



# Red Hat Mobile Application Platform ホ スト型 3

## モバイル開発者ガイド

Red Hat Mobile Application Platform ホスト型 3 向け



# Red Hat Mobile Application Platform ホスト型 3 モバイル開発者ガイド

---

Red Hat Mobile Application Platform ホスト型 3 向け

## 法律上の通知

Copyright © 2017 Red Hat, Inc.

The text of and illustrations in this document are licensed by Red Hat under a Creative Commons Attribution-Share Alike 3.0 Unported license ("CC-BY-SA"). An explanation of CC-BY-SA is available at

<http://creativecommons.org/licenses/by-sa/3.0/>

. In accordance with CC-BY-SA, if you distribute this document or an adaptation of it, you must provide the URL for the original version.

Red Hat, as the licensor of this document, waives the right to enforce, and agrees not to assert, Section 4d of CC-BY-SA to the fullest extent permitted by applicable law.

Red Hat, Red Hat Enterprise Linux, the Shadowman logo, JBoss, OpenShift, Fedora, the Infinity logo, and RHCE are trademarks of Red Hat, Inc., registered in the United States and other countries.

Linux ® is the registered trademark of Linus Torvalds in the United States and other countries.

Java ® is a registered trademark of Oracle and/or its affiliates.

XFS ® is a trademark of Silicon Graphics International Corp. or its subsidiaries in the United States and/or other countries.

MySQL ® is a registered trademark of MySQL AB in the United States, the European Union and other countries.

Node.js ® is an official trademark of Joyent. Red Hat Software Collections is not formally related to or endorsed by the official Joyent Node.js open source or commercial project.

The OpenStack ® Word Mark and OpenStack logo are either registered trademarks/service marks or trademarks/service marks of the OpenStack Foundation, in the United States and other countries and are used with the OpenStack Foundation's permission. We are not affiliated with, endorsed or sponsored by the OpenStack Foundation, or the OpenStack community.

All other trademarks are the property of their respective owners.

## 概要

本ガイドでは、Red Hat Mobile Application Platform ホスト型 3 でモバイルクライアントアプリを開発する方法を説明します。

## 目次

<b>第1章 ANDROID</b>	<b>5</b>
1.1. ANDROID へのアプリのデプロイ	5
1.1.1. ブラウザーメソッド	5
1.1.2. Dropbox メソッド	5
1.1.3. Android ツールメソッド	5
<b>第2章 IOS</b>	<b>6</b>
2.1. IOS へのアプリのデプロイ	6
<b>第3章 WINDOWS</b>	<b>8</b>
3.1. WINDOWS PHONE 8 へのアプリのデプロイ	8
3.1.1. 非企業向け Windows Phone 8 アプリ	8
3.1.1.1. ビルド	8
3.1.1.2. 配布およびデプロイメント	8
3.1.2. 企業向け Windows Phone 8 アプリ	8
3.1.2.1. エンタープライズ証明書の取得	9
3.1.2.2. アプリケーション登録トークン (AET: Application Enrollment Token) の生成	9
3.1.2.3. Windows Phone の認証情報バンドルの作成	9
3.1.2.4. アプリのビルドおよび認証情報バンドルを使った署名	10
3.1.2.5. 配布およびデプロイメント	10
3.2. WINDOWS プラットフォームのサポート	11
3.2.1. サポートされるプラットフォーム	11
3.2.2. Windows アプリのローカルビルド	12
3.2.3. Windows 8.1 の Cordova での動的コンテンツの有効化	12
3.3. WINDOWS のフォームアプリおよび CORDOVA アプリの開発	13
3.3.1. ステップ	13
3.3.1.1. 初期設定	13
3.3.1.2. Cordova セットアップ	14
3.3.1.3. バイナリーのビルド	15
<b>第4章 フォーム</b>	<b>16</b>
4.1. フォームの CORDOVA アプリへの統合	16
4.1.1. 実際例	16
4.1.1.1. 概要	16
4.1.1.2. Cordova アプリ	16
4.1.1.2.1. ジョブモデル	17
4.1.1.3. クラウドアプリ	17
4.1.2. 実際例の作成	17
4.1.3. 実装ガイド	18
4.1.4. 関連セクション	21
4.2. OPENSIFT 2 ターゲットでのフォームサポートの有効化	22
4.2.1. Platform のフォームのバックグラウンド	22
4.2.2. OpenShift 2 ターゲットのフォームサポートを有効にする方法	23
<b>第5章 クライアントアプリの開発</b>	<b>25</b>
5.1. RHMAP を使用した ANGULAR アプリの開発	25
5.1.1. サンプルプロジェクトの概要	25
5.1.1.1. クライアントアプリ	25
5.1.1.2. クラウドコードアプリ	25
5.1.2. AngularJS Hello World Project の新規作成	25
5.1.3. Android デバイス用クライアントアプリのビルド	26
5.1.4. デプロイメントの概要	26

5.1.4.1. クラウドコードアプリ	26
5.1.4.2. AngularJS クライアントアプリ	27
5.1.4.2.1. RHMAP SDK	27
5.1.4.2.2. fhconfig.json	27
5.1.4.2.3. \$fh.cloud	28
5.2. RHMAP を使用した BACKBONE アプリの開発	28
5.2.1. サンプルプロジェクトの概要	28
5.2.1.1. クライアントアプリ	29
5.2.1.2. クラウドコードアプリ	29
5.2.2. Backbone Hello World Project の新規作成	29
5.2.3. Android デバイス用クライアントアプリのビルド	29
5.2.4. デプロイメントの概要	30
5.2.4.1. クラウドコードアプリ	30
5.2.4.2. Backbone クライアントアプリ	30
5.2.4.2.1. RHMAP SDK	31
5.2.4.2.2. fhconfig.json	31
5.2.4.2.3. \$fh.cloud	31
5.3. RHMAP を使用した LONIC アプリの開発	32
5.3.1. サンプルプロジェクトの概要	32
5.3.1.1. クライアントアプリ	32
5.3.1.2. クラウドコードアプリ	32
5.3.2. Ionic Hello World Project の新規作成	32
5.3.3. Android デバイス用クライアントアプリのビルド	33
5.3.4. デプロイメントの概要	33
5.3.4.1. クラウドコードアプリ	34
5.3.4.2. Ionic クライアントアプリ	34
5.3.4.2.1. RHMAP SDK	34
5.3.4.2.2. fhconfig.json	34
5.3.4.2.3. \$fh.cloud	35
5.4. CORDOVA プラグインの使用	35
5.4.1. サポートされるプラットフォーム	35
5.4.2. プラグインのアプリへの追加	35
5.4.2.1. 仕様	36
5.4.3. Cordova Light アプリの考慮事項	37
5.4.3.1. デフォルトプラグイン	38
5.4.4. ブラウザーでのアプリのテスト	39
5.5. アプリでのセキュアキーの使用	39
5.5.1. CUID およびアプリ ID	39
5.5.2. キーの永続化	40
5.5.3. サンプルコード	41
5.6. アプリのデバッグ	41
5.6.1. Studio コンソール	41
5.6.2. Firebug	42
5.6.3. Chrome の Web Inspector	42
5.6.4. Safari / iOS 6+	42
5.6.5. Chrome / Android 4.0+	44
5.6.6. オンデバイス (On-Device)	47
5.6.6.1. Weinre	47
5.7. CORDOVA LIGHT CLIENT アプリの CORDOVA への移行	48
5.7.1. 概要	48
5.7.2. 既存アプリのバックアップ	49
5.7.3. アプリ移行の設定	49
5.7.4. アプリの移行	49

---

5.7.5. 移行後のアプリの設定	50
<b>第6章 アプリの公開</b> .....	<b>58</b>
6.1. アプリの GOOGLE PLAY への提出	58
6.1.1. 前提条件	58
6.1.2. 概要	58
6.1.3. アプリケーションのアップロード	58
6.1.4. アセットのアップロード	58
6.1.5. 詳細の一覧表示	59
6.1.6. 公開オプション	60
6.1.7. 連絡先情報	61
6.1.8. 同意	61
6.2. アプリの APPLE APP STORE への提出	61
6.2.1. 新規アプリケーションの作成	61
6.2.2. 販売用アプリケーションの提出	63
6.2.3. 外部リンク	63
6.3. アプリ認証情報バンドル	63
6.3.1. リソース	63
6.3.1.1. すべてのプラットフォーム	63
6.3.1.2. Android のみ	64
6.3.1.3. iOS のみ	64
6.3.2. Apple 開発者およびエンタープライズアカウント	65
6.3.2.1. 開発者アカウント	65
6.3.2.2. エンタープライズアカウント	65
6.3.2.3. iOS 証明書の更新	65





## 第1章 ANDROID

### 1.1. ANDROID へのアプリのデプロイ

.apk ファイルを Android デバイスまたはエミュレーターにインストールする方法は数多くあります。以下ではそれらの方法について詳しく説明しています。Android デバイスがない場合は、Android SDK インストールの一部としてインストールされている Android エミュレーターを使用します。詳細は、[Android エミュレーターでのアプリの実行](#)を参照してください。

#### 1.1.1. ブラウザーメソッド

[FHC コマンドラインツール](#)を使用してバイナリーを生成している場合、出力には以下のような URL 値が含まれます。

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

これは、バイナリーのダウンロード URL です。デバイスブラウザーでこの URL に移動すると、バイナリーをダウンロードし、インストールすることができます。

#### ヒント

[bit.ly](#) などの URL Shortner (短縮サービス) を使用すると、ダウンロード URL の入力が大幅に容易になります。

#### 1.1.2. Dropbox メソッド

[Dropbox](#) アカウントをお持ちの場合、[Google Play Store](#) にある [Dropbox アプリ](#)を利用できます。APK ファイルを (PC/Mac 上の) [Dropbox フォルダー](#)に置くと、このファイルを [Dropbox アプリ](#)で使用でき、これを開いたり、インストールしたりできます。

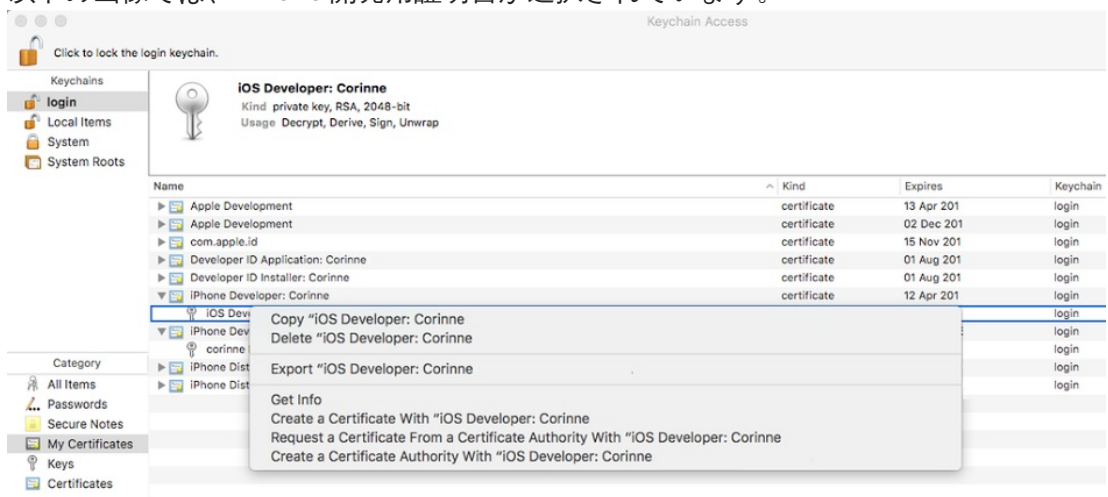
#### 1.1.3. Android ツールメソッド

[Android ツール](#) がインストールされている場合、[手順](#)に従って APK ファイルを USB ケーブル経由でデバイスに送信できます。

## 第2章 iOS

### 2.1. iOS へのアプリのデプロイ

1. iOS 開発用または配布用証明書がなければなりません。詳細については、[こちら](#)を参照してください。
2. [iPhone Provisioning Portal](#) にログインし、テストデバイスの UDID を使用してデバイスを作成します。
3. アプリの App ID を登録してからプロビジョニングプロファイルを作成します。詳細については、[こちら](#)を参照してください。
4. iOS dev p12 ファイルを生成します。以下のステップを実行するには **MAC** が必要です。
  - a. キーチェーンツールを開きます。
  - b. 「My Certificates」に移動し、iOS 開発用または配布用証明書を見つけます。
    - i. 以下の画像では、iPhone 開発用証明書が選択されています。



- c. プライベートキーを見つけて、これを右クリックします。
  - d. オプション「Export "iOS developer:"<Name of Developer>」を選択し、このファイルを p12 ファイルとしてエクスポートします。
    - i. p12 ファイルのパスフレーズの入力を求めるプロンプトが表示されたら、安全なパスフレーズを入力します。
    - ii. キーチェーンのパスワードを求めるプロンプトが出されたら、MAC パスワードを入力します。
      - A. p12 は MAC のキーチェーンに保存されます。
5. RMAP Studio で認証情報のバンドルを作成します。
  - a. RMAP Studio にログインします。
  - b. 「プロジェクト」セクションで iOS アプリを見つけてこれをクリックします。
  - c. 「認証情報」タブを見つけてこれをクリックしてから、「新規バンドル」をクリックします。

- d. 「iOS」アイコンを選択し、データをフィールドに入力します。
    - i. **バンドル名**: 分かりやすい名前を入力します。
    - ii. **プライベートキー**: エクスポートされた **p12** ファイルを選択します。
    - iii. **証明書**: エクスポートされた **p12** ファイルを選択します。
    - iv. **プロビジョニングプロファイル**: 上記のステップで作成されたプロビジョニングプロファイルを選択します。
  - e. 最後に、「バンドルの作成」をクリックして認証情報のバンドルを作成します。
6. RHMAP Studio で認証情報のバンドルを使用し、アプリケーションをビルドします。QR コードが作成されます。
7. アプリをインストール方法として 2 つの方法があります。
- a. QR コードを使用してテストデバイスにアプリをインストールします。
  - b. RHMAP Studio からバイナリー **.ipa** ファイルをダウンロードし、iTunes を使用して **.ipa** ファイルをインストールします。

## 第3章 WINDOWS

### 3.1. WINDOWS PHONE 8 へのアプリのデプロイ

Windows Phone 8 アプリの種類に応じて利用可能なデプロイメントのオプションは異なります。

#### 3.1.1. 非企業向け Windows Phone 8 アプリ

アプリバイナリーが Windows Phone の企業証明書で署名されていない場合、アプリは非企業向けのアプリになります。非企業向けの Windows Phone 8 アプリをデバイスに直接ワイヤレス (OTA) でインストールすることはできません。

##### 3.1.1.1. ビルド

- Visual Studio の使用  
Visual Studio では、署名オプションが指定されない場合、最終バイナリーは非企業向けアプリのバイナリーになります。
- App Studio の使用  
App Studio を使用して Windows Phone 用のビルドを行う場合に「Don't sign the binary (バイナリーに署名しない)」オプションが選択されていると、最終アプリバイナリーには署名されず、アプリは非企業向けになります。

The screenshot shows the 'Client Binary' configuration page. It includes a 'Platform' section with three buttons: 'Android', 'iOS', and 'Windows Phone' (which is highlighted in blue). Below this is a 'Version' dropdown menu set to '8.0', with a note: 'Version of Platform you want to produce a Binary for'. The 'Build Type' is set to 'Debug' with a note: 'The type of Binary you want to build - e.g. Debug or Distribution. [More info](#)'. The 'Sign Options' section has three radio buttons: 'Sign the binary with the FeedHenry company certificate.' (with a note about company certificates), 'Sign the binary with the credential bundle I selected.' (with a note about obtaining a certificate), and 'Don't sign the binary' (which is selected, with a note about uploading to the public app store). Each option has a corresponding [More info](#) link.

##### 3.1.1.2. 配布およびデプロイメント

通常、このタイプのアプリの配布とデプロイの方法として以下のような 2 つのオプションがあります。

- Windows Phone SDK で提供される Application Deployment ツールの使用  
このオプションは、主に開発のテスト用に使用されます。Windows Phone SDK がコンピューターにインストールされている場合、ターゲットデバイスは開発用に登録するデバイスになり、Application Deployment ツールを使って WP アプリをデプロイできます。
- Windows Phone App Store の使用  
アプリを Windows Phone App Store に公開し、ここからアプリを配布できます。

#### 3.1.2. 企業向け Windows Phone 8 アプリ

非企業向けアプリは、**Windows Phone** のエンタープライズ証明書で署名されると企業向けアプリになります。企業向けアプリはデバイスに直接ワイヤレス (OTA) でインストールでき、アプリストアは完全にバイパスされます。

**App Studio** を使用して企業アプリをビルドするには、以下のステップを実行する必要があります。

### 3.1.2.1. エンタープライズ証明書の取得

以下のステップに従ってエンタープライズ証明書を取得してください。

- **Publish Windows apps** のステップに従い、企業向け **Windows Phone** 開発者アカウントを作成し、アカウントタイプが「**Company**」であることを確認します。これには年間 \$99 の費用がかかります。企業情報の検証のために処理に数日かかることがあります。
- 開発者アカウントがセットアップされたら、**Enterprise Mobile Code Signing Certificate** にアクセスして証明書ファイルを申請し、「**Publisher ID**」フィールドに **Windows Phone** 開発者アカウントの情報を指定します。これには年間 \$299 の費用がかかります。
- このプロセスが完了すると、**Symantec** はコンピューターの証明書ストアにインポートできる証明書を配信します。[証明書のインストール方法](#)を参照してください。

#### ヒント

**Symantec** への証明書のリクエストや証明書の取得には同じコンピューターを使用してください。

- 最後に、証明書をそのプライベートキーと共に **PFX** ファイルでエクスポートし、これをパスワードで保護する必要があります。その手順については、[Export a Certificate with the Private Key](#)を参照してください。

さらに詳しくは、[Company app distribution for Windows Phone](#) を参照してください。

### 3.1.2.2. アプリケーション登録トークン (AET: Application Enrollment Token) の生成

企業アカウントで登録された電話のみがエンタープライズ証明書で署名されるアプリをワイヤレス (OTA) でインストールできます。デバイスを登録するには、アプリケーション登録トークン (AET) を生成し、これをデバイスにインストールする必要があります。

[Windows Phone 用のアプリケーション登録トークンを生成する方法](#)にあるステップを実行して、AET ファイルを手動で生成することができます。

または、**App Studio** では次のステップを実行して AET ファイルを生成することができます。

AET ファイルをデバイスにインストールする最も簡単な方法として、**Web** サーバーに AET を保存し、ユーザーにデバイスの **Web** ブラウザーを使用してこれをダウンロードするように求めることができます。次のいくつかのステップからも分かるように、**App Studio** でこれを実行できます。

### 3.1.2.3. Windows Phone の認証情報バンドルの作成

認証情報が利用可能になると、**App Studio** で新規の認証情報バンドルを作成することができます。

**Windows Phone Credentials**

**New "Windows Phone" Credentials Bundle**

Bundle Name

Certificate  fh.pfx  
It should be a PFX file which contains both the company certificate and its private key. [More info](#)

AET File Options ☒ Please generate the AET file using the certificate I uploaded.  
The AET file can be generated from the certificate file you uploaded above. If you choose this option, we will generate it for you. [More info](#)

☐ I have the AET file already and I will upload it.  
You can just upload the file if you already have the file generated. [More info](#)

☐ No, thanks. I will manage the AET file myself.  
You don't have to provide this file to us. But then you need to distribute the file to your users use other methods. [More info](#)




Certificate Password   
The password of your certificate is required to generate the AET file. We will not store the password.

スクリーンショットからも分かるように、**App Studio** は **AET ファイル** を生成できます。ただし、**AET ファイル** がすでに存在する場合、これを代わりにアップロードすることを選択できます。**AET ファイル** を自分で管理することを選択する場合は、**AET ファイル** がユーザーのデバイスにインストールされていることを確認してください。

#### 3.1.2.4. アプリのビルドおよび認証情報バンドルを使った署名

**Windows Phone** のアプリを **App Studio** でビルドする場合、アップロードしたばかりの認証情報バンドルを使ってアプリバイナリーに署名するオプションを選択できます。

**Client Binary**

Platform \*   

Version \*  Version of Platform you want to produce a Binary for

Build Type \*  The type of Binary you want to build - e.g. Debug or Distribution. [More info](#)

Sign Options ☐ Sign the binary with the FeedHenry company certificate.  
We have obtained a valid company certificate and the binary will be signed by us. The users' devices will be enrolled into our company account. [More info](#)

☒ Sign the binary with the credential bundle I selected.  
You can obtain your own company certificate and select it here. [More info](#)

☐ Don't sign the binary  
You don't need to sign the binary if you are going to upload it to the public app store. [More info](#)

Credential Bundle  Credential Bundle to sign this Binary with

Private Key Password  Optional - a Password for this Credential Bundle's Private Key

#### 3.1.2.5. 配布およびデプロイメント

アプリのビルドが終了すると、ダウンロードページには **OTA リンク** と **QR コード** が表示され、アプリをユーザーのデバイスに簡単にインストールできるようになります。**AET ファイル** が認証情報バンドルの一部として利用できる場合、そのダウンロードリンクと **QR コード** も表示されます。

## 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. WINDOWS プラットフォームのサポート

Red Hat Mobile Application Platform (RHMAP) は、Windows ベースのデバイスのクライアントアプリの開発をサポートします。詳細のチュートリアルについては、[Windows のフォームアプリ](#)と [Cordova アプリの開発](#)を参照してください。

### 3.2.1. サポートされるプラットフォーム

現行バージョンの RHMAP は Windows Phone バージョン 8 以降をターゲットにしています。本ガイドでは、プラットフォーム固有の問題の回避策の情報を詳しく記載しています。

- **Windows 10 ユニバーサルアプリ (電話、タブレットおよびデスクトップ)**
  - 完全サポート (以下を除く)
    - アプリは RHMAP でビルドできず、ローカルでビルドする必要があります。
- **Windows 8.1 ユニバーサルアプリ (電話、タブレットおよびデスクトップ)**
  - 完全サポート (以下を除く)
    - アプリは RHMAP でビルドできず、ローカルでビルドする必要があります。
    - Cordova アプリには動的コンテンツのサポートを有効にするための回避策がいくつか必要になります。
- **Windows Phone 8.1:**



- 完全サポート
- Windows Phone 7、7.5 以降
- Windows 8 Pro (タブレット)
- Windows 8 RT (タブレット)

### 3.2.2. Windows アプリのローカルビルド

Microsoft の開発ツールのライセンス方法の変更により、現時点で Windows 8.1 および Windows 10 のアプリを RHMAP で直接ビルドすることはできません。影響を受けるプラットフォームの RHMAP アプリはローカルの開発環境で手動でビルドする必要があります。

1. まず、クライアントアプリのリポジトリをクローンします。

```
git clone <Git_URL>
```

2. プロジェクトフォルダーでソリューションファイル (.sln) を見つけ、これを Visual Studio で開きます。
3. Visual Studio で F7 キーを押し、ソリューション設定およびプラットフォームを選択してからプロジェクトをビルドします。詳細は、公式のドキュメント [Building a Windows Phone app in Visual Studio](#) を参照してください。
4. ビルドが終了すると、拡張子が **.xap** または **.appx** のバイナリーが **<project\_name>/<solution\_name>/Bin/<solution\_configuration>** に置かれます。バイナリーファイルは Windows デバイスにデプロイしたり、RHMAP App Store にアップロードしたりできます。

### 3.2.3. Windows 8.1 の Cordova での動的コンテンツの有効化

Windows 8.1 については、アプリが **innerHTML** や **outerHTML** などのプロパティを使用することを防ぐセキュリティ上の制限があります。この制限により、動的コンテンツが挿入されることが妨げられ、ほとんどの JavaScript フレームワークが適切に機能しなくなります。さらに Cordova アプリと RHMAP のドラッグアンドドロップアプリも機能しなくなります。

この問題の回避策として Microsoft が提供する単一の JavaScript ファイルを参照することができます。これによりセキュリティ上の制限が緩和されます。

1. **winstore-jscompat.js** ファイルをダウンロードします:  
<https://raw.githubusercontent.com/MSPowerTech/winstore-jscompat/master/winstore-jscompat.js>
2. **winstore-jscompat.js** ファイルをプロジェクトの **www** フォルダーにコピーします。
3. 動的コンテンツの操作を必要とする HTML のファイルを参照します。**winstore-jscompat.js** は、**cordova.js** または **feedhenry.js** を含む他の JavaScript フレームワークの前に参照される必要があります。

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

この回避策についての詳細は、[MSPowerTech/winstore-jscompat](#) リポジトリの説明を参照してください。



### 3.3. WINDOWS のフォームアプリおよび CORDOVA アプリの開発

#### 概要

このチュートリアルでは、Red Hat Mobile Application Platform ホスト型 (RHMAP) を使って Windows プラットフォームをターゲットにするフォームアプリを作成する方法について説明します。このチュートリアルで説明されるステップは、フォームアプリだけではなくすべての Cordova アプリにも適用されます。

**Windows プラットフォームのサポート**で説明されているように、現時点ではいくつかの問題により、他のプラットフォームのサポートと同等のサポートが Windows プラットフォームでは利用できません。具体的には、Windows 8.1 および Windows 10 のネイティブアプリは Platform でビルドできず、手動でビルドする必要があります。さらに Cordova アプリには Windows 8.1 で適切に機能できるようにするための少しの回避策が必要です。

#### ヒント

このチュートリアルは、すぐに視聴可能な[ビデオウォークスルー](#)でもご利用いただけます。

#### 要件

- Windows 8.1 または Windows 10
- 最小 4 GB の RAM を搭載した 32 または 64 ビットマシン
- [Visual Studio 2013 Express](#) 以降
- Platform の既存のフォームアプリまたは Cordova アプリ。フォームアプリの作成方法については、[フォームプロジェクトの作成](#)を参照してください。

#### 3.3.1. ステップ

##### 3.3.1.1. 初期設定

1. Node.js をダウンロードし、インストールします:<https://nodejs.org/en/download/>  
この設定を指示するインストーラーを使用できます。npm パッケージマネージャーがインストールに組み込まれていることを確認します。これは本ガイドの後にも必要となるコアコンポーネントです。
2. git クライアントをダウンロードし、インストールします:<https://git-scm.com/download/win/>  
git の実行ファイル以外にも、Windows の Git インストーラーは、(Git Bash という) エミュレートされた Bash シェルおよびコマンドラインツールのセットを提供し、使い慣れた開発環境を提供します。Windows プロジェクトの Git についての詳細は、[Git for Windows](#) という公式 Web サイトを参照してください。



#### 注記

本ガイドに記載のコマンドラインの使用が関係するステップのほとんどでは、とくに記述がない限り **cmd** または **PowerShell** を使用できます。ただし、このステップでインストールされる **Git Bash** コマンドをすべてのステップで使用することをお勧めします。

3. Cordova パッケージをインストールします。

```
npm install -g cordova
```

これにより、**cordova** 実行可能ファイルを含む **Cordova** パッケージがグローバルにインストールされ、これがすべての **Node.js** アプリケーションで利用可能になります。

4. **Platform** からアプリのリポジトリをクローンします。  
まず **git** アクセスが適切に機能するように **SSH** キーをプラットフォームにアップロードする必要があります。詳細は、[SSH キーの設定](#) を参照してください。

アプリのリポジトリをクローンします。

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

アプリの **Git SSH** クローン URL は「**App Details** (アプリの詳細)」で確認できます。

### 3.3.1.2. Cordova セットアップ

通常 **Cordova** プロジェクトは **Platform** でのビルド時に **Cordova** アプリ用として自動的に作成されるため、**Cordova** 固有のファイルおよびフォルダーはアプリのリポジトリでは利用できません。ただし、手動でビルドする場合には **Cordova** ファイルの作成とコマンド呼び出しを手動で実行する必要があります。

フォームアプリについても **Cordova** をベースとしているため、**Cordova** アプリと同じビルド手順が必要になります。

1. **Cordova** プロジェクトを作成します。  
この例では **winforms** という名前を使用しますが、任意の名前を使用できます。

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

これにより新規の **Cordova** プロジェクトが作成され、**www** ディレクトリーから新規に作成されるプロジェクトの **www** ディレクトリーにアセットがコピーされます。

2. **Windows 8.1 のみ**: **Cordova** アプリでの動的コンテンツ操作の回避策を適用します。  
詳細は、[Windows 8.1 での Cordova の動的コンテンツの有効化](#) を参照してください。

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

**winstore-jscompat.js** は動的コンテンツ操作を実行する **Cordova** アプリのすべての **HTML** ファイルに含める必要があります。

```
<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. **windows** プラットフォームを **Cordova** プロジェクトに追加します。

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

これにより、ソリューションファイル (.sln) を含むバイナリーをビルドするために必要なすべてのソースファイルが作成されます。このファイルはプロジェクトを **Visual Studio** にインポートするために使用できます。

#### 4. デフォルトの Cordova プラグインを追加します。

```
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. バイナリーのビルド

アプリのバイナリーは **Cordova** を使用してコマンドラインからか、または **Visual Studio** からビルドでき、これは **Windows 8.1** および **Windows 10** の両方をサポートします。

**Cordova** を使用してビルドするには、以下を実行します。

```
cordova build
```

**Visual Studio** を使用してビルドするには、直前のステップで作成されたソリューションファイル (.sln) を **Visual Studio** にインポートし、ソリューションをビルドします。詳細は、[Windows アプリのローカルビルド](#)を参照してください。

**Cordova** は、アプリの実行をすぐに確認できるようにエミュレーターを開始する機能も提供します。

```
cordova emulate
```

## 第4章 フォーム

### 4.1. フォームの CORDOVA アプリへの統合

フォームの機能は既存アプリに統合できます。本ガイドでは、監督者がジョブを割り当て、作業者がモバイルデバイスでその割り当てを受信し、ジョブについての情報を送り返すまでの労働力管理の例を使ってこの統合について説明します。

本ガイドの手順では、[Cordova アプリ](#)および[クラウドアプリ](#)を使用します。これらのアプリをプロジェクトにインポートして、本ガイドで説明する実際例について説明します。

#### 4.1.1. 実際例

##### 4.1.1.1. 概要

このサンプルアプリケーションには以下の要件が設定されます。

監督者は、暴風雨で倒れた木を取り除くという作業者のジョブを作成します。監督者は作業者に対し、木の写真や位置情報、コメントおよび開始時間と終了時間などのジョブについての詳細情報の提出を求めます。このタスクはフォームアプリを使用して実行することができます。

監督者は以下を実行してジョブを作成します。

- 作業者が記入するフォームの選択
- フォームの選択およびジョブの詳細情報のフォームへの記入
- ジョブの固有 ID の指定

アプリケーションの要件には以下が含まれます。

- ジョブは、管理者が作成フォームに記入して作成する。
- 新規ジョブは管理者と管理者以外のユーザーのどちらも利用できる。アプリユーザーは記載済みの作成フォームを閲覧できる。
- アプリユーザーは別の完了フォームに記入してジョブを完了できる。
- 完了フォームはすべてのアプリユーザーが確認できるよう表示可能でなければならない。
- クライアントアプリのユーザーは、完了フォームをドラフトとして保存し、これに「In Progress (進行中)」または「Complete (完了)」のいずれかのマークを付けることができます。

##### 4.1.1.2. Cordova アプリ

クライアントアプリは以下のテクノロジーをベースとしています。

- **Backbone:** Cordova アプリは **backbone** モデルおよびビューを使用してジョブの作成および更新を管理します。
- **Handlebars:** ビューのテンプレートに使用されます。
- **Bootstrap:** スタイリングに使用されます。
- **Font-Awesome:** アイコンに使用されます。

Cordova アプリは以下を行います。

- さまざまな状態のジョブの一覧表示を管理
- フォームの表示を管理
- フォーム提出および提出のアップロードを管理
- フォームおよび提出データを含むジョブの作成を管理

#### 4.1.1.2.1. ジョブモデル

ジョブモデルは、ジョブについて説明する単純な **Backbone** モデルです。

ジョブコレクションは、ジョブモデルの集合です。

クライアントとクラウド間でジョブを同期するためにカスタム URL が組み込まれます。このカスタム URL はクラウドアプリで **RESTful /jobs** エンドポイントにアクセスするために使用されます。

#### 4.1.1.3. クラウドアプリ

クラウドアプリは、**\$fh.db API** を使用してジョブデータに対して **CRUD** 操作を実行するための **RESTful** エンドポイント (**/jobs**) で構成されています。



#### 注記

すべてのクラウドベースのフォームロジックは **クラウド API** に含まれているため、クラウドアプリではフォームを処理するためのロジックは表示されません。

### 4.1.2. 実際例の作成

1. App Studio で新規プロジェクトを作成します。
2. アプリをプロジェクトにインポートします。たとえば、本ガイドのサンプルでは **Cordova アプリ** および **クラウドアプリ** が使用されます。
3. プロジェクトの **フォーム** および **テーマ** を作成します。プロジェクトに関連付けて作成されるすべてのフォームとテーマは **Cordova アプリ** に表示されます。
4. オプションで、ユーザーおよびフィールドコードをプロジェクトに追加します。例えば以下のようになります。
  - 管理者: ジョブを作成し、完了することができる。
  - ユーザー: ジョブを完了することができる。

ユーザーには、関連フォームの表示前に提出に自動的に追加される **userId** および **userName** フィールドがあります。2つのテキストフィールドをこの実際例に関連付けられたフォームに追加します。

これらのフィールドが追加されたら、データブラウザーの **users (ユーザー)** コレクションに2名のユーザーを追加します。

#### 管理者

```
{
```

```

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

```

## ユーザー

```

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

```

### 4.1.3. 実装ガイド

以下を使用してフォームの機能をアプリに統合します。

1. **\$fh.forms.init** 関数をクライアントに追加してフォームの初期化 (Forms Initialization) を追加します。これにより、クライアントアプリのフォームが初期化され、アプリの他の場所で **\$fh.forms Client API** の使用が可能になります。  
**\$fh.forms.init** 関数は、アプリのログインプロセスの一部を構成します。
2. 管理者ユーザーとして完了フォームを選択します。これにより、ジョブを完了するために記載する必要のあるフォームが指定されます。**\$fh.forms.getForms** クライアント API 関数を使用してアプリで利用できるすべてのフォームを一覧表示します。



#### 注記

**\$fh.forms.getForms** クライアント API 呼び出しは、フォームの一覧のみをダウンロードしますが、各フォームの全体のフォーム定義はダウンロードしません。

3. **\$fh.forms.getForm** クライアント API を使用してフォームをクライアントにダウンロードします。  
フォームはジョブ作成、ジョブ詳細の表示、およびジョブの完了で使用されるため、この関数はヘルパー関数のセットに抽象化されます ([こちら](#)を参照してください)。

**\$fh.forms.getForm** クライアント API の使用法は、**FormFunctions.js** の **loadForm** 関数の一部として使用される場合は [こちら](#)で確認できます。

4. 提出をアプリにロードします。このプロセスは、**FormFunctions.js** ファイルの **loadSubmission** 関数の使用に関連して説明されています。  
フォームに入力されるデータは提出に設定されるため、フォームは提出に関連します。ただし、提出はクラウドにアップロードされる前にフォームに対して検証されます。

提出を作成する方法として、以下の 3 つの方法があります。

- **ローカルメモリから:** 提出をドラフトとしてローカルメモリに保存してから、提出モデルの **saveDraft** 関数を使用して保存します。この機能の実装については、**loadLocalSubmission** 関数で説明されています。
- **リモートからのダウンロード:** クラウドから提出をダウンロードします。たとえば、監督者がジョブの詳細を説明するためにフォームに記載する際に、提出の ID がジョブモデルに保存されます。アプリユーザーがジョブモデルをダウンロードすると、ユーザーは管理者



ユーザーが提出したフォームのリモート提出 ID にアクセスできます。このリモート提出 ID は、クラウドから提出の詳細定義をダウンロードするために使用されます。この機能の実装については、[downloadSubmission](#) 関数で説明されています。



### 注記

提出のフォーム定義はクラウドからダウンロードされる提出に含まれます。フォーム定義が複数の提出間で編集されている可能性があるためです。



### 注記

ダウンロードされた提出はクライアント上で編集できません。これらには読み取り専用アクセスのみが意図されています。ダウンロードされた提出のクラウドへの送信を試行すると、エラーが返されます。

- **新規の提出の作成:** フォームに関連付けられた提出がない場合に、新規の提出を作成できます。この場合、提出は、[フォームモデルで作成されます](#)。これにより、提出が適切なフォームに自動的に関連付けられます。

5. ユーザーが編集できるようにフォームをビューに表示します。  
フォームを既存の Cordova アプリに表示する方法として、以下の 2 つの方法があります。

- **\$fh.forms.backbone API** を使用したフォームの表示: これには **backbone/bootstrap SDK (\$fh.forms.backbone)** が含まれます。[Appforms Backbone](#) ファイルをダウンロードして Cordova アプリの一部として組み込みます。また、Cordova アプリは以下の JavaScript および CSS 依存関係に対応している必要があります。
  - Backbone
  - Bootstrap
  - Font-Awesome

**CSS** および **JavaScript** 依存関係はサンプルの Cordova アプリに組み込まれています。

**FormViewSDK.js** ファイルには、**Backbone SDK** バージョンのフォームビューが含まれます。Cordova アプリには **Backbone SDK** と手動によるフォーム表示間での切り換えを行うための「**Settings**」タブのオプションが含まれます。



### 注記

**Backbone SDK** は、**Backbone/Bootstrap** ベースの Cordova アプリのフォームアプリの統合を加速させることを目的としています。**\$fh.forms** クライアント API についてはすべての Cordova アプリで機能します。フォームのレンダリングとユーザーデータの送信内容への事前設定は開発者が行うタスクになります。

- フォームの手動表示



### 注記

フォームのユーザーへの表示は、提出を完了する最も簡単な方法ですが、フィールド入力値はどのソースからも提出に追加できます。提出はフィールドまたはページルールに対して有効な状態である必要があります。

**\$fh.forms** SDK はいずれのフレームワークにも依存しないため、どの Cordova アプリにも追加できます。このアプリは **Backbone** および **Bootstrap** をベースとしています。が、**\$fh.forms** APIを他の javascript ベースの UI フレームワーク (例: **Angular**) と共に使用することも同様に可能です。

基本的な **Bootstrap** フォームはフォーム定義に基づいて表示されます。このフォームは **FormView.js** ファイルで定義されます。フォームの表示、入力される提出内容および検証ロジックはすべて **\$fh.forms** API およびモデルを使用してアプリに定義されます。



### 注記

手動で表示されるフォームは説明用としてのみ実装されます。手動で実装できるのはテキストフィールドと数値フィールドのみですが、利用可能なフォームのフィールドタイプはすべて **\$fh.forms.backbone** SDK を使用して表示できます。

カスタムフォームビューの表示ロジックは **FormView.js** ファイルにあります。ここでは、このビューがフォームのユーザーへの表示に関係するすべてのイベントを処理することを確認できます。

さらに、**FormView.js** ファイルには以下を実行するためのロジックが含まれます。

- 入力時のフィールドデータの検証
- フィールドおよびページルールの検査
- 提出へのデータの設定
- 提出のドラフト保存
- クラウドへのフォームの提出

以下のステップでは、**\$fh.forms** SDK をカスタムで表示されたフォームに手動で統合する際に **Cordova** アプリがこれらの要件にどのように対応するかを説明します。

6. フィールドに入力できるデータを制限する検証パラメーターを定義します(たとえば、テキストフィールドではフィールドに入力できる最小/最大文字数を指定できます)。この機能をクライアントアプリに追加すると、フィールドの制限が反映されます。  
この要件に対応するために、**validateInput**関数は **FormView.js** ファイルの入力の **blur** イベントに登録されます。



### 注記

検証パラメーターは提出の有効性に影響を与えます。フィールドの検証がクライアントアプリで実行されない場合でも、すべての提出フィールドはデータベースに保存される前に検証されます。

7. フォームアプリには、フィールドおよびページルールが含まれます。**Studio** では、フォームエディターでフィールドルールを作成し、フィールド入力データに基づいてフィールドを表示/非表示にしたり、ページルールを作成してフィールドに入力されるデータに基づいてページを表示/スキップしたりすることができます。  
この機能は **\$fh.forms** API の実装に反映されます。ルールエンジンを使用して提出を処理することで、その提出で表示/非表示にする必要のあるフィールドまたはページを特定できます。

これは **FormView.js** ファイルの **checkRules** 関数で実装されます。





### 注記

フィールドおよびページルールは、提出の有効性に影響を与えます。クライアントアプリでフィールドおよびページルールにチェックが付けられていない場合でも、提出はデータベースに保存される前にすべてのルールに基づいてチェックされます。

8. **addInputValue** 関数を使って提出モデルにデータを追加します。このデータのソースとして、ユーザーに表示されたフォームか、アプリで利用できる外部データ、またはそれらの組み合わせのいずれかを使用することができます。

- 表示されたフォーム: この場合、フォームはユーザーが **\$fh.forms.backbone** SDK を使用してデータを入力できるように表示されるか、または手動で表示されます。**\$fh.forms** API をカスタムで表示されたフォームに手動で統合する際に、ビューから提出モデルへのデータ移行に対応する必要があります。

これは **FormView.js** ファイルの **saveFieldInputsToSubmission** 関数で説明されています。

- フィールドコードの使用による外部ソース: フィールドコードを追加してフォーム内のフィールドを一意に特定するためのフィールドを作成できます。このフィールドコードは外部データソース (例: CSV ファイルのヘッダー) に関連付けられます。この機能を使用して、外部データをフォームの提出にインポートすることができます。この機能は、サンプルの **Cordova** アプリにおいては **addSubmissionData** 関数で説明されています。この例では、ユーザーには **userId** および **userName** フィールドがあります。フォームにフィールドコード **userId** および **userName** のフィールドが含まれる場合、これらのフィールドにはユーザーモデルのデータが設定されます。



### 注記

フィールドコードはフォーム内で一意である必要があります。ただし、同じフィールドコードを複数のフォームに存在させることは可能です。

9. 提出をドラフトとして保存します。この機能は **FormView.js** ファイルの **saveDraft** 関数で説明されています。
10. 検証およびルール機能をフォームに追加すると、有効な提出をクラウドに送信して提出エディター (**submission editor**) で表示または編集を実行できます。フォームビューは、データの処理およびアップロード時に提出モデルで生成される提出関連のイベント (**validationerror**、**queued**、**progress**、**error**、**submitted**) をリスンします。

提出プロセスには以下の2つのステップが関係します。

- **Submit (提出):** 提出モデルで **submit** 関数を呼び出すと、提出がローカルフォームの定義に対して検証され、提出ステータスが **pending** (保留中) に変更されます。
- **Upload (アップロード):** 提出モデルで **upload** 関数を呼び出すと、提出がフォームデータベースへのアップロードのキューに入れられます。

#### 4.1.4. 関連セクション

- **\$fh.forms** クライアント API

- [\\$fh.db クラウド API](#)
- [フォームの作成](#)
- [テーマの作成](#)
- [フォームプロジェクトの作成](#)

## 4.2. OPENSIFT 2 ターゲットでのフォームサポートの有効化

OpenShift 2 ターゲットにデプロイされたクラウドアプリでフォームを使用するためのサポートはすぐに利用できる訳ではありません。このサポートは、本ガイドで説明されているステップを実行して有効にすることができます。



### 注記

本ガイドは、[openshift.feedhenry.com](https://openshift.feedhenry.com) などの OpenShift 2 ターゲットのサポートのある Platform インスタンスのみに適用されます。他のターゲットの場合も、フォームのサポートを追加の設定なしに利用可能にすることができます。

### 4.2.1. Platform のフォームのバックグラウンド

Platform のバックエンドには、**fh-mbaas** というフォーム関連の操作を処理するサービスがあります。この機能は **\$fh.forms** API で公開され、これにはフォーム定義へのアクセス、フォームの提出およびデプロイメントが含まれます。

Red Hat MBaaS ターゲットの場合、**fh-mbaas** のインスタンスは常に利用でき、これを手動で有効にする必要はありません。ただし、デフォルトで **fh-mbaas** サービスがデプロイされない OpenShift 2 ターゲットの場合にはこの限りではありません。この機能は Platform のサービステンプレートとして別途提供され、これを OpenShift 2 ターゲットにデプロイすることでフォームのサポートを有効にすることができます。

The screenshot displays the OpenShift console interface for an application named `openshiftmbaasservfhdovb3p-nzr.rhcloud.com`. The application is in the `Started` state. Below the application name, it indicates it was created about 2 hours ago in the `aws-us-east-1` region. The `Cartridges` section lists the following components:

Cartridge	Status	Gears	Storage
Node.js 0.10	Started	1 small	1 GB
MongoDB 2.4	Database: <code>openshiftmbaasservfhdovb3p</code> User: <code>admin</code> Password: <a href="#">show</a>		
Redis	<a href="#">details &gt;</a>		
FeedHenry	<a href="#">details &gt;</a>		

The `Source Code` section provides the SSH URL for cloning the repository: `ssh://55b239095973ca3b770000dd@opens`. Below this, it instructs the user to pass this URL to 'git clone' to copy the repository locally. The `Remote Access` section includes a link to log in to the application and a button to delete the application.

OpenShift 側では、フォームのサポートはカートリッジで実装されます。このカートリッジは、OpenShift アカウントにより個別アプリケーションの **Node.js** ランタイムおよびデータベースカートリッジと共にホストされ、この機能をフォームベースのすべてのアプリケーションに公開します。

## 4.2.2. OpenShift 2 ターゲットのフォームサポートを有効にする方法

前提条件:

- OpenShift 2 ターゲットのサポートのある RHMAP アカウント
- 既存の OpenShift 2 ターゲットおよび関連付けられた環境

以下のステップでは、アプリが OpenShift 2 ターゲットでフォームを使用できるようにするプロセスを説明します。

### 1. OpenShift mBaaS サービスのインスタンスの作成およびデプロイ

- Studio のサービス & API セクションで、MBaaS サービス/API のプロビジョニングをクリックします。
- 利用可能なサービスの一覧から **OpenShift mBaaS Service** を選択し、このサービスインスタンスに適切な名前を入力して**次へ進む**をクリックします。
- ログウィンドウに **Service is ready!** が表示されるまで待機し、**終了**をクリックします。  
「サービスの詳細」ページが表示されます。
- 左側のメニューにある**デプロイ**セクションに移動します。
- 画面右側のドロップダウンメニューから OpenShift ベースの環境を選択します。
- クラウドアプリをデプロイ**をクリックします。
- サービスがデプロイされるまで待機します。画面下部の **Deploy History (デプロイ履歴)** セクションにデプロイのステータスと結果が表示されます。これらのインジケーターには状態が即時に反映されない可能性があります。ページを再ロードしてステータスを更新し、正しい環境が選択されていることを確認します。デプロイメントが終了したら、ページを再度ロードし、画面上部の **Deploy Options** に **Current Host** フィールドがあることを確認します。

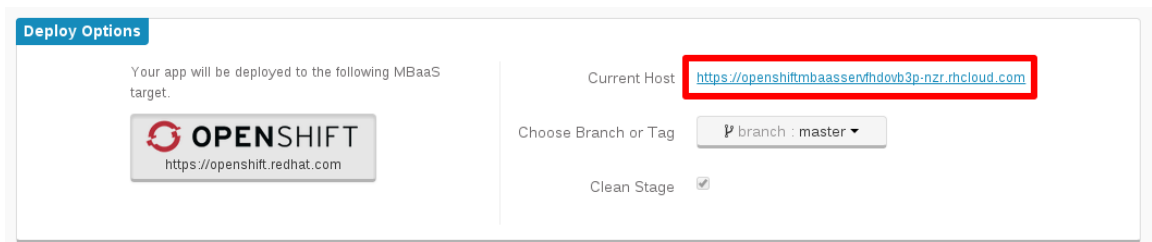


### 注記

サービスのデプロイには数分の時間がかかる可能性があります。これはギアの作成、いくつかのカートリッジのプロビジョニング、Git リポジトリおよび他のセットアップ操作のクローンが行われるためです。このデプロイは1つの OpenShift 2 ターゲットに対して1回のみ実行されます。同様に、クラウドアプリの初回デプロイメントの場合も、それに続くデプロイメントより多くの時間がかかる可能性があります。

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

- 画面の **Deploy Options** セクションの **Current Host** フィールドの値をメモするか、またはコピーします。この URL は fh-mbaas サービスとして機能する新たに作成された OpenShift アプリケーションを指します。



Deploy Options

Your app will be deployed to the following MBaaS target.

**OPENSIFT**  
https://openshift.redhat.com

Current Host <https://openshiftmbaaservfhdovb3p-nzr.rhcloud.com>

Choose Branch or Tag

Clean Stage ☒

## 2. OpenShift 2 ターゲットに正しい fh-mbaas ホスト URL を設定します。

- Studio の**管理/ MBaaS** ターゲットセクションに移動し、既存の OpenShift 2 MBaaS ターゲットを開きます。
- 先に保存した fh-mbaas サービスの URL を **fh-mbaas** ホストフィールドに入力します。
- MBaaS を保存** をクリックします。

この新たな OpenShift ベースの環境にデプロイされるすべてのクラウドアプリはフォーム機能にアクセスできます。

単一の fh-mbaas サービスを使用することでフォームサポートを複数の OpenShift 2 MBaaS ターゲットに提供することができ、このサービスを使用する OpenShift 2 MBaaS ターゲット以外の OpenShift インスタンスに配置することができます。フォームを使用するクラウドアプリは、ターゲットに有効な fh-mbaas ホストが設定されている限りすべての OpenShift 2 ターゲットにデプロイできます。

## 第5章 クライアントアプリの開発

### 5.1. RHMAP を使用した ANGULAR アプリの開発

#### 概要

ここでは、HTML5 AngularJS Cordova アプリ内で RHMAP Javascript SDK を使用方法を説明します。

これには新規の AngularJS Hello World Project を作成するために必要なステップが含まれ、ここではクラウドコードアプリと対話するために必要なコードが強調されます。

#### 要件

- RHMAP アカウント
- HTML、JavaScript、[AngularJS](#) および [Node.js](#) についての知識
- GitHub ユーザーアカウント
- [FHC](#)

AngularJS の使用を開始するユーザーは [AngularJS チュートリアル](#) をご利用できます。

#### 5.1.1. サンプルプロジェクトの概要

このサンプルプロジェクトは、1つのクライアントアプリと1つのクラウドアプリが含まれる単純なプロジェクトです。

##### 5.1.1.1. クライアントアプリ

クライアントアプリは、RHMAP Javascript SDK が含まれる単純な AngularJS アプリです。クライアントアプリは、ユーザーに対してそれぞれの名前をテキストボックスに入力し、**Say Hello From The Cloud** ボタンをクリックするように求めます。次にクライアントアプリは `$fh.cloud` 関数を使用して入力されたテキストをクラウドコードアプリに送信します。

##### 5.1.1.2. クラウドコードアプリ

クラウドコードアプリは **hello** エンドポイントを公開してクライアントアプリからリクエストを受信し、送信されたテキストを変更して **Hello** を追加し、応答をクライアントに返します。

#### 5.1.2. AngularJS Hello World Project の新規作成

以下のステップを使用して新規の AngularJS Hello World Project を作成します。

1. Studio にログインします。
2. 画面の左上にあるプロジェクトタブを選択します。
3. 画面上部の + **新規プロジェクト** ボタンを選択します。プロジェクトテンプレートの一覧が表示されます。
4. **AngularJS Hello World Project** テンプレートを選択し、新規プロジェクトの名前を指定します。

5. **次へ進む** ボタンを選択します。新規プロジェクトが作成されます。

6. プロジェクトが作成されたら、**終了** ボタンを選択します。

これで、**AngularJS** クライアントアプリとそれが通信するクラウドコードアプリが含まれる新規プロジェクトが作成されました。

アプリをプレビューするには、以下を実行します。

1. **アプリ、クラウドアプリ & サービスタグ** を選択します。
2. **アプリセクション** の下にある **クライアントアプリ** を選択します。ここから詳細画面に移動します。
3. **App Preview** には、クラウドコードアプリと通信する **AngularJS** クライアントアプリの作業用バージョンが含まれます。

### 5.1.3. Android デバイス用クライアントアプリのビルド

アプリをテストして適切に機能することを確認したら、**RHMAP Studio** を使用してクライアントアプリをビルドすることができます。

以下のステップに従って **Android** デバイスのクライアントアプリをビルドします。

1. クライアントアプリのインターフェースから **ビルド** タブを選択します。
2. **Android**、**master** ブランチ、**Debug** ビルドタイプ、**Dev** クラウドコードアプリを選択してから **ビルド** ボタンをクリックします。クライアントアプリの **Android** ビルドが作成されます。
3. **Android** ビルドが完了すると **QR** コードが表示されるので、**Android** デバイス上で **QR Code Scanner** アプリを開いてビルドをインストールします。または、ショート URL を電話のブラウザに入力します。



#### 注記

**Android** デバッグバイナリーは証明書なしにビルドできますが、いずれのタイプの **iOS/Windows Phone** バイナリーのビルドにも必要な認証情報が必要です。



#### 注記

**branch** セレクターにより、ビルドするクライアントアプリのブランチを選択できます。この場合、デフォルトの **master** ブランチが適切なブランチになります。

これで **RHMAP** を使用して **AngularJS HTML5 Cordova** アプリを作成することができました。

### 5.1.4. デプロイメントの概要

このセクションでは、サンプルのソリューションが機能するための必要なコードについて重点的に説明します。

#### 5.1.4.1. クラウドコードアプリ

まず、クラウドコードアプリについて考えてみましょう。この例ではクラウドコードアプリでクライアントアプリからのリクエストを受信し、**hello** パラメーターを変更し、以下のパラメーターを含む JSON オブジェクトを使用してクライアントアプリに応答できるようにします。

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

この機能をクラウドコードアプリに実装するには、以下を実行します。

1. **application.js** ファイルで、**lib** ディレクトリーにある **hello.js** ファイルを必要とする新規の **/hello** ルートが追加されます。
2. **hello.js** ファイルは 2 つのルートを作成します。どちらのルートも **hello** パラメーターを変更するための同じ操作を実行します。
  - **hello** パラメーターがクエリー文字列として追加される **GET** リクエスト。
  - **hello** パラメーターが **POST** リクエストの本文で送信される **POST** リクエスト。



#### 注記

このクラウドアプリは AngularJS クライアントアプリとは完全に切り離されています。クラウドコードアプリはプロジェクト内の任意の数のクライアントアプリ間で共有できます。

### 5.1.4.2. AngularJS クライアントアプリ

クラウドコードアプリの **/hello** エンドポイントを作成した後は、AngularJS クライアントアプリに追加した機能を検査し、この機能を使ってクラウドコードアプリで公開された **/hello** エンドポイントにリクエストを送信できるようにします。

クライアントアプリは、一部のテキストを受け入れる単一入力と入力をクラウドに送信し、結果をユーザーに表示する単一ボタンのある単純な AngularJS アプリです。

#### 5.1.4.2.1. RHMAP SDK

まず、すべての HTML5 Cordova アプリの場合のように RHMAP Javascript SDK (**feedhenry.js**) は **index.html** ファイルに含まれています。この SDK により、すべての **\$fh クライアント API** 関数にアクセスできます。



#### 注記

**feedhenry.js** ファイルはテンプレートリポジトリでは空になっていることを確認できます。クライアントアプリの作成時に **feedhenry.js** ファイルが **www** ディレクトリーにあると、最新の RHMAP Javascript SDK がファイルに自動的に追加されます。

#### 5.1.4.2.2. fhconfig.json

クライアントアプリには **fhconfig.json** ファイルも含まれます。このファイルには、RHMAP Javascript SDK がクラウドアプリと通信するために必要な情報が含まれます。





## 注記

すべての HTML5 クライアントアプリには、**\$fh** クライアント API 関数を使用するための **fhconfig.json** ファイルが含まれている必要があります。アプリの Studio での作成時に、このファイルに必要な情報が自動的に設定されます。

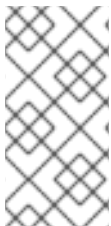
### 5.1.4.2.3. \$fh.cloud

この例では、クラウドコードアプリの **hello** エンドポイントにリクエストを送信するために **\$fh.cloud** クライアント API 関数が使用されます。

**\$fh.cloud** 関数は **cloud.js** ファイルにあります。ここで、**\$fh.cloud** 関数は **MainCtrl** コントローラーが使用する再利用可能なサービスとして公開されます。

AngularJS アプリには、**MainCtrl** という単一のコントローラーがあります。このコントローラーは以下を実行します。

1. **example.html** ビューからユーザーの入力を受け入れる。
2. **fhcloud** サービスを使用してクラウドコードアプリの **hello** エンドポイントを呼び出す。
3. **\$fh.cloud** 呼び出しが成功するかどうかに応じ、**success** または **error** 関数を使用してクラウドコードアプリからの応答を処理する。



## 注記

この場合、クライアントアプリは **GET** リクエストタイプを使用しています。クラウドコードアプリは **GET** および **POST** バージョンの **hello** エンドポイントの両方を公開し、**POST** リクエストタイプも機能します。これは RESTful アプリケーションを扱う場合にとくに役立ちます。

## 5.2. RHMAP を使用した BACKBONE アプリの開発

### 概要

ここでは HTML5 Backbone Cordova アプリ内で RHMAP Javascript SDK を使用方法を説明します。

これには **Backbone Hello World Project** を新たに作成するために必要なステップが含まれ、ここではクラウドコードアプリと対話するために必要なコードが強調されます。

### 要件

- RHMAP アカウント
- HTML、JavaScript、**Backbone** および Node.js についての知識
- GitHub ユーザーアカウント
- **FHC**

Backbone の使用を開始するユーザーは **Backbone Tutorial** をご利用できます。

### 5.2.1. サンプルプロジェクトの概要



このサンプルプロジェクトは、1つのクライアントアプリと1つのクラウドアプリが含まれる単純なプロジェクトです。

#### 5.2.1.1. クライアントアプリ

クライアントアプリは **RHMAP Javascript SDK** が含まれる単純な **Backbone** アプリです。クライアントアプリは、ユーザーに対してそれぞれの名前をテキストボックスに入力し、**Say Hello From The Cloud** ボタンをクリックするように求めます。次にクライアントアプリは **\$fh.cloud** 関数を使用して入力されたテキストをクラウドコードアプリに送信します。

#### 5.2.1.2. クラウドコードアプリ

クラウドコードアプリは **hello** エンドポイントを公開してクライアントアプリからリクエストを受信し、送信されたテキストを変更して **Hello** を追加し、応答をクライアントに返します。

### 5.2.2. Backbone Hello World Project の新規作成

以下のステップを使用して新規の **Backbone Hello World Project** を作成します。

1. **Studio** にログインします。
2. 画面の左上にある**プロジェクト**タブを選択します。
3. 画面上部の **+ 新規プロジェクト** ボタンを選択します。プロジェクトテンプレートの一覧が表示されます。
4. **Backbone Hello World Project** テンプレートを選択し、新規プロジェクトの名前を指定します。
5. **次へ進む** ボタンを選択します。新規プロジェクトが作成されます。
6. プロジェクトが作成されたら、**終了** ボタンを選択します。

これで **Backbone** クライアントアプリとそれが通信するクラウドコードアプリが含まれる新規プロジェクトが作成されました。

アプリをプレビューするには、以下を実行します。

1. **アプリ、クラウドアプリ & サービス** タグを選択します。
2. **アプリ** セクションの下にある**クライアントアプリ**を選択します。ここから詳細画面に移動します。
3. **App Preview** には、クラウドコードアプリと通信する **Backbone** クライアントアプリの作業用バージョンが含まれます。

#### 5.2.3. Android デバイス用クライアントアプリのビルド

アプリをテストして適切に機能することを確認したら、**RHMAP Studio** を使用してクライアントアプリをビルドすることができます。

以下のステップに従って **Android** デバイスのクライアントアプリをビルドします。

1. クライアントアプリのインターフェースから**ビルド**タブを選択します。

2. **Android**、**master** ブランチ、**Debug** ビルドタイプ、**Dev** クラウドコードアプリを選択してから**ビルド**ボタンをクリックします。クライアントアプリの **Android** ビルドが作成されます。
3. **Android** ビルドが完了すると **QR** コードが表示されるので、**Android** デバイス上で **QR Code Scanner** アプリを開いてビルドをインストールします。または、ショート **URL** を電話のブラウザに入力します。



#### 注記

**Android** デバッグバイナリーは証明書なしにビルドできますが、いずれのタイプの **iOS/Windows Phone** バイナリーのビルドにも必要な認証情報が必要です。



#### 注記

**branch** セレクターにより、ビルドするクライアントアプリのブランチを選択できます。この場合、デフォルトの **master** ブランチが適切なブランチになります。

これで **RHMAP** を使用して **Backbone HTML5 Cordova** アプリを作成することができました。

### 5.2.4. デプロイメントの概要

このセクションでは、サンプルのソリューションが機能するための必要なコードについて重点的に説明します。

#### 5.2.4.1. クラウドコードアプリ

まず、クラウドコードアプリについて考えてみましょう。この例ではクラウドコードアプリでクライアントアプリからのリクエストを受信し、**hello** パラメーターを変更し、以下のパラメーターを含む **JSON** オブジェクトを使用してクライアントアプリに応答できるようにします。

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

この機能をクラウドコードアプリに実装するには、以下を実行します。

1. **application.js** ファイルで、**lib** ディレクトリーにある **hello.js** ファイルを必要とする新規の **/hello** ルートが追加されます。
2. **hello.js** ファイルは 2 つのルートを作成します。どちらのルートも **hello** パラメーターを変更するための同じ操作を実行します。
  - **hello** パラメーターがクエリー文字列として追加される **GET** リクエスト。
  - **hello** パラメーターが **POST** リクエストの本文で送信される **POST** リクエスト。



#### 注記

このクラウドアプリは **Backbone** クライアントアプリとは完全に切り離されています。クラウドコードアプリはプロジェクト内の任意の数のクライアントアプリ間で共有できます。

#### 5.2.4.2. Backbone クライアントアプリ

クラウドコードアプリに **/hello** エンドポイントを作成した後は、**Backbone** クライアントアプリに追加した機能を検査し、この機能を使ってクラウドコードアプリで公開された **/hello** エンドポイントにリクエストを送信できるようにします。

クライアントアプリは、一部のテキストを受け入れる単一入力と入力をクラウドに送信し、結果をユーザーに表示する単一ボタンのある単純な **Backbone** アプリです。

#### 5.2.4.2.1. RHMAP SDK

まず、すべての HTML5 Cordova アプリの場合のように RHMAP Javascript SDK (**feedhenry.js**) は **index.html** ファイルに含まれています。この SDK により、すべての **\$fh クライアント API** 関数にアクセスできます。



#### 注記

**feedhenry.js** ファイルはテンプレートリポジトリでは空になっていることを確認できます。クライアントアプリの作成時に **feedhenry.js** ファイルが **www** ディレクトリーにあると、最新の RHMAP Javascript SDK がファイルに自動的に追加されます。

#### 5.2.4.2.2. fhconfig.json

クライアントアプリには **fhconfig.json** ファイルも含まれます。このファイルには、RHMAP Javascript SDK がクラウドアプリと通信するために必要な情報が含まれます。



#### 注記

すべての HTML5 クライアントアプリには、**\$fh** クライアント API 関数を使用するための **fhconfig.json** ファイルが含まれている必要があります。アプリの **Studio** での作成時に、このファイルに必要な情報が自動的に設定されます。

#### 5.2.4.2.3. \$fh.cloud

この **Backbone** クライアントアプリには、単一の**カウントビュー**があります。このビューは **index.html** ファイルの **hello** div にバインドされます。

カウントビューは **Get No. of Characters** ボタンの**クリックイベント**をリッスンします。このボタンがクリックされると、以下のようになります。

1. カウントビューは**クラウドヘルパー関数**を使用して **\$fh.cloud** クライアント API 関数を呼び出します。
2. 次に **\$fh.cloud** 関数は **GET** リクエストをクラウドコードアプリの**hello** エンドポイントに送信します。
3. **\$fh.cloud** 関数は、カウント操作の結果と共に **success** 関数を呼び出すか、またはリクエストが失敗した場合に **error** 関数を呼び出します。
4. カウントビューは関連するメッセージで更新されます。



## 注記

この場合、クライアントアプリは **GET** リクエストタイプを使用しています。クラウドコードアプリは **GET** および **POST** バージョンの **hello** エンドポイントの両方を公開し、**POST** リクエストタイプも機能します。これは **RESTful Backbone** モデルを扱う場合にとくに役立ちます。

## 5.3. RHMAP を使用した LONIC アプリの開発

### 概要

ここでは HTML5 Ionic Cordova アプリ内で RHMAP Javascript SDK を使用方法を説明します。

これには **Ionic Hello World Project** を新たに作成するために必要なステップが含まれ、ここではクラウドコードアプリと対話するために必要なコードが強調されます。

### 要件

- RHMAP アカウント
- HTML、JavaScript、[Ionic](#) および [Node.js](#) についての知識
- GitHub ユーザーアカウント
- [FHC](#) (インストール済み)

[Ionic のドキュメント](#) には Ionic アプリの開発を開始する上で必要なすべての情報が記載されています。

### 5.3.1. サンプルプロジェクトの概要

このサンプルプロジェクトは、1つのクライアントアプリと1つのクラウドアプリが含まれる単純なプロジェクトです。

#### 5.3.1.1. クライアントアプリ

クライアントアプリは、RHMAP Javascript SDK が含まれる単純な Ionic アプリです。クライアントアプリは、ユーザーに対してそれぞれの名前をテキストボックスに入力し、**Say Hello From The Cloud** ボタンをクリックするように求めます。次にクライアントアプリは `$fh.cloud` 関数を使用して入力されたテキストをクラウドコードアプリに送信します。

#### 5.3.1.2. クラウドコードアプリ

クラウドコードアプリは **hello** エンドポイントを公開してクライアントアプリからリクエストを受信し、送信されたテキストを変更して **Hello** を追加し、応答をクライアントに返します。

### 5.3.2. Ionic Hello World Project の新規作成

以下のステップを使用して新規の **Ionic Hello World Project** を作成します。

1. Studio にログインします。
2. 画面の左上にある **プロジェクト** タブを選択します。

3. 画面上部の **+** **新規プロジェクト** ボタンを選択します。プロジェクトテンプレートの一覧が表示されます。
4. **Ionic Hello World Project** テンプレートを選択し、新規プロジェクトの名前を指定します。
5. **次へ進む** ボタンを選択します。新規プロジェクトが作成されます。
6. プロジェクトが作成されたら、**終了** ボタンを選択します。

これで **Ionic** クライアントアプリとそれが通信するクラウドコードアプリが含まれる新規プロジェクトが作成されました。

アプリをプレビューするには、以下を実行します。

1. **アプリ、クラウドアプリ & サービスタグ** を選択します。
2. **アプリセクション** の下にある **クライアントアプリ** を選択します。ここから詳細画面に移動します。
3. **App Preview** には、クラウドコードアプリと通信する **Ionic** クライアントアプリの作業用バージョンが含まれます。

### 5.3.3. Android デバイス用クライアントアプリのビルド

アプリをテストして適切に機能することを確認したら、**RHMAP Studio** を使用してクライアントアプリをビルドすることができます。

以下のステップに従って **Android** デバイスのクライアントアプリをビルドします。

1. クライアントアプリのインターフェースから **ビルド** タブを選択します。
2. **Android**、**master** ブランチ、**Debug** ビルドタイプ、**Dev** クラウドコードアプリを選択してから **ビルド** ボタンをクリックします。クライアントアプリの **Android** ビルドが作成されます。
3. **Android** ビルドが完了すると **QR** コードが表示されるので、**Android** デバイス上で **QR Code Scanner** アプリを開いてビルドをインストールします。または、ショート URL を電話のブラウザに入力します。



#### 注記

**Android** デバッグバイナリーは証明書なしにビルドできますが、いずれのタイプの **iOS/Windows Phone** バイナリーのビルドにも必要な認証情報が必要です。



#### 注記

**branch** セレクターにより、ビルドするクライアントアプリのブランチを選択できます。この場合、デフォルトの **master** ブランチが適切なブランチになります。

これで **RHMAP** を使用して **Ionic HTML5 Cordova** アプリを作成することができました。

### 5.3.4. デプロイメントの概要

このセクションでは、サンプルのソリューションが機能するための必要なコードについて重点的に説明します。

### 5.3.4.1. クラウドコードアプリ

まず、クラウドコードアプリについて考えてみましょう。この例ではクラウドコードアプリでクライアントアプリからのリクエストを受信し、**hello** パラメーターを変更し、以下のパラメーターを含む JSON オブジェクトを使用してクライアントアプリに応答できるようにします。

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

この機能をクラウドコードアプリに実装するには、以下を実行します。

1. **application.js** ファイルで、**lib** ディレクトリーにある **hello.js** ファイルを必要とする新規の **/hello** ルートが追加されます。
2. **hello.js** ファイルは 2 つのルートを作成します。どちらのルートも **hello** パラメーターを変更するための同じ操作を実行します。
  - **hello** パラメーターがクエリー文字列として追加される **GET** リクエスト。
  - **hello** パラメーターが **POST** リクエストの本文で送信される **POST** リクエスト。



#### 注記

このクラウドアプリは **Ionic** クライアントアプリとは完全に切り離されています。クラウドコードアプリはプロジェクト内の任意の数のクライアントアプリ間で共有できます。

### 5.3.4.2. Ionic クライアントアプリ

クラウドコードアプリに **/hello** エンドポイントを作成した後は、**Ionic** クライアントアプリに追加した機能を検査し、この機能を使ってクラウドコードアプリで公開された **/hello** エンドポイントにリクエストを送信できるようにします。

クライアントアプリは、一部のテキストを受け入れる単一入力と入力をクラウドに送信し、結果をユーザーに表示する単一ボタンのある単純な **Ionic** アプリです。

#### 5.3.4.2.1. RHMAP SDK

まず、すべての HTML5 Cordova アプリの場合のように **RHMAP Javascript SDK (feedhenry.js)** は **index.html** ファイルに含まれています。この SDK により、すべての **\$fh クライアント API** 関数にアクセスできます。



#### 注記

**feedhenry.js** ファイルはテンプレートリポジトリでは空になっていることを確認できます。クライアントアプリの作成時に **feedhenry.js** ファイルが **www** ディレクトリーにあると、最新の **RHMAP Javascript SDK** がファイルに自動的に追加されます。

#### 5.3.4.2.2. fhconfig.json

クライアントアプリには **fhconfig.json** ファイルも含まれます。このファイルには、**RHMAP Javascript SDK** がクラウドアプリと通信するために必要な情報が含まれます。



## 注記

すべての HTML5 クライアントアプリには、**\$fh** クライアント API 関数を使用するための **fhconfig.json** ファイルが含まれている必要があります。アプリの Studio での作成時に、このファイルに必要な情報が自動的に設定されます。

### 5.3.4.2.3. \$fh.cloud

この例では、クラウドコードアプリの **hello** エンドポイントにリクエストを送信するために **\$fh.cloud** クライアント API 関数が使用されます。

**\$fh.cloud** 関数は **cloud.js** ファイルにあります。ここで **\$fh.cloud** 関数は **MainCtrl** コントローラーが使用する再利用可能なサービスとして公開されます。

Ionic アプリには、**MainCtrl** という単一のコントローラーがあります。このコントローラーは以下を実行します。

1. **example.html** ビューからユーザーの入力を受け入れる。
2. **fhcloud** サービスを使用してクラウドコードアプリの **hello** エンドポイントを呼び出す。
3. **\$fh.cloud** 呼び出しが成功するかどうかに応じ、**success** または **error** 関数を使用してクラウドコードアプリからの応答を処理する。



## 注記

この場合、クライアントアプリは **GET** リクエストタイプを使用しています。クラウドコードアプリは **GET** および **POST** バージョンの **hello** エンドポイントの両方を公開し、**POST** リクエストタイプも機能します。これは **RESTful** アプリケーションを扱う場合にとくに役立ちます。

## 5.4. CORDOVA プラグインの使用

Cordova プラグインは、ハイブリッドモバイルアプリのモバイルデバイス機能へのプラットフォームに依存しない **JavaScript** インターフェースを提供します。ストレージやカメラ、位置情報および他のサードパーティープラグインを含むデバイスの基本機能のための公式の **Apache** プラグインがいくつかあります。

Cordova プラグインの公式のレジストリーおよび配信チャンネルは **npm** であり、プラグイン ID は **npm** パッケージ ID に対応します。[Cordova プラグインページ](#)には公式のプラグインリストが記載されており、これはキーワード **ecosystem:cordova** を使用した **npm** パッケージのフィルターとして機能します。

### 5.4.1. サポートされるプラットフォーム

Cordova プラグインは **RHMAP** でサポートされるすべてのプラットフォームで使用できます。ただし、すべての Cordova プラグインがすべてのプラットフォームをサポートする訳ではありません。プラグインでサポートされるプラットフォームは **cordova.platforms** オブジェクトのプラグインの **package.json** ファイルで指定されるか、または Cordova プラグインの検索ページで指定されます。

### 5.4.2. プラグインのアプリへの追加

Cordova アプリケーションの開発時に、プラグインは **cordova プラグインアプリ** を使って追加されます。これにより、リポジトリからプラグインがダウンロードされ、必要なフォルダー構造が作成され



ると共にエントリーが **config.xml** ファイルに追加されます。詳細については、公式の Cordova ドキュメントの [Platforms and Plugins Version Management](#) を参照してください。

ただし RHMAP では、プラグインは RHMAP 固有の JSON ファイルの **config.json** でも宣言できます。これにより、Cordova Light アプリでプラグインを宣言でき、プラグインの仕様フォーマットが変更される場合もアプリがその変更に対応できるようになります。必要な変更は RHMAP Build Farm に 1 回で実装できるため、すべての開発者が各自のアプリを更新することを強制されることはありません。

```
{
  "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"
      }
    }
  ]
}
```

上記の例では、npm から利用可能な Device プラグインと Geolocation プラグイン、および指定の Github リポジトリからのみ利用可能な Local WebServer プラグインの 3 つのプラグインが指定されています。

#### 5.4.2.1. 仕様

**config.json** ファイルは Cordova アプリの **www** フォルダか、または Cordova Light アプリの **root** に置かれる必要があります。

このファイルには **plugins** というキーが含まれている必要があり、その値は JSON オブジェクトの配列であり、以下のプロパティを定義するために使用されます。

- **id (必須)**  
これは Cordova プラグインのグローバルに一意の ID です。これはその npm パッケージ ID に対応します。また、この ID は [Cordova プラグインの仕様](#) で説明されているようにプラグインの **plugin.xml** ファイルでも確認できます。
- **version (必須)**  
使用するプラグインのバージョンです。npm パッケージバージョンに対応します。Git で配信されるプラグインの場合にのみ、プラグインのバージョンをその **plugin.xml** で確認できます。

npm で配信されるプラグインの場合、**最新の**値を使用すると利用可能な最新バージョンを常に使用することができます。ただし、プラグインのアップグレードによって後方互換性が失われる可能性があるため、お使いのアプリで機能することが証明されている特定のバージョンを使



用することを強くお勧めします。

- **url**

有効な Cordova プラグインが含まれる公式の Git リポジトリ URL (**plugin.xml** ファイルを含む) です。

このフィールドに指定される値は、**id** の値または **version** フィールドの値にかかわらずプラグインをダウンロードするために使用されます。このフィールドが指定されていない場合、プラグインは **id** および **version** フィールドの値を使用して npm からダウンロードされます。

また [公式の Cordova CLI ドキュメント](#) で説明されているように、特定の Git の参照 (ref) およびリポジトリ内のプラグインへのパスを指定することもできます。

- Git の参照 (ref) オブジェクトは、URL に **#<git-ref>** を追加して指定できます。お使いのアプリで機能するかどうかについてテスト済みのタグまたはその他の安定した参照 (ref) を使用することを強くお勧めします。**master** を参照 (ref) として使用すると、ビルドのたびにプラグインコードが変更され、アプリケーションが破損する可能性があります。
- プラグインを含むディレクトリへのパスは URL の **:<path>** で指定できます。

たとえば、リポジトリの **release** ブランチの **plugin** サブディレクトリにあるプラグインを取得する必要がある場合、URL の形式は以下のようになります。

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

プラグインがダウンロードされた後に、プラグイン ID が **config.json** で指定される **id** フィールドと一致することを確認するためにその **plugin.xml** ファイルが解析されます。

- **preferences**

プラグイン設定を指定するには、上記の例の **CordovaLocalWebServerStartOnSimulator** のように **preferences** オブジェクトのキーと値のペアを使用します。

- **variables**

一部のプラグインは、文字列の値をパラメーター化するために **plugin.xml** で変数を使用します。このフィールドに変数の値をキーと値のペアで指定できます。変数の詳細については、[公式のドキュメント](#) を参照してください。

### 5.4.3. Cordova Light アプリの考慮事項

Cordova Light App の後方互換性を維持するため、有効な **config.json** ファイルが提供されない場合は、Build Farm でのアプリのビルド時にデフォルトのプラグインのセットが適用されます。**config.json** ファイルを提供し、アプリに必要な Cordova プラグインのみを指定することを強くお勧めします。これを実行することにより、以下が可能になります。

- プラグインの予期しない変更によるアプリへの影響を回避  
デフォルトプラグインの一覧は変更されまゝであるとは保証できず、これに変更が加わると、後方互換性が失われる可能性があります。独自の **config.json** ファイルを提供することにより、アプリはプラグインバージョンの一定のリストを常に使用できるため、ビルドの安定性が高まります。
- アプリのビルド時間とバイナリーのサイズを縮小  
プラグインはアプリのビルド時にダウンロードおよびインストールされ、ビルドされるバイナリーに組み込まれます。そのため、プラグインの数が減ると、ビルド時間やバイナリーのサイズが縮小することになります。

### 5.4.3.1. デフォルトプラグイン

#### 公式プラグイン

- 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)

#### オプション

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

#### カスタム RHMAP プラグイン:

- [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](#)

#### 5.4.4. ブラウザーでのアプリのテスト

プラグインが **config.json** ファイルに指定される場合、プラグインの **JavaScript** オブジェクトはビルドおよびデバイスへのインストールが終了するまで利用できません。そのため、アプリでプラグインの **JavaScript** オブジェクトを参照し、アプリを **Studio** の **App Preview** にロードしようとする、**object undefined** (オブジェクトが定義されていません) というエラーが出される可能性があります。

これを解決するには、プラグイン **API** を呼び出す際に防御的なチェック (**defensive checking**) を使用します。たとえば **Cordova camera API** の以下の呼び出しにより、「オブジェクトが定義されていません (**object undefined**)」というエラーが出されます。

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

ただし、**API** が定義されているかどうかを確認することで、エラーを処理できます。

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

### 5.5. アプリでのセキュアキーの使用

**\$fh.sec** **API** は、キーおよびデータの暗号化/復号化を生成する機能を提供します。ただし、キーの生成後は今後の使用に備えてそれらをどこかに保存する必要があるかもしれません。たとえば、シークレットキーで暗号化する必要のあるデータがあり、これをデバイスに保存したとします。次にアプリケーションが再度起動すると、同じシークレットキーを取得して、データを復号化する必要があります。

これを実行するための最適な方法として、キーをクラウド側に保存し、クライアントの固有 **ID (CUID)** およびアプリ **ID** を使用してそれらのキーをクライアントに関連付けることができます。

#### 5.5.1. CUID およびアプリ ID

**CUID** とアプリ **ID** の両方はすべての **\$fh.act** リクエストでクライアント **SDK** によって送信されます。これらは **\_\_fh** という特殊な **JSON** オブジェクトからアクセスできます。**CUID** は各デバイスに固有であり、アプリが削除されたり再インストールされる場合でもそのまま変更されません。アプリ **ID** はアプリがプラットフォームで作成されると生成され、これも変更されません。以下のコードを使用し、クラウドコードでそれらにアクセスできます。

```
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
    ....
}

```

### 5.5.2. キーの永続化

キーをクラウドに保存するために任意の永続的なメカニズムを使用できます。推奨される1つの方法として、**\$fh.db**を使用することができます。以下は、**\$fh.db**を使用してキーを保存し、取得する方法を示すサンプルコードです。

```

//read a key using $fh.db
var getKey = function(params, cb){
  if(typeof $fh !== "undefined" && $fh.db){
    $fh.db({
      act:'list',
      'type': 'securityKeys',
      eq: {
        "id": getKeyId(params), //The id is generated using the above
example code
        "keyType": params.type
      }
    }, function(err, data){
      if(err) return cb(err);
      if(data.count > 0){
        return cb(undefined, data.list[0].fields.keyValue);
      } else {
        return cb(undefined, undefined);
      }
    });
  } else {
    console.log("$fh.db not defined");
    cb("$fh.db not defined");
  }
}

//save a key using $fh.db
var saveKey = function(params, cb){
  if(typeof $fh !== "undefined" && $fh.db){
    //first check if a key with the same id and type already exists
    $fh.db({
      act:'list',
      'type': 'securityKeys',
      eq: {
        "id": getKeyId(params),
        "keyType": params.type
      }
    }, function(err, data){
      if(err) return cb(err);
      //a key with the same id and type already exists, update it
      if(data.count > 0){
        $fh.db({
          'act':'update',
          'type': 'securityKeys',
          'guid': data.list[0].guid,
          'fields' : {

```

```

        'id': getKeyId(params),
        'keyType': params.type,
        'keyValue' : params.value
    }
}, function(err, result){
    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");
}
}

```

### 5.5.3. サンプルコード

**\$fh.sec** API を使用方法を詳細に説明した参照アプリケーションが作成されており、このアプリケーションのコードは GitHub (<https://github.com/feedhenry-training/fh-security-demo-app>) で利用できます。

## 5.6. アプリのデバッグ

デバッグはアプリ開発に不可欠な部分です。Web アプリの場合、これはコードの特定部分の変数の値を表示するための **alert()** 関数を使用するだけで実行できます。コンソールのあるブラウザの場合、**console.log** 関数を使ってこの種の情報のログを受動的に記録することができます。Web アプリケーションのより高度なデバッグ方法には、DOM の状態検査、JavaScript コードのブレークポイント設定およびコードのステップ実行、またはオンザフライでの DOM および関連付けられたスタイルの更新などがあります。

このページでは、クラスプラットフォームアプリの開発時に使用できるデバッグツールについて説明します。これらのツールは **Studio** またはデバイスで使用できます。

デバッグ用のアプリや iOS および **Android** でデバッグを実行する方法の詳細については、この [ブログ](#) を参照してください。

### 5.6.1. Studio コンソール

ほとんどのブラウザはコンソールロガーをサポートしています。さまざまなログレベルをメッセージと共に呼び出すことができます。以下は例になります。

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

このコンソール出力をデバッグするには、ブラウザーで関連する Web デバッグツールを開く必要があります。

クラウドコードをデバッグするには、単純に `console.log()` および `console.error()` を使用します。対応するログファイルは「ログ」セクションの下に表示されるか、または FHC を使用して表示できます。

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

### 5.6.2. Firebug

開発用として選択したブラウザーが Firefox の場合、**Firebug** ツールを使うとデバッグが非常に簡単になります。**Firebug** は Firefox のアドオンですが、これは多数のアップデートが行われる非常にアクティブなアドオンです。以下は、**Firebug** を使用して実行できるいくつかの点になります。

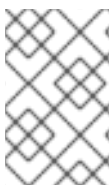
- コンソール出力の表示
- リソース要求の表示
- スクリプト実行のデバッグ
- ブレークポイントのデバッグ時のコードの動的実行
- DOM の表示/更新
- スタイルの表示/更新
- ローカルストレージ (DB) の表示/更新

上記の機能を使用すると、アプリのデバッグは非常に簡単になります。アプリが **Firebug** による警告またはエラーなしに起動する場合、そのアプリはクロスプラットフォームで機能できる可能性があります。

### 5.6.3. Chrome の Web Inspector

**Web Inspector** は **Firebug** によく似ています。これは **Firebug** とほぼ同じ機能を提供しますが、設定なしに **WebKit** ブラウザーに組み込まれるという利点があります。つまり、**Web Inspector** は **Google Chrome** および **Safari** で使用することができます。**Chrome** ではこのツールをデフォルトで有効にでき、web ページの任意のオブジェクトをクリックして「**Inspect Element (要素を検証)**」を選択することにより、コンテキストで開始できます。

### 5.6.4. Safari / iOS 6+



#### 注記

Safari での iOS でのリモートデバッグは、開発プロビジョニングプロファイルでビルドされたアプリケーション用にのみ有効にできます。Ad Hoc および Distribution アプリをこの方法でデバッグすることはできません。

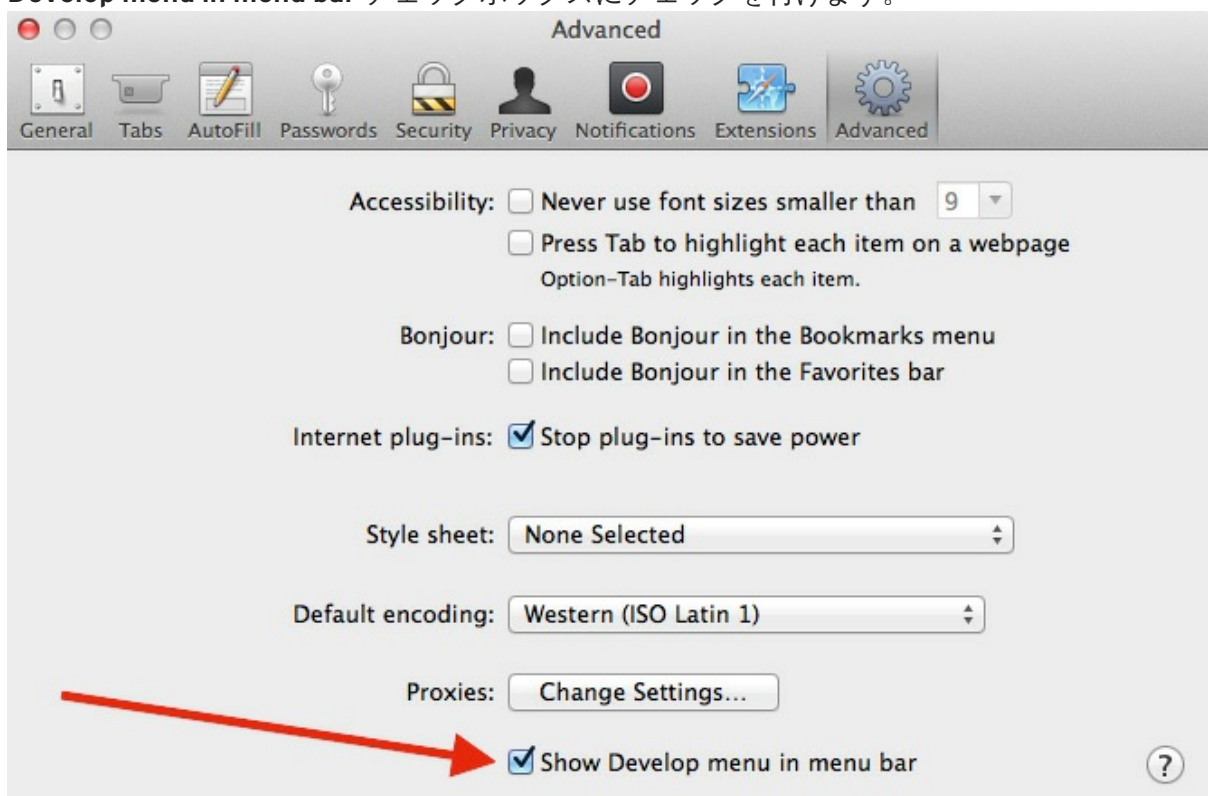
1. iOS デバイスの「**Settings app**」に移動します。



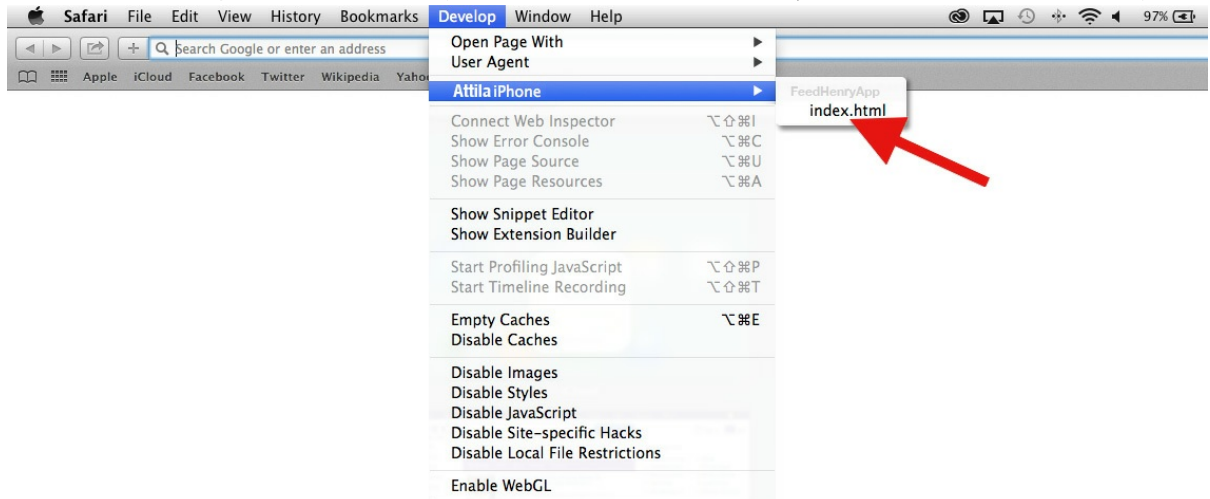
2. Safari → **Advanced** に移動してから **Web Inspector** スイッチを切り替えます。



3. デスクトップ Safari で、**Safari → Preferences** に移動し、**Advanced** を選択してから **Show Develop menu in menu bar** チェックボックスにチェックを付けます。



4. USB 経由で iOS デバイスを開発マシンに接続します。
5. デバッグに必要なアプリを開きます。これは Safari の **Develop** メニューから使用できます。



6. **Develop** メニューには、**User Agent** スイッチャーのような追加ツールがあります。これにより、ブラウザを **Mobile Safari** などの異なるブラウザとして表示することができます。この機能は、アプリのモバイル Internet 版を開発する場合に役立ちます。

### 5.6.5. Chrome / Android 4.0+



#### 注記

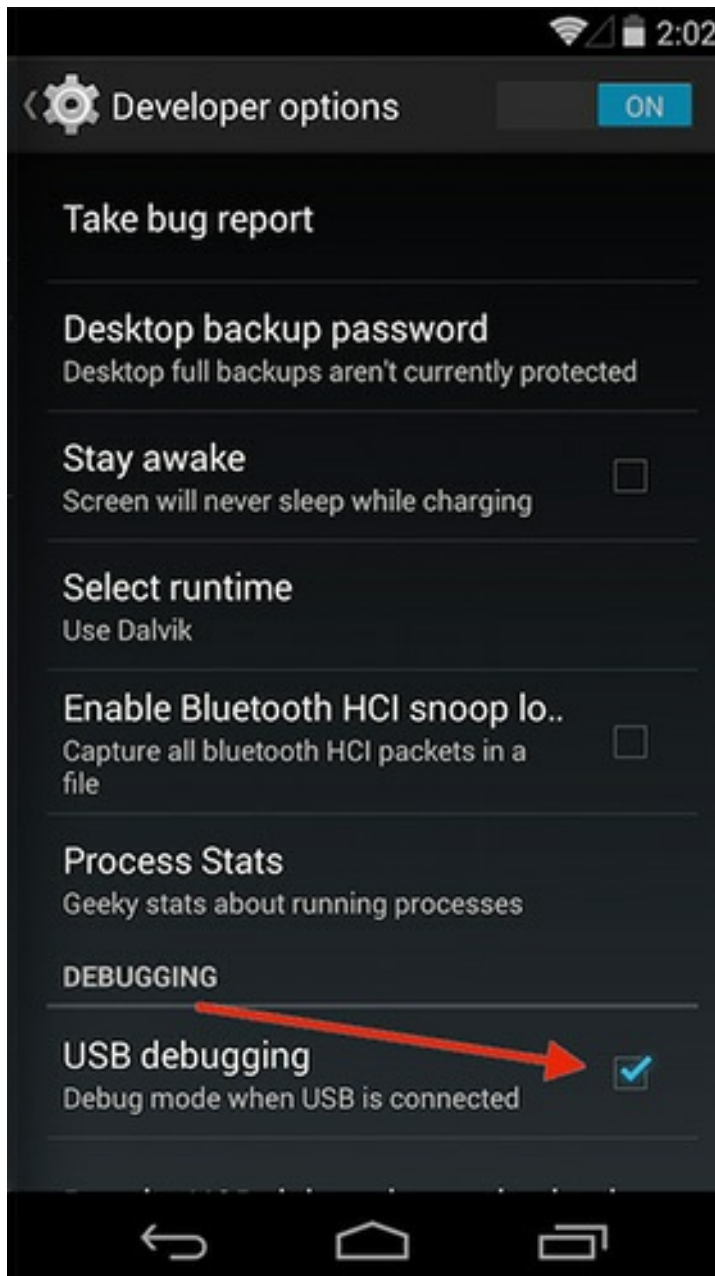
Chrome での Android のリモートデバッグは、デバッグ可能フラグが付けられたアプリケーション用にのみ有効にできます。Cordova 3.3 以降を使用している場合、**android:debuggable="true"** を **AndroidManifest.xml** ファイルの **<application>** 要素に追加します。Cordova 3.2 以前のバージョンを使用している場合には、WebView デバッグを有効にする必要があります。

1. Android デバイスで USB デバッグを有効にします。Android 4.2 以降のバージョンでは、開発者のオプションは非表示にされています。これを利用可能にするには、**Settings** → **About phone** に移動し、**Build number** を 7 回タップします。





2. 直前の画面に戻り、**Developer options** を見つけて **USB debugging** チェックボックスにチェックを付けます。**Stay awake** オプションを選択することも推奨されています。



3. デスクトップ Chrome ブラウザーの右上のメニューから、**Tools → Inspect Devices** を選択します。
4. **Discover USB devices** にチェックを付けます (選択されていない場合)。
5. USB 経由で Android デバイスに接続します。
6. このデバイスを開発用に初めて接続する場合、開発マシンから USB デバッグのパーミッションを要求するアラートがデバイスに表示される可能性があります。このアラートが毎回表示されないようにするには、**Always allow from this computer** にチェックを付けてから、OK をタップします。
7. アプリをデバッグするには、これをデバイス上で実行する必要があります。
8. Chrome では、**Inspect** リンクをクリックして DevTools を開きます。



### 5.6.6. オンデバイス (On-Device)

デバイス上のデバッグは、デスクトップブラウザ内でのデバッグに比べると単純な初期段階のデバッグと言えます。iOS などの一部のプラットフォームでは、コンソールを利用でき、Developer モードが有効にされると javascript エラーが表示されますが、これは完全に機能が搭載された Firebug または Web Inspector などの高度な機能とは比較になりません。

#### 5.6.6.1. Weinre

Weinre は、現時点で最も良いオンデバイスデバッグツールと言えるでしょう。これは webkit ブラウザーのみを正式にサポートします。つまり、このツールは Android および iOS で機能します。詳細一覧およびバージョン番号はこのサイトで確認できます。

Weinre は、デバイス上のアプリから Weinre サーバーへのリモートのデバッグセッションをセットアップすると機能します。Weinre は、Web Inspector にアクセスし、アプリをリモートでデバッグするために使用できる web サーバーを実行します。本書の作成時点で、この Web Inspector セッションの機能は DOM のデバッグおよび更新を可能にします。リモートコンソールも表示され、開発者はコンソールからアプリのコードを動的に実行できます。

アプリケーションで Weinre のデバッグを有効にするには、いくつかのステップを実行する必要があります。

1. デバイスの接続先となる Weinre jar を稼働させます。これを実行する方法として 2 つの方法があります。
  - インターネット上でアクセス可能なマシンで jar ファイルを実行する。
  - デバイスと同じネットワーク上のマシンで jar を実行します。つまり、このデバイスは、WiFi 経由で開発者マシンと同じルーター/アクセスポイントに接続されていることとなります。

jar をコマンドラインから実行する場合、すべてのインターフェースでリッスンするように **boundHost** に値 **-all-** を使用することをお勧めします。

2. アプリをデバイスにビルドし、デプロイする前にアプリの HTML にスクリプトを追加します。Weinre のドキュメントによると、組み込まれるスクリプトは以下のようになります。

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

使用されるアドレス **some.server.xyz** は Weinre サーバーを実行しているマシンのアドレスと一致する必要があります (ip アドレスも機能します)。

3. アプリをデバイスにデプロイし、これを起動します。(開発者マシンの) Weinre サーバーの web ページを開くと、デバッグユーザーインターフェースへのリンクが表示されます。このリンクから **Web Inspector** が開き、アプリのリモートデバッグが可能になります。  
ここまではサードパーティーツールについての説明であり、このセットアッププロセスに関する更新情報については、公式サイトをご確認いただくことをお勧めします。

## 5.7. CORDOVA LIGHT CLIENT アプリの CORDOVA への移行

### 5.7.1. 概要

Cordova Light アプリの標準 Cordova への移行により、アプリの最新 RHMAP 機能との互換性が確保されますが、以下の点に注意してください。

- Cordova Light では、iOS、iPhone および iPad という iOS アプリの 3 つの異なる種類を作成することができました。標準 Cordova アプリは (バリエーションなしの) iOS をサポートします。移行時には Cordova Light の iPhone 設定オプションのみが送信されます。移行前に、[この手順](#)で説明されているように保存する必要のある iOS および iPad 設定オプションが Studio の iPhone 設定オプションに適用されていることを確認してください。
- Cordova Light アプリに **config.json** ファイルが含まれていない場合は、移行ツールが **config.xml** ファイルに、Build Farm でサポートされるデフォルトのプラグインを設定します。[この手順](#)で説明されているように不要なプラグインは削除することが推奨されています。

#### 注記

Cordova Light は、標準 Cordova が導入されたため非推奨となりました。

Cordova Light アプリから Cordova への移行は必須ではありません。テンプレートから新規の Cordova Light アプリを作成することはできなくなりましたが、それらを RHMAP Studio のプロジェクトにインポートしたり、Build Farm を使用してビルドしたりすることができます。

既存の Cordova Light クライアントアプリを Studio にインポートするには、以下を実行します。

1. プロジェクトに移動します。
2. アプリ、クラウドアプリ & サービスに移動します。
3. + ボタンをクリックして新規アプリを追加します。
4. 既存アプリのインポート ボタンをクリックします。
5. Cordova Light アプリタイプを選択します。

1. [既存アプリのバックアップ](#)
2. [アプリ移行の設定](#)
3. [アプリの移行](#)

## 4. 移行後のアプリの設定

### 5.7.2. 既存アプリのバックアップ

既存の **Cordova Light** アプリをバックアップするには、アプリの **詳細** に移動し、**クローン** をクリックします。これにより、元のアプリと同じソースコードを持つ新規の **Cordova Light** アプリが作成されます。

### 5.7.3. アプリ移行の設定

1. アプリの **設定 (Config)** ページに移動します。



#### 注記

**設定 (Config)** オプションは **Cordova Light** アプリの左側のメニューにのみ表示されます。

2. 保存する必要がある **iOS** および **iPad** 設定オプションが **Studio** の **iPhone** 設定オプションに適用されていることを確認します。設定の移行方法についての詳細は、[移行後のアプリの設定](#) を参照してください。
3. **Cordova Light** アプリが **RHMAP 3.11** よりも前にビルドされている場合、[RHMAP 3.11 アップグレードドキュメント](#) を使用してアプリをアップグレードし、最新のプラグインを使用できるようにします。

### 5.7.4. アプリの移行

**Cordova Light** アプリは、**Studio** または **FHC コマンドラインツール** のいずれかを使用して移行することができます。

**Studio** を使用してアプリを移行するには、以下の手順を実行します。

1. アプリの **設定 (Config)** ページに移動します。
2. **Full Cordova への移行 (Migrate to Full Cordova)** をクリックします。
3. **iPad** および **iOS** 設定を **iPhone** 設定に移動したことを確認します。

移行が完了すると、**Studio** に通知が表示されます。

**FHC** を使用してアプリを移行するには、以下の手順を実行します。

1. **FHC コマンドラインツール** をセットアップします。
2. 以下のコマンドを使用してアプリを移行します。

```
fhc cordova migrate --app=<CORDOVA_LIGHT_APP>
```

ここで、**CORDOVA\_LIGHT\_APP** は **Cordova Light** アプリのアプリ ID です。これはアプリの **詳細 (Details)** ページで参照できます。

移行ツール:

1. 標準 **Cordova config.xml** ファイルを作成します。

2. **config.xml** ファイルに値がアプリの名前に設定された **name** タグを作成します。
3. **config.xml** にクラウドアプリドメインが設定された **access** タグを作成し、すべてのサブドメインを許可します。
4. **access** タグを **config.xml** に作成し、**cdvfile:/\*** を許可します。
5. **cordova.js** を **index.html** に追加します (存在しない場合)。これはプラグインを実行するのに必要です。
6. 「設定 (Config)」タブの値を該当する Cordova の値に移行し、**config.xml** にプラグインまたは設定を追加します。
7. **www/config.json** プラグインオブジェクトを **config.xml** で標準 Cordova プラグインエンタリに変換します。
8. **cordova-plugin-splashscreen** プラグインをインストールします。
9. スプラッシュ画像およびアイコンの参照を **config.xml** ファイルに追加します。
10. アプリタイプを **Cordova Light** から **Cordova** に変更します。
11. **www/config.json** ファイルを削除します。

### 5.7.5. 移行後のアプリの設定

1. アプリが外部サービスに接続する必要がある場合に **access** タグを追加します。移行ツールはクラウドアプリドメインが設定された **access** タグを自動的に作成し、すべてのサブドメインと **cdvfile:/\*** を許可します。**access** タグについての詳細は、[Cordova Whitelist Guide](#) を参照してください。
2. 不要なプラグインを削除します。**Cordova Light** アプリに **config.json** ファイルが含まれない場合、移行ツールは **config.xml** ファイルに **Cordova 6.3.1** と互換性のある以下のプラグインを設定します。ご使用のアプリではこれらのプラグインのいずれも不要な場合があります。

```
<plugin name="cordova-plugin-device" spec="1.1.2"/>
<plugin name="cordova-plugin-network-information" spec="1.2.1"/>
<plugin name="cordova-plugin-battery-status" spec="1.1.2"/>
<plugin name="cordova-plugin-device-motion" spec="1.2.1"/>
<plugin name="cordova-plugin-device-orientation" spec="1.0.3"/>
<plugin name="cordova-plugin-geolocation" spec="2.4.1"/>
<plugin name="cordova-plugin-file" spec="4.2.0"/>
<plugin name="cordova-plugin-camera" spec="2.3.1"/>
<plugin name="cordova-plugin-media" spec="2.4.1"/>
<plugin name="cordova-plugin-media-capture" spec="1.4.1"/>
<plugin name="cordova-plugin-file-transfer" spec="1.5.1"/>
<plugin name="cordova-plugin-dialogs" spec="1.2.1"/>
<plugin name="cordova-plugin-vibration" spec="0.3.7"/>
<plugin name="cordova-plugin-contacts" spec="2.1.0"/>
<plugin name="cordova-plugin-globalization" spec="1.0.3"/>
<plugin name="cordova-plugin-inappbrowser" spec="1.4.0"/>
<plugin name="cordova-plugin-console" spec="1.0.3"/>
<plugin name="cordova-plugin-whitelist" spec="1.2.2"/>
<plugin name="cordova-plugin-screen-orientation" spec="v1.4.2"/>
<plugin name="com.feedhenry.plugins.apkdownloader"
spec="https://github.com/feedhenry/fh-cordova-plugin-
```



```
apkdownloader.git#fh0.0.1"/>
<plugin name="fh-cordova-plugin-device "
spec="https://github.com/feedhenry/fh-cordova-plugin-
device.git#fh0.0.3"/>
<plugin name="cordova-sms-plugin" spec="v0.1.11"/>
<plugin name="cordova-plugin-statusbar" spec="~2.2.1"/>
<plugin name="fh-cordova-plugin-smsbuilder"
spec="https://github.com/feedhenry/sms-builder.git#fh0.1.2"/>
<plugin name="fh-cordova-plugin-ftputil"
spec="https://github.com/feedhenry/fh-cordova-plugin-
ftputil.git#fh0.0.2"/>
```

3. 標準 Cordova アプリは (バリエーションなしの) iOS をサポートします。以下の Cordova Light から標準 Cordova への設定オプションの移行について注意してください。

a. 以下の iPhone オプションの移行に注意してください。

- アクティビティースピナー (Activity Spinner)

アクティビティースピナー (Activity Spinner) フィールドの値が **上 (Top)** または **中央 (Center)** の場合、移行ツールは値 **true** の **ShowSplashScreenSpinner** 設定を **config.xml** ファイルに追加します。たとえば、**<preference name="ShowSplashScreenSpinner" value="true"/>** のようになります。値が **非表示 (Hidden)** の場合、値が **false** の **ShowSplashScreenSpinner** 設定を追加します。

- バンドル識別子 (Bundle Identifier)

バンドル識別子 (Bundle Identifier) フィールドの値が **Android パッケージ名 (Android Package Name)** の値と同一の場合、その値は **config.xml** ファイルの **id** ウィジェット属性に使用されます。値が異なる場合は、**バンドル識別子 (Bundle Identifier)** フィールドの値が **config.xml** ファイルの **ios-CFBundleIdentifier** ウィジェット属性に使用されます。値が指定されていない場合は、デフォルトの **org.feedhenry.dart.helloworld.cordova** が使用されます。**ios-CFBundleIdentifier** 属性は iOS アプリのバンドル識別子を設定するために使用されます。

- 自動回転 (Auto Rotate)

自動回転 (Auto Rotate) オプションにチェックが付けられていない場合、**画面の向き (Orientation)** 設定タグが **config.xml** ファイルの **ios** プラットフォームタグ内に **画面の向き (Orientation)** フィールドの値と共に追加されます。たとえば、**<preference name="Orientation" value="landscape" />** のようになります。

- 自動非表示スプラッシュ (Auto Hide Splash)

自動非表示スプラッシュ (Auto Hide Splash) オプションにチェックが付けられている場合、値が **true** の **AutoHideSplashScreen** 設定が **config.xml** ファイルに追加されます。たとえば、**<preference name="AutoHideSplashScreen" value="true" />** のようになります。このオプションにチェックが付けられていない場合は、値 **false** が使用されます。

- 背景音 (Background Audio): 背景音 (Background Audio) オプションにチェックが付けられている場合、**cordova-plugin-settings-hook** プラグインタグおよびこのコードが **config.xml** ファイルの **widget** 要素内に追加されます。

```
<plugin name="cordova-plugin-settings-hook" spec="~0.2.4"/>
<config-file parent="UIBackgroundModes" platform="ios"
```

```
target="*-Info.plist">
  <array>
    <string>audio</string>
  </array>
</config-file>
```

- ステータスバーの非表示 (Hide Status Bar): ステータスバーの非表示 (Hide Status Bar) オプションにチェックが付けられている場合、**cordova-plugin-settings-hook** プラグインタグおよびこのコードが **config.xml** ファイルの **widget** 要素内に追加されます。

```
<plugin name="cordova-plugin-settings-hook" spec="~0.2.4"/>
<platform name="ios">
  <config-file parent="UIStatusBarHidden" platform="ios"
target="*-Info.plist">
    <true/>
  </config-file>
  <config-file
parent="UIViewControllerBasedStatusBarAppearance"
platform="ios" target="*-Info.plist">
    <false/>
  </config-file>
</platform>
```

- 旧スタイルのステータスバー (Old Style Status Bar)
- 旧スタイルのステータスバー (Old Style Status Bar) オプションにチェックが付けられている場合、**cordova-plugin-statusbar** のプラグインタグ、値が **false** の **StatusBarOverlaysWebView** 設定タグ、および値が **#000000** の **StatusBarBackgroundColor** 設定タグが **config.xml** ファイルの **widget** 要素内に追加されます。

```
<plugin name="cordova-plugin-statusbar" spec="~2.2.1"/>
<preference name="StatusBarOverlaysWebView"
value="false"/>
<preference name="StatusBarBackgroundColor"
value="#000000"/>
```

- 画面の向き (Orientation)  
自動回転 (Auto Rotate) にチェックが付けられていない場合、画面の向き (Orientation) の値が使用され、**config.xml** ファイルの **ios** プラットフォームタグ内に 画面の向き (Orientation) 設定タグが作成されます。
- リモートデバッグ (Remote Debug): この値は使用されません。
- iphone5 Retina スプラッシュ画像 (iphone5 Retina Splash Image) : この値は使用されませんが、**config.xml** ファイルの末尾にコメントとして記録されます。
- Retina スプラッシュ画像 (Retina Splash Image) : この値は使用されませんが、**config.xml** ファイルの末尾にコメントとして記録されます。
- スプラッシュ画像 (Splash Image): この値は使用されませんが、**config.xml** ファイルの末尾にコメントとして記録されます。
- バージョン名 (Version Name)



バージョン名 (Version Name) は `config.xml` ファイルの `widget` 要素の `ios-CFBundleVersion` 属性を設定するために使用されます。`widget` 要素の `version` 属性の場合、Android タブのバージョン名 (Version Name) が代わりに使用されます。Cordova は `ios-CFBundleVersion` を使用してアプリのビルドバージョンである `CFBundleVersion` を設定し、`version` を使用してアプリの表示されるバージョンである `CFBundleShortVersionString` を設定します。アプリを異なる iOS バージョン名でビルドする場合には、移行後に `config.xml` ファイルの `version` 属性を `ios-CFBundleVersion` 属性の値に置き換えます。たとえば、生成される `widget` 要素を以下のように更新します。

```
<widget version="1.2" ios-CFBundleVersion="1.2">
  ...
</widget>
```

b. 以下の iPad オプションの移行に注意してください。

- **アクティビティスピナー (Activity Spinner):** この値は使用されず、代わりに iPhone タブのこのフィールドの値が使用されます。
- **バンドル識別子 (Bundle Identifier):** この値は使用されず、代わりに iPhone タブのこのフィールドの値が使用されます。
- **自動回転 (Auto Rotate):** この値は使用されず、代わりに iPhone タブのこのフィールドの値が使用されます。
- **自動非表示スプラッシュ (Auto Hide Splash):** この値は使用されず、代わりに iPhone タブのこのフィールドの値が使用されます。
- **背景音 (Background Audio):** この値は使用されず、代わりに iPhone タブのこのフィールドの値が使用されます。
- **Flurry アプリケーションキー (Flurry Application Key):** この値は使用されず、代わりに iPhone タブのこのフィールドの値が使用されます。
- **ステータスバーの非表示 (Hide Status Bar):** この値は使用されず、代わりに iPhone タブのこのフィールドの値が使用されます。
- **旧スタイルのステータスバー (Old Style Status Bar):** この値は使用されず、代わりに iPhone タブのこのフィールドの値が使用されます。
- **iPad 横向き Retina スプラッシュ画像 (iPad Landscape Retina Splash Image) :** この値は使用されませんが、`config.xml` ファイルの末尾にコメントとして記録されます。
- **iPad Retina スプラッシュ画像 (iPad Retina Splash Image) :** この値は使用されませんが、`config.xml` ファイルの末尾にコメントとして記録されます。
- **横向きスプラッシュ画像 (Landscape Splash Image):** この値は使用されませんが、`config.xml` ファイルの末尾にコメントとして記録されます。
- **画面の向き (Orientation):** この値は使用されず、代わりに iPhone タブのこのフィールドの値が使用されます。
- **リモートデバッグ (Remote Debug):** この値は使用されません。
- **スプラッシュ画像 (Splash Image):** この値は使用されませんが、`config.xml` ファイルの末尾にコメントとして記録されます。

- **バージョン名 (Version Name):** この値は使用されず、代わりに iPhone タブのこのフィールドの値が使用されます。
- c. iOS タブのすべての設定は移行ツールによって無視されます。iOS の設定を保存するには、移行前にこれを iPhone タブに移動する必要があります。
- d. 以下の Android オプションの移行に注意してください。
  - **Android パッケージ名 (Android Package Name)**  
**Android パッケージ名 (Android Package Name)** フィールドの値が **バンドル識別子 (Bundle Identifier)** の値と同一の場合、その値は **config.xml** の **id** ウィジェット属性に使用されます。値が異なる場合は、**Android パッケージ名 (Android Package Name)** フィールドの値が **config.xml** ファイルの **android-packageName** ウィジェット属性に使用されます。値が指定されていない場合は、デフォルトの値である **org.feedhenry.dart.helloworld.cordova** が使用されます。この属性は Android アプリパッケージ名を設定するために使用されます。
  - **自動回転 (Auto Rotate)**  
**自動回転 (Auto Rotate)** オプションにチェックが付けられていない場合、**画面の向き (Orientation)** 設定タグが **config.xml** ファイルの Android プラットフォームタグ内に **画面の向き (Orientation)** フィールドの値と共に追加されます。たとえば、`<preference name="Orientation" value="landscape" />` のようになります。
  - **Flurry アプリケーションキー (Flurry Application Key)**  
iPhone タブの **Flurry アプリケーションキー (Flurry Application Key)** フィールドに値がない場合、この値が **config.xml** ファイルの末尾にコメントとして記録されますが、Flurry プラグインはインストールされていないためにこの値は使用されません。iPhone および Android タブのどちらにも **Flurry アプリケーションキー (Flurry Application Key)** の値がある場合には、Android の値が無視されます。
  - **フルスクリーン (Full Screen)**  
**フルスクリーン (Full Screen)** にチェックが付けられている場合、**フルスクリーン (Full Screen)** 設定タグが **config.xml** ファイルの android プラットフォームタグ内に、値 **true** と共に追加されます。たとえば、`<preference name="Fullscreen" value="true" />` のようになります。
  - **画面の向き (Orientation)**  
**自動回転 (Auto Rotate)** にチェックが付けられていない場合、**画面の向き (Orientation)** の値が使用され、**config.xml** ファイルの android プラットフォームタグ内に **画面の向き (Orientation)** 設定タグが作成されます。
  - **音声パーミッション (Permission Audio):** このフィールドは無視されます。パーミッションは関連付けられたプラグインで処理されます。
  - **カメラパーミッション (Permission Camera):** このフィールドは無視されます。パーミッションは関連付けられたプラグインで処理されます。
  - **連絡先パーミッション (Permission Contacts):** このフィールドは無視されます。パーミッションは関連付けられたプラグインで処理されます。
  - **場所パーミッション (Permission Location):** このフィールドは無視されます。パーミッションは関連付けられたプラグインで処理されます。

- **電話の状態読み取りパーミッション (Permission Read Phone State):** このフィールドは無視されます。パーミッションは関連付けられたプラグインで処理されます。
- **SMS 受信パーミッション (Permission Receive SMS):** このフィールドは無視されます。パーミッションは関連付けられたプラグインで処理されます。
- **バイブレーションパーミッション (Permission Vibrate):** このフィールドは無視されます。パーミッションは関連付けられたプラグインで処理されます。
- **リモートデバッグ (Remote Debug):** この値は使用されません。
- **スプラッシュ画像 (Splash Image)**  
この値は使用されませんが、**config.xml** ファイルの末尾にコメントとして記録されます。

#### バージョンコード (Version Code)

バージョンコード (Version Code) フィールドは **config.xml** ファイルの **widget** 要素の **android-versionCode** 属性を設定するために使用されます。

```
<widget android-versionCode="18">
  ...
</widget>
```

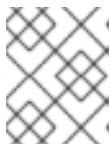
#### バージョン名 (Version Name)

バージョン名 (Version Name) フィールドは、**config.xml** ファイルの **widget** 要素の **version** 属性の値を設定するために使用されます。

```
<widget version="1.8">
  ...
</widget>
```

- e. Blackberry タブのすべての設定は無視され、記録されません。
  - f. Windows Phone タブのすべての設定は無視され、記録されません。
4. 必要に応じてスプラッシュ画像およびアイコンを **config.xml** ファイルに追加します。  
スプラッシュ画像は Cordova プラットフォームの機能ですが、この機能にはプラグインが必要になります。そのため、**<plugin name="cordova-plugin-splashscreen" spec="~4.0.1"/>** が **config.xml** ファイルに追加されます。

移行ツールは、スプラッシュ画像およびアイコンが対応するフォルダーにあり、[クライアントアプリの開発](#) で示されるような正しい名前を持つ場合にスプラッシュおよびアイコンタグを追加します。画像がそれらのパスに正しい名前で見つからない場合、スプラッシュまたはアイコンタグは追加されません。スプラッシュ画像およびアイコンが必要な場合でパスと名前が一致しない場合、移行後にスプラッシュおよびアイコンタグを手動で **config.xml** に追加できます。



#### 注記

アイコンまたはスプラッシュ画像がない場合、Cordova はそれらのデフォルト画像を使用します。

## a. iOS スプラッシュ画像を追加します。

iOS スプラッシュ画像は **www/res/splash/ios/** フォルダになければなりません。iOS ではスプラッシュ画像に正しい解像度が設定されている必要があります。画像の解像度が正しくないと、それが使用されないだけでなく、**Cordova** デフォルト画像も使用されません。そのため、アプリが一部のデバイスのスケール (**scaled**) モードで **320x480** の解像度で実行される可能性があります。以下の一覧には、正しい名前と解像度が設定されている必要なすべてのスプラッシュ画像が表示されています。

```
<splash src="www/res/splash/ios/Default-568h@2x.png"
width="640" height="1136" />
<splash src="www/res/splash/ios/Default-667h.png" width="750"
height="1334" />
<splash src="www/res/splash/ios/Default-736h.png"
width="1242" height="2208" />
<splash src="www/res/splash/ios/Default-Landscape-736h.png"
width="2208" height="1242" />
<splash src="www/res/splash/ios/Default-
Landscape@2x~ipad.png" width="2048" height="1536" />
<splash src="www/res/splash/ios/Default-Landscape.png"
width="1024" height="768" />
<splash src="www/res/splash/ios/Default-Portrait@2x~ipad.png"
width="1536" height="2048" />
<splash src="www/res/splash/ios/Default-Portrait.png"
width="768" height="1024" />
<splash src="www/res/splash/ios/Default@2x.png" width="640"
height="960" />
<splash src="www/res/splash/ios/Default.png" width="320"
height="480" />
```

## b. Android スプラッシュ画像を追加します。

Android スプラッシュ画像は **www/res/splash/android/** ディレクトリになければなりません。名前が **splash.png** のスプラッシュ画像を密度 (**density**) を設定せずに指定する場合、その画像はすべての密度で使用されます。以下の一覧には、移行ツールが処理できるすべての画像が表示されています。

```
<splash src="www/res/splash/android/splash.png" />
<splash src="www/res/splash/android/splash-ldpi.png"
density="ldpi" />
<splash src="www/res/splash/android/splash-mdpi.png"
density="mdpi" />
<splash src="www/res/splash/android/splash-hdpi.png"
density="hdpi" />
<splash src="www/res/splash/android/splash-xhdpi.png"
density="xhdpi" />
```

## c. iOS アイコンを追加します。

iOS アイコンは **www/res/icons/ios/** フォルダになければなりません。アイコンには正しい解像度が設定されている必要があります。以下の一覧には、移行ツールが処理できる正しい名前と解像度が設定された必要なすべてのアイコンが表示されています。

```
<icon src="www/res/icons/ios/Icon-40.png" width="40"
height="40" />
<icon src="www/res/icons/ios/Icon-40@2x.png" width="80"
height="80" />
```

```

        <icon src="www/res/icons/ios/Icon-50.png" width="50"
height="50" />
        <icon src="www/res/icons/ios/Icon-50@2x.png" width="100"
height="100" />
        <icon src="www/res/icons/ios/Icon-60.png" width="60"
height="60" />
        <icon src="www/res/icons/ios/Icon-60@2x.png" width="120"
height="120" />
        <icon src="www/res/icons/ios/Icon-60@3x.png" width="180"
height="180" />
        <icon src="www/res/icons/ios/Icon-72.png" width="72"
height="72" />
        <icon src="www/res/icons/ios/Icon@2x~ipad.png" width="144"
height="144" />
        <icon src="www/res/icons/ios/Icon-76.png" width="76"
height="76" />
        <icon src="www/res/icons/ios/Icon-76@2x.png" width="152"
height="152" />
        <icon src="www/res/icons/ios/Icon-small.png" width="29"
height="29" />
        <icon src="www/res/icons/ios/Icon-small@2x.png" width="58"
height="58" />
        <icon src="www/res/icons/ios/Icon.png" width="57" height="57"
/>
        <icon src="www/res/icons/ios/Icon@2x.png" width="114"
height="114" />

```

d. Android アイコンを追加します。

Android アイコンは **www/res/icons/android/** ディレクトリーになければなりません。以下の一覧には、移行ツールが処理できるすべてのアイコンファイルが表示されます。

```

        <icon src="www/res/icons/android/icon-ldpi.png"
density="ldpi" />
        <icon src="www/res/icons/android/icon-mdpi.png"
density="mdpi" />
        <icon src="www/res/icons/android/icon-hdpi.png"
density="hdpi" />
        <icon src="www/res/icons/android/icon-xhdpi.png"
density="xhdpi" />

```

## 第6章 アプリの公開

### 6.1. アプリの **GOOGLE PLAY** への提出

#### 6.1.1. 前提条件

- 「Release」アプリ (APK ファイル) が Platform からビルドされているか、または Android Studio を使用してビルドされている
- 開発者には高解像度のアプリケーションアイコンファイルが必要です。
  - 512x512 JPEG または 24 bit PNG (no Alpha)
- 開発者には1つ以上のアプリのスクリーンショットが必要です。
  - 320x480 または 480x854 JPEG または 24 bit PNG (no Alpha) 形式



#### 注記

スクリーンショットはフルブリードで、境界線や横長 (landscape) のサムネイルは切り取られます。すべてのサイズ (WxH) は 320wx480h などのように表示されます。

#### 6.1.2. 概要

Google Play にアプリを公開するに多数のステップを実行する必要があり、これらはすべて web ブラウザーを使って実行されます。ステップには以下が含まれます。

- アプリケーションのアップロード
  - アセットのアップロード
  - 詳細の一覧表示
  - 公開オプション
  - 連絡先情報
  - 同意

#### 6.1.3. アプリケーションのアップロード

- 開発者ログインを使って Google Play (<https://play.google.com/apps>) にログインします。
  - Google Play にすでにあるアプリの一覧が表示されます。
  - Upload Application ボタンをクリックします。
  - Upload an Application 画面に移動します。

#### 6.1.4. アセットのアップロード

- ドラフト版アプリケーション APK ファイル
  - APK ファイルのアップロード

- スクリーンショット
  - スクリーンショットのアップロード -- JPEG または 24-bit PNG (Alpha Transparency はサポートされません)
- 高解像度アプリケーションアイコン
  - アプリアイコンのアップロード -- JPEG または 24-bit PNG (Alpha Transparency はサポートされません)
- 販促用画像 (Promotional Graphic)
  - これは 1.6 以上が搭載されているデバイスの Google Play で使用されるオプションです。
- フィーチャーグラフィック (Feature Graphic)
  - これはオプションです。
- プロモーション動画 (Promotional Video)
  - これは YouTube URL をプロモーション動画に指定するためのオプションです。
- マーケティングの除外 (Marketing Opt-out)
  - Google でのマーケティング/アプリのプロモーションを禁止するオプションもあります。

#### 6.1.5. 詳細の一覧表示

- 言語
  - アプリの言語を選択します。
- タイトル
  - Google Play に表示されるアプリケーションの名前
- 説明
  - Google Play に表示されるアプリの説明を入力します。
- 最近の変更点
  - アプリケーションの更新をアップロードする際に変更点を入力します。
- 販促用の説明文 (Promo text)
  - これは販促用画像と関連して使用されます。
- アプリケーションタイプ
  - "アプリケーション"/"ゲーム"からアプリケーションタイプを選択します。
- カテゴリー
  - アプリケーションタイプの選択に基づいてアプリが表示されるカテゴリーを選択します。
  - アプリケーション
    - コミック

- 通信
- エンタメ
- ファイナンス
- 健康
- ライフスタイル
- マルチメディア (Multimedia)
- ニュース & 天気 (News & Weather)
- 仕事効率化
- 参考書
- ショッピング
- ソーシャルネットワーク
- スポーツ
- テーマ (Themes)
- ツール
- 旅行
- デモ
- ソフトウェアライブラリー
- ゲーム
  - アーケード & アクション
  - 頭脳系 & パズル
  - カード & カジノ (Casino)
  - ミニゲーム (Casual)
- 価格
  - 価格は自動的に **Free** に設定されます。アプリの料金を請求をする場合は、**Google Checkout** で販売者アカウントをセットアップする必要があります。

#### 6.1.6. 公開オプション

- コピープロテクション
  - アプリのデバイスからのコピーを防止します。近日中に廃止が予定されており、代わりに **Licensing Service** (ライセンス管理サービス) が使用されます。
- 場所



- アプリを表示させる **Google Play** マーケットを選択します。

### 6.1.7. 連絡先情報

- ウェブサイト
  - ウェブサイト URL
- メール
  - メールアドレス
- 電話
  - 電話番号

### 6.1.8. 同意

アプリケーションが **Android** コンテンツガイドラインを満たしており、アプリが場所や国を問わず米国法に準拠するものであり、かつ当該法に基づくアプリケーションの米国からの輸出が認可されていることを確認する必要があります。

最後に、入力した内容を保存するか、入力した内容を削除するか、またはアプリケーションを公開するよう選択します(これにより、アプリが公開され、**Google Play** で利用可能になります)。

## 6.2. アプリの **APPLE APP STORE** への提出

### 前提条件

ここでは、開発者がすでに数多くのタスクを実行していることを前提とします。これらには以下が含まれます。

- 「**Release**」アプリが **Platform** からビルドされているか、または **xCode** を使用してビルドされている
- 開発者にはアプリケーション用の 2 つのアイコンファイルがある
  - 57x57 PNG 形式
  - 512x512 PNG 形式
- 開発者には 1 つ以上のアプリの 320x480 PNG 形式が必要

### 概要

アプリを **iPhone App Store** に公開するには、数多くのステップを実行する必要があり、これらはすべて **web** ブラウザーを使って実行されます。ステップには以下が含まれます。

- 新規アプリケーションの作成
- アプリ内課金 (**in-app purchasing**) のセットアップ
- 販売用アプリケーションの提出

### 6.2.1. 新規アプリケーションの作成

1. 開発者ログイン (メールアドレス) を使って iTunes Connect (<http://itunesconnect.apple.com>) にログインします。
  - a. **Manage Your Applications** をクリックします。
  - b. **Add New Application** をクリックします。
  - c. 「**Export Laws and Encryption (輸出法および暗号化)**」の質問についての選択肢を選びます。
2. **Overview** タブ
  - a. アプリの詳細情報を **Overview** セクションに記入します。
  - b. 「**Restrict this binary to a specific platform (このバイナリーの特定プラットフォームへの制限)**」の質問についての選択肢を選びます。
  - c. バージョン番号 (1.0 など) を入力します。
  - d. SKU 番号の固有の値を入力します (例: 現在の日時)。
  - e. 画面の下部にある (青色の) **Continue** ボタンをクリックします。
3. **Ratings** タブ
  - a. アプリケーションに該当するチェックボックスを選択します。
  - b. (青色の) **Save Changes** ボタンをクリックします。
4. **Upload** タブ
  - a. 「**Upload application binary later (アプリケーションバイナリーを後でアップロード)**」オプションをクリックします。
  - b. 先の[前提条件](#)で扱われているアイコンおよびスクリーンショットをアップロードします (Large Icon と Primary Screen Shot は必須です)。ヒント: スクリーンショットを追加する場合、スクリーンショットがアップローダーで追加される際に正しい順序で表示されるよう、逆の順序で選択する必要があるかもしれません (選択した順序と逆の順序で追加/アップロードされるように見えます)。
  - c. (青色の) **Continue** ボタンをクリックします。
5. **Localization** タブ
  - a. アプリに適した言語を選択します。
  - b. **Continue** ボタンをクリックします。
6. **Pricing** タブ
  - a. アプリの **Availability Date (入手可能日)** を選択します (例: 本日)。
  - b. 適切な **Price Tier (価格帯)** を選択します。
  - c. アプリを入手できるストアを選択し、続行します。
7. **Review** タブ
  - a. 適切なストア (例: 会社の所在地のある国) を選択します。

- b. (黒色の) **Submit Application** ボタンをクリックします。

## 6.2.2. 販売用アプリケーションの提出

### 1. アプリケーションバイナリーのアップロード

- a. **iTunes Connect** にログインしたら、**Manager your Applications** をクリックします。
- b. **Apple** による承認を得るために提出する必要があるアプリケーションをクリックします (現在のステータスは、アプリケーションバイナリーがこれからアップロードされることを示すオレンジ色で表示されているはずです)。
- c. **Upload Binary** ボタンをクリックします (許可される最大バイナリーサイズは **64MB** です)。
- d. **Choose File** をクリックし、[前提条件](#)に記載されるバイナリー **zip** ファイルを見つけます。
- e. **Upload File** をクリックします。
- f. 右下にある **Save Changes** ボタンをクリックします。

### 2. Review App Store を確認します。

- a. **Manager your Applications** をクリックします。
- b. 承認用に提出されたアプリケーションをクリックします。
- c. **Edit Information** ボタンをクリックし、**Review** タブを開きます。
- d. **Review App Store** がアプリケーションの作成時に選択した国に設定されていることを確認します (**iTunes Connect** 内のバグがある可能性により、この確認は必要です)。
- e. **Done** をクリックすると、アプリケーションが **Apple** による承認を受ける準備ができます。

## 6.2.3. 外部リンク

[In depth information on managing apps](#)

## 6.3. アプリ認証情報バンドル

認証情報バンドルは、特定のビルドを実行するために必要な数多くのリソースで構成されています。以下には、各種のリソースがそれらの目的についての簡単な説明と共に一覧表示されています。

### 6.3.1. リソース

ビルド操作を実行する際に、認証情報バンドルが必要になることがあります (ビルドによって異なる)。認証情報バンドルは、証明書、プロビジョニングプロファイルおよびプライベートキーなど、開発ビルド、ディストリビューションビルド、デバッグビルドなどの特定タイプのビルドを実行するために必要なリソースの組み合わせです。プラットフォームとビルドタイプの両方に基づいて、バンドルを構成する各種リソースが分類されます。

以下は、認証情報バンドルに追加できるリソースの内訳の一覧とそれらの目的についての簡単な説明です。

#### 6.3.1.1. すべてのプラットフォーム

- プライベートキー
  - これは、所有者にのみ知らされるコンテンツのファイルです。アプリのビルドプロセスで、アプリはこのキーを使ってデジタル署名されます。これは、開発者のデジタル署名はアプリに残るため、アプリを開発者に再び関連付けることができることを意味します。

### 6.3.1.2. Android のみ

- **Android Distribution Certificate - Google Play Store** にアップロードするアプリをビルドするために使用されます。この証明書は、マーケットへのアップロード時に開発者を識別するために使用されます。
- 署名キー
  1. **キーツール** を使用して署名キーを作成します。

```
keytool -genkey -v -keystore redhat.keystore -alias rhmap -keyalg  
RSA -keysize 2048 -validity 10000
```

2. **java keystore key intp pkcs#12** 形式をエクスポートします。

```
keytool -importkeystore -srckeystore redhat.keystore -  
destkeystore rhmap.p12 -deststoretype PKCS12 -srcalias rhmap
```

3. 配信証明書を抽出します。

```
openssl pkcs12 -in rhmap.p12 -nokeys -out rhmap-cert.pem
```

4. プライベートキーを抽出します。

```
openssl pkcs12 -in rhmap.p12 -nodes -nocerts -out rhmap-key.pem
```

ここで、

- プライベートキーは **rhmap-key.pem** です。
- 証明書は **rhmap-cert.pem** です。

### 6.3.1.3. iOS のみ

- iOS 開発証明書
  - 開発時にデバイス上で **iOS App** を実行するために使用されます。
- iOS 配布用証明書
  - iOS アプリを **App Store** に提出するため、またアプリをオンデバイスのテスト用に配布するために使用されます。これは開発者の識別にも使用されます。

**警告**

Build Farm を使用してビルドされた Swift ベースの iOS アプリに署名するためにエンタープライズ向けの配布用証明書を使用する場合、結果として作成されるアプリは起動時にクラッシュする可能性があります。この問題が生じた場合は、この問題の解決方法の詳細について Red Hat のナレッジベース記事 [Swift-based iOS application crashes upon startup when signed using an enterprise distribution certificate without Organisational Unit field](#) を参照してください。

- プロビジョニングプロファイル
  - これは iOS デバイスに開発アプリケーションをインストールするために必要です。

### 6.3.2. Apple 開発者およびエンタープライズアカウント

アプリを Apple App Store で公開するには、アクティブな Apple アカウント (開発者またはエンタープライズアカウント) を持っている必要があります。このアカウントは、関連アプリを App Store で利用可能な状態にするために毎年更新する必要があります。

#### 6.3.2.1. 開発者アカウント

開発者アカウントは、iOS 配布用証明書を作成するために使用され、またアプリを Apple App Store に公開するために使用されます。

配布用証明書の期限が切れる場合でも、iOS 開発者アカウントがアクティブな場合には App Store の既存アプリは影響を受けず、App Store 内で引き続き利用可能であり、デバイス上にすでにあるアプリは予想どおりに機能し続けます。

#### 6.3.2.2. エンタープライズアカウント

エンタープライズアカウントは (社内の) 配布用証明書を作成するために使用されます。このアカウントは、アプリを RHMAP App Store またはお客様の社内 MDM に公開するために必要です。

既存の社内証明書の期限が切れると、その証明書を使ってビルドされたすべてのアプリが実行されなくなり、このバージョンのアプリを追加でインストールすることができなくなります。

この場合、アプリは再ビルドし、新規の証明書で署名され、関連するストアに再度公開してから、すべてのユーザーによって再びダウンロードされる必要があります。

#### 6.3.2.3. iOS 証明書の更新

Apple の企業向け証明書および開発者証明書は 3 年ごとに再作成される必要があります。

開発者アカウントに関連付けられたお客様のメールアドレスでは、先の再更新の必要性についての事前通知が受信されます。

Red Hat はお客様の Apple 開発者アカウントまたはエンタープライズアカウントの初期セットアップを支援しますが、作成される Apple アカウントおよび証明書の所有権と責任の所在はお客様にあります。

お客様が証明書の有効期限切れについての通知を受ける際に、アプリの利用に支障をきたさないよう証明書を更新しておくことを推奨いたします。