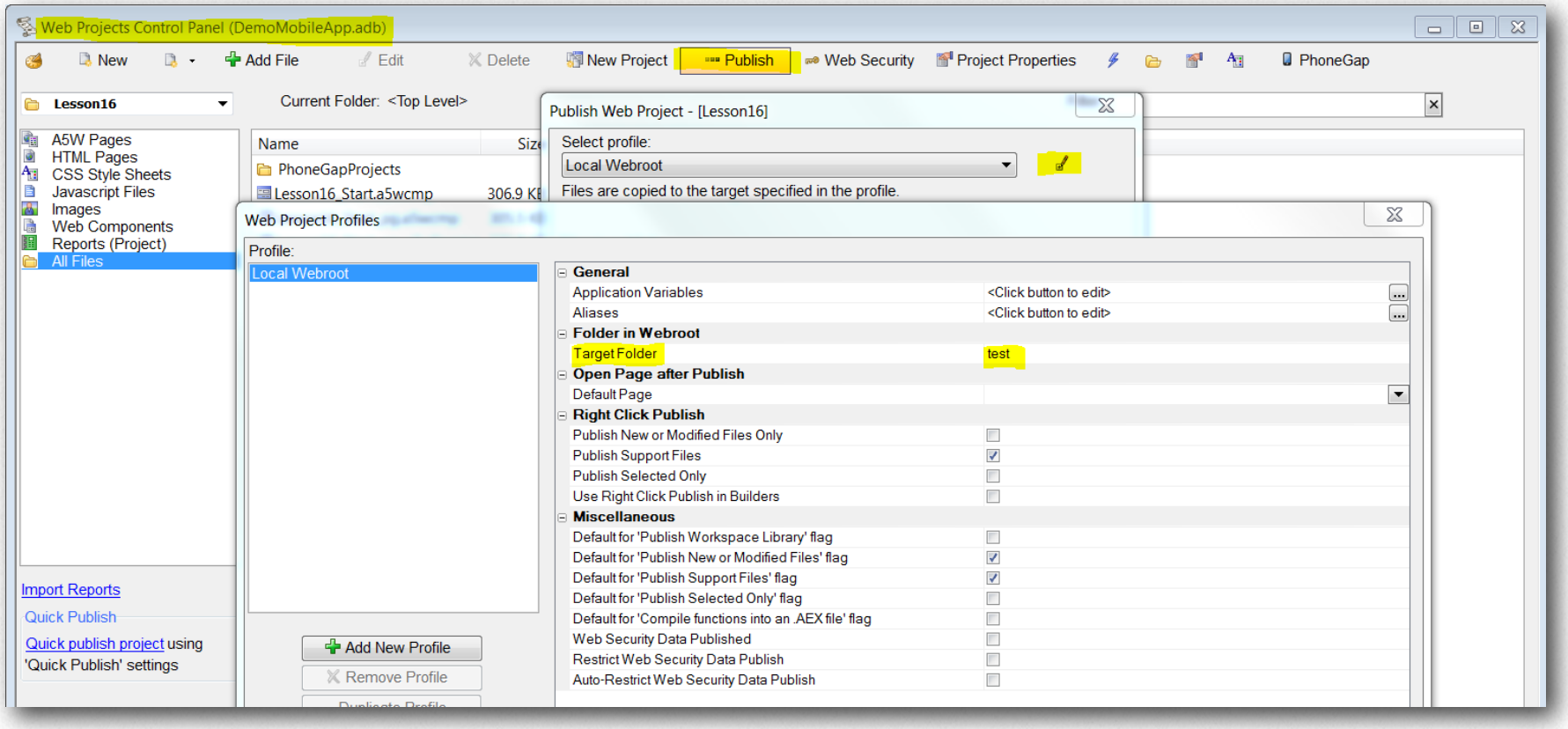


Make a note of the Server Port, ***and the status of the Enable SSL checkbox***. In this example, the port in use is 81 and SSL is not being used. When SSL is not in use, the protocol used is HTTP. When Enable SSL is checked, the protocol used is HTTPS. You may also want to make a note of the document root so you can verify the component file exists (after publishing) within the target folder, should you run into any problems when running your app on a mobile device.

Determining the server publishing directory

The UX component that serves as the basis for the PhoneGap application (as specified within the PhoneGap App Builder Genie) must be published to the Alpha Anywhere server in order for the PhoneGap application installed on the mobile device to make callbacks to the Alpha Anywhere Server. To determine the publishing location, from within the Web Projects Control Panel, open the Publish settings dialog.

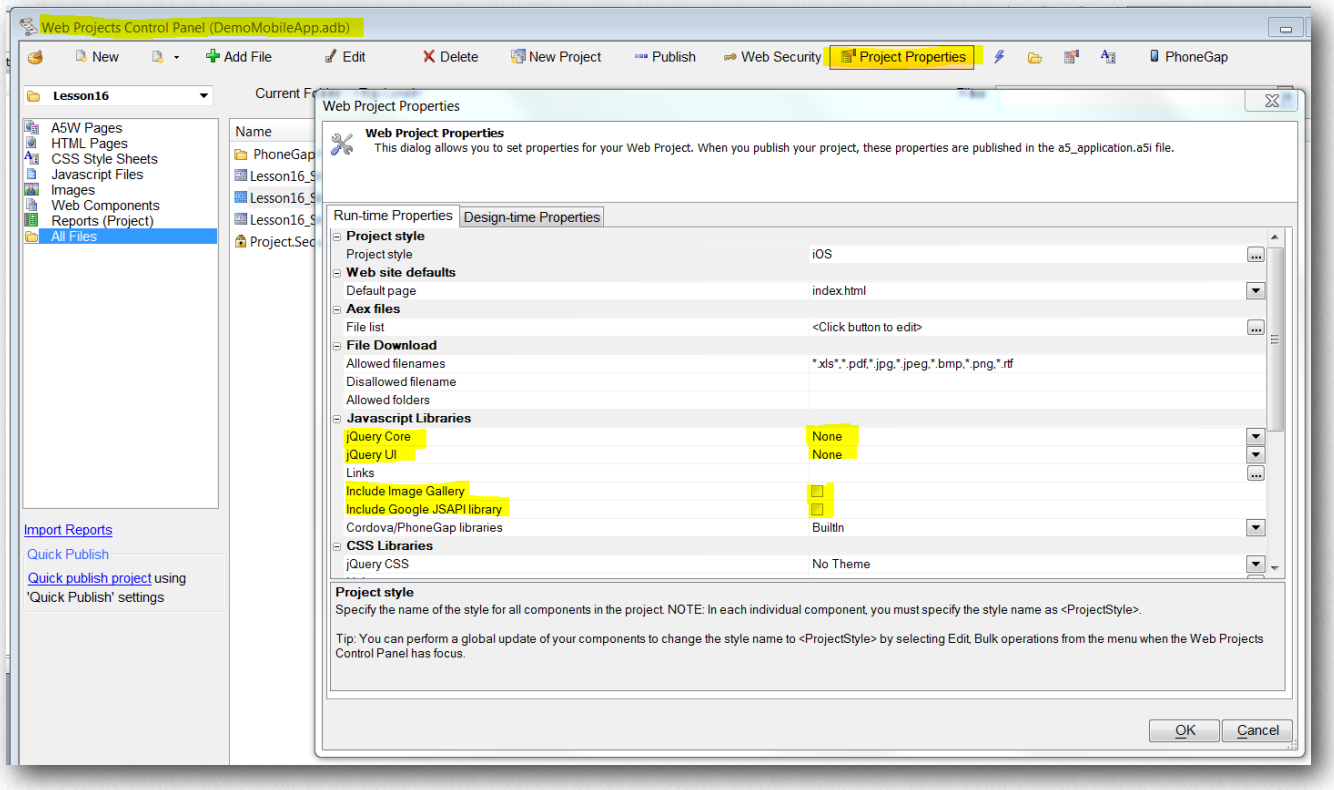
In the example below, the component is being published to a directory named test. Make a note of this directory.



If the Target Folder is left blank, the component will be published to the document root, identified in the previously discussed Server Settings dialog.

Verify the web project properties

Before publishing the component or generating the PhoneGap App, it is a good idea to take a look at your Web Project Properties to ensure that you are only loading the JavaScript Libraries that you actually need for your component. In our case, we don't need any additional libraries. The idea is to keep the client side payload as small as possible. From the Web Projects Control Panel select Project Properties. The Run-time Properties should match the dialog shown below.



Specifying the name of the component

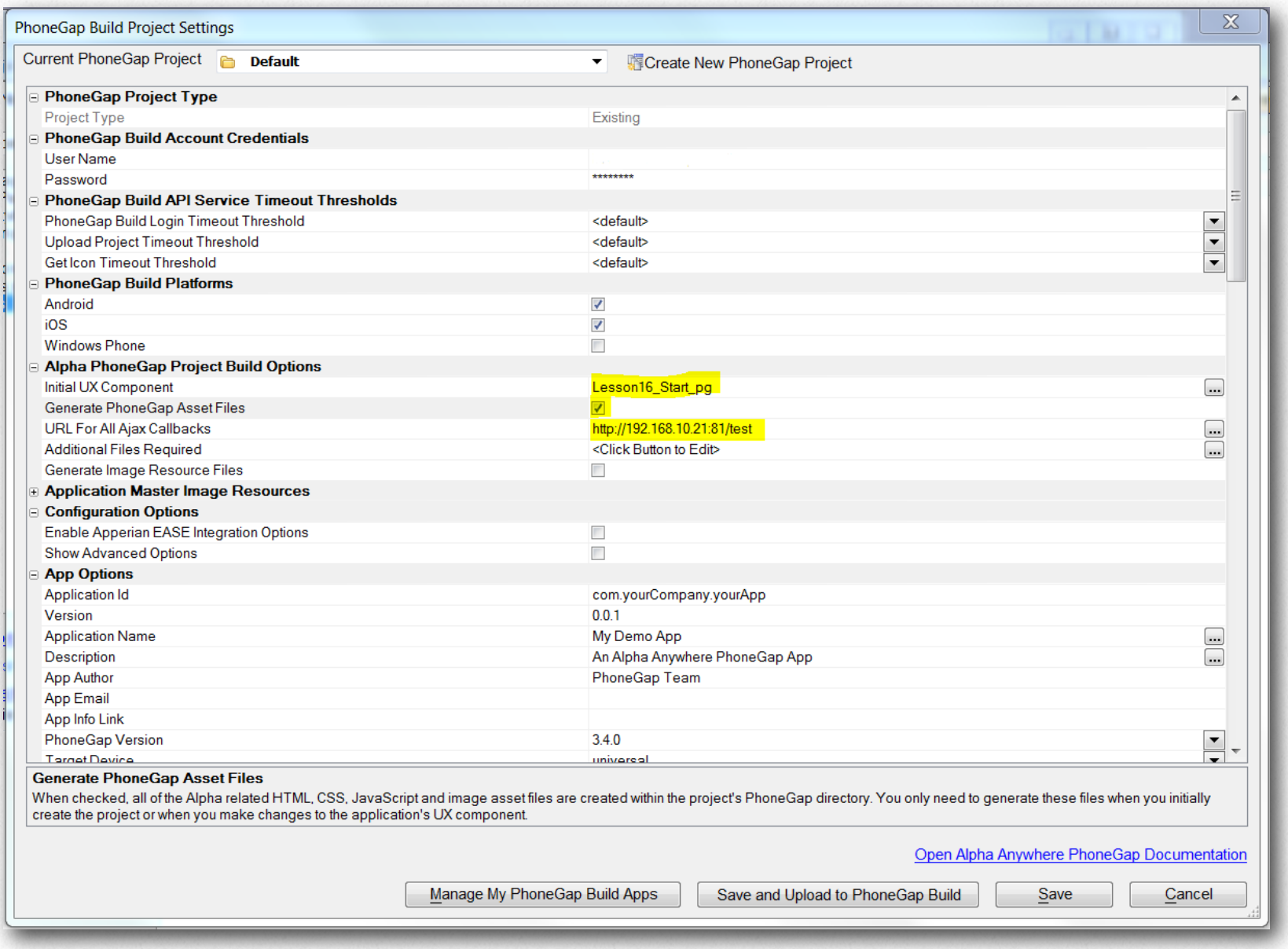
We will use the PhoneGap App Builder to enter the name of the component and the application server details.

Launch the PhoneGap App Builder from **Web Projects Control Panel | PhoneGap**

Under the Alpha PhoneGap Project Build Options select the Initial UX component, Lesson16_Start_pg. Make sure the Generate PhoneGap Asset Files checkbox is checked.

You need to specify the URL to communicate with the Alpha Anywhere server in the URL For All Ajax Callbacks property. You must specify the protocol, IP Address, port number (if you are using anything other than the default, port 80) and if publishing to a directory within the A5Webroot, the name of that directory.

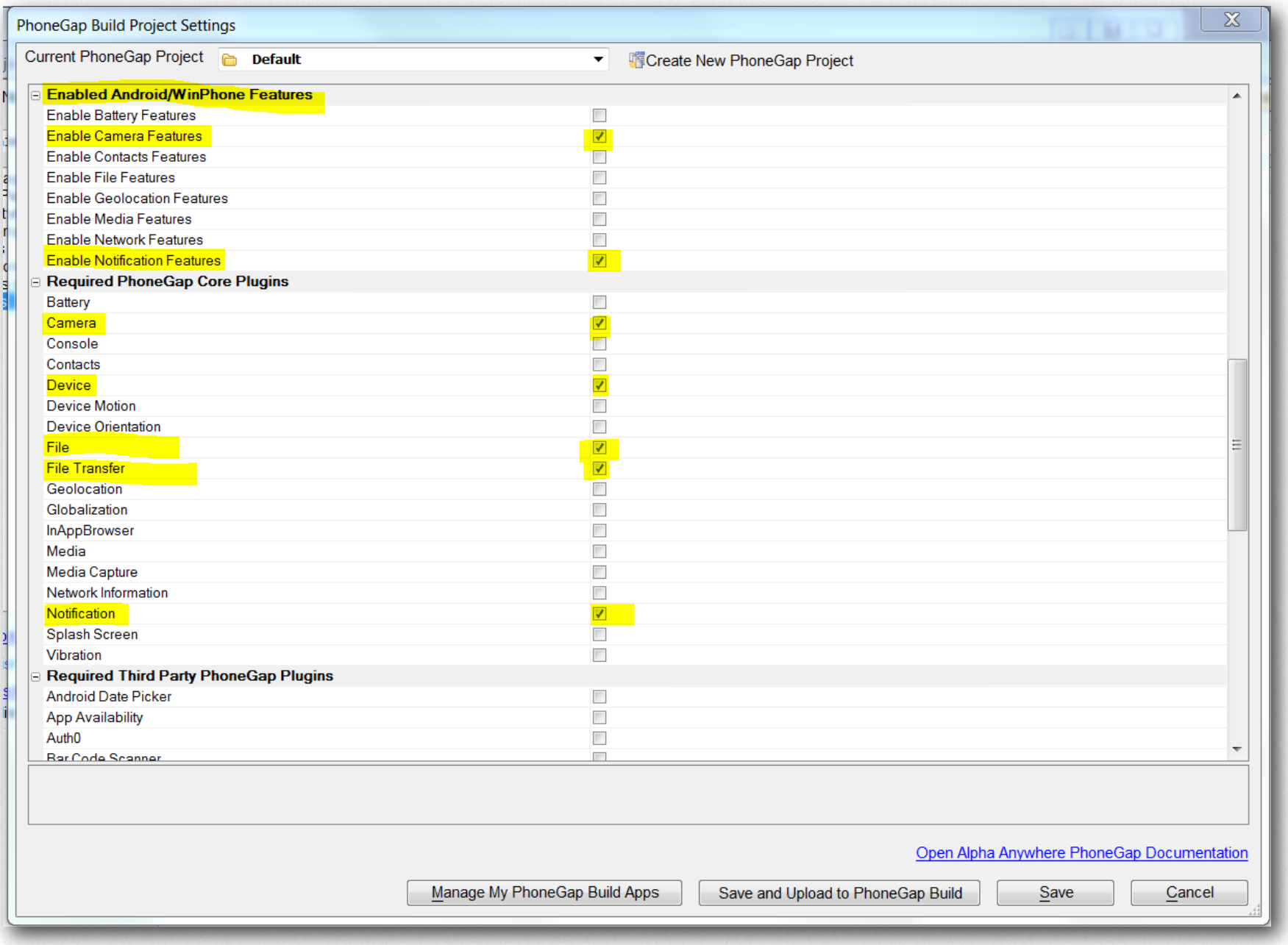
In the image shown below, the protocol in use is http, the IP address is 192.168.10.21, the port in use is 81 and the component was published to the test directory within the A5Webroot. It is designated as http://192.168.10.21:81/test



Adjust PhoneGap App Builder Properties for Android Features and Required Plugins

You will not need to select all of the required PhoneGap plugins required for native camera access and image uploads (but feel free to verify the plugins are in fact selected) because the plugins have been automatically selected for you. This occurs when you use Action JavaScript to include this native PhoneGap functionality in your component.

For Android apps, ***make sure to enable the Camera and Notification features.***

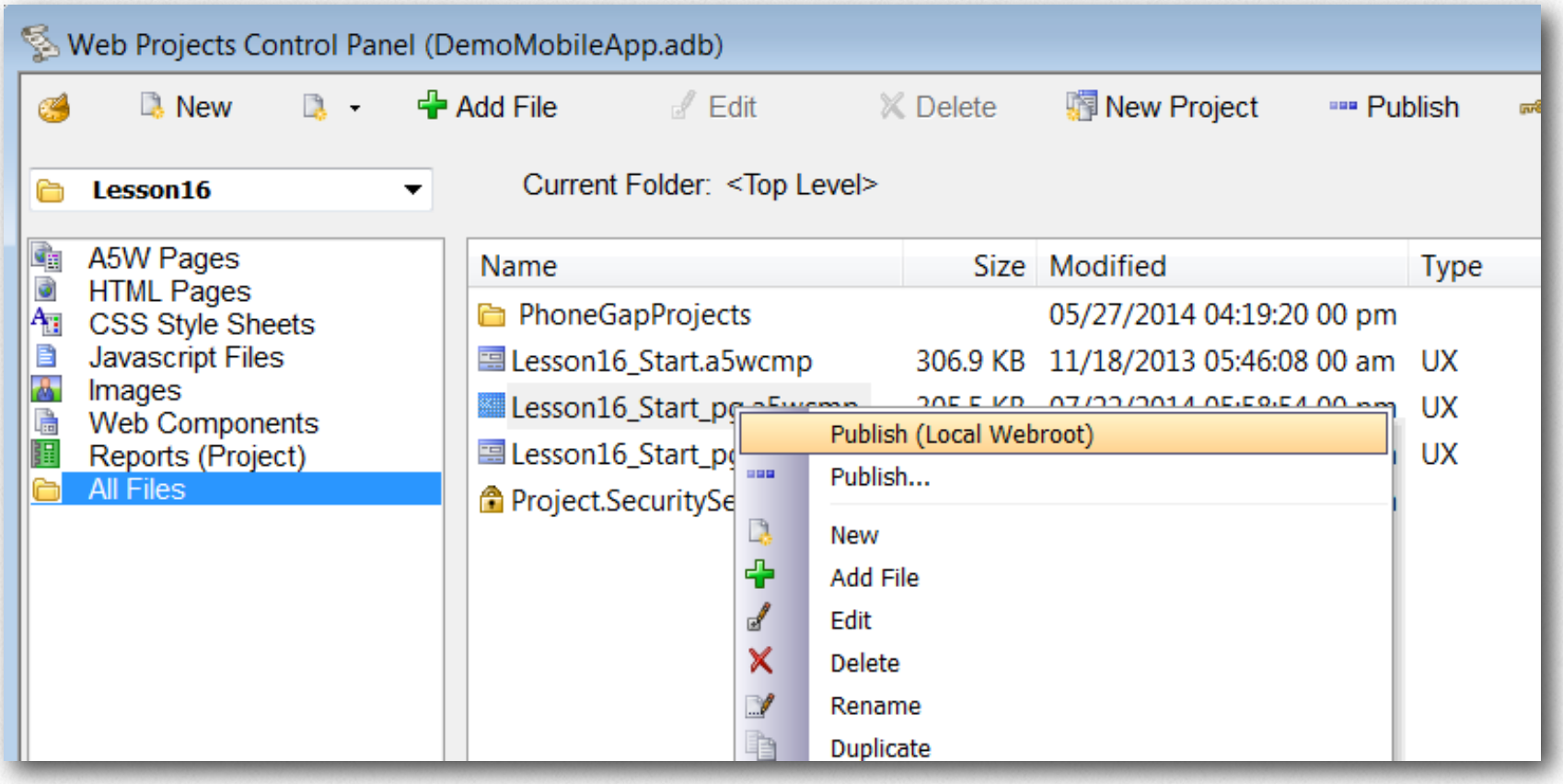


Publish The App To PhoneGap Build

Assuming you have entered your PhoneGap Build credentials, you can ***Save and Upload the app to PhoneGap Build***, however, the app will not run properly (callbacks will fail) on your mobile device unless you publish the component designated in the initial UX Component property to the Alpha Development Server.

Publishing the component to the Development Server

From within the Web Projects Control Panel, right click on the component named Lesson16_Start_pg.a5wcmp and select Publish (Local Webroot).



Install and Test

The app is now ready for testing. You should be able to install the app on your mobile device, take a picture and save the image to the Alpha Development Server.

Reference Video

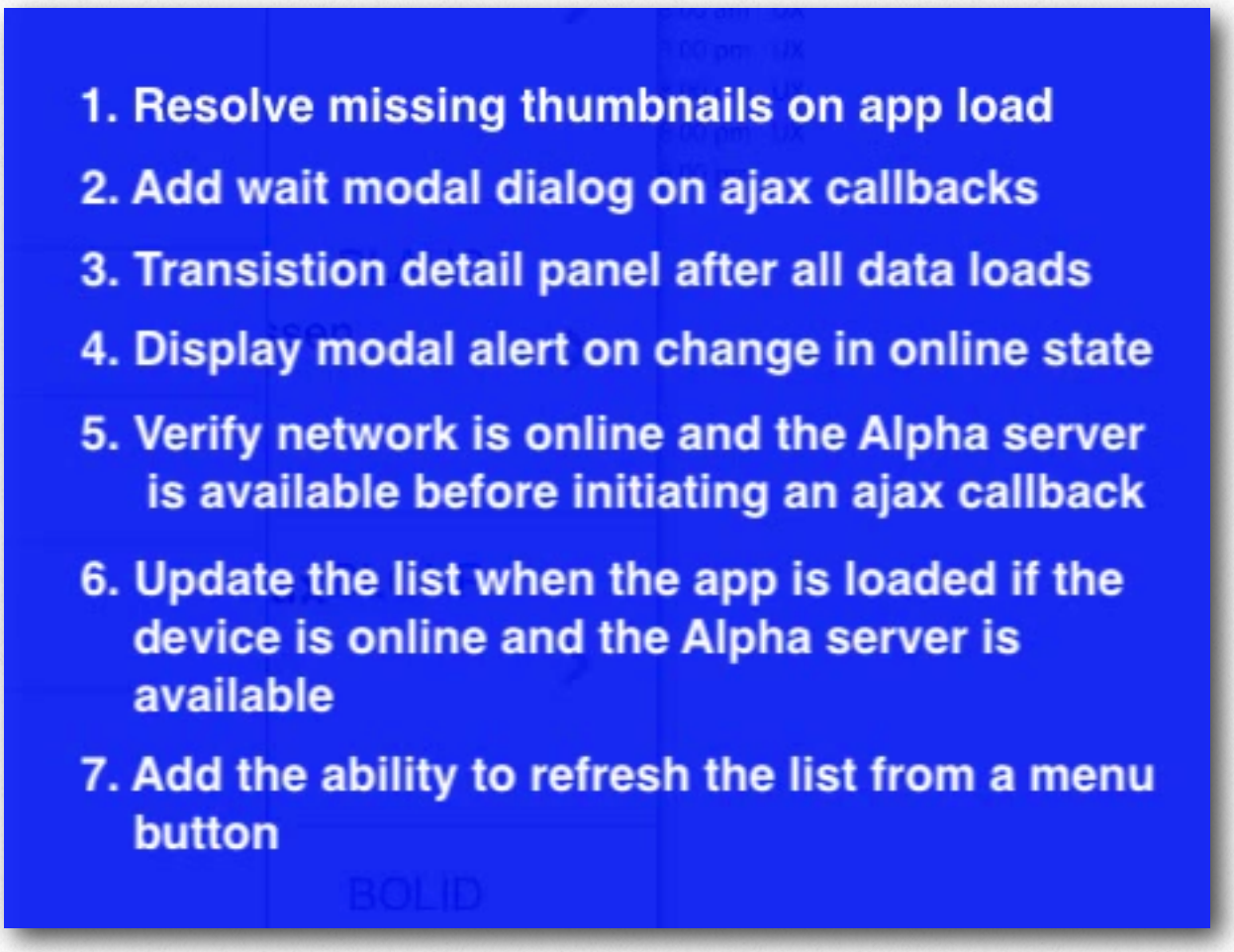
See: [Adopting the Demo Mobile Component for use with PhoneGap, Part 1]

9

Optimizing The Demo Mobile App for PhoneGap

In this chapter, we will incorporate a number of enhancements to the Demo Mobile App to improve it's functionality and improve the user experience when the app is deployed as a native PhoneGap application.

While we have successfully deployed the Demo Mobile App as a native PhoneGap app (as covered in Chapter 8), there are a number of shortcomings and features that require further enhancements. Some of the issues that need to be addressed to make this a polished native PhoneGap app include:



Source code: The source code for this component will all revisions described herein is located in the Alpha Anywhere executable directory under PhoneGap/examples/2 folder. The filename is Lesson16_Start_pg_v2.a5wcmp.

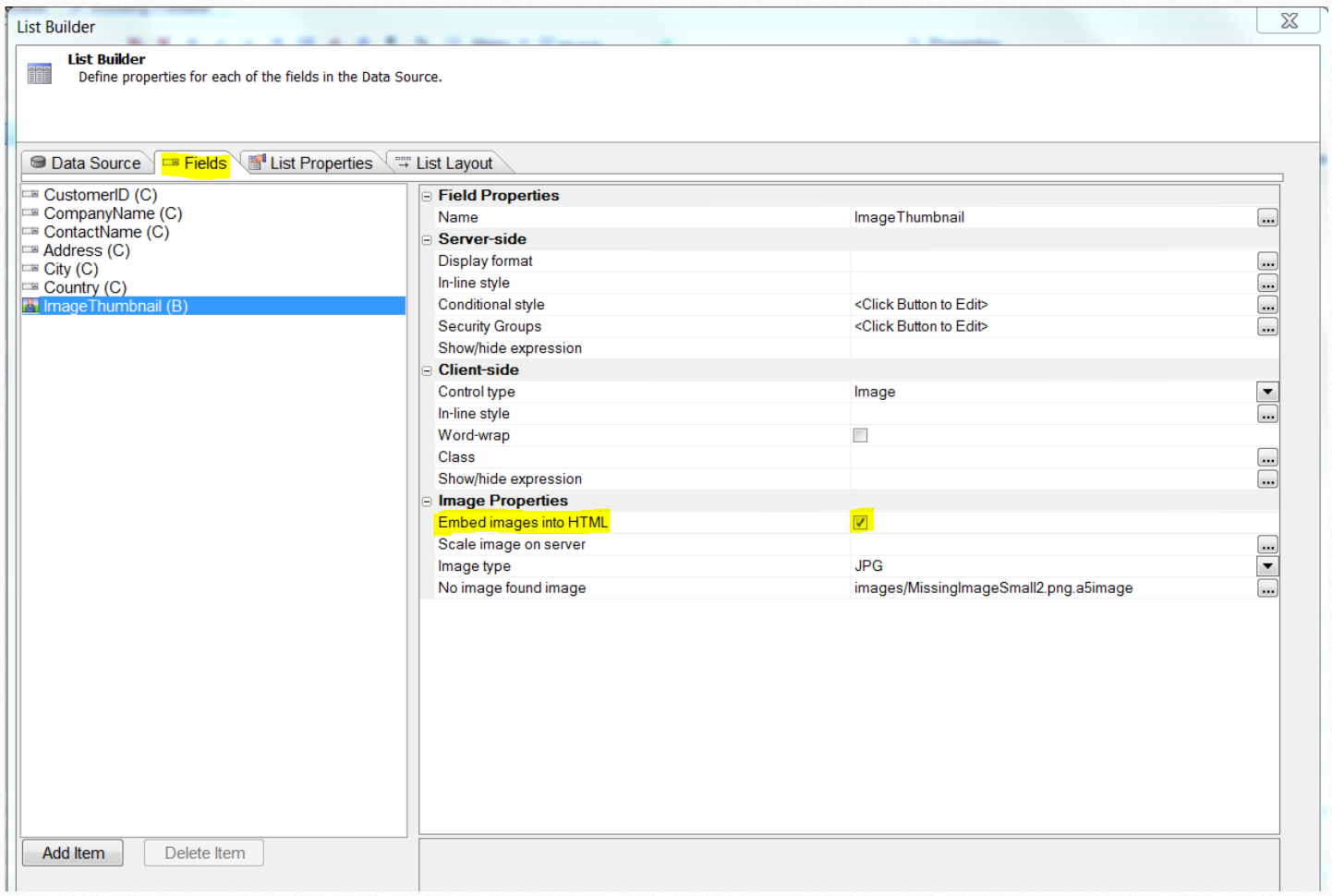
We'll look at each item listed above in detail.

1. Resolve missing image thumbnails

When you run the demo app on your mobile device, you will notice the thumbnail images are not being displayed. This occurs because the images were not embedded within the list control when the app was packaged up for PhoneGap Build.

The original component makes an Ajax Callback to update these images when initially loaded. We could do that from within the PhoneGap app however, that will slow down the app and it's possible that the network may not be available. It is best to try to provide the initial data and populate the UI from the list control. This makes loading the app quite fast and if the network is not available, at least the UI is complete.

To embed the thumbnail images in the list control, ***open the component*** and ***select the CUSTOMERLIST control***. ***Click on List Properties*** and ***select the ImageThumbnail*** in the Fields tab. ***Check Embed images into HTML.***



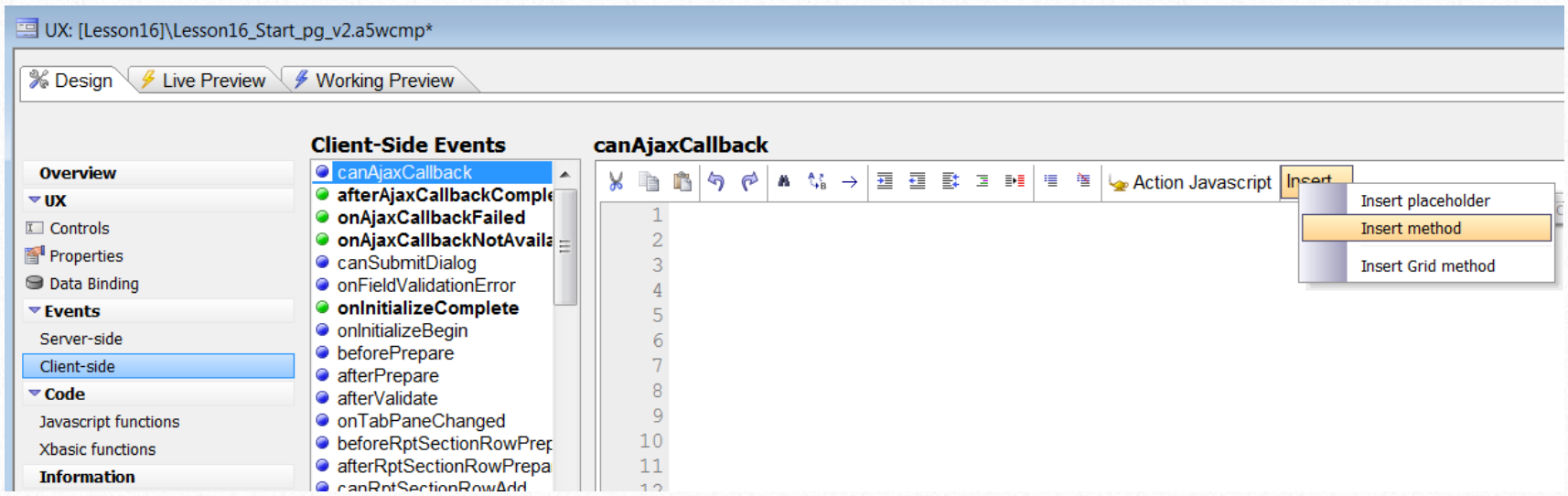
Now the thumbnail images will be stored in the list control as Base64 images, when the app is packaged up with the PhoneGap App Builder. This increases the app code size, but the true impact is small because the app is loaded on the mobile device. It will load quite fast.

2. Add Wait Modal Dialog On Ajax Callbacks

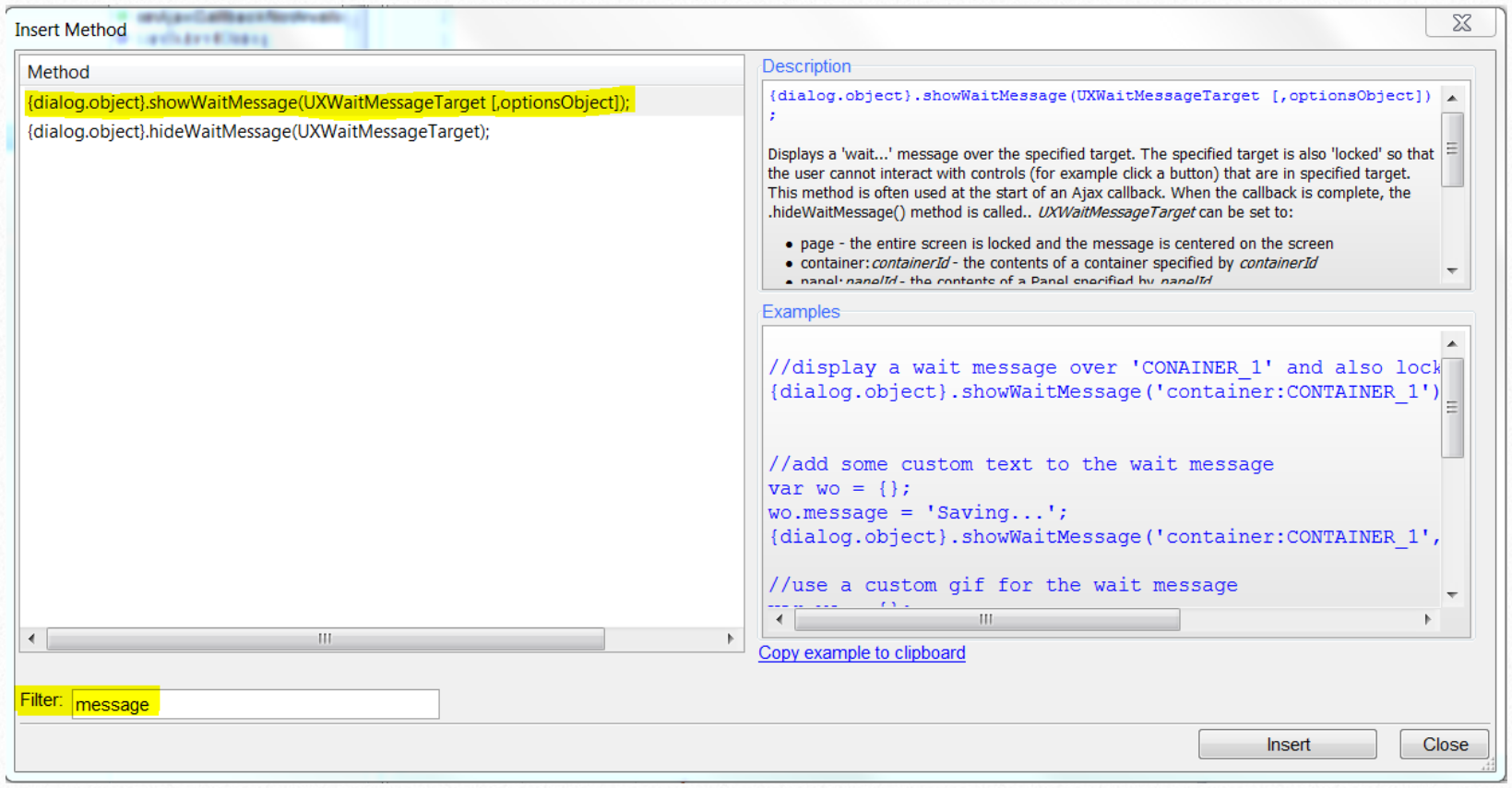
It is normally best to let the user know when the app is making an Ajax Callback because they are asynchronous. The Ajax request is made and in our case, the app waits for a response. If the network or server is not available, a timeout occurs.

To display a message, we will use the showWaitMessage method of the UX component. ***From the Client side events, choose the canAjaxCallback event***. This event always fires before an Ajax callback. The placeholder {dialog.object} is used to reference the current UX component name.

An easy way to explore the methods available for the {dialog.object} is to use the Insert method button in the JavaScript editor tab.



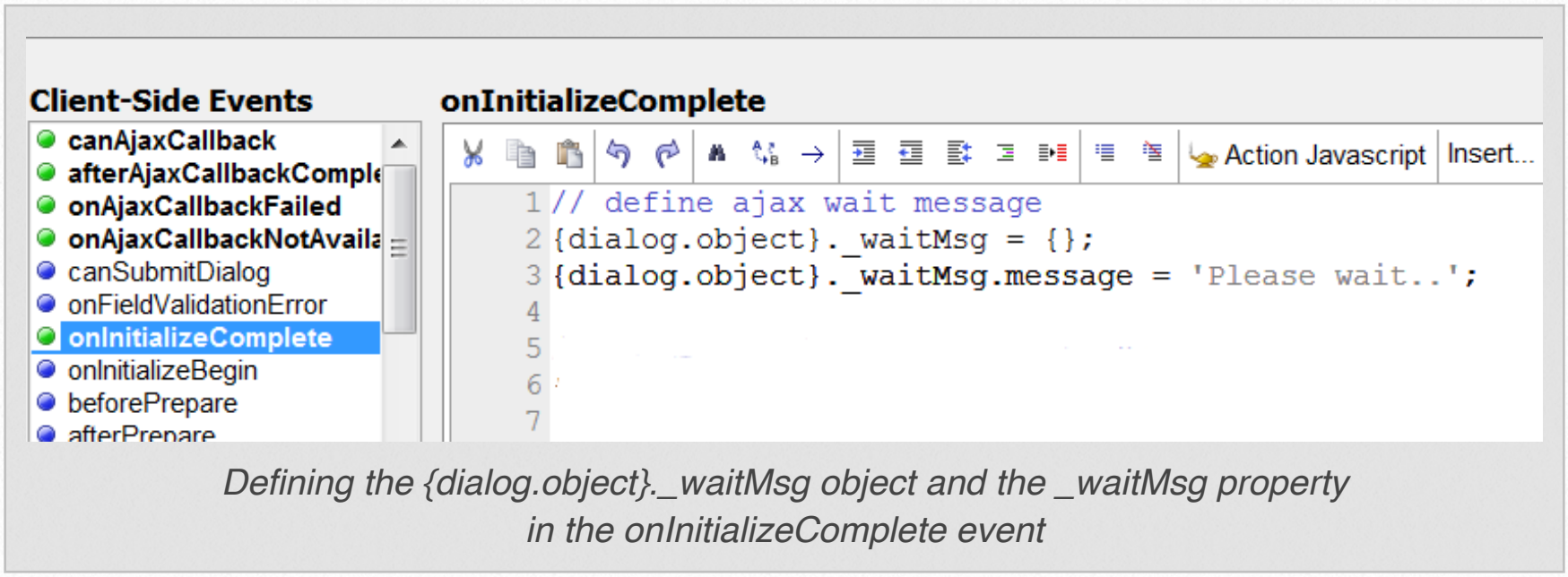
Set the Filter to message and take a look at the help and example code for the showWaitMessage method.



I am going to use the ‘page’ option to display the message centered on the existing page. The second parameter is an object. Rather than define the object every time this event fires (which is inefficient), I will create the object once and attach it to the {dialog.object}. This is done using onInitializeComplete event. Then I can use the object at any time. The code is efficient and I have not polluted the global name space.

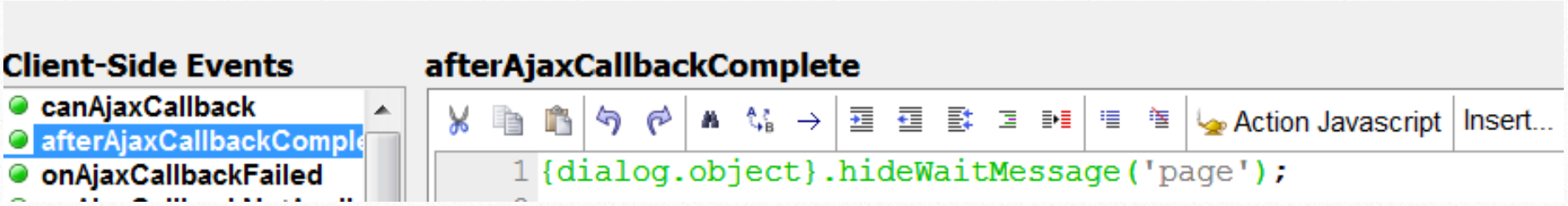
```
{dialog.object}.showWaitMessage('page',{dialog.object}._waitMsg);
```

To hide the wait message when the callback is complete, the hideWaitMessage method of the UX component is used. This method should be called from within the afterAjaxCallbackComplete event and from within the

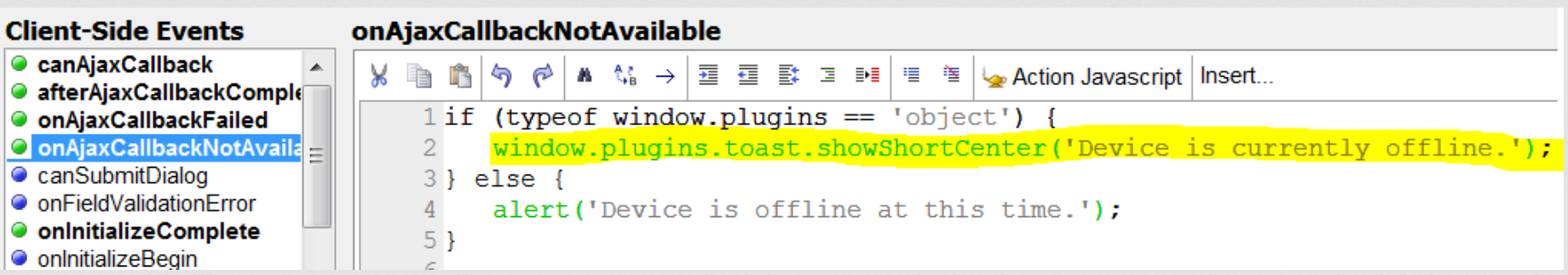


Defining the {dialog.object}._waitMsg object and the _waitMsg property in the onInitializeComplete event

onAjaxCallbackFailed event to cover all possible outcomes.



Should the device be offline, it's impossible to make an Ajax callback. The UX component is able to track the online state. The onAjaxCallbackNotAvailable event will fire if an Ajax callback is attempted and the device is offline. In this case, we want to inform the user that the device is offline. I'm using a PhoneGap third party plugin called Toast to display a message that appears for a moment and then fades out. This requires no interaction from the user. If the plugin is missing, a JavaScript alert is used.

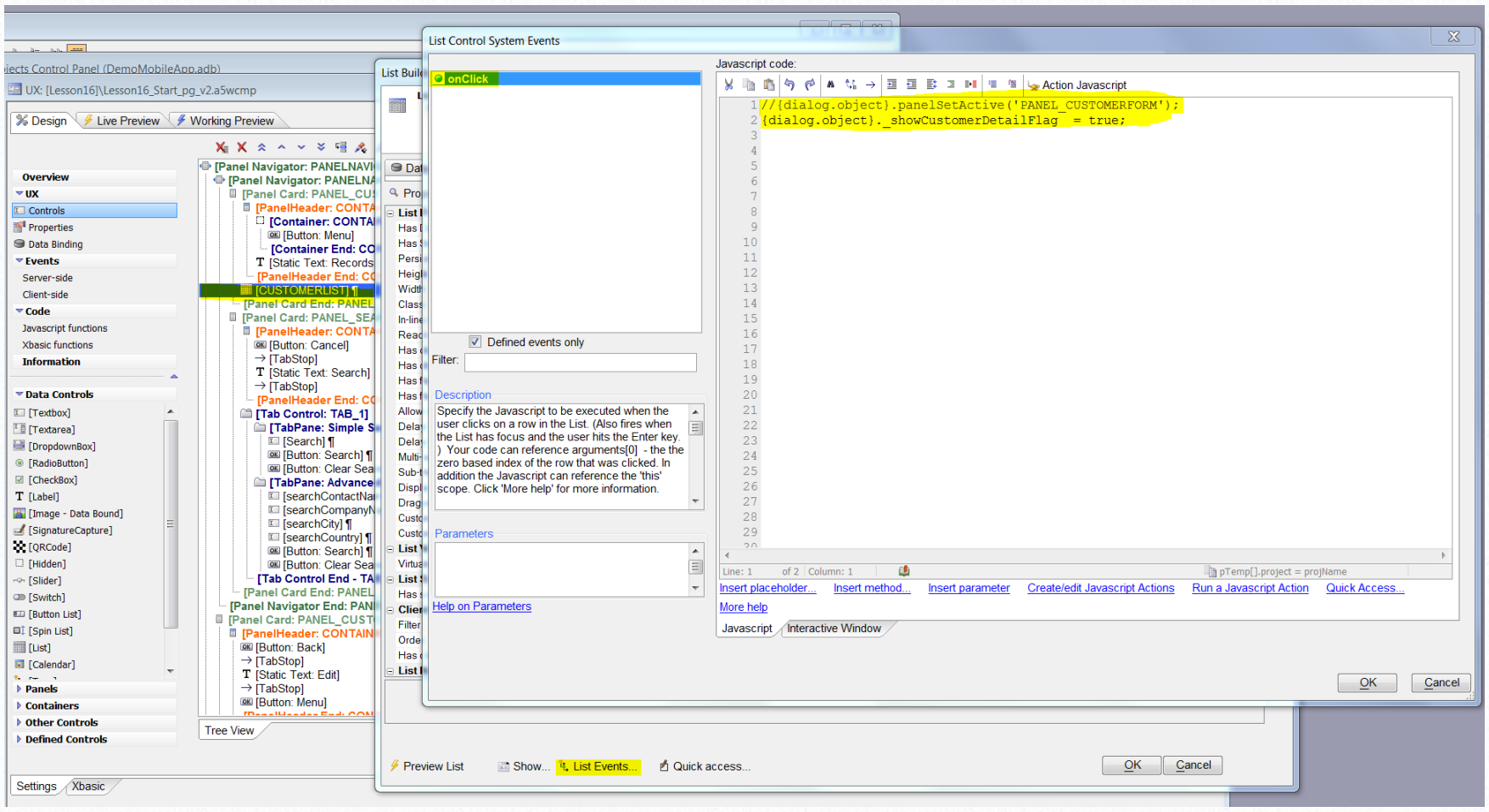


*Using the PhoneGap **Toast plugin** to display a message that fades out over time*

3. Transition to detail panel after Ajax callback is completed and all data loaded

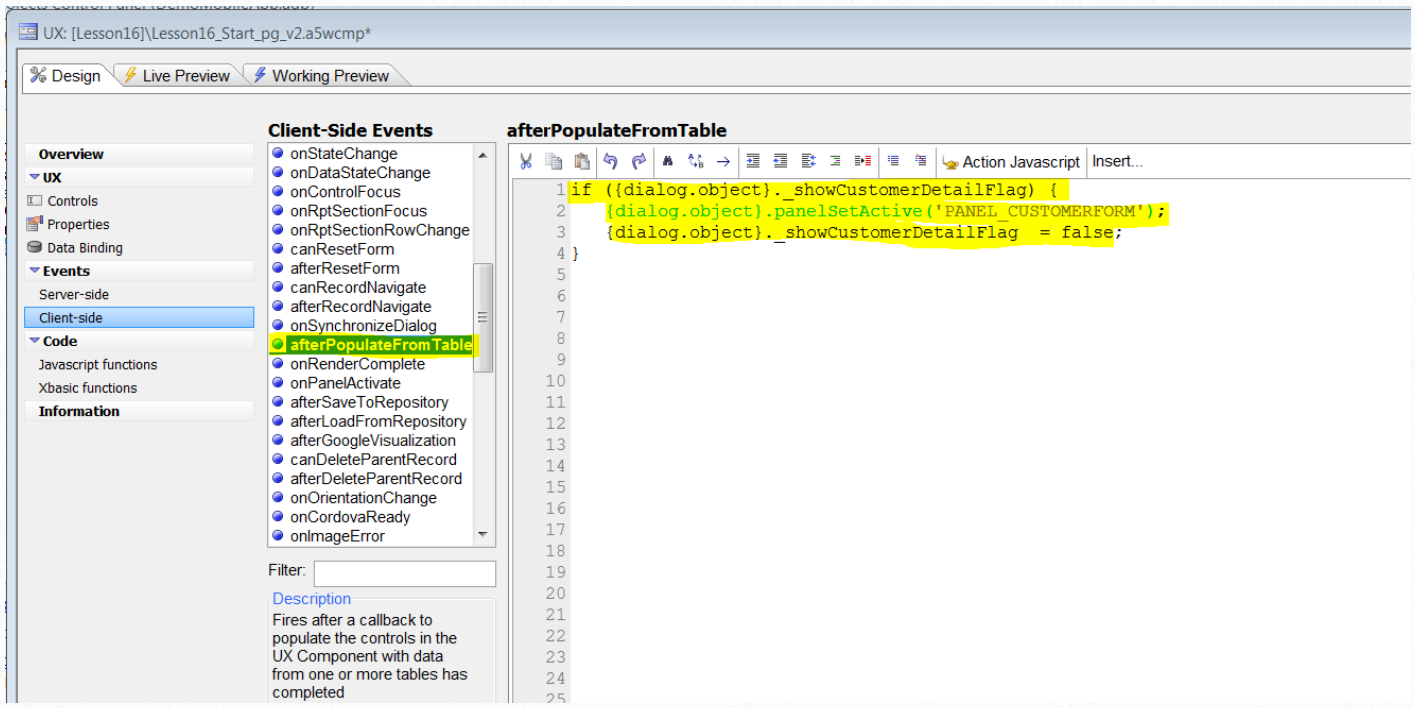
When a row is tapped in the CUSTOMERLIST control, a callback is made to populate the data bound controls contained within the PANEL_CUSTOMERFORM which represents a detail view. Because the callback is asynchronous, it is best practice to wait for the callback to complete prior to transitioning to the detail view panel card. This prevents the previous image from appearing when the detail view panel card is transitioned into view.

Rather than initiate the transition to the PANEL_CUTOMERFORM on the click event of the CUSTOMERLIST list control (that code is commented out in the image below), I'm setting a JavaScript variable attached to the {dialog.object} placeholder (which is the UX component) called _showCustomerDetailFlag to true.



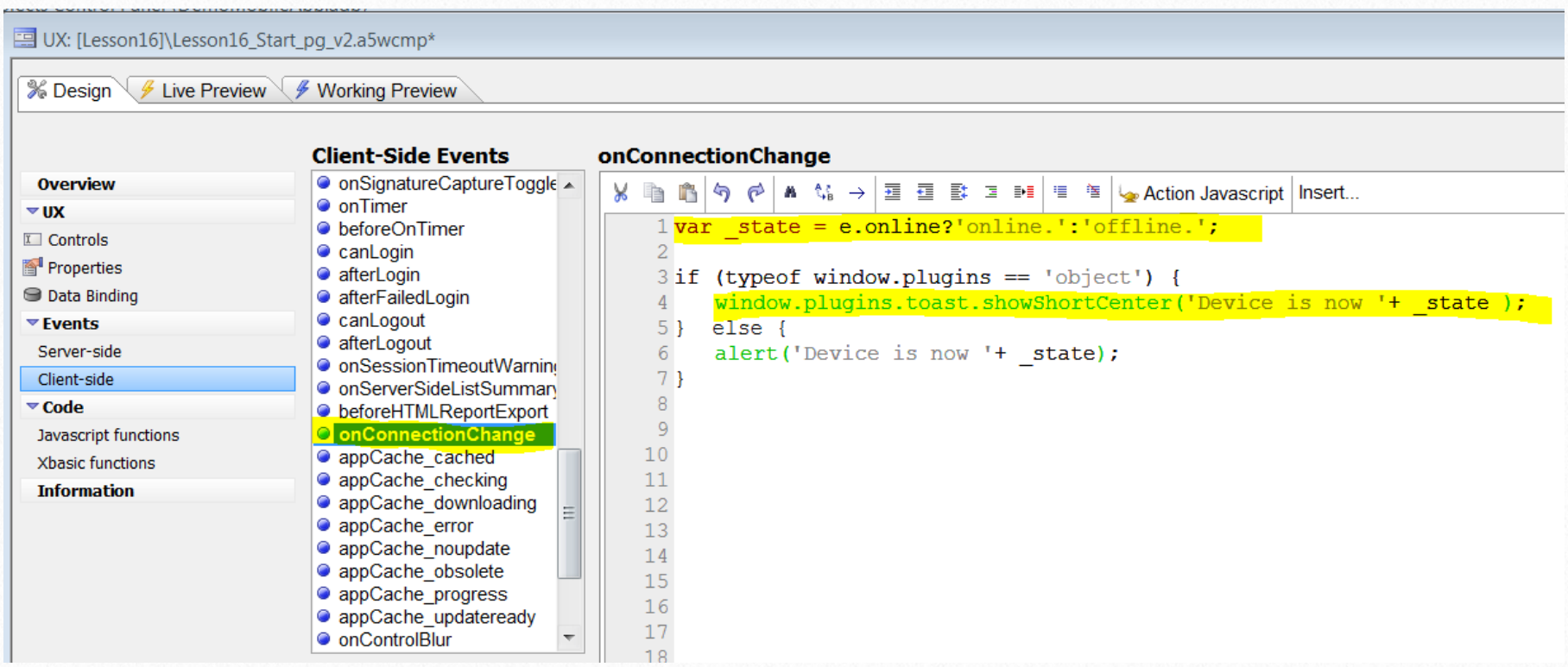
Next, we are going to now use the client side afterPopulateFromTable event (which fires after a callback to populate the data bound controls of a UX component) to transition the PANEL_CUSTOMERFORM into view, only if the {dialog.object)._showCustomerDetailFlag is true. This flag is required because this event will fire after any callback that may populate populate data bound controls, such as the CUSTOMERLIST.

Notice the flag is set to false once the panel is transitioned into view.



4. Displaying a modal alert when a change is detected in the online state

When a mobile app is dependent upon remote server access, it is a good idea to let the user know when the online/offline state of the device changes.



The client side onConnectionChange event fires every time the network connection state changes.

In the example code above, I am setting a variable called _state with the string value of “online” or “offline” based on the boolean value of the e.online parameter that is passed when the event fires. If the PhoneGap third party plugin Toast is available, it is used to display a modal message box that indicates the online state. The message box will fade out after a short period of time and requires no user intervention.. If the Toast plugin is not available, a JavaScript alert is used.

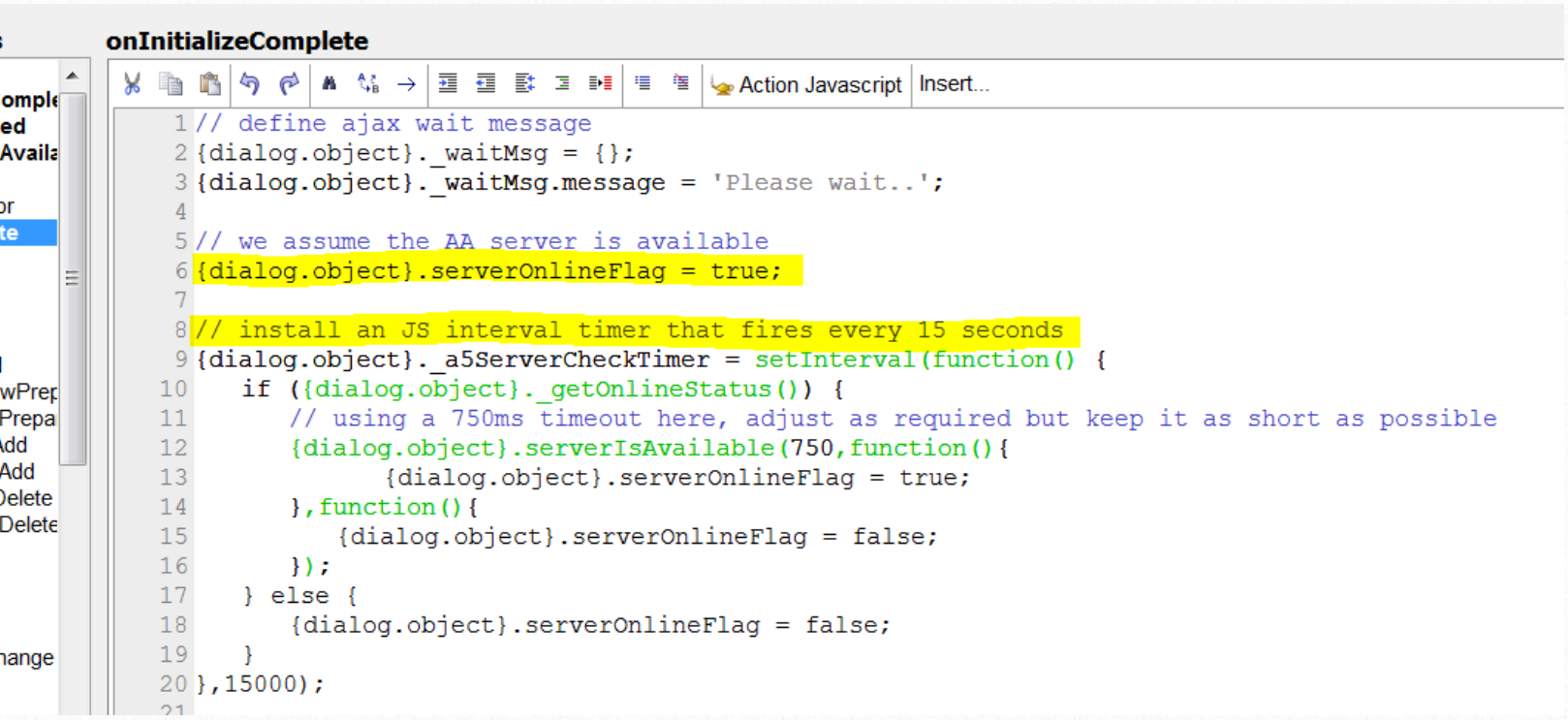
5. Verify the device is online and the Alpha Server is available prior to making an Ajax callback

In an effort to make the app as fast and responsive as possible, we can verify that the device is online and that the Alpha Server is available before initiating any Ajax callbacks. I consider this best practice. It's a bit tricky to do because a timer is required to check the availability of the server on a periodic basis. It is also important that the server overhead to process this server heartbeat check is absolutely minimal, otherwise the server could become sluggish, wasting time and resources processing simple heartbeat requests for possibly thousands of clients.

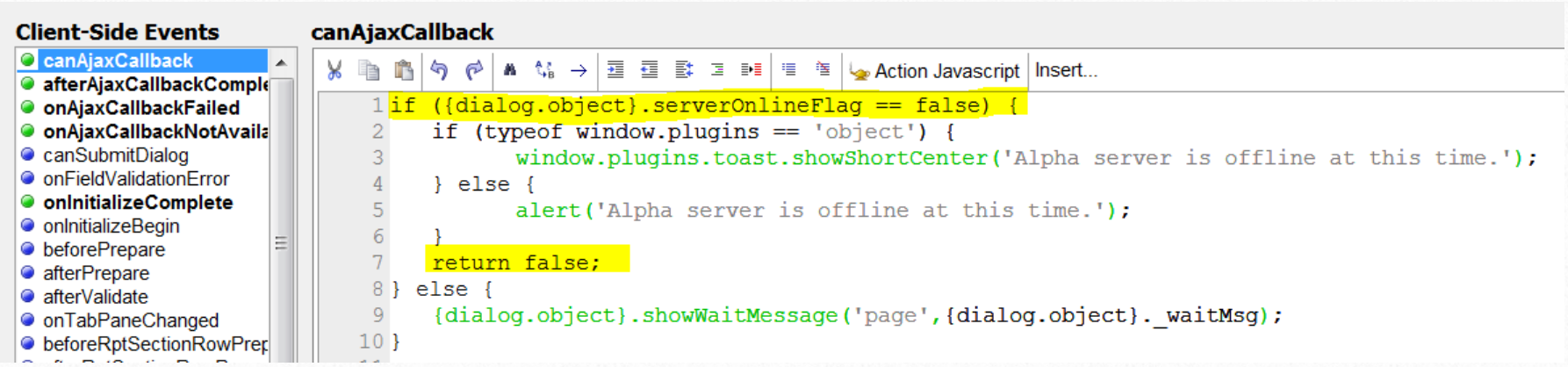
To monitor the state of the server, a global flag is required. I am using a {dialog.object}.serverOnlineFlag in the code below. This is set to an initial state of true, assuming the server is available.

Next I am creating a JavaScript interval timer that will fire every 15 seconds. You can adjust this value as required but I do not recommend anything less than 15 seconds. The timer initially calls the {dialog.object}.getOnlineStatus method to determine if the device is online. If the device is online, then the {dialog.object}.serverIsAvailable method is called. This method takes a timeout value (I am using 750ms) and a success and fail function, which I am defining as anonymous functions. The success and fail functions toggle the state of {dialog.object}.serverOnLineFlag. If the device is not online, the serverOnlineFlag defaults to false.

The serverIsAvailable method makes a very simple request to determine if the Alpha Server is available. Every effort has been made to make sure this request has minimal impact on the server.



Next we need to modify the canAjaxCallback client side event.



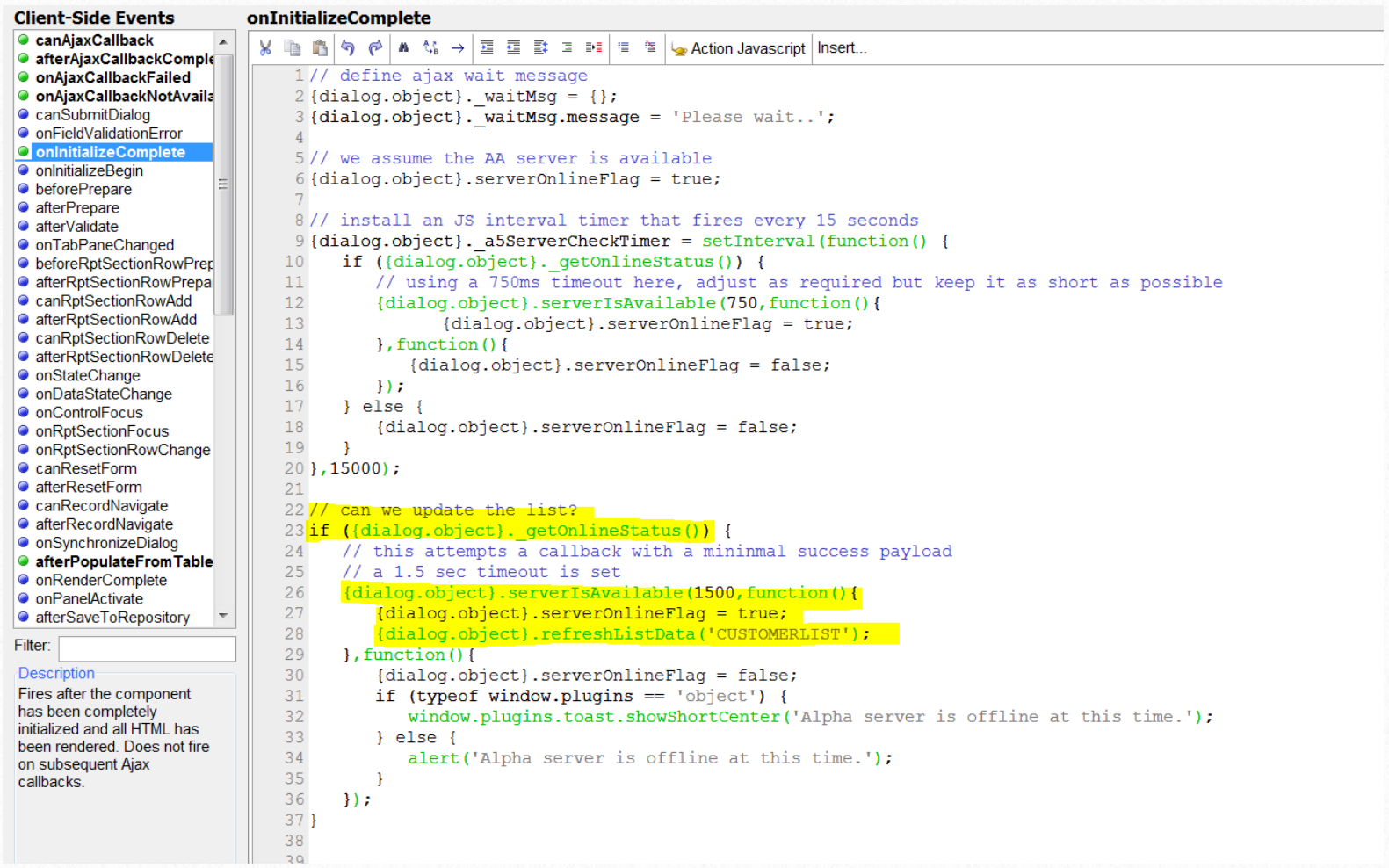
This event fires prior to the initiation of any Ajax callback made by the UX component. Since we are now setting the {dialog.object}.serverOnlineFlag in the interval {dialog.object}._a5ServerCheckTimer previously defined in the onInitializeComplete event, we can monitor its state to allow an Ajax callback to occur. If the flag is set to false, a message is displayed indicating that the server is offline and the callback is not initiated.

If the server is online the callback will fire and a wait message is displayed.

6. Updating the Customer List when the app is loaded if the device is online and the Alpha Server is available.

When the app initially loads on the device, the Customer List data that is displayed is the data that was embedded within the component when the PhoneGap app was created within the PhoneGap App Builder.

It is a good idea to refresh this data as long as the device is online and the server is available. To do so we will add some more code to the onInitializeComplete event. Remember, this event only fires once, after the UX component initialization is complete.



Here we are initially checking the device online status with the {dialog.object}.getOnlineStatus method. If the device is online, the {dialog.object}.serverIsAvailable method is called, with a timeout value of 1.5 seconds. I set the timeout to be a bit longer here than in the previously installed interval timer because it's important to try to initially update the Customer list data and this timeout is masked by the app time associated with the app startup.

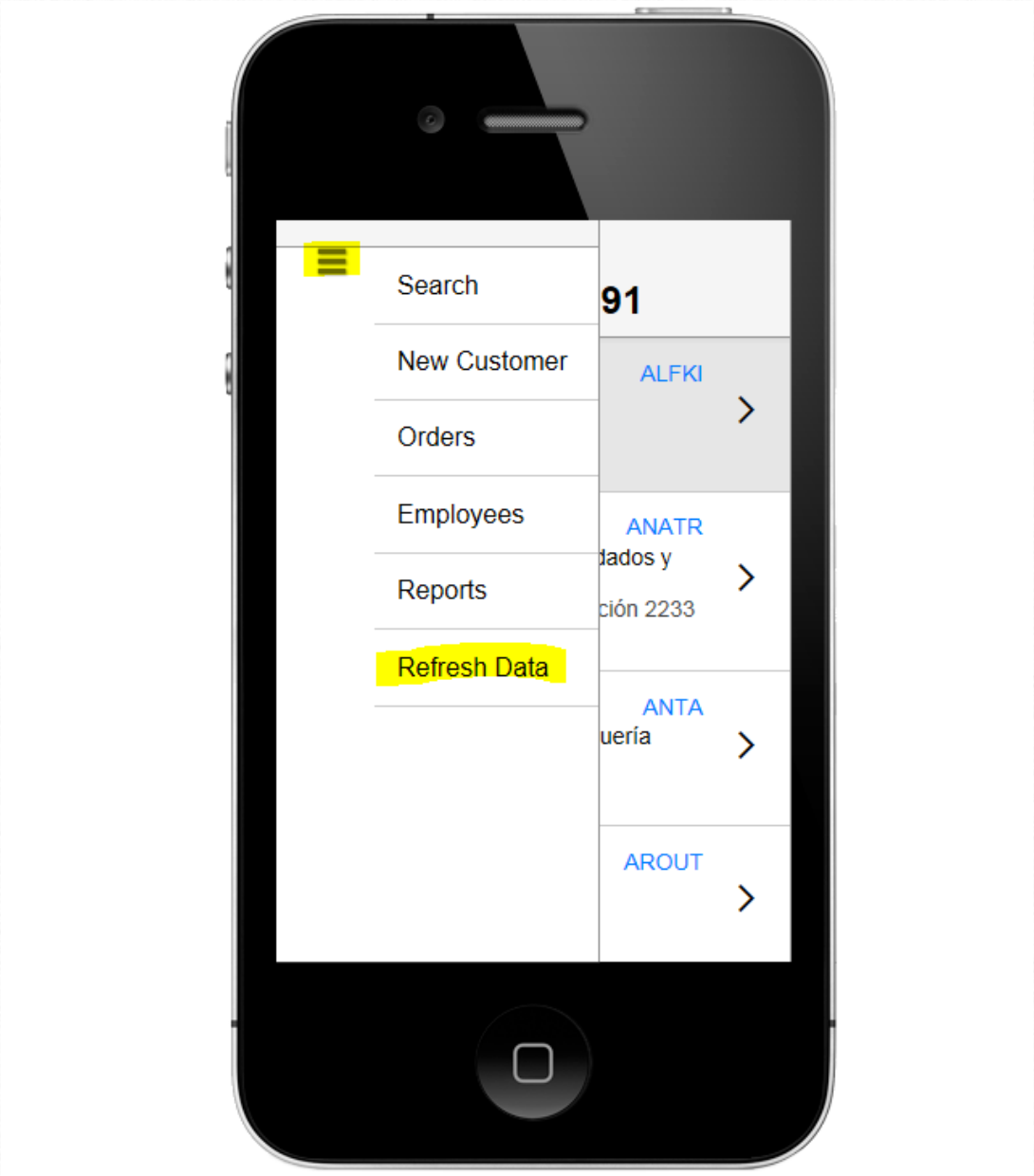
The success and fail functions are once again defined as anonymous functions.

The success function sets the {dialog.object}.serverOnlineFlag to true and calls the {dialog.object).refreshListData method to refresh the CUSTOMERLIST.

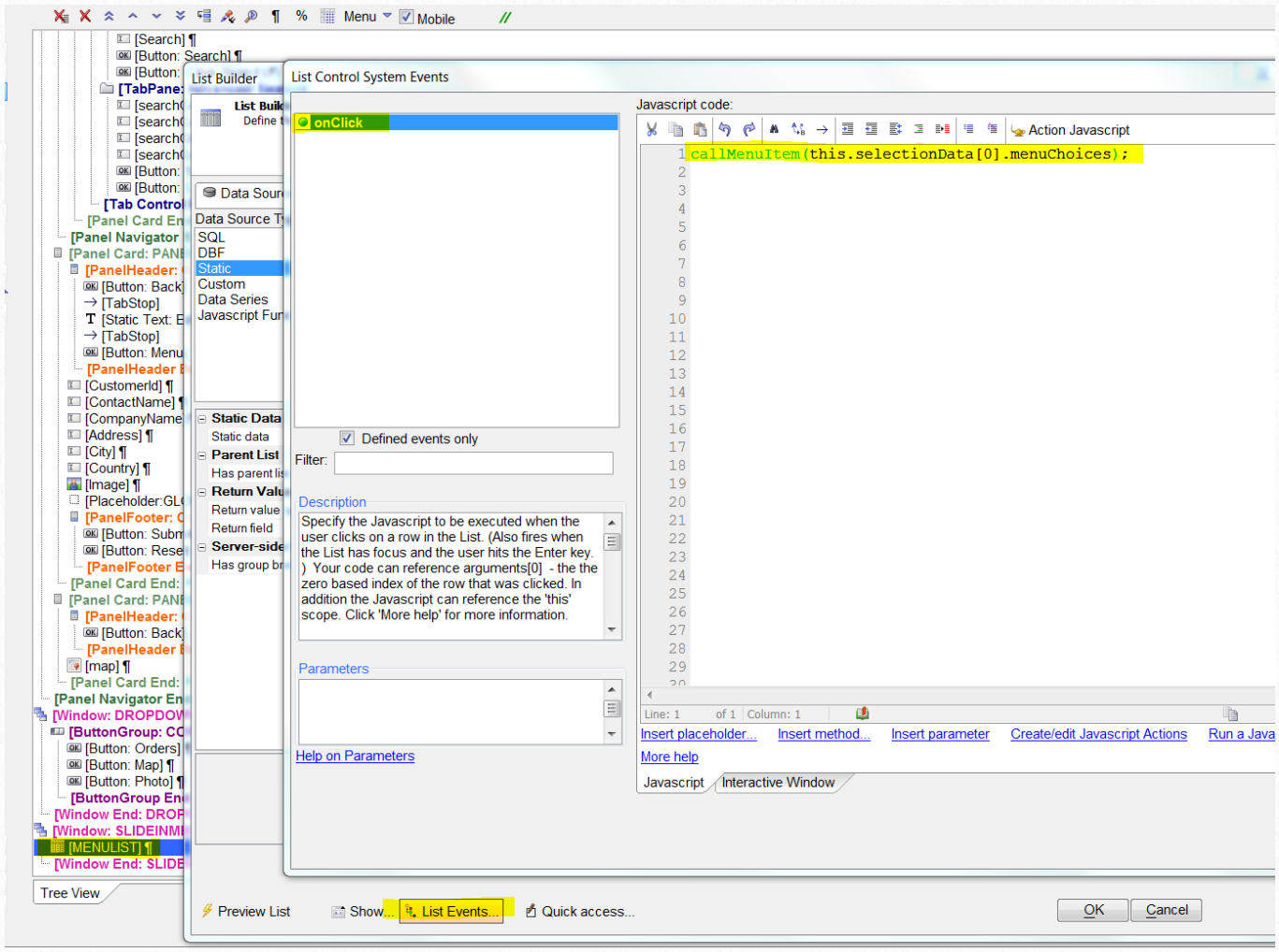
The fail function sets the {dialog.object}.serverOnlineFlag to false and displays a message indicating the server is offline.

7. Refresh the list from a menu button

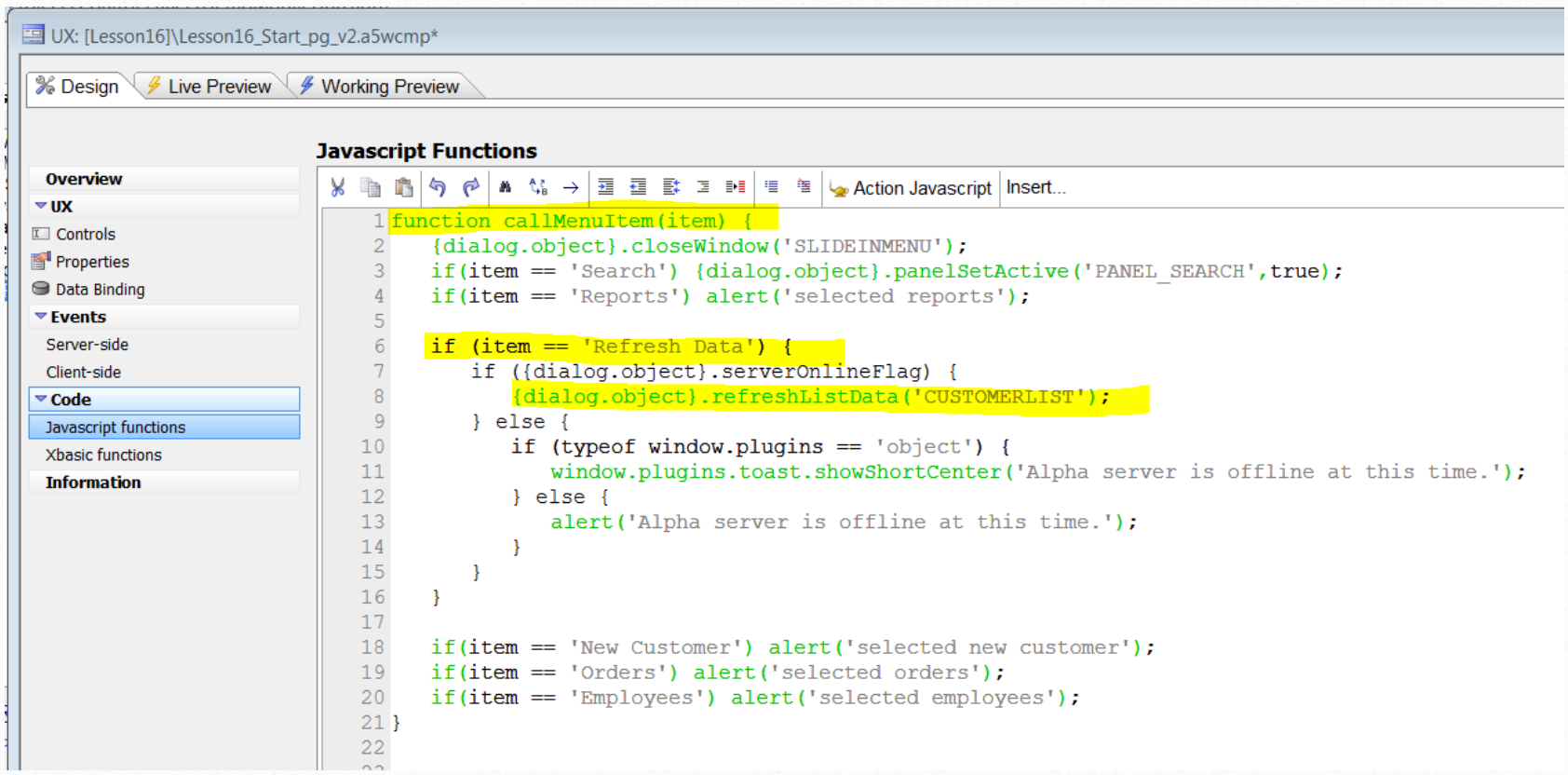
The goal is to refresh the CUSTOMERLIST from a menu button.



The MENULIST control contains the list properties that we need to examine.



The onclick event of the MENULIST control calls a function called callMenuItem and passes in the selected item as an argument. We'll need to modify this function to refresh the CUSTOMERLIST when refresh is selected.



The code should look familiar. If the {dialog.object}.serverOnlineFlag is set then the CUSTOMERLIST is refreshed by the {dialog.object}.refreshListData method. If the server is offline, a message is displayed by either the Toast PhoneGap plugin (fades in then fades out with no user intervention) or by a JavaScript alert if the Toast plugin has not been installed.

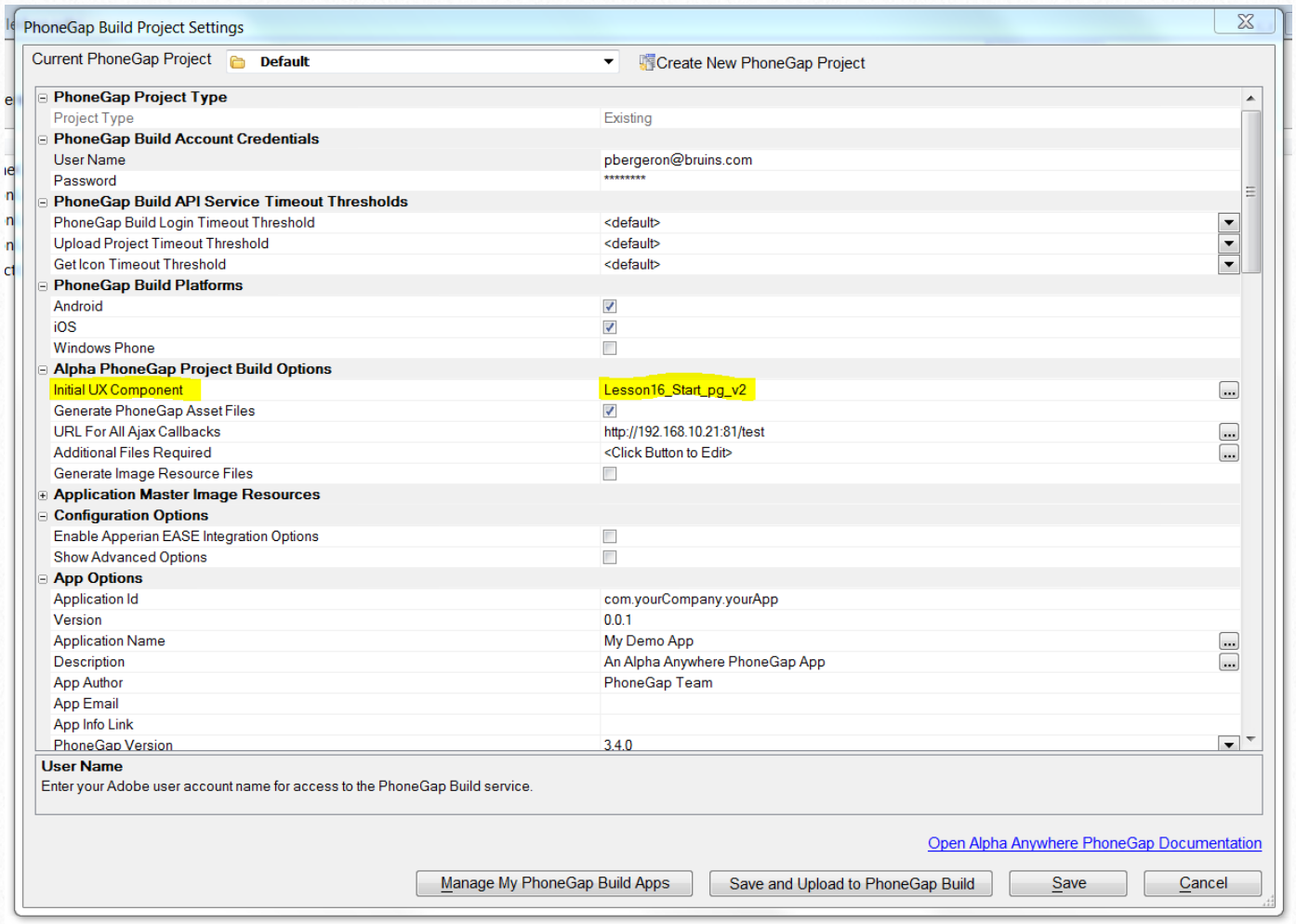
Modify the previous PhoneGap project to use this component

Rather than start a new PhoneGap project, you can always edit an existing project and use a new component as the basis for the PhoneGap app.

This first step is to publish the component to your development server. This process was covered in the previous chapter. This time ***publish the component named Lesson16_Start_pg_v2.a5wcmp to your development server.*** You are required to publish this component to your server because the PhoneGap app will be making callbacks to your server.

Next, ***open the PhoneGap App Builder Genie from the Web Projects Control Panel.***

Modify the name of the initial component to Lesson16_Start_pg_v2 and ***save and upload to PhoneGap Build.***



You should now be able to install the app on your device (scan in the QRCode) and test out all of the the new functionality.

Reference Video

See: Creating a PhoneGap App From The Demo Mobile Component Part 2

10

Apperian EASE Integration

Mobile Application Deployment and Management

See Apperian EASE Integration Overview Video

Apperian offers mobile app deployment management through the Apperian EASE Platform.

The Apperian EASE platform can secure, manage and deploy enterprise mobile apps for all mobile platforms.

Apperian EASE provides:

- An enterprise app store for distributing public, custom and web apps as well as configuration profiles
- Rock-solid mobile app security using enterprise credentials
- Diverse app wrapping policies without requiring code modifications or SDKs
- One-time app distribution option for getting a single app deployed quickly
- Seamless mobile enterprise integration with development platforms and identity providers Flexible content management solution for distributing virtually any type of file content for mobile users

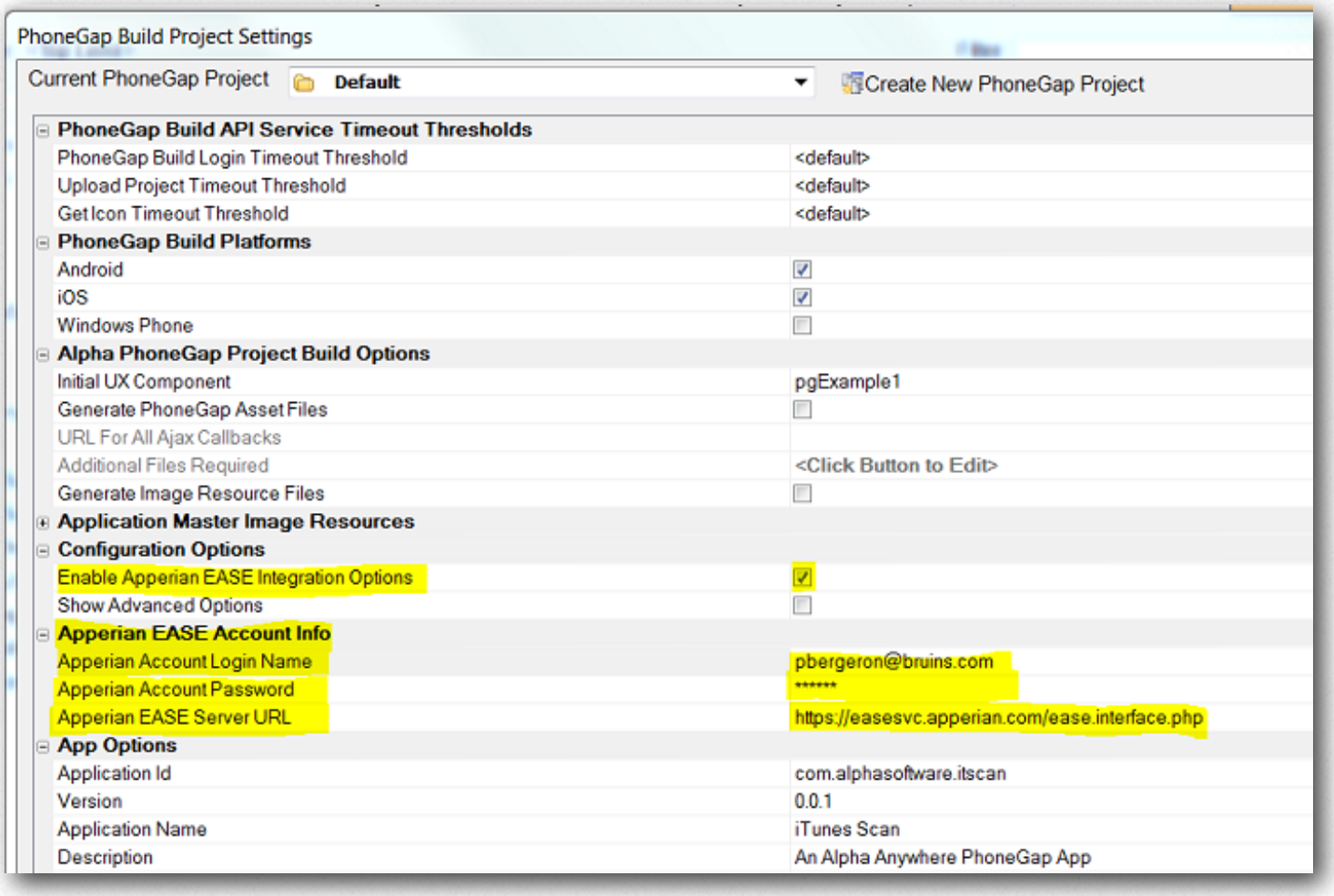
Alpha Anywhere now has direct support for Apperian EASE allowing you to upload new apps or publish app updates directly to the Apperian EASE database directly from within the PhoneGap App Builder.

To publish apps to the Apperian Ease platform, you will need to setup an Apperian account.

See the Apperian Web Site for more information.

Enabling the Apperian Features Within The PhoneGap App Builder

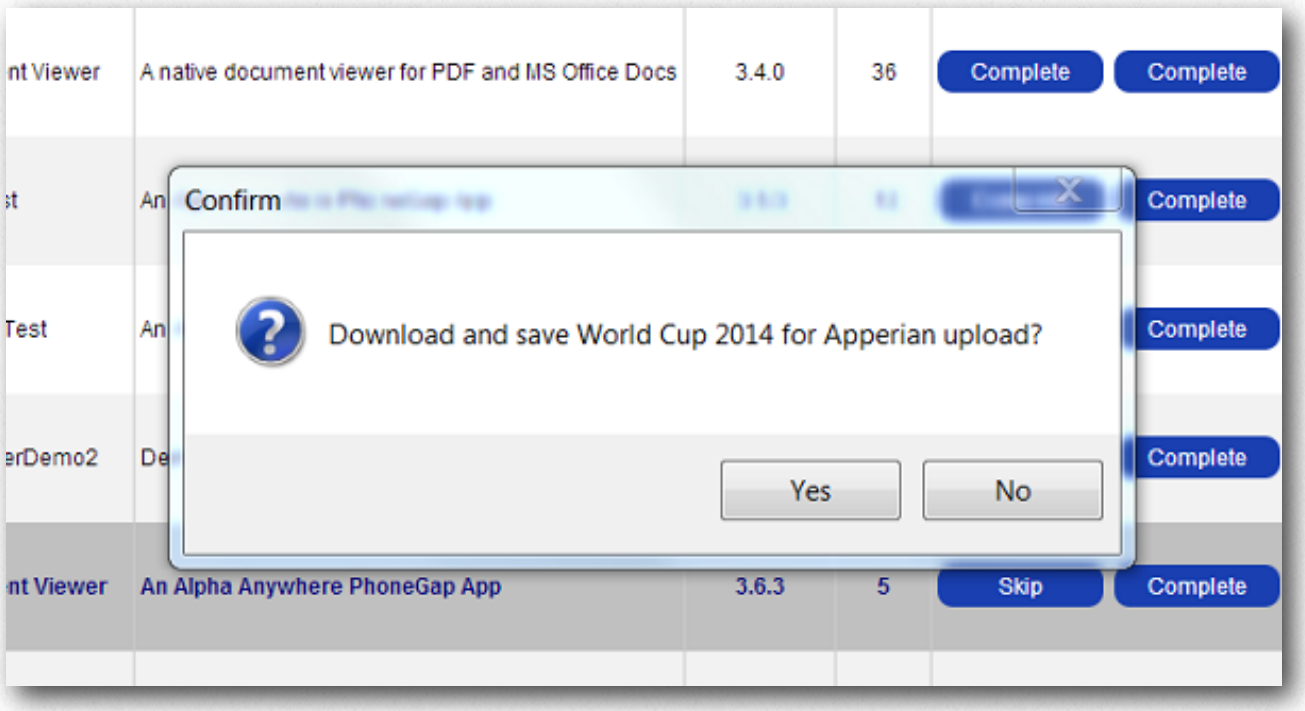
To enable the Apperian features, from within the PhoneGap App Builder, check the Enable Apperian EASE Integration Options from within the PhoneGap App Builder.



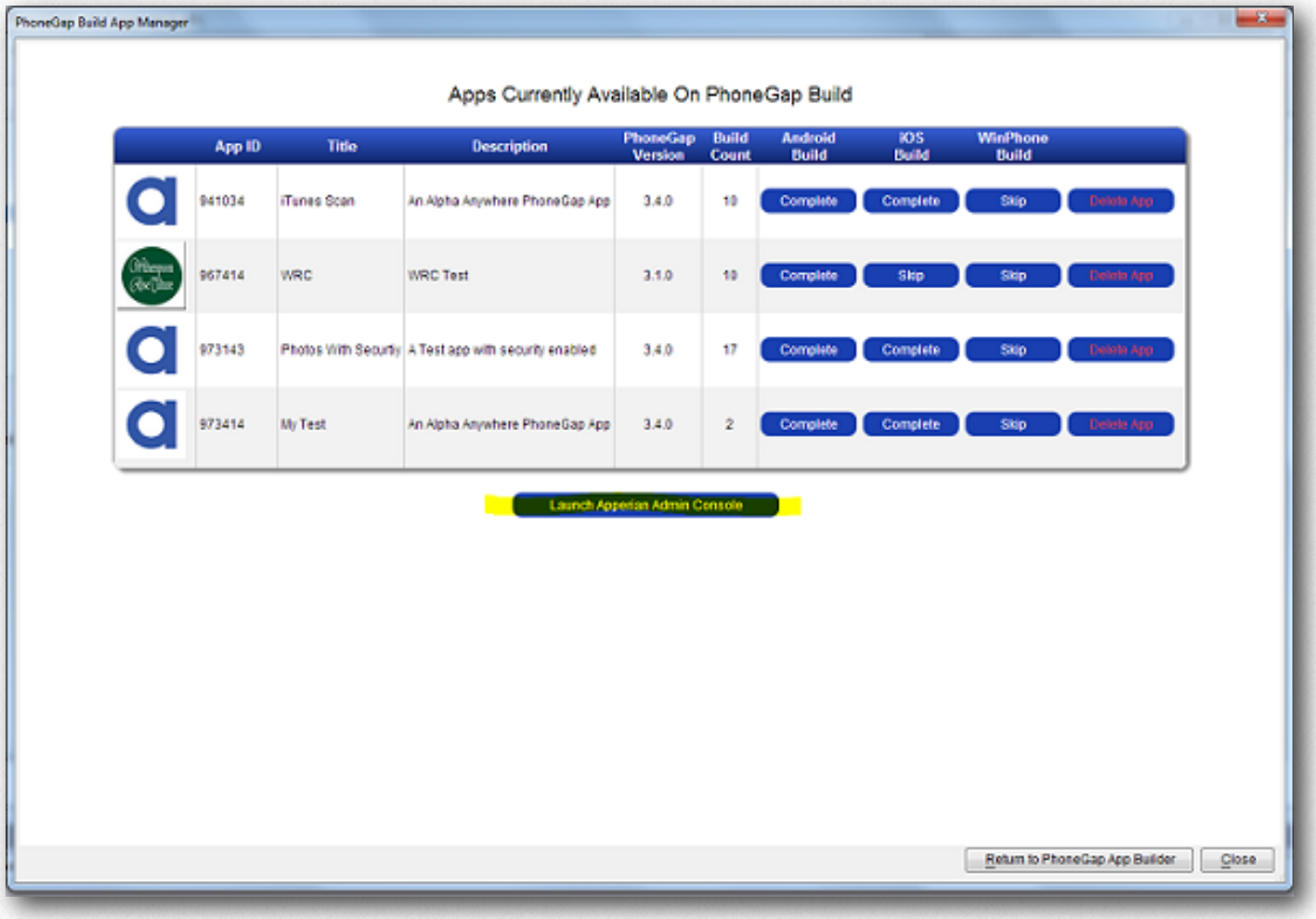
Next, enter your Apperian account credentials and make sure to specify the correct Apperian server. Different servers are used for US and non US accounts.

Once the Apperian features have been enabled, a number of optional items are enabled within the PhoneGap App Builder.

- You will be able to save the App bundle for uploading to Apperian when you bring up the QRCode within the PhoneGap App Builder.



•A button will be displayed at the base of the PhoneGap Build App Manager table to launch the Apperian Admin Console.



The Apperian Admin Console will display all of the apps you may have previously published to the Apperian EASE platform.

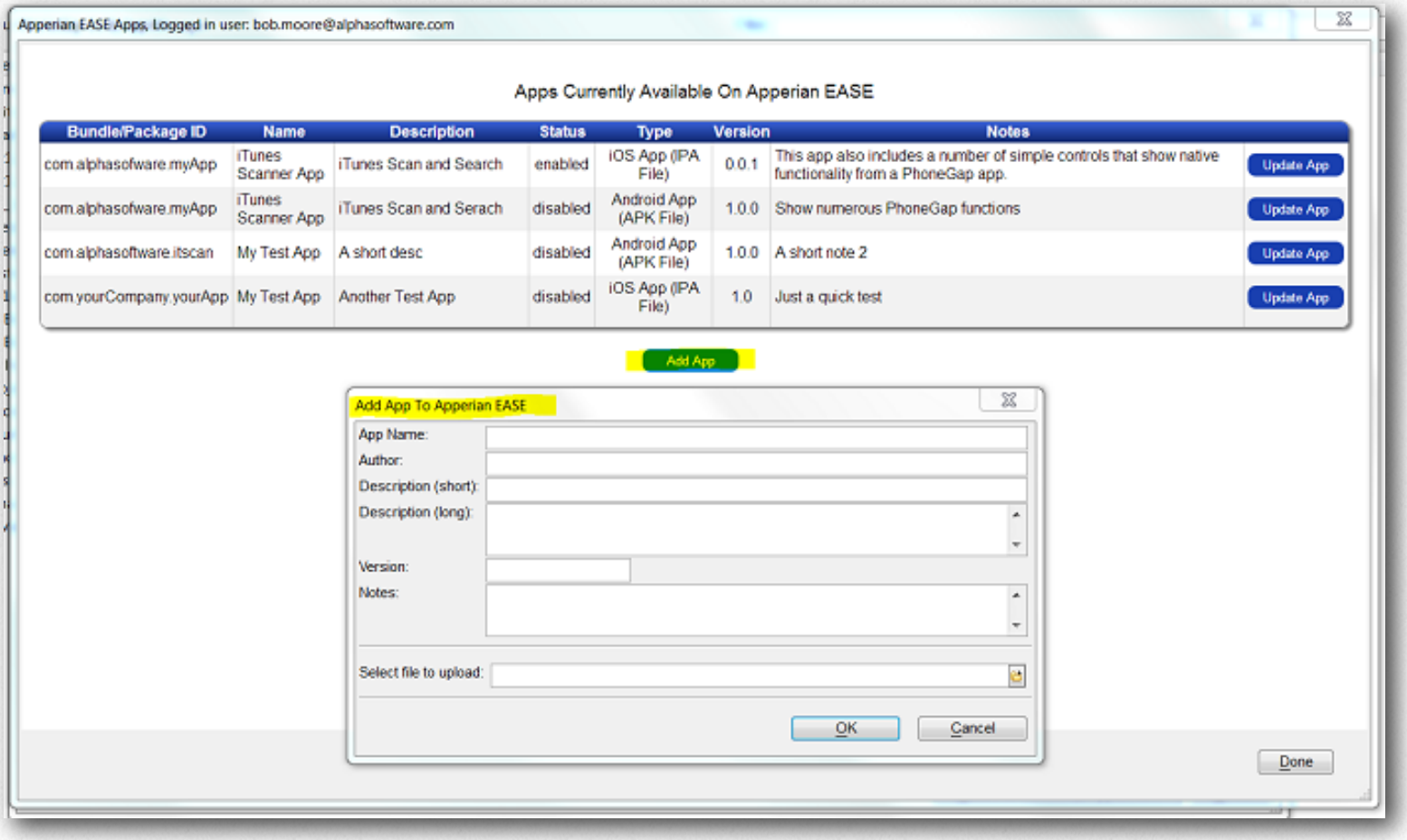
To Add A New App To Apperian

Make sure to download and save the app bundle for the target platform. The app bundle files will be automatically saved to the appDownloads folder of the PhoneGapProjects target project directory.

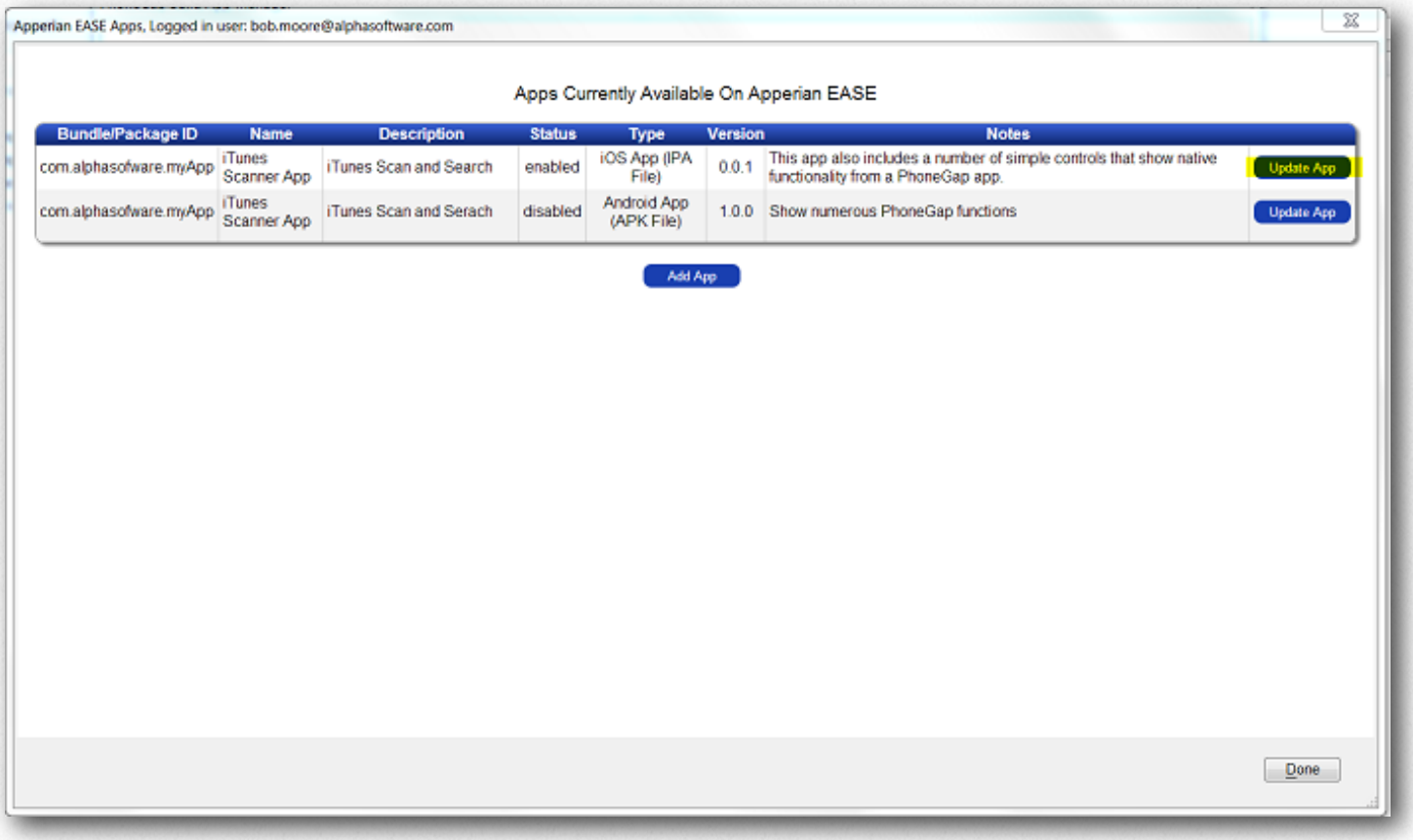
Next click the Add App button and fill out all of the required information for submission to Apperian.

All of the fields must has valid data and the version number should include a decimal, ie: 1.20, 0.20, 2.52, etc.

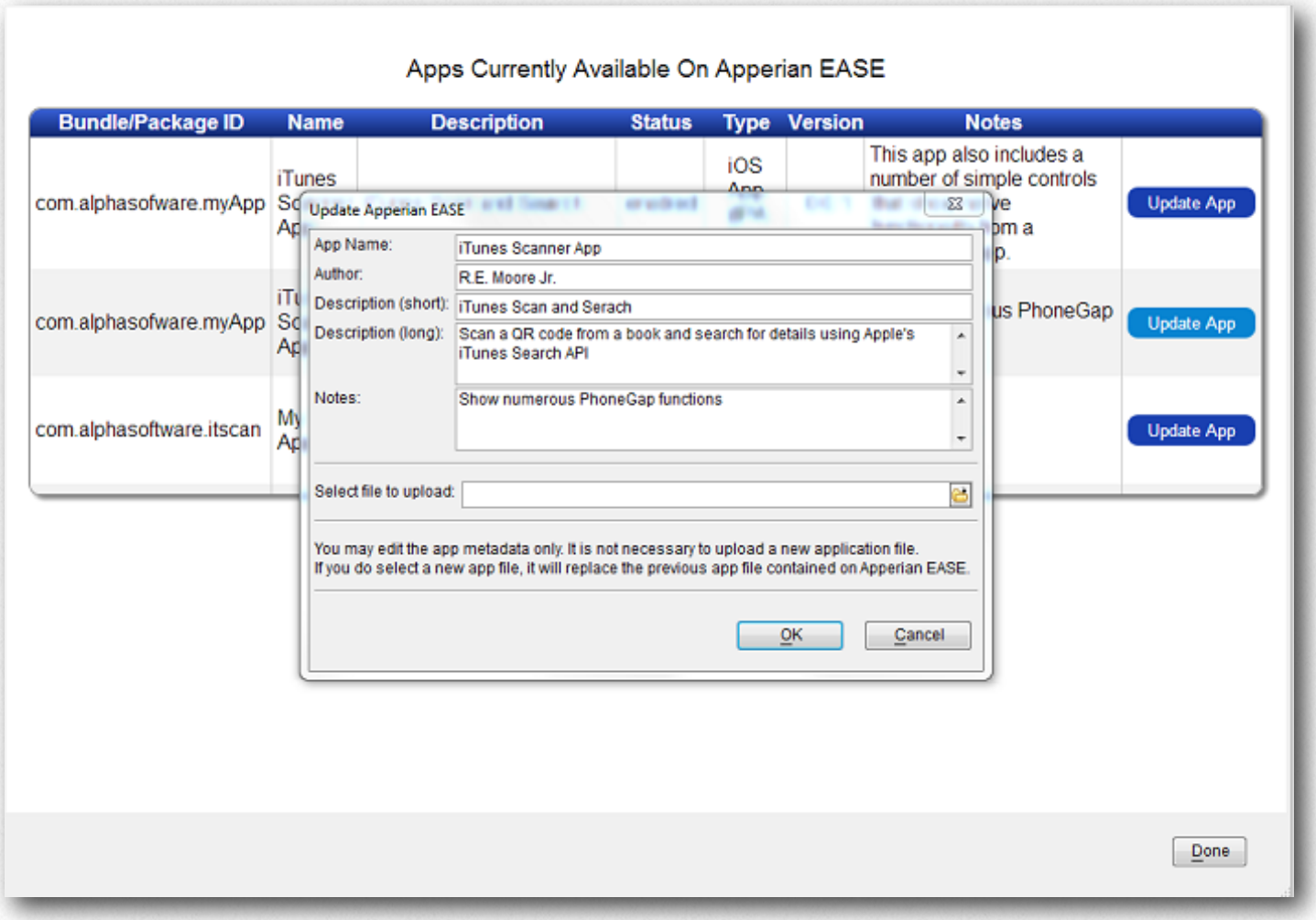
The app.bundle.id will be automatically retrieved from the app bundle.



To Update An Existing App



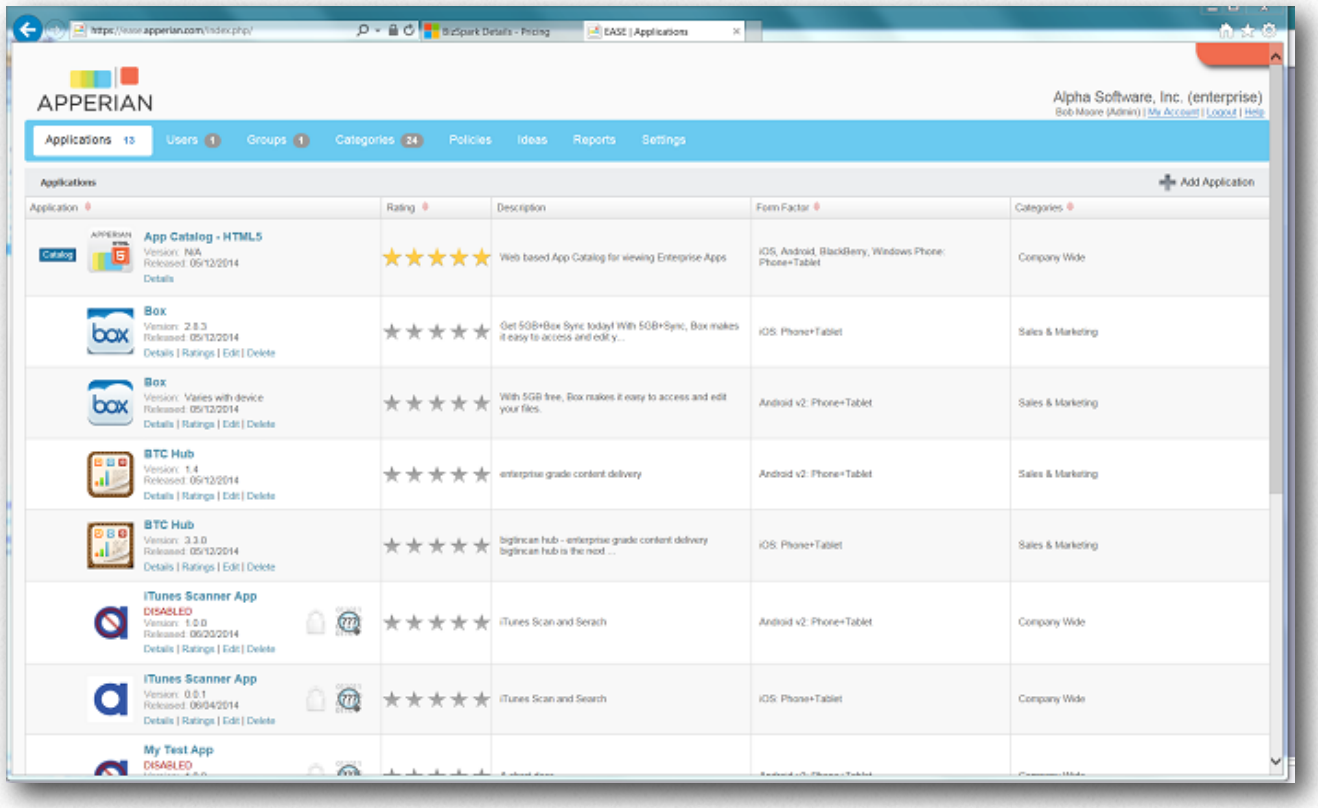
Click the update button.



You do not need to upload a new app bundle unless the files have changed. You can simply edit the information and save.

To Access The Apperian EASE Platform

Log in to your Apperian EASE account at Apperian.



Once you've logged in, you will see the files and data that have been uploaded from Alpha Anywhere. Further deployment options will be set within the Apperian EASE Management Console. Keep in mind that you can modify the app within Alpha Anywhere and update the app bundle and app meta data on Apperian at anytime.

See Apperian EASE Integration Overview Video

11

Updates And More Information

This document represents an ongoing effort to keep you informed about the latest information on PhoneGap and Alpha Anywhere integration.

Additional chapters, with code examples and additional sample components will be added to this document on a continuing basis to cover a wide variety of new features and use cases.

Keep an eye on the following for information updates.

Alpha Software Message Board: http://msgboard.alphasoftware.com

Alpha Software Blog: http://blog.alphasoftware.net

Alpha Software Website: http://www.alphasoftware.com

For more information on PhoneGap

PhoneGap Website: http://phonegap.com/

PhoneGap Blog: http://phonegap.com/blog/

PhoneGap Build Website: http://build.phonegap.com

For more information on Apperian

Apperian Website: http://www.apperian.com

12

Reference Video Links

I have completed a number of instructional videos that either are documented herein or they will be documented and added to this document as soon as time allows.

Adopting and Modifying the Demo Mobile Component for Use with PhoneGap

Adopting the Demo Mobile Component for use with PhoneGap, Part 1

Adopting the Demo Mobile Component for use with PhoneGap, Part 2

The 2014 World Cup Application, using RESTful Web Services

The 2014 World Cup Series Video Collection