Red Hat Mobile Application Platform 4.2
Mobile Developer Guide

For Red Hat Mobile Application Platform 4.2

Red Hat Mobile Application Platform 4.2 Mobile Developer Guide

For Red Hat Mobile Application Platform 4.2

Legal Notice

Abstract

This document provides guides for development of mobile client apps in Red Hat Mobile Application Platform 4.2.

Table of Contents

CHAPTER 1. ANDROID

1.1. DEPLOYING AN APP ON ANDROID

There are many ways to get an APK file installed on an Android device or emulator. Some of them are detailed here. If you don't have an Android device, the emulator can be installed as part of the Android SDK.

1.1.1. Browser Method

If you are using the FHC command line tool to generate your binary, the output will contain a URL value for example,

```
http://[your-studio-
domain].redhatmobile.com/digman/A/B/C/android~2.2~50~MyApp.apk
```

This is the download URL of the binary. By navigating to this URL on the device browser, the binary can be downloaded and installed.

> **Tip**
>
> Using a URL shortener like bit.ly makes entering the download URL a lot easier

1.1.2. Dropbox Method

If you have a Dropbox account, there is a Dropbox app available in the Google Play Store. Placing the APK file in your Dropbox folder (on your PC/Mac) will make the file available through the Dropbox app, allowing you to open and install.

1.1.3. Android Tools Method

If you have the android tools installed, you can follow the instructions for sending the APK file to the device over a USB cable.

CHAPTER 2. IOS

2.1. DEPLOYING AN APP ON IOS

Note

Steps 1 to 4 below are for App Developers only.

App Testers who have been sent a build to test can skip straight to step 5.

1. Get the UDID of your test device.

2. Log on to your iPhone Provisioning Portal. Create a device using the UDID you got from your iPhone, and create a provisioning profile with that device.

3. Upload the provisioning profile through My Account section of the RHMAP Studio (`https://<your-studio-domain>.redhatmobile.com`).

4. Build the application in Studio (`https://<your-studio-domain>.redhatmobile.com`) and download the binary for iPhone.

5. Add the provisioning profile and the app to your iTunes.

6. Install the app to your iPhone.
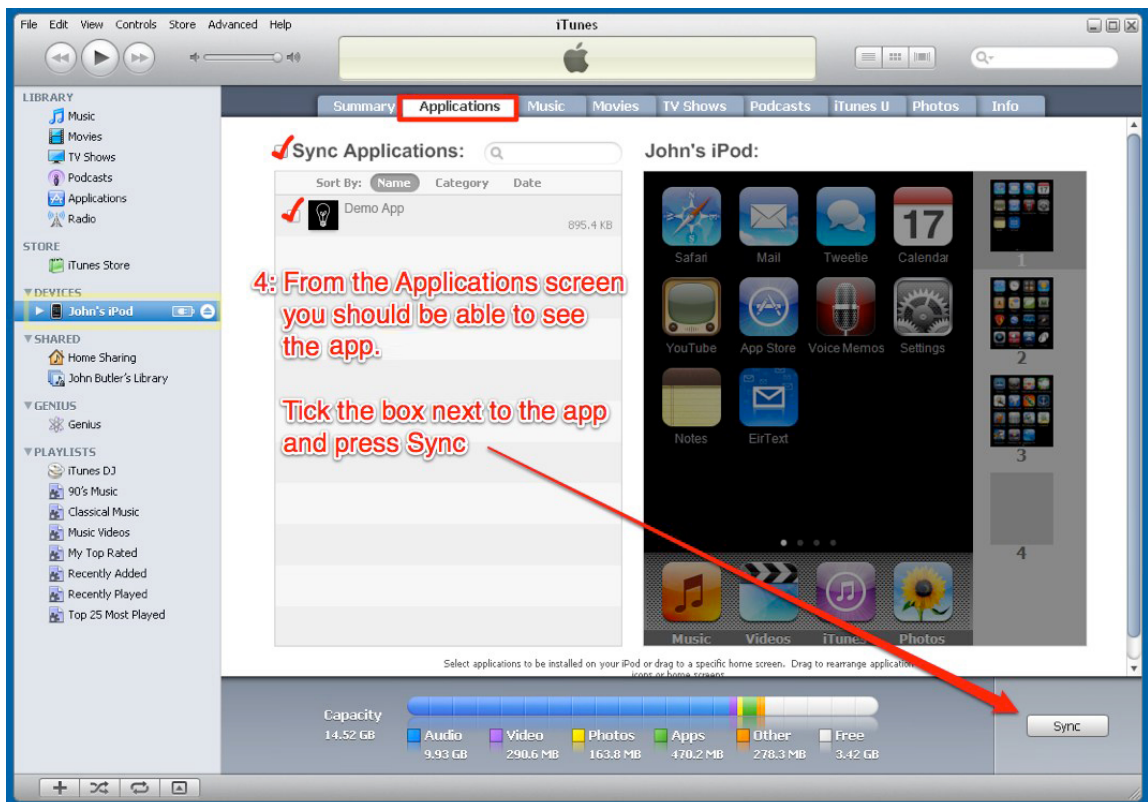
2.2. BUILDING APPS FOR IOS 7

Apple has announced that all the apps submitted to the App Store after February 1st will have to be built with the iOS 7 SDK. We have been working on our app building infrastructure to make sure the transition is as smooth as possible for our developers.

We have done most of the heavy lifting. However, there are still a few things need to be done by developers to ensure the apps are fully compatible with iOS 7.

2.2.1. Status Bar

From iOS 7, a UI view will use full screen by default which means the status bar will cover the top part of the application. Depending the application's user interface, it may become a problem when running on iOS 7. To help developers deal with this issue, we added a new configuration option in the App Studio called *Old Style Status Bar* (iOS destinations only). When this option is checked, we will make sure the app uses the old style status bar as seen on iOS 6 devices and it won't overlap with the application. It won't change anything if the app is running on pre iOS 7 devices.

This is done by using a Cordova plugin called Statusbar. It also provides some APIs to allow developers to change the style and behavior of the status bar programmatically. Check out the link for more details.

2.2.2. New Icon Dimensions

The dimensions of icons on iOS 7 have been changed as well. Check out the Icons and Splashscreens page for updated icon requirements.

These are the things you need to do to make sure the apps are fully compatible with iOS 7. We also suggest you to have a look at the iOS 7 Human Interface Guidelines from Apple and make any other necessary changes.

CHAPTER 3. WINDOWS

3.1. DEPLOYING AN APP ON WINDOWS PHONE 8

Different deployment options are available depending on the types of the Windows Phone 8 apps.

3.1.1. Non-company Windows Phone 8 Apps

Unless an app binary is signed by a Windows Phone company certificate, the app will be non-company app. Non-company Windows Phone 8 apps cannot be OTA installed on to devices directly.

3.1.1.1. Building

» Using Visual Studio

In Visual Studio, if no signing options are specified, the final binary will be a non-company app.

» Using the App Studio

When using the App Studio to build for Windows Phone, if the "Don't sign the binary" option is selected, the final app binary won't be signed and it will be a non-company app.

3.1.1.2. Distribution & Deployment

Normally you will have two options to distribute and deploy this type of apps:

» Using the Application Deployment Tool provided by the Windows Phone SDK

This option is mainly for development testing. If you have Windows Phone SDK installed on a computer, and the target device is a developer registered device, you can deploy WP apps using the Application Deployment Tool.

» Using the Windows Phone App Store

You can publish the app to the Windows Phone App Store and distribute from there.

3.1.2. Company Windows Phone 8 Apps

A non-company app will become a company app once it's been signed by a Windows Phone

Enterprise Certificate. Company apps can be OTA installed on to devices directly and bypassing the app store completely.

To build company apps using the App Studio, you should follow these steps:

3.1.2.1. Acquiring the enterprise certificate

Following these steps to get your enterprise certificate:

» Follow the steps in the Publish Windows apps guide to create a company Windows Phone developer account. Make sure the account type is "Company". This will cost $99 per year. It may take a few days to process as your company information will be validated.

» Once your developer account is setup, visit the Enterprise Mobile Code Signing Certificate to apply for the certificate file. Specify the "Publisher ID" field as provided by your Windows Phone developer account. This will cost $299 per year.

» Once the process is completed, Symantec will deliver a certificate that can be imported into the certificate store on a computer. See Certificate install instructions.

> **Tip**
>
> Use the same computer when requesting and picking up the certificate from Symantec.

» Finally, you should export the certificate with its private key as a PFX file and protect it with a password. See the Export a Certificate with the Private Key guide for instructions.

Check out the Company app distribution for Windows Phone guide for more details.

3.1.2.2. Generating the application enrollment token (AET)

Only the phones enrolled in your company account can OTA install apps signed by your enterprise certificate. To enroll devices, you need to generate the application enrollment token (AET) and install it on those devices.

You can follow the steps in the How to generate an application enrollment token for Windows Phone guide to generate the AET file manually.

Alternatively, the App Studio can generate the AET file for you during the next step.

To install the AET file on to devices, the easiest way is to store the AET on a web server and ask your users to download it using their devices' web browser. The App Studio can help with that as you will see in the next few steps.

3.1.2.3. Creating a credential bundle for Windows Phone

Once the certificate is available, you can create a new credential bundle in the App Studio.

As you can see in the screenshot, the App Studio can generate the AET file for you. However, if you have the AET file already, you can choose to upload it instead. If you choose to manage the AET file yourself, make sure the AET file is installed on your users' devices.

3.1.2.4. Build the app and sign it with the credential bundle

When building an app for Windows Phone in the App Studio, you will have the options to sign the app binary with the credential bundles you just uploaded.

3.1.2.5. Distribution & Deployment

Once the app finishes building, there will be OTA link and QR code on the download page to easily install it to the users' devices. If the AET file is available as part of the credential bundle, there will be download link & QR code for it as well.

Download Artifact

⊕ Download

-- or --

Use this OTA link or scan the QR code to install this build directly onto a device

http://henr.ie/1kQ7Qam

Need the AET file? Click here.

Close

3.2. SUPPORT FOR WINDOWS PLATFORMS

Red Hat Mobile Application Platfrom (RHMAP) supports the development of client apps for Windows-based devices. For a detailed tutorial, see Developing Forms Apps and Cordova Apps for Windows.

3.2.1. Supported Platforms

The current version of RHMAP targets Windows Phone version 8 and higher. Workarounds for platform-specific issues are provided further in this guide.

- **Windows 10 Universal Apps (Phone, Tablet and Desktop)**:

  - Full support, except:

    - Apps can't be built in the RHMAP and must be built locally

- **Windows 8.1 Universal Apps (Phone, Tablet and Desktop)**:

  - Full support, except:

    - Apps can't be built in the RHMAP and must be built locally

    - Cordova apps need a small workaround to enable support for dynamic content

- **Windows Phone 8**:

  - Full support

- **Windows Phone 7,7.5 and older**,

- **Windows 8 Pro for Tablets**,

- **Windows 8 RT for Tablets**,

- **Windows 7 and 8 Desktop**:

  - No support

3.2.2. Building Windows Apps Locally

Due to changes in Microsoft's licensing of its development tools, it is currently not possible to build Apps for Windows 8.1 and Windows 10 directly in RHMAP. Windows 8 apps are not affected by this. RHMAP apps for the affected platforms have to be built manually in your local development environment.

1. First, clone the repository of your client app.

   ```
   git clone <Git_URL>
   ```

2. Locate the solution file (.sln) in the project folder and open it in Visual Studio.

3. In Visual Studio, select a solution configuration and platform and build the project by pressing the F7 key. For more details, see the official documentation: Building a Windows Phone app in Visual Studio.

4. Once the build is finished, a binary with a `.xap` or `.appx` extension will be located at `<project_name>/<solution_name>/Bin/<solution_configuration>`. The binary file can be deployed to Windows devices or uploaded to the RHMAP App Store.

3.2.3. Enabling Dynamic Content in Cordova on Windows 8.1

On Windows 8.1, security restrictions are in place that prevent apps from using properties such as `innerHTML` and `outerHTML`. This in turn prevents any dynamic content to be injected and most JavaScript frameworks will therefore fail to function properly. This also prevents Cordova apps and RHMAP drag and drop apps from functioning.

A workaround for this issue is to reference a single JavaScript file provided by Microsoft, which relaxes the security restriction:

1. Download the `winstore-jscompat.js` file: https://raw.githubusercontent.com/MSOpenTech/winstore-jscompat/master/winstore-jscompat.js.

2. Copy the `winstore-jscompat.js` file into the `www` folder of your project

3. Reference the file from any HTML that requires dynamic content manipulation. `winstore-jscompat.js` must be referenced before any other JavaScript framework including `cordova.js` or `feedhenry.js`.

   ```
   <!-- above feedhenry.js or cordova.js-->
   <script src="winstore-jscompat.js" type="text/javascript"></script>
   ```

For more information about this workaround, see the description in the MSOpenTech/winstore-

jscompat repository.

3.3. DEVELOPING FORMS APPS AND CORDOVA APPS FOR WINDOWS

Overview

This tutorial shows you how to create a Forms app targeting the Windows platform using Red Hat Mobile Application Platform (RHMAP). The same steps as demonstrated in this tutorial also apply to any Cordova app, not just Forms apps.

Currently, there are several issues preventing support for the Windows platform equivalent to the other platforms, as described in Support for Windows Platforms. Specifically, native Windows 8.1 and Windows 10 apps can't be built by the Platform and have to be built manually. Also, Cordova apps need a small workaround to work properly in Windows 8.1.

> **Tip**
>
> This tutorial is also available as a quick video walkthrough.

Requirements

- Windows 8.1 or Windows 10

- 32 or 64-bit machine with minimum 4 GB of RAM

- Visual Studio 2013 Express or higher

- An existing Forms app or a Cordova app in the Platform. See the Create a Forms Project guide for help with creating a Forms app.

3.3.1. Steps

3.3.1.1. Initial Setup

1. Download and install Node.js: https://nodejs.org/en/download/.

    There's an installer available which guides you through the setup. Make sure to include the *npm package manager* in the installation. It's a core component which will also be required further in this guide.

2. Download and install a git client: https://git-scm.com/download/win
    Aside from the `git` executable itself, the installer of Git for Windows also provides an emulated Bash shell (called *Git Bash*) and a set of command-line tools in order to provide a familiar development environment. To learn more about the Git for Windows project, see the official website of Git for Windows.

Note

Most steps in this guide involving the use of command line can use **cmd** or PowerShell, unless otherwise noted. However, we suggest using the *Git Bash* command line installed in this step for all the steps.

3. Install the Cordova package

```
npm install -g cordova
```

This installs the Cordova package including a `cordova` executable globally and thus makes it available to all Node.js applications.

4. Clone the repository of your app from the Platform.

First, you need to upload your SSH key to the platform for git access to work properly. See SSH Key Setup for details.

Clone the repository of your app:

```
git clone <git-ssh-clone-url-of-your-app>
```

The Git SSH Clone URL of your app can be found in the App Details.

3.3.1.2. Cordova Setup

Normally, Cordova projects are automatically created for your Cordova apps during build in the Platform and the Cordova-specific files and folders are not available in the app's repository. However, when building manually, the Cordova files need to be created and command invoked manually.

Forms apps are based on Cordova and as such they need the same procedure for building as Cordova apps.

1. Create a Cordova project

We use the name *winforms* in this example, but it can be an arbitrary name.

```
cordova create winforms --copy-from <location-of-forms-app>\www
```

This creates a new Cordova project and copies the assets from the `www` directory into the newly created project's `www` directory.

2. **Windows 8.1 only**: Apply workaround for dynamic content manipulation in Cordova apps.

See Enabling Dynamic Content in Cordova on Windows 8.1 for more information.

```
git clone https://github.com/MSOpenTech/winstore-jscompat.git
copy winstore-jscompat\winstore-jscompat.js forms\www
```

The `winstore-jscompat.js` must be included in every HTML file in your Cordova app which does dynamic content manipulation.

```
<meta name="format-detection" content="telephone=no">
<!-- these three js files should be included exactly in this order
-->
<script src="winstore-jscompat.js" type="text/javascript"></script>
<script src="cordova.js" type="text/javascript"></script>
<script src="feedhenry.js" type="text/javascript"></script>
```

3. Add the **windows** platform to the Cordova project.

```
cd winforms
cordova platform add windows
```

This creates all the source files necessary to build the binary, including a Solution File (.sln) which can be used to import the project into Visual Studio.

4. Add the default Cordova plugins.

```
cordova plugin add \
    cordova-plugin-file \
    cordova-plugin-camera \
    cordova-plugin-file-transfer \
    cordova-plugin-device \
    cordova-plugin-network-information \
    cordova-plugin-battery-status \
    cordova-plugin-device-motion \
    cordova-plugin-device-orientation \
    cordova-plugin-geolocation \
    cordova-plugin-media \
    cordova-plugin-media-capture \
    cordova-plugin-dialogs \
    cordova-plugin-vibration \
    cordova-plugin-contacts \
    cordova-plugin-globalization \
    cordova-plugin-inappbrowser \
    cordova-plugin-console \
    https://github.com/fheng/fh-cordova-plugins-api.git
```

3.3.1.3. Building the Binary

The app binary can be built from the command line using Cordova or from Visual Studio and supports both Windows 8.1 and Windows 10.

To build using Cordova:

```
cordova build
```

To build using Visual Studio, import the solution file (.sln) created in previous step into Visual Studio and build the solution. See Building Windows Apps Locally for details.

Cordova also provides the ability to start an emulator to quickly see the app running:

```
cordova emulate
```

CHAPTER 4. FORMS

4.1. INTEGRATING FORMS INTO A CORDOVA APP

Forms functionality can be integrated into existing apps. This guide demonstrates the integration on an example of a workforce management application — a supervisor assigns jobs, a worker receives the assignments on their mobile device, and sends back information about the job.

The procedures in this guide use a Cordova app and a cloud app. These apps can be imported into a project to demonstrate the working example described in this guide.

4.1.1. Working Example

4.1.1.1. Overview

The following requirements are set for the example application:

A supervisor creates a job for a worker to remove a fallen tree after a storm. The supervisor asks the worker for details of the job such as photos of the tree, location, comments, and the time started and finished. This can be achieved using forms apps.

The supervisor creates a job by:

» Selecting the form for the worker to complete.

» Selecting and filling out a form with the details for the job.

» Giving the job a unique ID.

The requirements for the application include:

» Jobs are created by an admin by filling out a creation form.

» New jobs are available to both admin and non-admin users. App users can view the completed creation form.

» App users can complete a job by filling in a separate completion form.

» The completion form must be visible to any app user to review.

» The client app user can mark the job as either "In Progress" or "Complete" by saving the completion form as a draft.

4.1.1.2. Cordova App

The client app is based on the following technologies:

» Backbone: The Cordova app uses backbone models and views to manage the creation and update of jobs.

» Handlebars: Used for view templating.

» Boostrap: Used for styling.

» Font-Awesome: Used for icons.

The Cordova app is responsible for:

- » Managing the listing of jobs in various states.

- » Managing the rendering of any forms.

- » Managing the submission and upload of any form submissions.

- » Managing the creation of jobs containing form and submission data.

4.1.1.2.1. Job Model

The job model is a simple Backbone model describing a job.

The jobs collection is a collection of job models.

A custom URL is included for synchronizing jobs between the client and cloud. This custom URL is used to access RESTful **/jobs** endpoints on the cloud app.

4.1.1.3. Cloud App

The cloud app consists of RESTful endpoints (/jobs) for performing CRUD operations on job data using the $fh.db API.

> **Note**
>
> There is no visible logic in the cloud app to deal with forms, because all cloud-based forms logic is contained in the cloud APIs.

4.1.2. Creating A Working Example

1. Create a new project in the App Studio.

2. Import an app into your project. For example, the examples in this guide use the Cordova app and cloud app.

3. Create forms and themes for the project. Any created forms and themes associated with the project will now be visible in the Cordova app.

4. Optionally, add users and field codes to the project. For example:

   - » Admin: Able to create and complete jobs.

   - » User: Able to complete jobs.

   A user has the **userId** and **userName** fields that are automatically added to a submission before rendering the related form. Add two text fields to any forms associated with this working example.

   When the fields have been added, add two users to the **users** collection in your Data Browser.

   Admin

```
{
    "userId": "admin",
    "userName": "<<Any Name>>"
}
```

User

```
{
    "userId": "user",
    "userName": "<<Any Name>>"
}
```

4.1.3. Implementation Guide

Use the following to integrate forms functionality into an app.

1. Add Forms Initialization by adding the **`$fh.forms.init`** function to the client. This initializes forms on the client app to enable the usage of the $fh.forms Client API in the rest of the app.

   The **`$fh.forms.init`** function is part of the log in process for the app.

2. As an admin user, select a completion form. This specifies the form that needs to be completed in order to complete the job. List all of the forms available to the app using the **`$fh.forms.getForms`** client API function.

   > **Note**
   >
   > The **`$fh.forms.getForms`** client API call only downloads a list of forms, it does not download the entire form definition for each form.

3. Download a form to the client using the **`$fh.forms.getForm`** client API.

   As forms are used in job creation, viewing job details, and completing jobs, this function is abstracted to a set of helper functions here.

   The **`$fh.forms.getForm`** client API usage can be seen here as part of the **`loadForm`** function in **`FormFunctions.js`**.

4. Load a submission into your app. This process is illustrated using the loadSubmission function in the FormFunctions.js file.

   Forms are related to submissions, in that any data entered into a form is populated to a submission. However, a submission is validated against a form before being upload to the cloud.

   There are three ways to create a submission:

   - **From Local Memory**: Save a submission as a draft to local memory then edit later using the **`saveDraft`** function on the submission model. The implementation of this functionality is shown in the loadLocalSubmission function.

   - **Download From Remote**: Download a submission from the cloud. For example, when

the supervisor completes a form to describe the details of the job, the ID of the submission is saved to the job model. When the app user downloads the job model, they have access to the remote submission ID of the form submitted by the admin user. This remote submission ID is used to download the full submission definition from the cloud. The implementation of this functionality is shown in the downloadSubmission function.

> **Note**
>
> The form definition for the submission is contained in the submission downloaded from the cloud. This is because the form definition may have been edited between submissions.

> **Note**
>
> Downloaded submissions should not be edited on the client. They are intended for read-only access. Any attempt to submit a downloaded submission to the cloud will return an error.

- **Create A New Submission**: If there is no submission associated with a form, a new submission can be created. In this case, the submission is created from a form model. This ensures that the submission is automatically related to the correct form.

5. Render the form into the view for editing by a user.

   There are two methods of rendering a form into an existing Cordova app:

   - Rendering the form using the **$fh.forms.backbone** API, which includes a backbone/bootstrap SDK (**$fh.forms.backbone**), by downloading the Appforms Backbone file and include it as part of your Cordova app. In addition, the Cordova app must satisfy the following JavaScript and CSS dependencies:

     - Backbone

     - Bootstrap

     - Font-Awesome

     The CSS and JavaScript dependencies are included in the example Cordova app.

     The **FormViewSDK.js** file contains the Backbone SDK version of the form view. The Cordova app contains an option in the "Settings" tab to switch between the Backbone SDK and manual form rendering.

     > **Note**
     >
     > The Backbone SDK is intended to speed up forms apps integration for Backbone/Bootstrap based Cordova apps. However, the **$fh.forms** client API will work with any Cordova app. The rendering of the form and managing the population of user data to a submission will be the responsibility of the developer.

   - Rendering a form manually.

Note

Rendering a form to the user is the simplest method of completing a submission. However, field input values can be added to a submission from any source. The submission is still required to be valid against any field or page rules.

The `$fh.forms` SDK does not depend on any framework, and can therefore be added to any Cordova app. This app is based on Backbone and Bootstrap, however it is equally possible to use the `$fh.forms` API with other javascript-based UI frameworks (for example, Angular).

A basic Bootstrap form is rendered based on the form definition. This form is defined in the `FormView.js` file. All of the rendering, submission input, and validation logic of the form is defined in the app using the `$fh.forms` API and models.

Note

The manually rendered form is implemented for illustration purposes only. Only the text and number fields are manually implemented. However, all available form field types can be rendered using the `$fh.forms.backbone` SDK.

The rendering logic for the custom form view is located in the `FormView.js` file. Here, you can see that the view handles all of the events related to rendering the form to the user.

In addition, the `FormView.js` file contains logic for:

- Validating field data when entered.

- Checking field and page rules.

- Populating data to a submission.

- Saving a submission as a draft.

- Submitting a form to the cloud.

The following steps illustrate how the Cordova app addresses these requirements when manually integrating the `$fh.forms` SDK into a custom rendered form.

6. Define the validation parameters that restrict the data that can be entered into the field (for example, a text field can specify a minimum/maximum number of characters that can be entered into the field). Adding this functionality to the client app reflects the restrictions of the field.

   To satisfy this requirement, the validateInput function is registered to the blur event of an input in the `FormView.js` file.

> **Note**
>
> Validation parameters influence whether a submission is valid. Even if field validation is not performed on the client app, all submission fields will be validated before saving to the database.

7. Form apps include field and page rules. In the Studio, forms editors can create field rules to show and hide fields based on field input data and page rules to show and skip pages based on field input data.

   This functionality is reflected in the implementation of the **$fh.forms** API. By processing a submission using a rules engine, the submission can identify fields or pages that need to be shown or hidden.

   This is implemented in the checkRules function in the **FormView.js** file.

   > **Note**
   >
   > Field and page rules influence whether a submission is valid. Even if field and page rules are not checked on the client app, the submission will be checked against all rules before saving to the database.

8. Add data to a submission model using the **addInputValue** function. The source of this data can either be the form rendered to the user, external data available to the app, or a mixture of both.

   » From a rendered form: In this case, a form is rendered for the user to input data using the **$fh.forms.backbone** SDK or by manually rendering a form.

     When manually integrating the **$fh.forms** API into a custom rendered forms, it is necessary to handle the migration of data from the view to the submission model.

     This is illustrated by the saveFieldInputsToSubmission function in the **FormView.js** file.

   » From an external source using field codes: You can add field codes to form fields to uniquely identify a field within a form. This field code can relate to an external data source (for example, a header in a CSV file). Using this functionality, it is possible to import external data into a form submission.

     This functionality is demonstrated in the example Cordova app by the addSubmissionData function. In this example, a user has **userId** and **userName** fields. If a form contains fields with fields codes **userId** and **userName**, these fields will be populated with the data from the User model.

     > **Note**
     >
     > Field codes must be unique within a form. However, the same field code can be present in multiple forms.

9. Save a submission as a draft. This functionality is illustrated by the saveDraft function in the **FormView.js** file.

10. Having added validation and rules functionality to the form, we can now submit valid submissions to the cloud for viewing/editing on the submission editor.

    The form view listens for submission-related events (validationerror, queued, progress, error, submitted) emitted by the submission model as the data is being processed and uploaded.

    The submission process has two distinct steps:

    » Submit: Calling the submit function on a submission model validates the submission against the local form definition and changes the submission status to pending.

    » Upload: Calling the upload function on a submission model will queue the submission for upload to the forms database.

4.1.4. Related Sections

» $fh.forms Client API

» $fh.db Cloud API

» Creating A Form

» Creating A Theme

» Creating A Forms Project

4.2. ENABLING FORMS SUPPORT IN OPENSHIFT 2 TARGETS

Support for using Forms in cloud apps deployed to OpenShift 2 targets is not immediately available. It can be enabled by following the steps outlined in this document.

Note

This guide only applies to Platform instances with support for OpenShift 2 targets, for example openshift.feedhenry.com. Other targets have Forms support available without any extra configuration.

4.2.1. What's behind Forms in the Platform

There's a service in the back end of the Platform responsible for handling Forms-related operations, called **fh-mbaas**. Its functions are exposed through the **$fh.forms** API and include access to form definitions, submissions and deployment of forms.

With FeedHenry MBaaS targets an instance of the **fh-mbaas** is always available and doesn't need to be manually enabled. This is however not the case for OpenShift 2 targets which, by default, don't have the **fh-mbaas** service deployed. This functionality is provided separately as a service template in the Platform which can be deployed to an OpenShift 2 target to enable Forms support.

On the OpenShift side the Forms support is implemented through a cartridge. This cartridge is hosted, along with a Node.js runtime and database cartridges, in a separate application in your OpenShift account and exposes its functionality to all of your Forms-based applications.

4.2.2. How to enable Forms support for OpenShift 2 targets

Prerequisites:

» a RHMAP account with support for OpenShift 2 targets

» an existing OpenShift 2 target and an associated environment

The following steps will guide you through the process of enabling your app to use Forms in an OpenShift 2 target:

1. Create and deploy an instance of the *OpenShift mBaaS Service*.

   a. In the *Services & APIs* section of the Studio, click *Provision MBaas Service/API*

   b. From the list of available services, choose *OpenShift mBaaS Service*, enter any suitable name for the service instance and click *Next*

   c. Wait until the log window shows *Service is ready!* and click *Finish*. The Service Details page will be displayed.

   d. Navigate to the *Deploy* section in the left-hand side menu

   e. Select an appropriate OpenShift-backed environment from the dropdown menu on the right-hand side of the screen

   f. Click *Deploy Cloud App*

   g. Wait until the service gets deployed. In the *Deploy History* section at the bottom of the screen, the deployment status and result are displayed. It might happen that these indicators don't reflect the immediate state. Reload the page to update the status and check that the correct environment is selected. After the deployment is

finished, reload the page again and check that there's a *Current Host* field in *Deploy Options* at the top of the screen.

> **Note**
>
> It may take several minutes for the service to get deployed, as this involves creating a gear, provisioning several cartridges, cloning a git repository and other setup operations. This is only performed once for an OpenShift 2 target. Similarly, the first deployment of a cloud app might take longer than any subsequent deployments.

Deploy History

Deploy Target	Status	Result	Started At	Finished At	Runtime	Git Branch/Tag	Commit	Clean Stage
OPENSHIFT	Complete	Success	Jul 24th 2015 3:09:28 pm	Jul 24th 2015 3:16:39 pm	0.10.25	Branch : master	69c7121b6a6ce2c1	Yes

h. Write down or copy the value of *Current Host* field of the *Deploy Options* section of the screen. The URL points to the newly created OpenShift application which acts as the fh-mbaas service.

Deploy Options

Your app will be deployed to the following MBaaS target.

OPENSHIFT
https://openshift.redhat.com

Current Host   https://openshiftmbaasservfhdovb3p-nzr.rhcloud.com

Choose Branch or Tag   ⑂ branch : master ▾

Clean Stage   ✔

2. Set the correct fh-mbaas host URL in an OpenShift 2 target.

a. Navigate to the *Admin* / *MBaaS Targets* section of Studio and open an existing OpenShift 2 MBaaS target.

b. Enter the URL of the fh-mbaas service that you saved earlier into the **fh-mbaas Host** field.

c. Click *Save MBaaS*.

Any cloud app deployed to this new OpenShift-backed environment will have access to Forms functionality.

A single fh-mbaas service can be used to provide Forms support to multiple OpenShift 2 MBaaS targets and can reside on a different OpenShift instance than the OpenShift 2 MBaaS targets using it. Cloud apps using Forms can be deployed to any OpenShift 2 target as long as that target has a valid *fh-mbaas Host* set.

CHAPTER 5. DEVELOPING CLIENT APPS

5.1. DEVELOPING AN ANGULAR APP USING RHMAP

Overview

This guide will introduce you to using the RHMAP Javascript SDK within a HTML5 AngularJS Cordova App.

This guide includes the steps necessary to create a new AngularJS Hello World Project and highlights the code necessary to interact with a Cloud Code App.

Requirements

- RHMAP Account

- Knowledge of HTML, JavaScript, AngularJS and Node.js

- GitHub User Account

- FHC.

A useful AngularJS Tutorial is available for users wishing to start with AngularJS.

5.1.1. Sample Project Overview

The example project is a simple project containing one Client App and one Cloud App.

5.1.1.1. Client App

The Client App is a simple AngularJS App that includes the RHMAP Javascript SDK. The Client App asks the user to enter their name into a text box and click a **Say Hello From The Cloud** button. The Client App then uses the $fh.cloud function to send the text entered to the Cloud Code App.

5.1.1.2. Cloud Code App

The Cloud Code App exposes a **hello** endpoint to recieve a request from the Client App, change the text sent to add **Hello** and return the response to the client.

5.1.2. Create A New AngularJS Hello World Project

Use the following steps to create a new AngularJS Hello World Project.

1. Log into the Studio.

2. Select the **Projects** tab located at top-left of the screen.

3. Select the **+ New Project** button located at the top of the screen. You will then see a list of Project Templates.

4. Select the **AngularJS Hello World Project** Template and give the new project a name.

5. Select the `Next` button. The new project will now be created.

6. When the project has been created, select the `Finish` button.

You now have a new Project containing an AngularJS Client App and a Cloud Code App that it will communicate with.

To Preview the app:

1. Select the `Apps, Cloud Apps & Services` tab.

2. Select the `Client App` under the `Apps` section. This will take you to the Details Screen.

3. The `App Preview` contains a working version of the AngularJS Client App that will communicate with the Cloud Code App.

5.1.3. Build The Client App For An Android Device

Now that you have tested your app and are happy that it is working correctly, you can now build the Client App using the RHMAP Studio.

Use the following steps to build the Client App for an Android device:

1. From the Client App interface selected the `Build` tab.

2. Select `Android`, the `master` branch, a `Debug` build type, the `Dev` Cloud Code App and then click the `Build` button. An Android build of the Client App will now be built.

3. When the Android build has completed, you will be presented with a QR Code, simply open a QR Code Scanner app on your Android device and install the build. Alternatively, type the short URL into your phone's browser.

Note

You can build Android debug binaries without any certificates but you will need the requisite credentials to build any type of iOS/Windows Phone binaries.

Note

The **branch** selector allows you to select which branch of the Client App you wish to build. In this case, the default **master** branch is the correct branch.

Congratulations. You have just created an AngularJS HTML5 Cordova App using RHMAP.

5.1.4. Development Overview

This section will highlight the code necessary for the example solution to work correctly.

5.1.4.1. Cloud Code App

First, let's consider the Cloud Code App. In this example, we want the Cloud Code App to recieve a request from the Client App, change the **hello** parameter and respond to the Client App using a JSON object containing the following parameters:

```
{
    msg: "Hello <<hello parameter sent by the client app>>"
}
```

To implement this functionality in the Cloud Code App:

1. In the application.js file, a new **/hello** route is added which requires a **hello.js** file located in the **lib** directory.

2. The hello.js file creates two routes. Both routes perform the same operation of changing the **hello** parameter.

   » A **GET** request where the **hello** parameter is appended as a query string.

   » A **POST** request where the **hello** parameter is sent in the body of a POST request.

> **Note**
>
> This Cloud Code App is completely independent of the AngularJS Client App. The Cloud Code App can be shared between any number of Client Apps within a project.

5.1.4.2. AngularJS Client App

Having created the **/hello** endpoint in the Cloud Code App, we now proceed to examine the functionality added to the AngularJS Client App to allow it to send requests to the **/hello** endpoint exposed in the Cloud Code App.

The Client App is a simple AngularJS App with a single input that accepts some text and a single button that sends the input to the cloud and displays the result to the user.

5.1.4.2.1. RHMAP SDK

Firstly, as in all HTML5 Cordova Apps, the RHMAP Javascript SDK (**feedhenry.js**) is included in the index.html file. This allows access to all of the $fh Client API functions.

> **Note**
>
> You will notice that the feedhenry.js file is empty in the template repository. When a Client App is created, if the **feedhenry.js** file is present in the **www** directory, the latest RHMAP Javascript SDK will be added to the file automatically.

5.1.4.2.2. fhconfig.json

The Client App also contains a fhconfig.json file. This file contains the information needed for the RHMAP Javascript SDK to communicate with the cloud app.

Note

All HTML5 Client Apps must contain a `fhconfig.json` file to use the $fh Client API functions. This file is automatically populated with the required information when the app is created in the Studio.

5.1.4.2.3. $fh.cloud

In this example, the $fh.cloud Client API function is used to send requests to the **hello** endpoint in the Cloud Code App.

The **$fh.cloud** function is located in the cloud.js file. Here, the $fh.cloud function is exposed as a reusable service for the **MainCtrl** Controller to use.

There is a single controller in the AngularJS App called MainCtrl. This controller is responsible for

1. Accepting the input from the user from the example.html view.

2. Using the **fhcloud** service to call the **hello** endpoint in the Cloud Code App.

3. Processing the response from the Cloud Code App using the **success** or **error** functions, depending on whether the $fh.cloud call was successful.

Note

In this case, the Client App is using a **GET** request type. As the Cloud Code App exposes both a **GET** and **POST** version of the **hello** endpoint, a **POST** request type will also work. This is especially useful when dealing with RESTful applications.

5.2. DEVELOPING A BACKBONE APP USING RHMAP

Overview

This guide will introduce you to using the RHMAP Javascript SDK within a HTML5 Backbone Cordova App.

This guide includes the steps necessary to create a new Backbone Hello World Project and highlights the code necessary to interact with a Cloud Code App.

Requirements

- RHMAP Account

- Knowledge of HTML, JavaScript, Backbone and Node.js

- GitHub User Account

- FHC.

A Backbone Tutorial is available for users wishing to start with Backbone.

5.2.1. Sample Project Overview

The example project is a simple project containing one Client App and one Cloud App.

5.2.1.1. Client App

The Client App is a simple Backbone App that includes the RHMAP Javascript SDK. The Client App asks the user to enter their name into a text box and click a **Say Hello From The Cloud** button. The Client App then uses the $fh.cloud function to send the text entered to the Cloud Code App.

5.2.1.2. Cloud Code App

The Cloud Code App exposes a **hello** endpoint to recieve a request from the Client App, change the text sent to add **Hello** and return the response to the client.

5.2.2. Create A New Backbone Hello World Project

Use the following steps to create a new Backbone Hello World Project.

1. Log into the Studio.

2. Select the **Projects** tab located at top-left of the screen.

3. Select the **+ New Project** button located at the top of the screen. You will then see a list of Project Templates.

4. Select the **Backbone Hello World Project** Template and give the new project a name.

5. Select the **Next** button. The new project will now be created.

6. When the project has been created, select the **Finish** button.

You now have a new Project containing an Backbone Client App and a Cloud Code App that it will communicate with.

To Preview the app:

1. Select the **Apps, Cloud Apps & Services** tab.

2. Select the **Client App** under the **Apps** section. This will take you to the Details Screen.

3. The **App Preview** contains a working version of the Backbone Client App that will communicate with the Cloud Code App.

5.2.3. Build The Client App For An Android Device

Now that you have tested your app and are happy that it is working correctly, you can now build the Client App using the RHMAP Studio.

Use the following steps to build the Client App for an Android device:

1. From the Client App interface selected the **Build** tab.

2. Select **Android**, the **master** branch, a **Debug** build type, the **Dev** Cloud Code App and then click the **Build** button. An Android build of the Client App will now be built.

3. When the Android build has completed, you will be presented with a QR Code, simply open a QR Code Scanner app on your Android device and install the build. Alternatively, type the short URL into your phone's browser.

Note

You can build Android debug binaries without any certificates but you will need the requisite credentials to build any type of iOS/Windows Phone binaries.

Note

The **branch** selector allows you to select which branch of the Client App you wish to build. In this case, the default **master** branch is the correct branch.

Congratulations. You have just created an Backbone HTML5 Cordova App using RHMAP.

5.2.4. Development Overview

This section will highlight the code necessary for the example solution to work correctly.

5.2.4.1. Cloud Code App

First, let's consider the Cloud Code App. In this example, we want the Cloud Code App to recieve a request from the Client App, change the **hello** parameter and respond to the Client App using a JSON object containing the following parameters:

```
{
    msg: "Hello <<hello parameter sent by the client app>>"
}
```

To implement this functionality in the Cloud Code App:

1. In the application.js file, a new **/hello** route is added which requires a **hello.js** file located in the **lib** directory.

2. The hello.js file creates two routes. Both routes perform the same operation of changing the **hello** parameter.

   ≫ A **GET** request where the **hello** parameter is appended as a query string.

   ≫ A **POST** request where the **hello** parameter is sent in the body of a POST request.

Note

This Cloud Code App is completely independent of the Backbone Client App. The Cloud Code App can be shared between any number of Client Apps within a project.

5.2.4.2. Backbone Client App

Having created the **/hello** endpoint in the Cloud Code App, we now proceed to examine the functionality added to the Backbone Client App to allow it to send requests to the **/hello** endpoint exposed in the Cloud Code App.

The Client App is a simple Backbone App with a single input that accepts some text and a single button that sends the input to the cloud and displays the result to the user.

5.2.4.2.1. RHMAP SDK

Firstly, as in all HTML5 Cordova Apps, the RHMAP Javascript SDK (**feedhenry.js**) is included in the index.html file. This allows access to all of the $fh Client API functions.

> **Note**
>
> You will notice that the feedhenry.js file is empty in the template repository. When a Client App is created, if the **feedhenry.js** file is present in the **www** directory, the latest RHMAP Javascript SDK will be added to the file automatically.

5.2.4.2.2. fhconfig.json

The Client App also contains a fhconfig.json file. This file contains the information needed for the RHMAP Javascript SDK to communicate with the cloud app.

> **Note**
>
> All HTML5 Client Apps must contain a **fhconfig.json** file to use the $fh Client API functions. This file is automatically populated with the required information when the app is created in the Studio.

5.2.4.2.3. $fh.cloud

In this Backbone Client App, there is a single Count View. This view is bound to the **hello** div in the index.html file.

The Count View listens for the **click** event of the **Get No. of Characters** button. When the button is clicked:

1. The Count View will use the Cloud Helper Function to call the $fh.cloud Client API function.

2. The **$fh.cloud** function will then send a **GET** request to the **hello** endpoint of the Cloud Code App.

3. The **$fh.cloud** function will call the **success** function with the result of the counting operation or the **error** function if the request has failed.

4. The Count View is then updated with the relevant message.

Note

In this case, the Client App is using a `GET` request type. As the Cloud Code App exposes both a `GET` and `POST` version of the `hello` endpoint, a `POST` request type will also work. This is especially useful when dealing with RESTful Backbone Models.

5.3. DEVELOPING AN IONIC APP USING RHMAP

Overview

This guide will introduce you to using the RHMAP Javascript SDK within a HTML5 Ionic Cordova App.

This guide includes the steps necessary to create a new Ionic Hello World Project and highlights the code necessary to interact with a Cloud Code App.

Requirements

- RHMAP Account

- Knowledge of HTML, JavaScript, Ionic and Node.js

- GitHub User Account

- FHC installed.

The Ionic Documentation contains all of the information needed to start developing Ionic Apps.

5.3.1. Sample Project Overview

The example project is a simple project containing one Client App and one Cloud App.

5.3.1.1. Client App

The Client App is a simple Ionic App that includes the RHMAP Javascript SDK. The Client App asks the user to enter their name into a text box and click a `Say Hello From The Cloud` button. The Client App then uses the $fh.cloud function to send the text entered to the Cloud Code App.

5.3.1.2. Cloud Code App

The Cloud Code App exposes a `hello` endpoint to recieve a request from the Client App, change the text sent to add `Hello` and return the response to the client.

5.3.2. Create A New Ionic Hello World Project

Use the following steps to create a new Ionic Hello World Project.

1. Log into the Studio.

2. Select the `Projects` tab located at top-left of the screen.

3. Select the **+ New Project** button located at the top of the screen. You will then see a list of Project Templates.

4. Select the **Ionic Hello World Project** Template and give the new project a name.

5. Select the **Next** button. The new project will now be created.

6. When the project has been created, select the **Finish** button.

You now have a new Project containing an Ionic Client App and a Cloud Code App that it will communicate with.

To Preview the app:

1. Select the **Apps, Cloud Apps & Services** tab.

2. Select the **Client App** under the **Apps** section. This will take you to the Details Screen.

3. The **App Preview** contains a working version of the Ionic Client App that will communicate with the Cloud Code App.

5.3.3. Build The Client App For An Android Device

Now that you have tested your app and are happy that it is working correctly, you can now build the Client App using the RHMAP Studio.

Use the following steps to build the Client App for an Android device:

1. From the Client App interface selected the **Build** tab.

2. Select **Android**, the **master** branch, a **Debug** build type, the **Dev** Cloud Code App and then click the **Build** button. An Android build of the Client App will now be built.

3. When the Android build has completed, you will be presented with a QR Code, simply open a QR Code Scanner app on your Android device and install the build. Alternatively, type the short URL into your phone's browser.

Note

You can build Android debug binaries without any certificates but you will need the requisite credentials to build any type of iOS/Windows Phone binaries.

Note

The **branch** selector allows you to select which branch of the Client App you wish to build. In this case, the default **master** branch is the correct branch.

Congratulations. You have just created an Ionic HTML5 Cordova App using RHMAP.

5.3.4. Development Overview

This section will highlight the code necessary for the example solution to work correctly.

5.3.4.1. Cloud Code App

First, let's consider the Cloud Code App. In this example, we want the Cloud Code App to recieve a request from the Client App, change the **hello** parameter and respond to the Client App using a JSON object containing the following parameters:

```
{
    msg: "Hello <<hello parameter sent by the client app>>"
}
```

To implement this functionality in the Cloud Code App:

1. In the application.js file, a new **/hello** route is added which requires a **hello.js** file located in the **lib** directory.

2. The hello.js file creates two routes. Both routes perform the same operation of changing the **hello** parameter.

   ≫ A **GET** request where the **hello** parameter is appended as a query string.

   ≫ A **POST** request where the **hello** parameter is sent in the body of a POST request.

Note

This Cloud Code App is completely independent of the Ionic Client App. The Cloud Code App can be shared between any number of Client Apps within a project.

5.3.4.2. Ionic Client App

Having created the **/hello** endpoint in the Cloud Code App, we now proceed to examine the functionality added to the Ionic Client App to allow it to send requests to the **/hello** endpoint exposed in the Cloud Code App.

The Client App is a simple Ionic App with a single input that accepts some text and a single button that sends the input to the cloud and displays the result to the user.

5.3.4.2.1. RHMAP SDK

Firstly, as in all HTML5 Cordova Apps, the RHMAP Javascript SDK (**feedhenry.js**) is included in the index.html file. This allows access to all of the $fh Client API functions.

Note

You will notice that the feedhenry.js file is empty in the template repository. When a Client App is created, if the **feedhenry.js** file is present in the **www** directory, the latest RHMAP Javascript SDK will be added to the file automatically.

5.3.4.2.2. fhconfig.json

The Client App also contains a fhconfig.json file. This file contains the information needed for the RHMAP Javascript SDK to communicate with the cloud app.

> **Note**
>
> All HTML5 Client Apps must contain a `fhconfig.json` file to use the $fh Client API functions. This file is automatically populated with the required information when the app is created in the Studio.

5.3.4.2.3. $fh.cloud

In this example, the `$fh.cloud` Client API function is used to send requests to the `hello` endpoint in the Cloud Code App.

The `$fh.cloud` function is located in the cloud.js file. Here, the $fh.cloud function is exposed as a reusable service for the `MainCtrl` Controller to use.

There is a single controller in the Ionic App called MainCtrl. This controller is responsible for

1. Accepting the input from the user from the example.html view.

2. Using the `fhcloud` service to call the `hello` endpoint in the Cloud Code App.

3. Processing the response from the Cloud Code App using the `success` or `error` functions, depending on whether the $fh.cloud call was successful.

> **Note**
>
> In this case, the Client App is using a `GET` request type. As the Cloud Code App exposes both a `GET` and `POST` version of the `hello` endpoint, a `POST` request type will also work. This is especially useful when dealing with RESTful applications.

5.4. DEVELOPING A PUSH NOTIFICATION APPLICATION USING AEROGEAR UNIFIEDPUSH SERVER

Overview

This tutorial illustrates how to connect the community version of AeroGear UnifiedPush Server (UPS) deployed in your OpenShift instance to the Red Hat Mobile Application Platform (RHMAP).

You will build a set of sample applications for receiving sports news:

- **Mobile application** (*Push Notifications Mobile Client*): lets users subscribe to categories and receive a push notification whenever a new story is posted in any of the chosen categories.

- **Web application** (*Push Console App*): lets administrators create and update the category list, as well as create news stories and push them to clients.

- **Server application** (*Push Cloud App*): This cloud app is the back end, containing the server-side logic. It exposes API methods to manage categories (that is, different sports) and news stories associated with one or more categories. The API of this app is called by the web and mobile application. This app communicates with the UnifiedPush Server.

> **Warning**
>
> The community version of UPS is not supported. For the officially supported way to use push notifications in your apps, see Developing An Application Using Push Notifications

Requirements

- RHMAP account

- OpenShift Online account

- Mobile platform developer accounts (Google, Apple)

5.4.1. Create a Project using the *AeroGear Community Push* Template

To create a project using the push template in the Studio:

1. Log into the Studio.

2. Select the *Projects* tab located at top-left of the screen.

3. Select the *New Project* button located at top-left of the screen.

4. Find the *AeroGear Community Push* template and click the corresponding *Choose* button on the right side of the screen.

5. Enter a name for the project in the *Name your Project* box and click the *Create* button on the right side of the screen.

6. Once the project creation has completed, click the *Finish* button at the bottom of the screen.
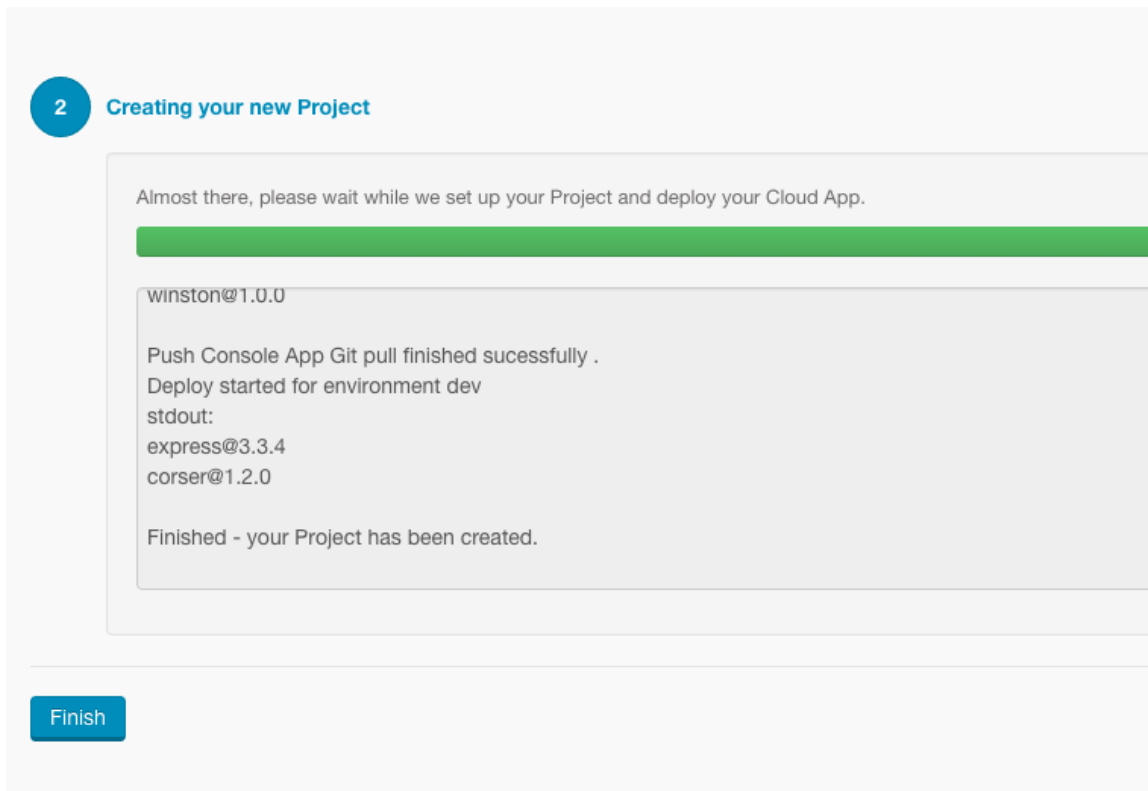
At this point, your project contains three apps:

» Push Cloud App: a simple CRUD application for storing categories and news,

» Push Console App: a *management interface* where administrators can create and delete categories,

» Push Notifications Mobile Client: a client app where users *subscribe* to categories and receive push notifications when new stories are created.

5.4.2. Create an instance of UPS

In order for the application to send and receive push notifications, an instance of the *UnifiedPush Server* (*UPS*) must be set up.

To create an instance of UPS running on *Openshift Online*:

1. Log into your Openshift Online account

2. Create a new *AeroGear Push 1.x Quickstart* application. For more details on creating, see the AeroGear Documentation
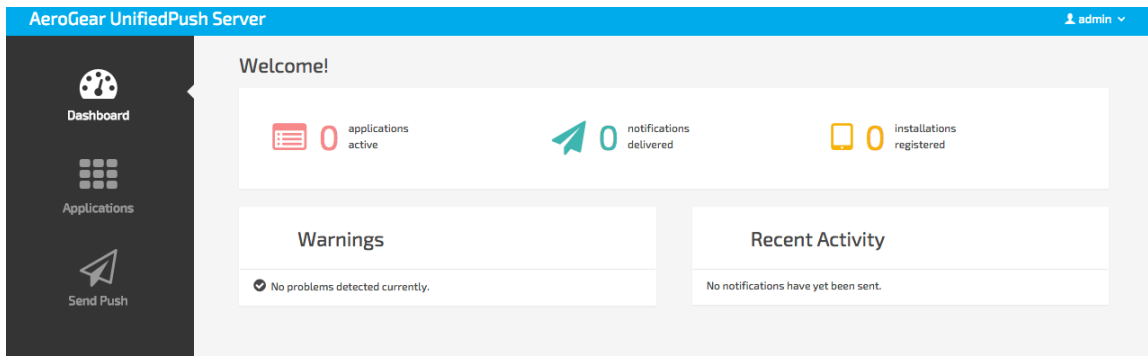
3. Log into your *AeroGear Push 1.x Quickstart* application and click on *Applications* on the left side of the screen.
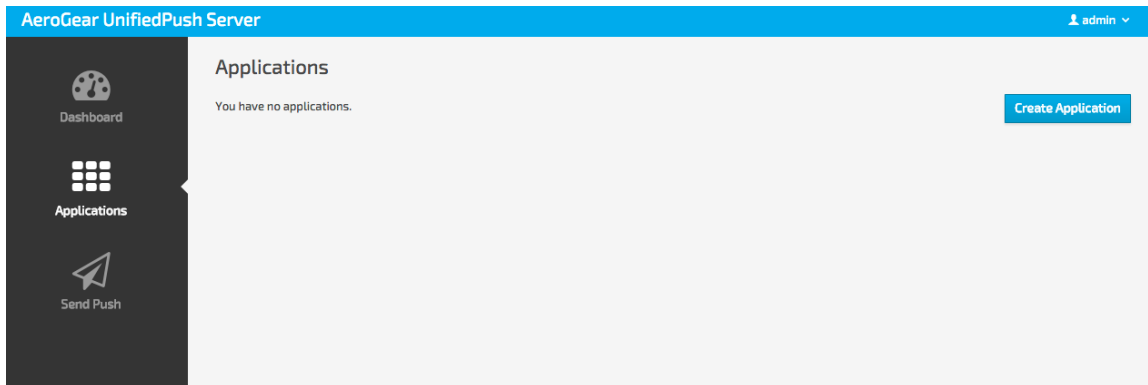
> **Note**
>
> The default username and password can be found in the AeroGear documentation.

4. Click on *Create Application* on the right side of the screen.

5. Enter a name and description for the application and click *Create*
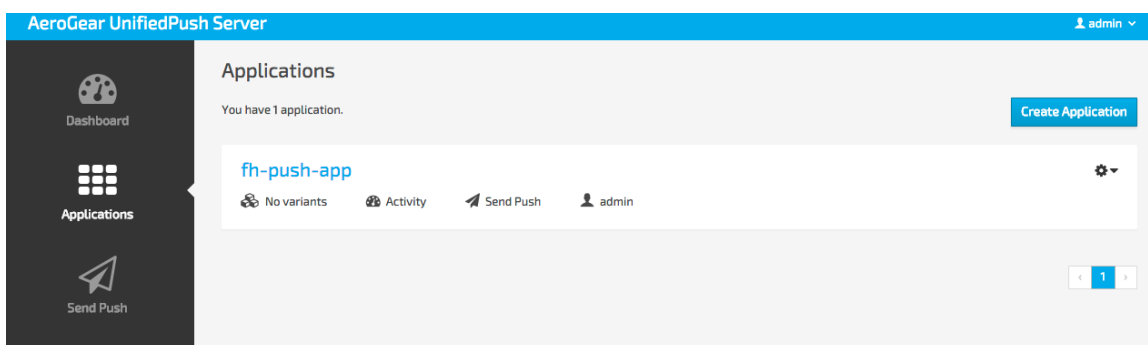
6. Once the application has been created, reveal its details by clicking on its name in the list of applications in the middle of the screen.

7. Note the following fields since this information will be used later:

    ▷ Server URL

    ▷ Application ID

    ▷ Master Secret

8. You will also need to set up a *Variant* for each targeted platform (for example, iOS or Android). For more details on creating a *Variant*, see the AeroGear documentation for iOS and documentation for Android. After setting up variants, you will need to note down the following information for each specific variant since it will be used later:
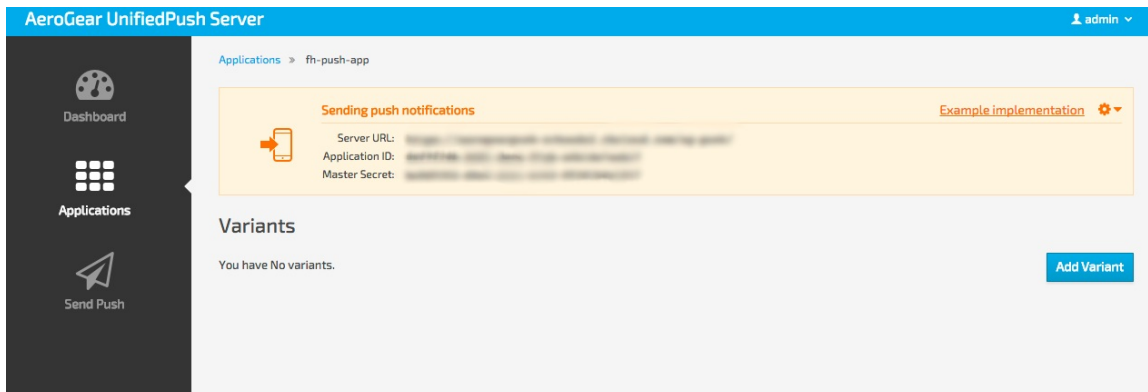
    ▷ Server URL

    ▷ Variant ID

    ▷ Secret

    ▷ Project Number (Android only, used for *senderID*)
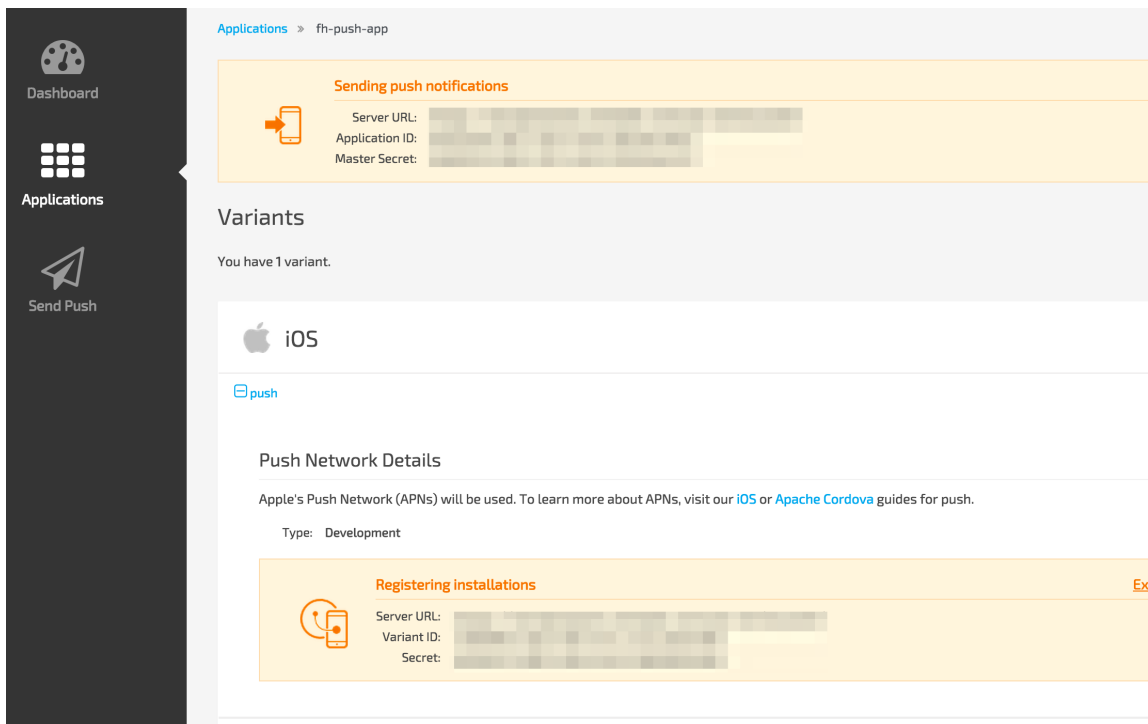
5.4.3. Create a Push Connector instance

In order to connect the Push Cloud App to UPS, a *cloud service* must be instantiated in RHMAP and configured to connect to the UPS instance created in the previous step.

1. Click on *Services and APIs* at the top of the screen.
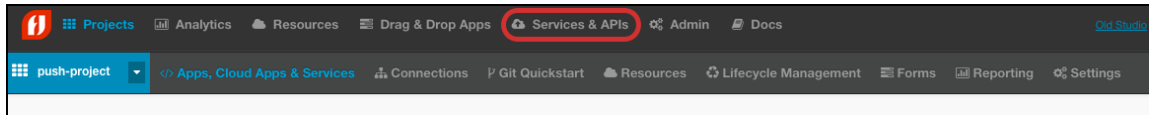
2. Click on the *Provision MBaaS Service/API* button at the top left of the screen.
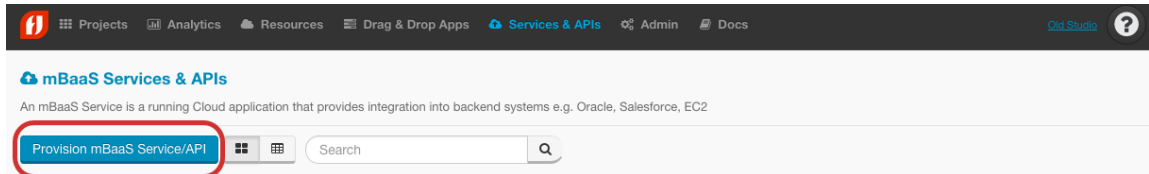
3. Locate *AeroGear Community Push Connector* from the list and click the *Choose* button on the right of the screen.
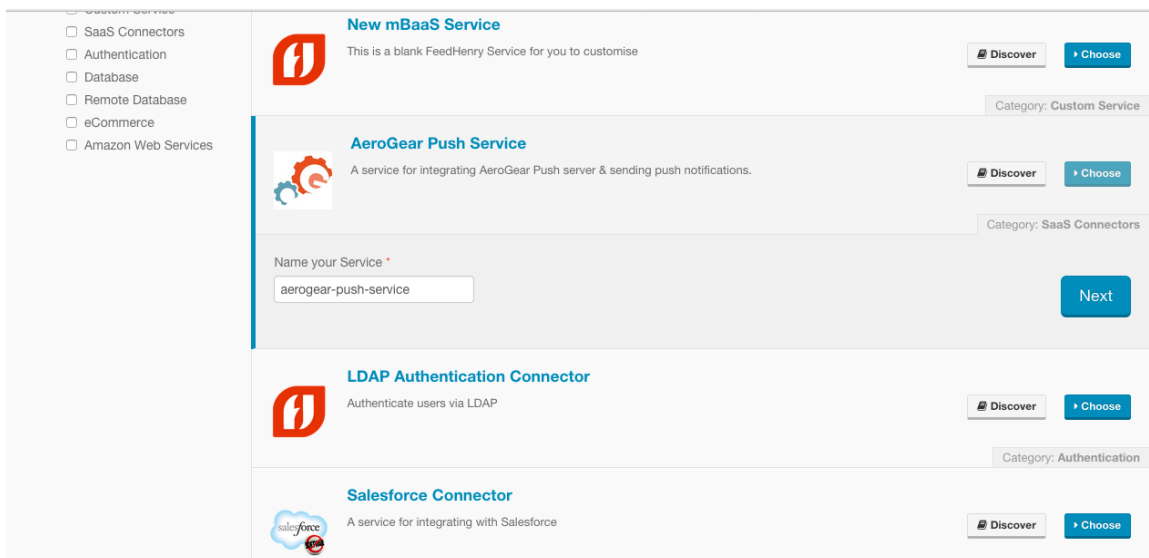
4. Enter a name and click *Next* on the right side of the screen.

5. Enter the *Server URL*, *Application ID* and *Master Secret* from step 2.7 and click the *Next* button.

6. Once the creation process is complete, click on the *Finish* button.

5.4.4. Associate the Push Connector with your project

After the Push Connector has been created and configured, your project must be configured to use it.

1. Select the *Projects* tab located at top-left of the screen.

2. Click on your project to open it.

3. Click on the + symbol in the *MBaaS Services* column on the right side of the screen.
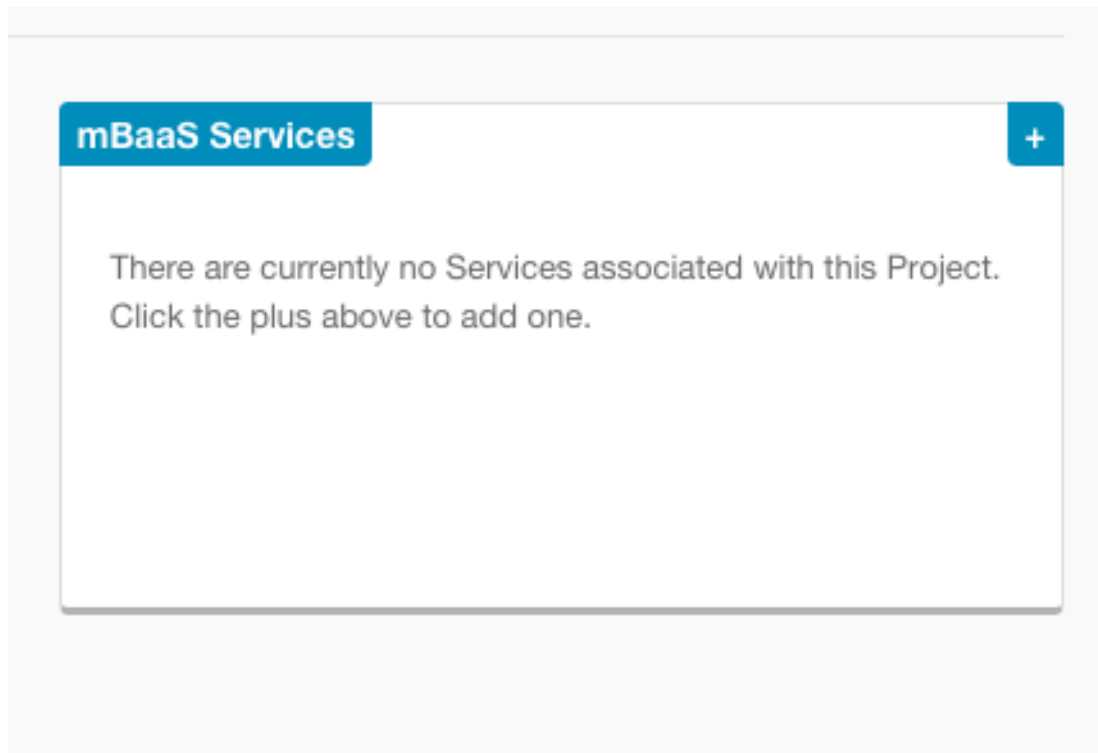
4. Click on the Push Connector instantiated in the previous step and then click on the *Associate Services* button at the bottom of the screen.

5. You will be returned to the *Apps, Cloud Apps & Services* page of your project and the Push Connector should appear in the *MBaaS Services* column.

5.4.5. Configure the Push Cloud App to use the Push Connector

1. If you are not already on the *Apps, Cloud Apps & Services* page of your project, navigate to it.

2. Click on the *Push Connector* in the *MBaaS Services* column to reveal its details.

3. Note the *Service ID* (GUID) of the *Push Connector* since it will be used later. Also, ensure that your project name appears under the *Access Control* section in the middle of the screen.

> **Note**
>
> Verify that your project has been added in the *Access Control* area. Failure to do so will result in the *Push Cloud App* being unable to send push notifications.

4. If any changes were made, click *Save Service* at the bottom of the screen.

5. Click on *Environment Variables* on the left side of the screen.

6. Click on *Push Environment Variables* on the right side of the screen and click*Confirm* when prompted.

7. Return to the *Apps, Cloud Apps & Services* page of your project.

8. Click on the *Push Cloud App* application under the *Cloud Code Apps* column in the middle of the screen.

9. Click on *Environment Variables* on the left side of the screen.

10. Click on *Add Variable* on the top right side of the *App Environment Variables* box.

11. Enter *AEROGEAR_SERVICE_GUID* for *Name* and paste the *Service ID* of the Push Connector (noted in step 3) for *Development*

12. Click the *Push Environment Variables* button on the top right side of the *App Environment Variables* box to publish the variables. Upon completion, *AEROGEAR_SERVICE_GUID* should appear in **both** the *App Environment Variables* section as well as the *Deployed Environment Variables* section.

> **Note**
>
> Ensure that both the *Service Connector* and *Push Cloud App* are running. The status for each of these may be found under the *Cloud App Status* field on the *Details* page of each. Both of these items must be deployed, running, and the environment variable pushed for push notifications to be properly sent.

5.4.6. Configure the mobile client

The *Push Notifications Mobile Client* app must be updated with the previously noted credentials for access to the UPS. These credentials authorize the client to register with the push server.

1. If you are not already on the *Apps, Cloud Apps & Services* page of your project, navigate to it.

2. Click on the *Push Notifications Mobile Client* under the *Apps* column on the left side of the screen.

3. Click on *Editor* on the left side of the screen.

4. Expand the `www` directory and click on the `push-config.json` file.
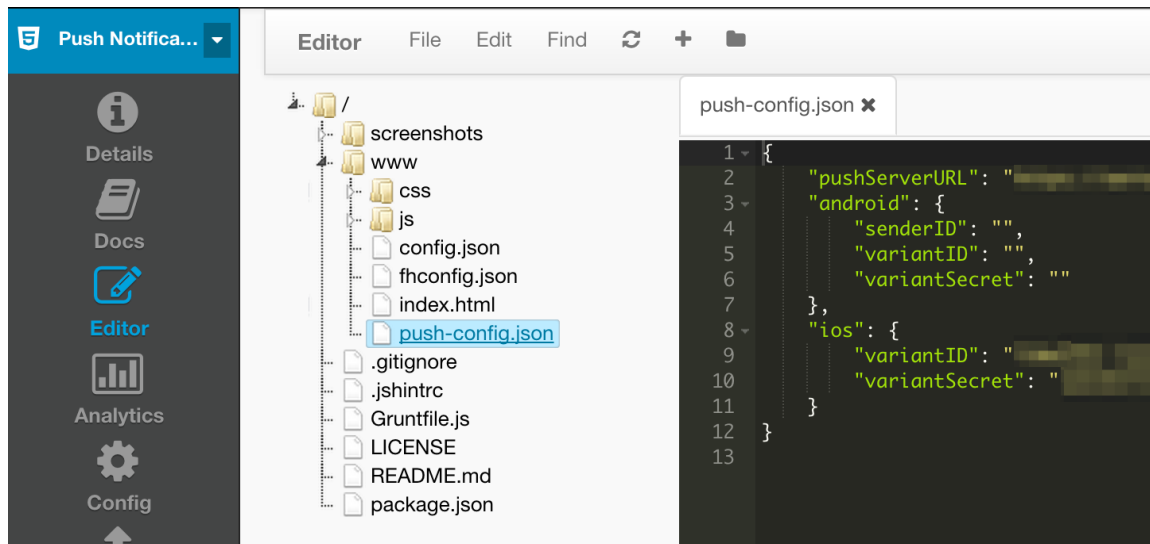
5. For each client platform (for example, *Android*, *iOS*), update the following fields using the appropriate variant from step 2.8:

   » for *pushServerURL* enter the *Server URL*

   » for *variantID* enter the *Variant ID*

   » for *variantSecret* enter the *Secret*

   » for *senderID* (Android only) enter the *Project Number*

6. In the *Editor* menu, click *File* then *Save* to save the changes.

5.4.7. Set up build credentials

This step is only required if you're building for iOS. If your target platform is Android, skip to .

5.4.7.1. Create a credential bundle

After you have obtained all necessary keys, certificates, and provisioning profiles from the mobile platform's developer portals, a *Credential Bundle* must be created to use them.

Note

These are the *Developer* private keys and certificates for the mobile platform (that is, iOS, Android, etc) which are **not** the same as the push certificate and private keys for the platform. For iOS, you will also need the *Provisioning Profile* for the application. If you did not already create one in step 2, see these instructions. For more details on obtaining the *Developer* private keys and certificates see each platform's documentation.

To create a credential bundle:

1. If you are not already on the *Apps, Cloud Apps & Services* page of your project, navigate to it.

2. Click on the *Push Notifications Mobile Client* under the *Apps* column on the left side of the screen.

3. Click on *Credentials* in the menu on the left side of the screen.
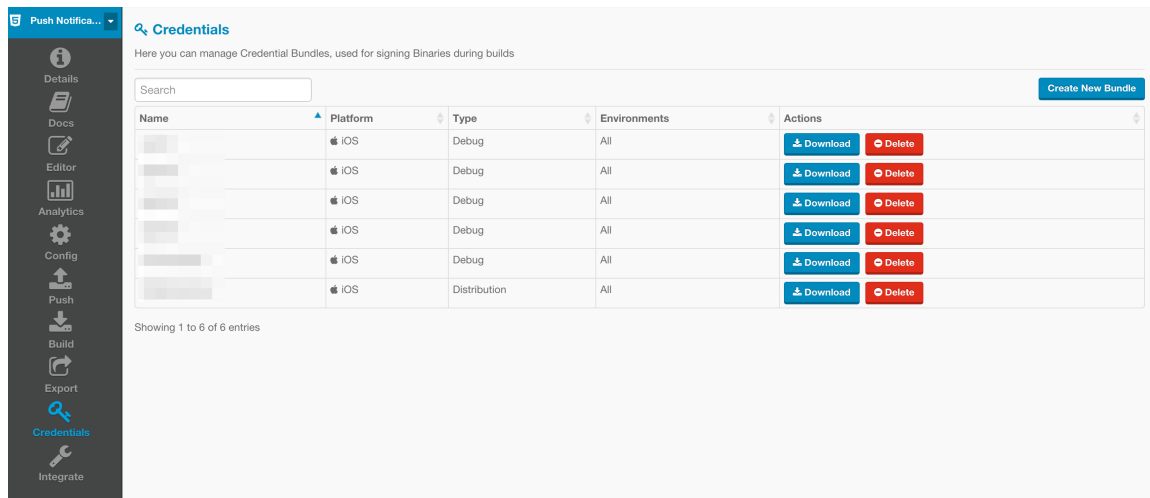
4. Click on the *Create New Bundle* button on the right side of the screen

5. Choose the desired platform, enter the relevant information, and upload any profiles, keys, or certificates.
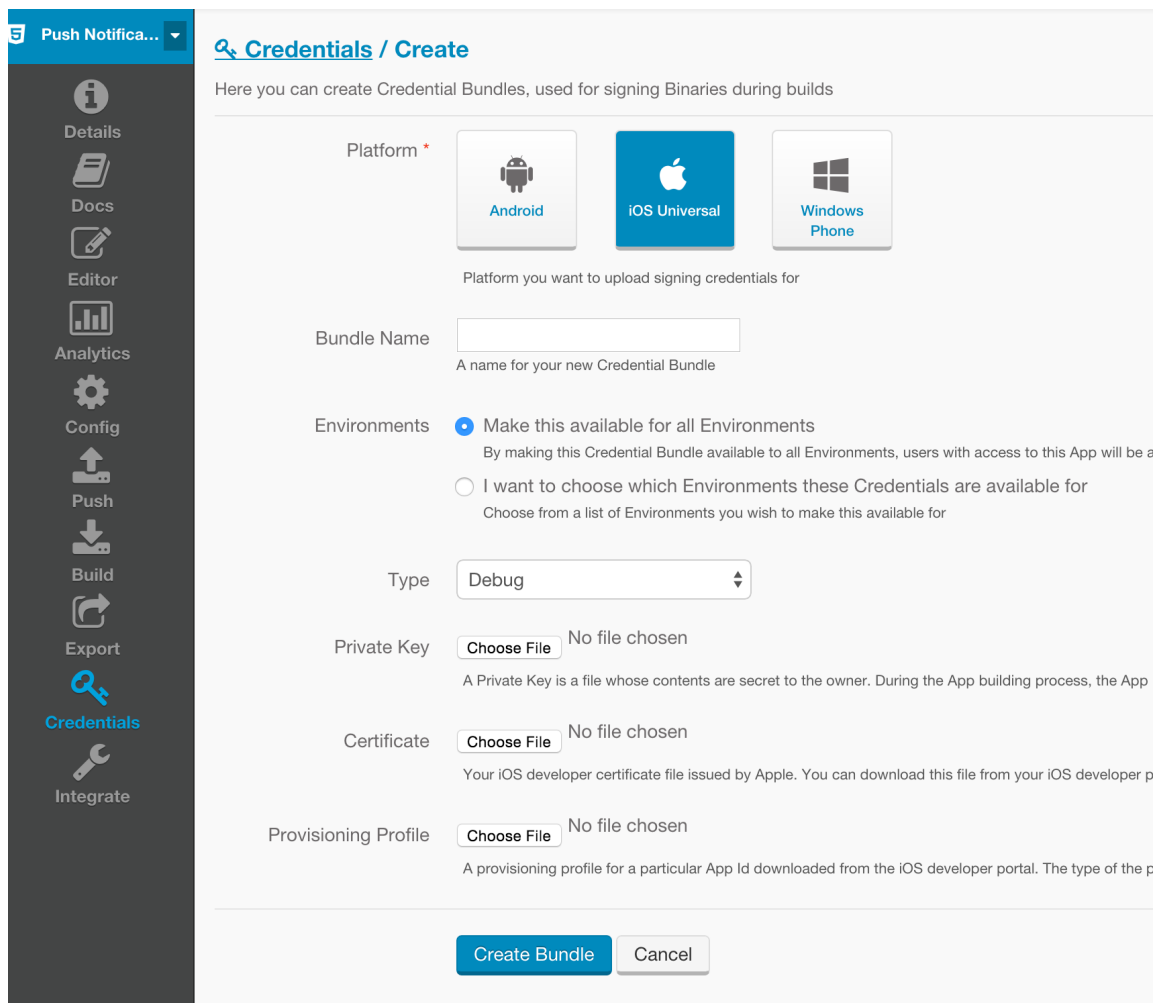
6. Click the *Create Bundle* button at the bottom of the screen.

Note

Ensure that you upload the correct profiles, keys, and certificates intended for the mobile client. In addition, if any of those items are secured with passwords, you will need to enter that information when building the client using any credential bundle with that secured item. If *any* of this information (profiles, keys, certificates, passwords, etc) is incorrect or not properly configured, your mobile client may not build or install on your device.

5.4.7.2. Set up the Bundle Identifier or Package Name

Before building, depending on the desired platform, the *Bundle Identifier* or *Package Name* information must be set.

To set the *Bundle Identifier* or *Package Name*:

1. If you are not already on the *Apps, Cloud Apps & Services* page of your project, navigate to it.

2. Click on the *Push Notifications Mobile Client* under the *Apps* column on the left side of the screen.

3. Click on *Config* in the menu on the left side of the screen.

4. Choose the desired platform (for example, iOS, Android, etc) from the menu on the left side of the screen.
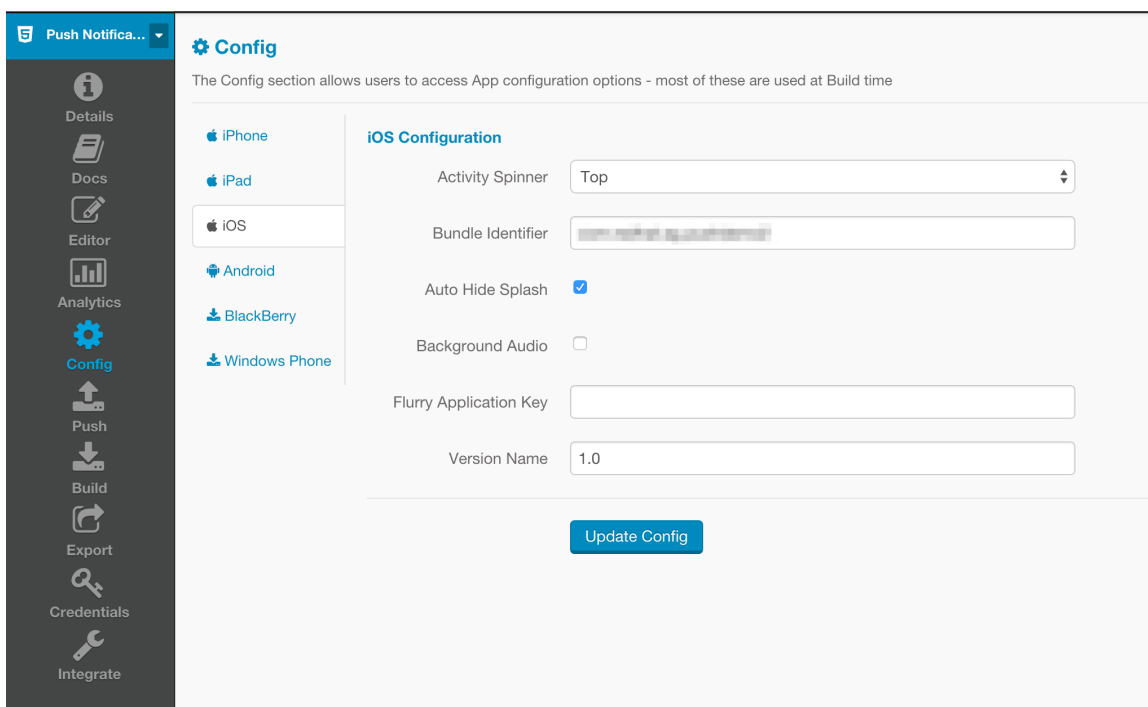
5. Enter the *Bundle Identifier* or *Package Name* in the appropriate text area and click the *Update Config* button at the bottom of the screen.

> **Note**
>
> Depending on the platform, you may need to set the same Bundle Identifier or Package Name in multiple locations. For example, depending on how an iOS app is built, the Bundle Identifier may need to be set in a combination of the iPhone, iPad, and/or iOS sections. When setting the Bundle Identifier or Package Name in multiple locations, you will need to click the *Update Config* button on **each** screen before moving to the next location. If you change locations *before* clicking *Update Config*, your updates will be lost.

5.4.8. Build the app

Once all the certificates and credentials have been configured, the *Push Notifications Mobile Client* may be built for a platform.

To build the *Push Notifications Mobile Client* for a platform:

1. If you are not already on the *Apps, Cloud Apps & Services* page of your project, navigate to it.

2. Click on the *Push Notifications Mobile Client* under the *Apps* column on the left side of the screen.

3. Click on *Build* in the menu on the left side of the screen.

4. Choose the desired platform.

5. Ensure that the proper *Build Type* is configured.

6. On certain platforms, other fields (for example, *Credential Bundle*, *Private Key Password*, etc.) may also need to be entered.

7. Click the *Build* button at the bottom of the screen.

> **Note**
>
> If any item in the credential bundle being used is secured with a password, it must be entered when performing the build.

8. Wait until the app is built. You can monitor progress at the bottom of the screen in the *Artifact History* section.

9. Once the build is finished, click *Download* in the first line of the table.

10. With your mobile device, scan the presented QR code and install the app.

5.4.9. Test the app

At this point, the *Push Notifications Mobile Client* is built for the desired platforms and is ready to receive push notifications sent from the *Push Console App*, which uses the *Push Cloud App* connected through the *Push Connector* to the UPS deployed on OpenShift.

To test if this is working:

1. Navigate to the *Apps, Cloud Apps & Services* page of your project.

2. Click on the *Push Console App* application under the *Apps* column.

3. Use the *App Preview* box on the right side of the screen to test the application. You could also navigate to the URL listed next to *Current Host*.
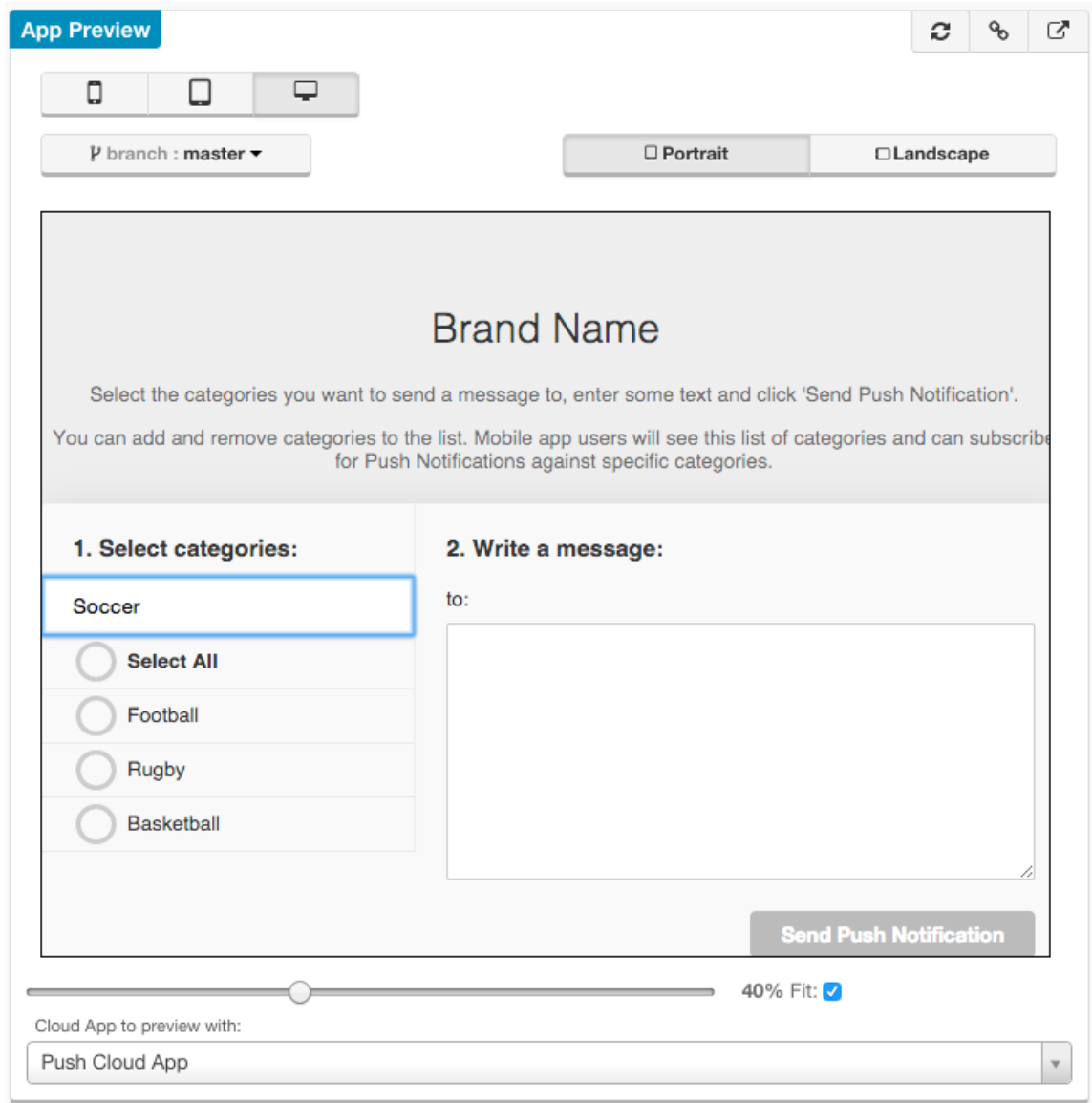
4. Create a couple of categories (for example, Football, Rugby and Basketball).

5. Open the application on the device and select the categories you want to receive.

6. In the studio, navigate to the *Apps, Cloud Apps & Services* page of your project.

7. Return to the *Push Console App* and use the *App Preview* box (or visit the *Current Host* URL) to create a news story and associate it with a category you subscribed to on your mobile device.
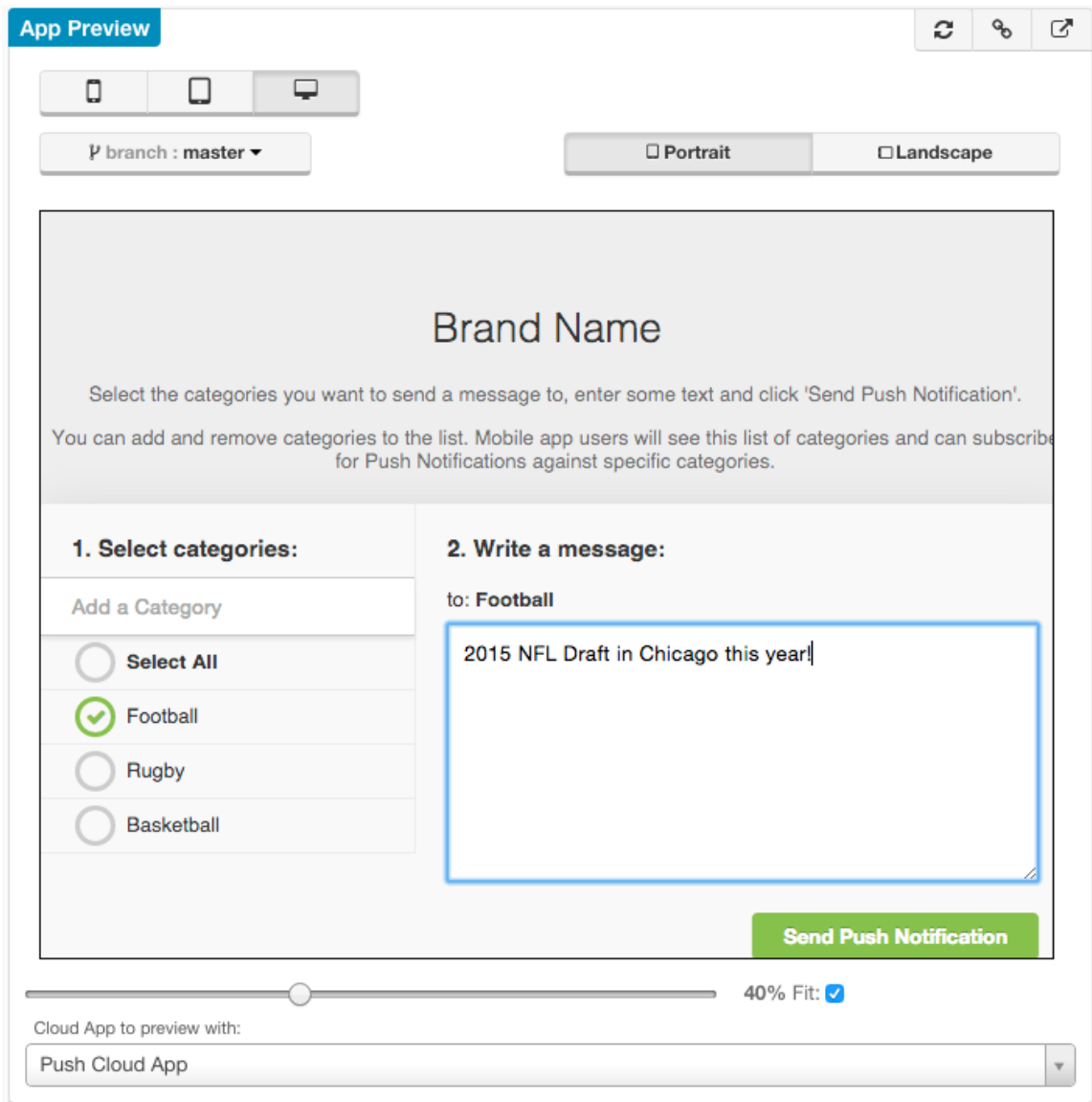
8. Click *Send Push Notification* and your mobile device should receive a push notification shortly.
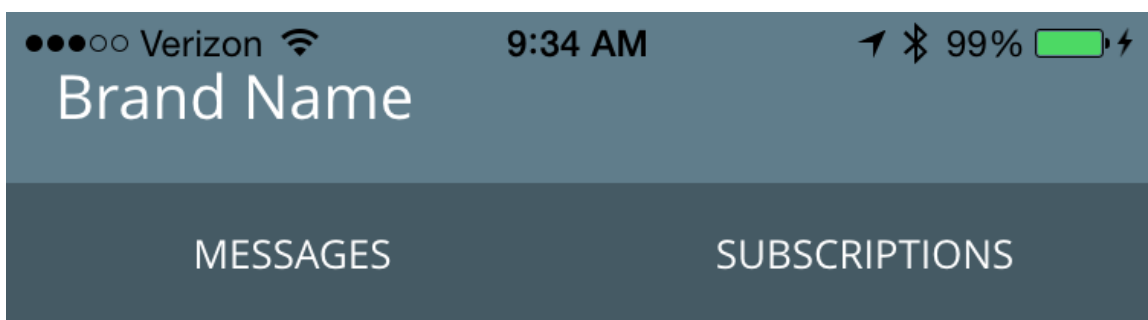
Note

In addition to building this example exercise, this template may also be used as a starting point to create your own application that integrates with UPS.

5.4.10. Common Issues

5.4.10.1. Checking the Logs

If you are seeing issues during testing, review the RHMAP and UnifiedPush Server (in OpenShift Online) logs for more details. The *Push Cloud App* and *Push Console App* logs can be found by navigating to each app in the project and clicking on *Logs* on the left side of the screen. To access the logs in OpenShift Online, consult the Openshift Online documentation. For more information on debugging UPS, see Debugging the UnifiedPush Server.

5.4.10.2. Errors being logged in *Push Cloud App* and push messages not reaching

client

If push messages are being sent from the *Push Console App*, but not reaching the client apps, check the logs in the *Push Cloud App* for the following:

```
AeroGear Community Push Connector - err =  null  :: data =  code=401,
message=You do not have permission to access this service.
```

This is most likely the result of the project not being added to the *Access Control* area of the *Service Connector*. Make sure you have completed Step 5, all environment variables have been pushed, and that **both** the *Service Connector* and *Push Cloud App* are deployed and running. In the *Service Connector*, try updating the *Access Control* area, saving the changes, re-pushing the environment variables, and restarting the service. Additionally, you could try re-pushing the environment variables for the *Push Cloud App* as well as restarting it.

5.4.10.3. Mobile client build failing due to *No matching provisioning profiles found*

If your client app is failing to build with the following error message:

```
[BUILD] 'Code Sign error: No matching provisioning profiles found: No
provisioning profiles matching the bundle identifier
"com.feedhenry.fhPushNotificationsMobileClient" were found.'
```

you may not have your Bundle IDs properly configured. Revisit section Set up the Bundle Identifier or Package Name and ensure:

- your bundle ID is correct and matches what you set up in the platform's (for example, Apple, Google, etc.) developer portal,

- your bundle ID is specified for **every** platform that you're building for (for example, make sure that your bundle ID is set in *iOS*, *iPhone*, and *iPad* if building for those platforms).

5.5. USING CORDOVA PLUG-INS

Cordova plug-ins provide a platform-independent JavaScript interface to mobile device capabilities in hybrid mobile apps. There are several official Apache plug-ins for basic device functions such as storage, camera, or geolocation, and other third-party plug-ins. For example, the RHMAP Push Notification functionality in Cordova apps is implemented using the **aerogear-cordova-push** plug-in.

The official registry and distribution channel for Cordova plug-ins is npm and the plug-in ID corresponds to the npm package ID. The Cordova Plugins page contains the official plug-in list and acts as a filter for npm packages with the keyword **ecosystem:cordova**.

5.5.1. Supported Platforms

You can use Cordova plug-ins with all platforms supported by RHMAP. However, not all Cordova plug-ins support all platforms. Platforms supported by a plug-in are specified in a plug-in's **package.json** file in the **cordova.platforms** object, or on the Cordova Plug-ins search page.

5.5.2. Adding Plug-ins to Apps

When developing Cordova applications, plug-ins are added using **`cordova plugin add`**, which downloads the plug-in from the repository, creates the necessary folder structure, and adds an entry in the **`config.xml`** file. See Platforms and Plugins Version Management in the official Cordova documentation for more information.

In RHMAP, however, plug-ins can also be declared in an RHMAP-specific JSON file **`config.json`**, which allows Cordova Light apps to declare plug-ins, and makes the apps future-proof in case the plug-in specification format changes. The necessary changes can be implemented once in the RHMAP Build Farm instead of forcing all developers to update their apps.

```
{
  "plugins": [
    {
      "id": "cordova-plugin-device",
      "version": "latest"
    },
    {
      "id": "cordova-plugin-geolocation",
      "version": "1.0.1"
    },
    {
      "id": "cordova-labs-local-webserver",
      "url": "https://github.com/apache/cordova-plugins#master:local-
webserver",
      "version": "2.3.1",
      "preferences": {
          "CordovaLocalWebServerStartOnSimulator": "false"
      }
    }
  ]
}
```

Three plug-ins are specified in the above example: the Device plug-in and Geolocation plug-in, which are available through npm, and the Local WebServer plug-in which is available only from the indicated Github repository.

5.5.2.1. Specification

The **`config.json`** file must be located in the **`www`** folder of a Cordova app, or in the root of a Cordova Light app.

The file must contain a key called **`plugins`**, the value of which is an array of JSON objects, which can define the following properties:

» **id** (Required)

   This is the globally unique ID of the Cordova plug-in, which corresponds to its npm package ID. It can also be found in the plug-in's **`plugin.xml`** file, as described in the Cordova Plugin Specification.

» **version** (Required)

   The version of the plug-in to use. Corresponds to the npm package version. For plug-ins distributed through Git only, you can find the version of a plug-in in its **`plugin.xml`**.

For plug-ins distributed through npm, you can also use the **latest** value to always use the latest available version. We strongly advise you to use a specific version that is proved to work with your app, since a plug-in upgrade could break backward compatibility.

» **url**

The URL of a public Git repository, containing a valid Cordova plugin (contains a **plugin.xml** file).

The provided value for this field is used to download the plug-in, regardless of the values of the **id** and **version** fields. If this field is not provided, the plug-in will be downloaded from npm with the values of **id** and **version** fields.

You can also specify a particular Git ref and a path to the plug-in within the repository, as described in official Cordova CLI documentation.

- A Git ref object can be specified by appending **#<git-ref>** in the URL. We strongly recommend using a tag or other stable ref that is tested as working with your app. Using **master** as the ref could result in the plug-in code changing on every build and potentially breaking your application.

- Path to the directory containing the plug-in can be specified with **:<path>** in the URL.

For example, if we want to get a plug-in that resides in the **plugin** subdirectory in the **release** branch of the repository, the URL should have the following format:

```
https://example.com/example.git#release:plugin
```

After the plug-in is downloaded, its **plugin.xml** file will be parsed to make sure the plug-in ID matches the **id** field specified in the **config.json**.

» **preferences**

To provide plug-in configuration, use key-value pairs in the **preferences** object, like **CordovaLocalWebServerStartOnSimulator** in the example above.

» **variables**

Some plug-ins use variables in **plugin.xml** to parameterize string values. You can provide values for variables as key-value pairs in this field. See official documentation for more information on variables.

5.5.3. Considerations For Cordova Light Apps

To preserve the backward compatibility of Cordova Light apps, if a valid **config.json** file is not provided, a default set of plug-ins will be applied when building the app in the Build Farm. We strongly advise you to provide the **config.json** file and to specify only the Cordova plug-ins needed by your app. By doing that, you will be able to:

» Prevent unexpected changes of plug-ins affecting your apps.

The list of default plug-ins is not guaranteed to stay unchanged and any possible changes to it may not be backward-compatible. By providing your own **config.json** file, you ensure that your apps will always use a stable list of plugin versions, thus making builds more stable.

» Reduce app build time and size of binary.

Plug-ins are downloaded and installed during app build time, and included in the built binary. Therefore, reducing the number of plug-ins also reduces the build time and size of binary.

5.5.3.1. Default Plugins

Official plug-ins

- cordova-plugin-device (1.1.2)
- cordova-plugin-network-information (1.2.1)
- cordova-plugin-battery-status (1.1.2)
- cordova-plugin-device-motion (1.2.1)
- cordova-plugin-device-orientation (1.0.3)
- cordova-plugin-geolocation (2.2.0)
- cordova-plugin-file (4.2.0)
- cordova-plugin-camera (2.2.0)
- cordova-plugin-media (2.3.0)
- cordova-plugin-media-capture (1.3.0)
- cordova-plugin-file-transfer (1.5.1)
- cordova-plugin-dialogs (1.2.1)
- cordova-plugin-vibration (2.1.1)
- cordova-plugin-contacts (2.1.0)
- cordova-plugin-globalization (1.0.3)
- cordova-plugin-inappbrowser (1.4.0)
- cordova-plugin-console (1.0.3)
- cordova-plugin-whitelist (1.2.2)
- cordova-plugin-splashscreen (3.2.2)

Optional

- cordova-sms-plugin (v0.1.9)
- com.arnia.plugins.smsbuilder (0.1.1)
- cordova-plugin-statusbar (2.1.3)

Custom RHMAP plug-ins:

- com.feedhenry.plugins.apis (0.0.6)
- com.feedhenry.plugins.apkdownloader (0.0.1)
- com.feedhenry.plugin.device (0.0.2)

➤ com.feedhenry.plugins.ftputil

5.5.4. Testing Apps in the Browser

If a plug-in is specified in the **config.json** file, the JavaScript object of the plug-in won't be available until it is built and installed on the device. So if you reference the plug-in's JavaScript object in your app, and try to load your app in *App Preview* in the Studio, you may get *object undefined* errors.

To solve this, use defensive checking when calling the plug-in APIs. For example, the following call to the Cordova camera API would result in an "object undefined" error:

```
navigator.camera.getPicture(success, fail, opts);
```

However, checking whether the API is defined lets you handle the error gracefully:

```
if(typeof navigator.camera !== "undefined"){
  navigator.camera.getPicture(success, fail, opts);
} else {
  //fail gracefully
  fail();
}
```

5.6. USING SECURE KEYS IN YOUR APP

$fh.sec APIs provides the functionality to generate keys and data encryption/decryption. However, after the keys are generated, you may need to save them somewhere for future usage. For example, you have some data that needs to be encrypted with a secret key and saved on the device. Next time, when the app starts again, you need to get the same secret key and decrypt the data.

The best practice to achieve this is to save the keys on the cloud side, and associate the keys with the client using the client unique id (CUID) and app id.

5.6.1. CUID and App Id

Both the CUID and app id are sent by the client SDK in every $fh.act request. They are accessible via a special JSON object called __**fh**. The CUID is unique for each device and remain unchanged even if the app is deleted and re-installed. The app id is generated when the app is created on the platform and remain unchanged. You can use the following code to access them in the cloud code:

```
getKeyId = function(params){
  var cuid = params.__fh.cuid;
  var appid = params.__fh.appid;
  var keyid = cuid + "_" + appid;
  return keyid;
}

exports.getKey = function(params, callback){
  var keyid = getKeyId(params);
  //get a key using this keyid
  ....
}
```

```
      if(err) return cb(err);
      return cb(undefined, result);
    })
  } else {
    //a key with the same id and type is not found, create it
    $fh.db({
      'act': 'create',
      'type': 'securityKeys',
      'fields': {
        'id' : getKeyId(params),
        'keyType': params.type,
        'keyValue': params.value
      }
    }, function(err, result){
      if(err) return cb(err);
      return cb(undefined, result);
    });
  }
});
} else {
  console.log("$fh.db not defined");
  cb("$fh.db not defined");
}
}
```

5.6.3. Sample Code

A reference application has been created which fully demonstrates how to use **$fh.sec** APIs. The code for this application is available on GitHub: https://github.com/feedhenry-training/fh-security-demo-app.

5.7. ICONS AND SPLASHSCREENS

Apps can be installed on a number of different platforms including iOS, Android, and Windows Phone, and each has a different requirement for their app icons, and splashscreen and launch images. This topic outlines the naming conventions for icons and splashscreen images for each platform.

Note

This topic is for Cordova Light apps only. For Cordova apps, icons and splashscreen images can be configured in each platform's native projects.

5.7.1. Image File Locations

Place icons and splashscreen images in the following locations:

- **Icons**: **/www/res/icons/<platform_name>**

- **Splashscreens**: **/www/res/splash/<platform_name>**

> **Note**
>
> To reduce the size of the binary file, the **res/icons** and **res/splash** directories are removed once the image resources are copied to their correct locations. Do not reference files in those directories in your app.

5.7.2. iOS

iOS devices include a range of differing display types (retina/non-retina) and resolutions.

5.7.2.1. Icons

The following table displays the file names and paths to specific icon types expected by the build farm. Binaries (or exports) compiled for iOS look for icons in these locations to enable them. Note that the file names are case sensitive.

File name (case sensitive)	Target(s)	OS Version	Expected Location	Dimensions
Icon.png	iPhone (Non-Retina), iPad (Non-Retina)	iOS 7 or earlier	**/www/res/icons/ios/Icon.png**	57 x 57
Icon@2x.png	iPhone (Retina), iPad (Retina)	iOS 7 or earlier	**/www/res/icons/ios/Icon@2x.png**	114 x 114
Icon-72.png	iPad (Non-Retina)	iOS 7 or earlier	**/www/res/icons/ios/Icon-72.png**	72 x 72
Icon@2x~ipad.png	iPad (Retina)	iOS 7 or earlier	**/www/res/icons/ios/Icon@2x~ipad.png**	144 x 144
Icon-60@2x.png	iPhone (Retina)	iOS 7 or later	**/www/res/icons/ios/Icon-60@2x.png**	120 x 120
Icon-76.png	iPad (Non-Retina)	iOS 7 or later	**/www/res/icons/ios/Icon-76.png**	76 x 76

File name (case sensitive)	Target(s)	OS Version	Expected Location	Dimensions
`Icon-76@2x.png`	iPad (Retina)	iOS 7 or later	`/www/res/icons/ios/Icon-76@2x.png`	152 x 152
`Icon-60@3x.png`	iPhone 6 Plus	iOS 8 or later	`/www/res/icons/ios/Icon-60@3x.png`	180 x 180

Note

If none of the above icons are provided, a default set of icons is included automatically. To avoid that, provide at least one icon.

5.7.2.2. App Launch Image and Splashscreens

The following table displays file names and paths to specific app launch image types as expected by the build farm. Binaries (or exports) compiled for iOS look for app launch images in these locations to enable them. Note that the file names are case sensitive.

File name (case sensitive)	Target(s)	Expected Location	Dimensions
`Default.png`	iPhone 3.5" (Non-Retina)	`/www/res/splash/ios/Default.png`	320 x 480
`Default@2x.png`	iPhone 3.5" (Retina)	`/www/res/splash/ios/Default@2x.png`	640 x 960
`Default-568h@2x.png*`	iPhone 4" (Retina)	`/www/res/splash/ios/Default-568h@2x.png`	640 x 1136
`Default-Landscape.png`	iPad (Non-Retina)	`/www/res/splash/ios/Default-Landscape.png`	1024 x 768

File name (case sensitive)	Target(s)	Expected Location	Dimensions
`Default-Portrait.png`	iPad (Non-Retina)	`/www/res/splash /ios/Default-Portrait.png`	768 x 1004
`Default-Landscape@2x~ip ad.png`	iPad (Retina)	`/www/res/splash /ios/Default-Landscape@2x~ip ad.png`	2048 x 1496
`Default-Portrait@2x~ipa d.png`	iPad (Retina)	`/www/res/splash /ios/Default-Portrait@2x~ipa d.png`	1536 x 2008
`Default-667h.png`	iPhone 6	`/www/res/splash /ios/Default-667h.png`	750 x 1334
`Default-736h.png`	iPhone 6 Plus	`/www/res/splash /ios/Default-736h.png`	1242 x 2208
`Default-Landscape-736h.png`	iPhone 6 Plus Landscape	`/www/res/splash /ios/Default-Landscape-736h.png`	2208 x 1242

5.7.2.2.1. Supporting 4" Retina Displays (iPhone 5)

In order for your build to support iOS devices with 4" screens (for example, the iPhone 5), you must include a `Default-568h@2x.png` splashscreen image in your app.

5.7.2.2.2. Supporting iPhone 6 and iPhone 6 Plus

To avoid app zooming on iPhone 6 and iPhone 6 Plus, include `Default-667h.png`, `Default-736h.png`, and `Default-Landscape-736h.png` splashscreen images in your app.

5.7.2.3. Icons for Spotlight and Settings (Optional)

The following table displays file names and paths to specific icons for Spotlight or settings as expected by the build farm. Binaries (or exports) compiled for iOS look for icon files in these locations to enable them. Note that the file names are case sensitive.

File name (case sensitive)	Target(s)	OS Version	Expected Location	Dimensions
`Icon-small.png`	iPhone (Non-Retina)	iOS 7 or earlier	`/www/res/icons/ios/Icon-small.png`	29 x 29
`Icon-small@2x.png`	iPhone (Retina)	iOS 7 or earlier	`/www/res/icons/ios/Icon-small@2x.png`	58 x 58
`Icon-50.png`	iPad (Non-Retina)	iOS 7 or earlier	`/www/res/icons/ios/Icon-50.png`	50 x 50
`Icon-50@2x.png`	iPad (Retina)	iOS 7 or earlier	`/www/res/icons/ios/Icon-50@2x.png`	100 x 100
`Icon-40.png`	iPad (Non-Retina)	iOS 7 or later	`/www/res/icons/ios/Icon-40.png`	40 x 40
`Icon-40@2x.png`	iPad (Retina), iPhone (Retina)	iOS 7 or later	`/www/res/icons/ios/Icon-40@2x.png`	80 x 80

5.7.3. Android

5.7.3.1. Icons

The Android platform allows specifying image resources using qualifiers to provide the best possible image resources for a particular device. To support that, you can provide your app's image resources using qualifiers as well:

File name (case sensitive)	Expected Location	DPI	Recommended Dimensions
`icon.png`	`/www/res/icons/android/icon.png`	auto scale	48 x 48
`icon-ldpi.png`	`/www/res/icons/android/icon-ldpi.png`	~120dpi	36 x 36
`icon-mdpi.png`	`/www/res/icons/android/icon-mdpi.png`	~160dpi	48 x 48
`icon-hdpi.png`	`/www/res/icons/android/icon-hdpi.png`	~240dpi	72 x 72
`icon-xhdpi.png`	`/www/res/icons/android/icon-xhdpi.png`	~320dpi	96 x 96

Note

If none of the above icons are provided, a default set of icons will be included automatically. To avoid that, provide at lease one icon.

5.7.3.2. Splashscreen

Similar to icons, you can use qualifiers for splashscreen images as well.

File name (case sensitive)	Expected Location	DPI	Recommended Dimensions
`splash.png`	`/www/res/splash/android/splash.png`	auto scale	320 x 480

File name (case sensitive)	Expected Location	DPI	Recommended Dimensions
`splash-ldpi.png`	`/www/res/splash/android/splash-ldpi.png`	~120dpi	240 x 320
`splash-mdpi.png`	`/www/res/splash/android/splash-mdpi.png`	~160dpi	320 x 480
`splash-hdpi.png`	`/www/res/splash/android/splash-hdpi.png`	~240dpi	480 x 800
`splash-xhdpi.png`	`/www/res/splash/android/splash-xhdpi.png`	~320dpi	640 x 960

To make the same image fit different screen sizes, you can use nine-patch images. Follow the same conventions above to use nine-patch images, but use `.9.png` as the extension.

You can use other qualifiers as part of the file name, for example, `icon-tvdpi.png`, `icon-small.png`, `icon-normal.png`, etc. More details about the available qualifiers can be found at Providing Resources.

5.7.4. Windows Phone

5.7.4.1. Icons

Only one small icon and one tile icon can be specified for the Windows Phone platform:

File name (case sensitive)	Expected Location	Recommended Dimensions
`icon.png`	`/www/res/icons/windows phone/icon.png`	62 x 62
`icon-tile.png`	`/www/res/icons/windows phone/icon-tile.png`	173 x 173

> **Note**
>
> If none of the above icons are provided, a default set of icons will be included automatically. To avoid that, provide all icons.

5.7.4.2. Splashscreen

File name (case sensitive)	Expected Location	Recommended Dimensions
`splash.jpg`	`/www/res/splash/windowsphone/splash.jpg`	480 x 800

5.7.4.3. Additional Documentation

» Apple Developer Documentation - App Icon

» Apple Developer Documentation - Launch Files

» Android Providing Resources Guide

5.8. DEBUGGING APPS

Debugging is an essential part of app development. With web applications this can be as simple as using the **alert()** function to show the value of a variable at a certain point in the code. For browsers that have a console, the **console.log** function can be used to log this kind of information passively. More advanced forms of debugging web applications include inspecting the state of the DOM, breakpointing JavaScript code and stepping through it, or updating the DOM and the associated styles on the fly.

This page gives an explanation of the debugging tools that can be used while developing cross platform apps. These tools can be used in the Studio or on device.

5.8.1. Studio console

Most browsers support the console logger. Various log levels can be called with a message for example,

```
console.log(message);
console.error(message);
```

To debug this console output, you will need to open the relevant web debugging tools in your browser.

For debugging cloud code, simply use **console.log()** and **console.error()**. The corresponding log files can be viewed in the Studio under the 'Logs' section, or using FHC.

```
console.log('this goes to stdout.log');
console.error('this goes to stderr.log');
```

5.8.2. Firebug

If your browser of choice for development is Firefox, the http://getfirebug.com/[Firebug^] tool makes debugging very easy. Firebug is an add-on for Firefox that is quite active and has many updates to it. Here are some of the things that can be done using Firebug:

» view console output

» view resource requests

» debug script execution

» dynamically run code, even when breakpoint debugging

» view/update the DOM

» view/update styles

» view/update local storage (DB)

Using these features, debugging an app is made very easy. Getting an app working without Firebug showing any errors or problems gives the app a good chance of working cross platform.

5.8.3. Web Inspector with Chrome

Web Inspector is very similar to Firebug. It offers more or less the same features as Firebug, but has the advantage of being included with WebKit browsers out of the box. This means the Web Inspector can be used with Google Chrome and Safari. The tool is enabled by default with Chrome and can be started by context clicking any object on a web page, and selecting **Inspect Element**.

5.8.4. Safari / iOS 6+

Note

Remote debugging on iOS with Safari can only be enabled for applications built with a Development provisioning profile. Ad Hoc and Distribution apps cannot be debugged this way.

1. Go to the **Settings app** on your iOS device.

2. Navigate to **Safari → Advanced**, and then toggle on the **Web Inspector** switch.

3. In desktop Safari, go to **Safari** → **Preferences**, select **Advanced**, then check the **Show Develop menu in menu bar** check box.
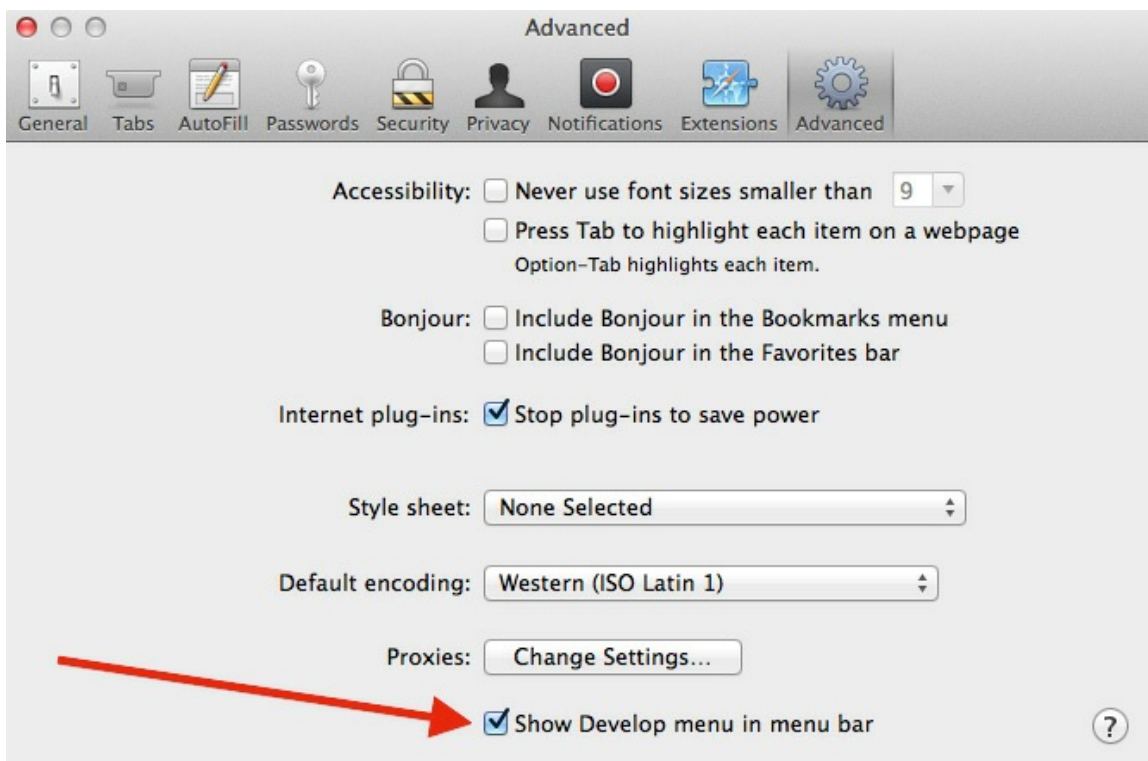
4. Connect your iOS device to your development machine via USB.

5. Open the app you want to debug, it will be available in Safari's **Develop** menu.
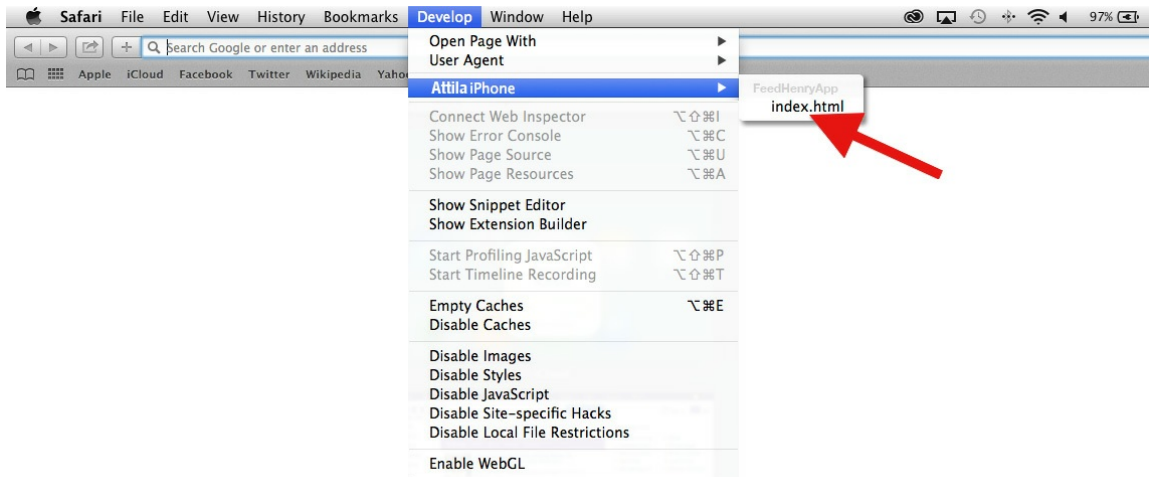
6. The **Develop** menu has additional tools such as the **User Agent** switcher. This allows the browser to pretend to be a different browser for example, Mobile Safari. This feature can be useful if developing a mobile Internet version of an app.

5.8.5. Chrome / Android 4.0+

Note

Remote debugging on Android with Chrome can only be enabled for applications that flagged debuggable. If you are using Cordova 3.3 or newer, add `android:debuggable="true"` inside the `<application>` element in the `AndroidManifest.xml` file. If you are using Cordova 3.2 or older version, you have to enable WebView debugging.

1. Enable USB debugging on your Android device On Android 4.2 and newer versions, the developer option is hidden. To make it available, navigate to **Settings** → **About phone**, and tap the **Build number** seven times.

2. Go back to the previous screen to find **Developer options**, then check the **USB debugging** check box. Choosing the **Stay awake** option is also recommended.

3. In the desktop Chrome browser, in the menu on the upper-right corner select **Tools** → **Inspect Devices**

4. Check **Discover USB devices**, if it is not selected.

5. Connect your Android device via USB

6. If it is the first time you attached this device for developing, you may see an alert on device requesting permission for USB debugging from your development machine. To avoid appearing this alert every time you debug, check **Always allow from this computer**, then tap OK.

7. To debug your app, you need to run it on device.

8. In Chrome, click the **Inspect** link to open the DevTools.

5.8.6. On-Device

Debugging on device is still in an early stage compared to debugging in a desktop browser. Some platforms, such as iOS, make the console available and show any javascript errors when Developer mode is enabled, but it is nowhere near as fully featured as Firebug or Web Inspector.

5.8.6.1. Weinre

Weinre is probably the best on-device debugging tool available at the moment. It officially only supports webkit browsers. This means it will work for Android, iOS and Blackberry. The full list and version numbers are available on its site.

Weinre works by setting up a remote debugging session from the app on the device to the Weinre server. Weinre runs a web server which can be used to access the Web Inspector, and remotely debug the app. At the time of writing, the features of this Web Inspector session allow debugging the DOM and updating it. The remote console is also shown, and the developer can dynamically run code in the app from the console.

To enable Weinre debugging in an application, there are a few steps.

1. Get the Weinre jar up and running on a machine that the device can connect to. There are 2 ways of doing this:

   ▹ Run the jar file on a machine accessible over the Internet

   ▹ Run the jar on a machine on the same network as the device, that is, device connected via WiFi to the same router/access point as the developers machine.

   Note that when running the jar from the command line, it is advised to use the value **-all-** for the **boundHost** so that it is listening on all interfaces.

2. Add a script to the HTML of the app before building and deploying it to the device. According to the documentation, the script include will look something like this.

   ```
   <script src="http://some.server.xyz/target/target-script-
   min.js#anonymous"></script>
   ```

   The address used, **some.server.xyz**, must match the address (ip address should work too) of the machine that is running the Weinre server.

3. Deploy the app to the device and launch it. Opening the web page of the Weinre server (on the developer machine) should present a link to the debug user interface. This link opens up the Web Inspector and allows remote debugging of the app.

   As these instructions are for a third party tool, it is best to check with the official site for any updates around this setup process.

CHAPTER 6. PUBLISHING APPS

6.1. SUBMITTING AN APP TO GOOGLE PLAY

6.1.1. Prerequisites

❯ A Release build of an app (APK file) is generated (either from the Platform or from the Android IDE/Command line tools)

❯ The developer has a high resolution application icon file

  ▫ 512x512 JPEG or 24 bit PNG (no Alpha)

❯ The developer has at least one screenshot of the app

  ▫ 320x480 or 480x854 JPEG or 24 bit PNG (no Alpha) format

Note

Screenshots must be full bleed, have no border and landscape thumbnails will be cropped. All measurements WxH for example, 320wx480h

6.1.2. Overview

There are a number of steps for publishing an app to Google Play, all of which are done with a web browser. They are:

❯ Upload an Application

  ▫ Upload Assets

  ▫ Listing Details

  ▫ Publishing Options

  ▫ Contact Information

  ▫ Consent

6.1.3. Uploading an Application

❯ Log into Google Play at https://play.google.com/apps using your Developer Login

  ▫ You will be presented with a list of apps you already have in Google Play

  ▫ Click on the Upload Application button

  ▫ You will now be brought to the **Upload an Application** Screen

6.1.4. Upload Assets

❯ Draft application APK file

  ▫ Upload your APK file

» Screenshots

■ Upload your screenshot(s) -- JPEG or 24-bit PNG (Alpha Transparency is unsupported)

» High Resolution Application Icon

■ Upload your App Icon — JPEG or 24-bit PNG (Alpha Transparency is unsupported)

» Promotional Graphic

■ This is optional to be used in Google Play on devices with 1.6 or higher

» Feature Graphic

■ This is optional

» Promotional Video

■ This is optional to supply a YouTube URL to your promotional video

» Marketing Opt-out

■ There is also the option to opt out of Google marketing/ promoting your app

6.1.5. Listing Details

» Language

■ Select the Language for your app

» Title

■ The name for your application as it will appear in Google Play

» Description

■ Enter the description of your to appear in Google Play

» Recent Changes

■ Enter the changes as you are uploading updates of your application

» Promo text

■ This will be used in conjunction with the Promotional Graphic

» Application Type

■ Select the type of application it is "Applications" / "Games"

» Category

■ Select the category you want the app to be displayed in based on choice of application type

■ **Applications**

■ Comics

■ Communication

■ Entertainment

- Finance

- Health

- Lifestyle

- Multimedia

- News & Weather

- Productivity

- Reference

- Shopping

- Social

- Sports

- Themes

- Tools

- Travel

- Demo

- Software libraries

- **Games**

  - Arcade & Action

  - Brain & Puzzle

  - Cards & Casino

  - Casual

» Price

- Price is automatically set to **Free**. If you want to charge for apps you must set up a merchant account at Google Checkout.

6.1.6. Publishing Options

» Copy Protection

- Protect apps being copied from the device - **Soon to be deprecated and superceded by the Licensing Service**

» Locations

- Select Google Play markets you want the app to appear in

6.1.7. Contact Information

» Website

- Your website URL

» Email

- Your Email address

» Phone

- Your phone number

6.1.8. Consent

You must verify that the application meets the Android Content Guidelines and that your app may be subjected to US laws regardless of your location or nationality and that, as such, your application is authorised for export from the US under these laws.

Finally, You must choose to Save what you have entered, delete what you have entered or Publish the application (This will publish the app and make it available in Google Play)

6.2. SUBMITTING AN APP TO THE APP STORE

Prerequisites

This guide assumes that a number of things have already been done by the developer. They are:

» A 'Release' App has been build either form the Platform or using xCode

» The developer has 2 icon files for the application

- 57x57 PNG format

- 512x512 PNG format

» The developer has at least one screenshot of the app 320x480 PNG format

Overview

There are a number of steps for publishing an app to the iPhone App Store, all of which are done with a web browser. They are:

» Create a new Application

» Set up any in-app purchasing

» Submit Application for Sale

6.2.1. Creating a new Application

1. Log into iTunes Connect at http://itunesconnect.apple.com using your Developer Login (that is, email address)

   a. Click on Manage Your Applications

   b. Click on Add New Application

c. Select a choice for the "Export Laws and Encryption" question

2. Overview Tab

a. Fill in the Overview section using your app details

b. Select a choice for "Restrict this binary to a specific platform" question

c. Enter a Version Number for example, 1.0

d. Enter a unique value for the SKU Number: for example, the current date and time

e. Click the (blue) Continue button at the end of the screen.

3. Ratings Tab

a. Select the appropriate check boxes for you application

b. Click (blue) Save Changes button

4. Upload Tab

a. Click the Upload application binary later option

b. Upload the icons and screen shots mentioned in the Prerequisites (Large Icon and Primary Screen Shot are mandatory). *TIP: When adding additional screen shots, you may like to select them in reverse order so that they appear in the correct order when added by the uploader (which seems to add/upload them in the opposite order that you selected them).*

c. Click the (blue) Continue button

5. Localization Tab

a. Choose the appropriate language for you app

b. Click the Continue button

6. Pricing Tab

a. Choose the Availability Date (Start Date) of your app for example, today

b. Choose the appropriate Price Tier

c. Select the stores you want your app to be available in and continue

7. Review Tab

a. Choose the appropriate review store (for example, the country your company resides in)

b. Click the (black) Submit Application button.

6.2.2. Submit Application for Sale

1. Upload Application Binary

    a. Once logged into iTunes Connect, click Manager your Applications

    b. Click the application you want to submit for approval by Apple (Note the current status should be orange, indicating the Application binary is yet to be uploaded)

    c. Click the Upload Binary button

    d. Click Choose File and locate your binary zip file mentioned in the Prerequisites.

    e. Click Upload File

    f. Click the Save Changes button in the bottom right

2. Confirm Review App Store

    a. Click Manager your Applications

    b. Click your application that was submitted for approval

    c. Click the Edit Information button and open the Review Tab

    d. Verify that the Review App Store is set to the country selected while creating the application (necessary due to a suspected bug in iTunes Connect)

    e. Click Done and your application is now ready for approval by Apple

6.2.3. External Links

In depth information on managing apps

6.3. APP CREDENTIALS BUNDLES

A Credentials Bundle consists of a number of resources needed to perform a particular build. Here, the different resources are listed, along with a brief explanation of their purpose.

6.3.1. Resources

When performing build operations, a Credentials bundle can sometimes be required (depending on the build). A Credential bundle is a combination of resources, such as certificates, provisioning profiles, and private keys, necessary for performing specific types of builds, be it a development build, distribution build, debug build etc. Depending on both the platform, and the build type, different resources will be grouped together to constitute a bundle.

Listed below is a breakdown of resources that can be added to a Credentials Bundle, along with a brief description of what they are used for.

6.3.1.1. All Platforms

» Private Key

    ▫ This is a file whose contents are known only to the owner. During the app building process, the app is digitally signed using this key. This means the developers digital signature is left on the App, allowing the App to be tied back to the developer.

6.3.1.2. Android Only

» Android Distribution Certificate (Android only)

 ▫ Used to build Apps for upload to the Google Play Store. This certificate is used to identify you as the developer upon upload to the market.

6.3.1.3. iOS Only

» iOS Development Certificate

 ▫ Used to run an iOS App on devices during development.

» iOS Distribution Certificate

 ▫ Used for submitting your iOS App to the App Store, and for distributing the App for On-Device testing. This is also used to identify you as the developer.

> **Warning**
>
> If you use an enterprise distribution certificate to sign a Swift-based iOS app built using the build farm, the resulting app may crash upon startup. If you encounter this problem, refer to the Red Hat Knowledge Base article Swift-based iOS application crashes upon startup when signed using an enterprise distribution certificate without Organisational Unit field for detailed instructions on how to resolve the problem.

» Provisioning Profile

 ▫ Necessary in order to install development applications on iOS devices.

6.3.2. Apple Developer and Enterprise Accounts

In order to publish an app in the Apple App Store you must have an active apple account (developer or enterprise). This account will need to be renewed annually in order for associated apps to continue to be available in the App Store.

6.3.2.1. Developer Account

A developer account is used create an iOS distribution certificates used to publish apps to the apple App Store.

When a distribution certificate expires, if the iOS Developer account is still active, existing apps on the App Store will not be affected, they will continue be available within the App Store and apps already on device will continue to function as expected.

6.3.2.2. Enterprise Account

An enterprise account is used to create (in-house) distribution certificates which are needed to publish apps to the RHMAP app store or customer in-house MDM.

When an existing in-house certificate expires all apps built with that certificate will not run and further installs of this version of the app will not be possible.

The app will need to be rebuilt, signed with a new certificate, republished to the relevant store and then re-downloaded by all users.

6.3.2.3. iOS Certificate Renewal

Apple enterprise and developer certificates must be recreated every three years.

The customer email address associated with the developer account will receive advance notification of the impending renewal requirement.

While the FeedHenry team may have been involved in assisting a customer with the initial setup of an Apple developer or enterprise account the ownership and responsibility for the apple account and the certificates created remains with the customer.

Upon receiving the certificate expiry notification it is recommended that the customer proactively renew the certificate in order to avoid interruption to their apps availability.