Date: 12-May-2025

CARD SCANNER

VERSION: 1.0.0

# **OVERVIEW**

The Card Scanner Component is a specialized tool designed to automatically capture and retrieve Credit/Debit card details by scanning the card's physical surface. This technology significantly reduces the need for manual data entry, enhancing efficiency and accuracy.

## **Use case:**

A payment or e-commerce app, where the user scans a credit/debit card using the camera to automatically extract and fill in card details like the number and expiry date for faster and error-free checkout.

## **Percentage of re-use:**

Approximate 80% of reuse.

## **Features**

1. **Faster checkout** by eliminating the need to type card numbers.
2. **Improved accuracy** by reducing errors from manual entry.
3. **Accessibility:** Helpful for users with difficulty typing or seeing small characters.

# **GETTING STARTED**

## **Prerequisites**

Before you start using the Card Scanner component, ensure the following

• Volt MX Iris

## **Platforms Supported**

1. Mobile
2. iOS
3. Android
4. Tablet, iPad

## **Importing the component**

You can import the Forge components only into the apps that are of the Reference Architecture type.

### **To import the Card Scanner component, do the following:**

1. Open your app project in Volt MX Iris.
2. In the Project Explorer, click the **Templates** tab.

Graphical user interface, text, application

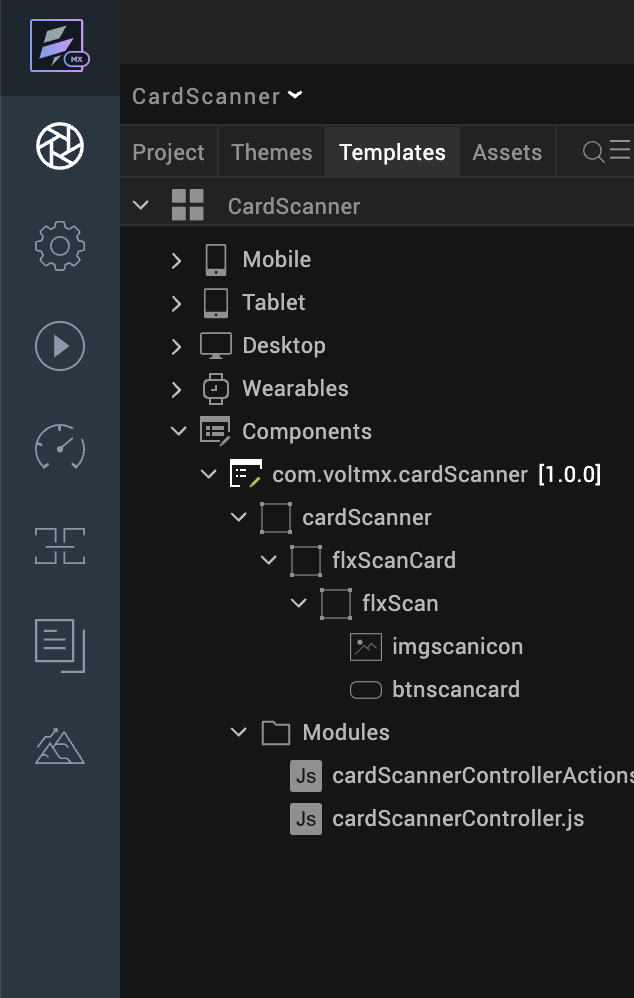
Description automatically generated

1. Right-click **Components**, and then select **Import Component**. The **Import Component** dialog box appears.

Graphical user interface, text, application, Teams

Description automatically generated

1. Click **Browse** to navigate to the location of the component, select the component, and then click **Import**. The component and its associated widgets and modules are added to your project.



Once you have imported a component to your project, you can easily add the component to a form. For more information, refer [Add a Component to a Form](https://opensource.hcltechsw.com/volt-mx-docs/docs/documentation/Iris/iris_user_guide/Content/C_UsingComponents.html#add-a-component-to-a-form)

## **D. Building and previewing the app**

After performing all the above steps, you can build your app and run it on your device. For more information, you can refer to the [Building and Viewing an Application](https://opensource.hcltechsw.com/volt-mx-docs/docs/documentation/Iris/iris_user_guide/Content/Cloud_Build_in_VoltMX_Iris.html#cloud) section of the Volt MX User Guide.

You can then run your app to see the Card Scanner work in real time.

# **REFERENCES**

## **Dynamic Usage**

You can also add a **Card Scanner** component dynamically. To do so,

1. In the **Project Explorer**, on the **Projects** tab, click **Controllers** section to access the respective **Form Controller**. Create a method and implement the code snippet similar to the sample code mentioned below.

createComponent: function(){

try{

/\* Creating cardScanner component instance \*/

var cardScanner = new com.voltmx.cardScanner({

id: "cardScanner",

isVisible: true,

top:"0dp",

left:"0dp",

width:"100%",

height:"100%",

clipBounds: true,

autogrowMode: voltmx.flex.AUTOGROW\_NONE,

skin: "slFbox",

zIndex:1

},{},{});

/\*Adding the Card Scanner component to a form\*/

this.view.add(cardScanner);

/\* Event \*/

this.view.cardScanner.onSuccessResult = function(cardDetails){

try{

if(cardDetails){

alert(JSON.stringify(cardDetails));

}

else{

alert("No cardDetails found");

}

}

catch(err){

alert("error= "+err);

}

}.bind(this);

}catch(e){

alert("e= "+e);

}

}

In the code snippet, you can edit the properties of the component as per your requirement. For more information, see Setting Properties.

2. Save the file

**Configuring Native Settings (iOS)**

To configure the native settings for iOS, follow these steps:

1. From the **Project** explorer, go to **Assets** and expand **Media**.
2. Right-click **Common**, and then select **Resource Location**. Volt MX Iris opens the common resources folder in a file explorer.  
   Graphical user interface, text

   Description automatically generated
3. Open the **infoplist\_configuration.json** file with a text or code editor.
4. At the end of the file, type the following code. You can change the description based on your preference.

{

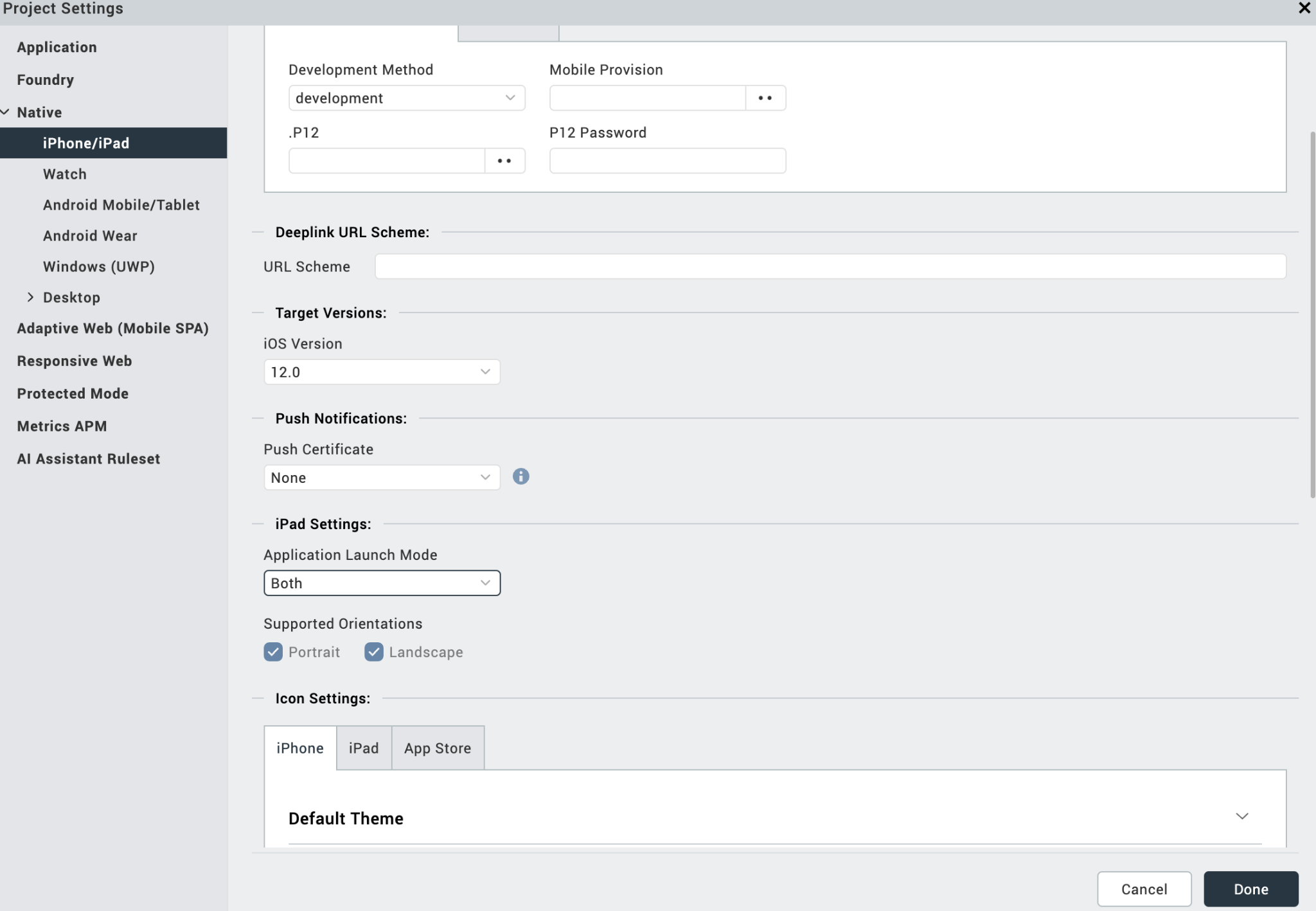
"NSCameraUsageDescription" : "This uses camera Permission"

}



1. Save the file.

**Configuring Orientation**1.From the left navigation menu, click **Project Settings**.  
2. In the **Project Settings** window, go to **Native → iPhone/iPad**.  
3. Set the application launch mode to **Both** and select both **Portrait** and **Landscape** for supported orientations. Similarly change the orientation for the Form.

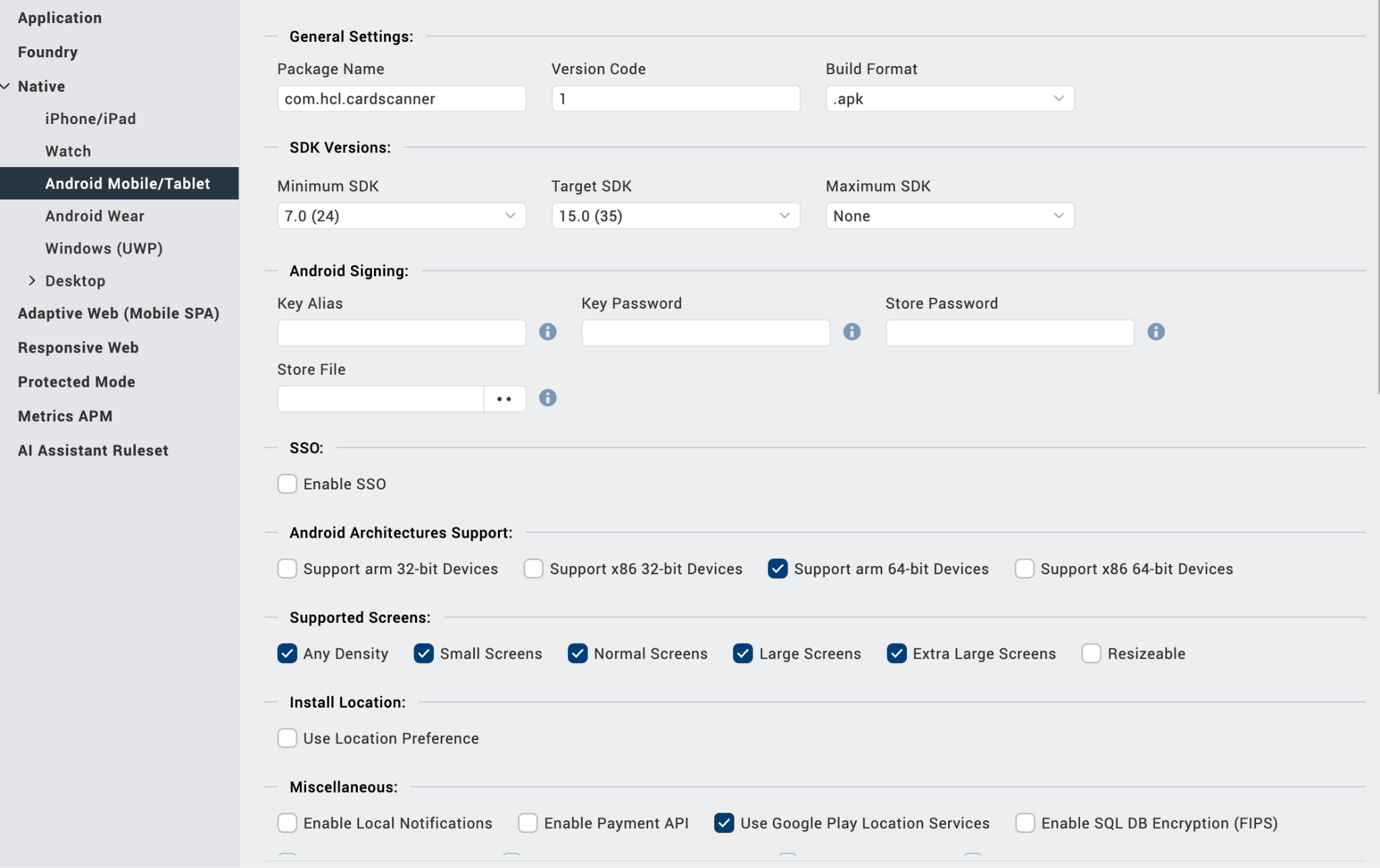
****

**Configuring Native Settings (Android)**

To configure the native settings for Android, follow these steps:

1. From the left navigation menu, click **Project Settings**.

2. In the Project Settings window, go to **Native** → **Android Mobile/Tablet**.

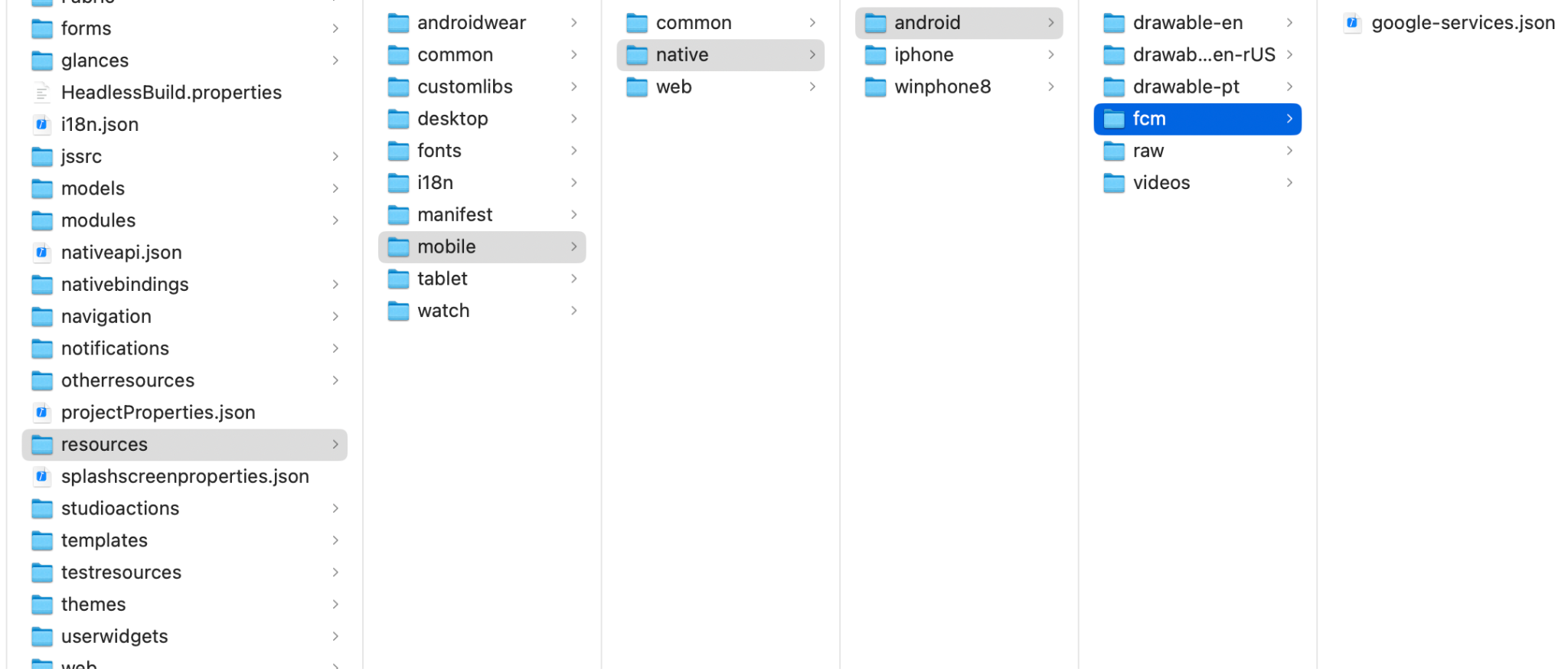
3. Set the **Minimum SDK version** to **24** and under **Miscellaneous**→ **Select Use Google Play Location Services.  
  
**

4.In the Project Settings window, go to **Native** → **Android Mobile/Tablet** → **Push Notification** → **Select FCM.**

5. As we use Firebase, the **google-services.json** file is essential in Firebase Cloud Messaging (FCM) for Android. It contains the Firebase project configuration and enables the Firebase SDKs to authenticate and securely communicate with the project.

**How to get google-services.json**

1. Go to the Firebase Console. Select your project and Go to **Project Settings.**
2. In the **General** tab, under **Your apps**, select your Android app.
3. Click **Download google-services.json**.
4. After getting google-services.json, place the google-services.json file in your project.  
   Path: Project -> resources -> mobile -> native -> android -> Create a folder as fcm and place the google-services.json file.



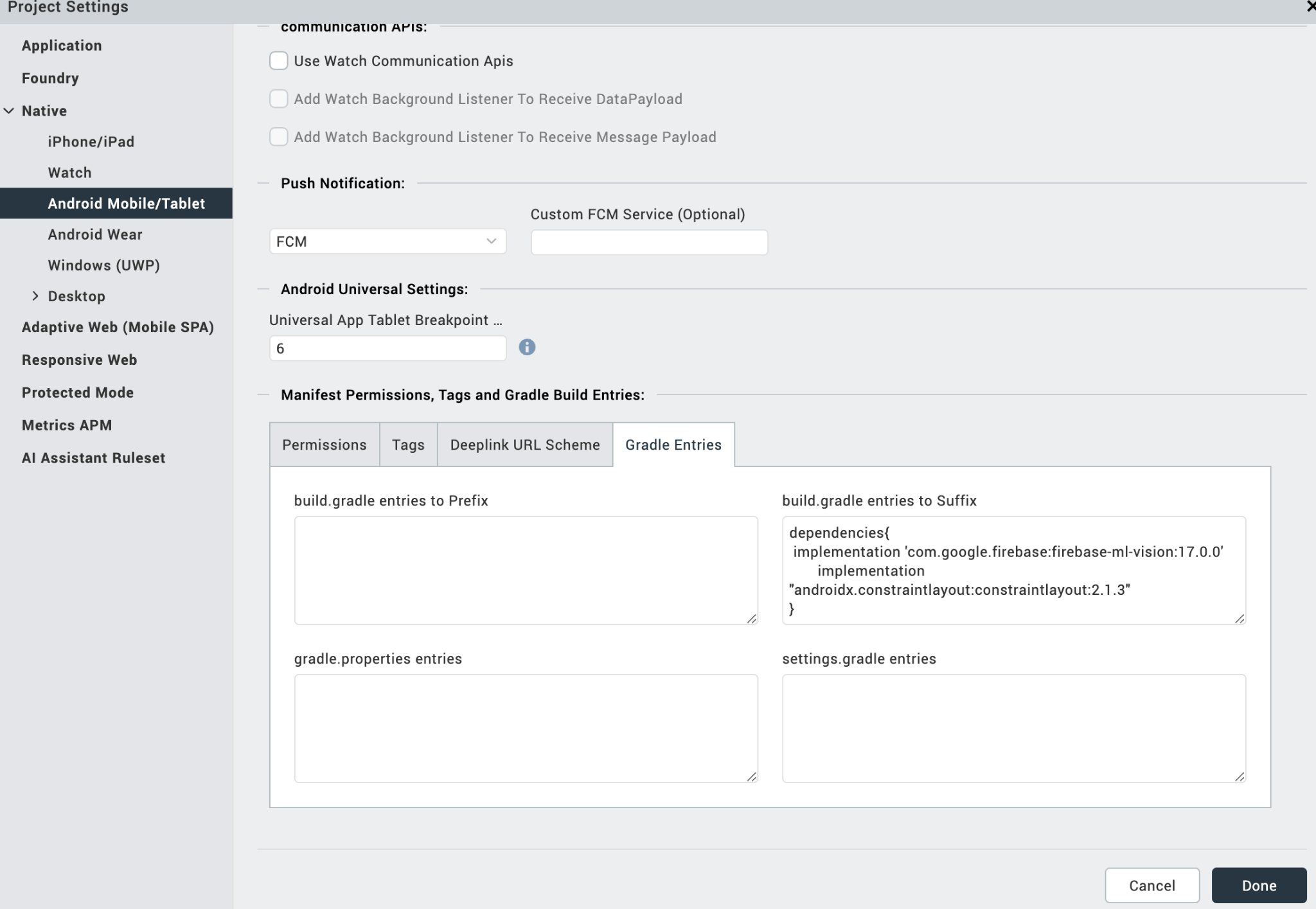
1. In google-services.json file you will have a package name. Add that package name in **Native** → **Android Mobile/Tablet** **→ General Settings → Package Name.  
   Similarly for Tablet also add the fcm folder.**
2. Switch to the **Gradle Entries** tab.
3. In the **build.gradle entries to Suffix** box, type the given code based on the version of the component.

dependencies{

implementation 'com.google.firebase:firebase-ml-vision:17.0.0'

implementation "androidx.constraintlayout:constraintlayout:2.1.3"

}

**** Click Done.

## **Events**

The component invokes events when its corresponding action is performed. You can configure any logic you want the component to perform whenever an event occurs. You can configure the events directly on the Actions tab or by writing a JavaScript, For more information, refer to [Add Actions](https://help.hcl-software.com/voltmx/v9.2/tutorials/ActionEditor.html) in the Volt MX Iris User Guide.

### **onErrorCallback**

| **Description:** | Invoked when any error occurs in the component. |
| --- | --- |
| **Syntax:** | onErrorCallback |
| **Parameters:** | errObj [JSON]: Information about the error such as the error code and error message. |
| **Example:** | this.view.componentID.onErrorCallback = function(errObj) {  alert("Error Occurred "+errObj); }.bind(this); |

### **onSuccessResult**

| **Description:** | Receives the scanned card details after the scanning process is completed. |
| --- | --- |
| **Syntax:** | onSuccessResult |
| **Parameters:** | cardDetails [JSON]: Details of the scanned card such as card number, expiry date, cardholder name. |
| **Example:** | this.view.componentID.onSuccessResult = function(cardDetails) {  alert(JSON.stringify(cardDetails)); }.bind(this); |

# **REVISION HISTORY**

App version 1.0.0:

## **A. Limitations**

* If the card details are incomplete or the OCR is unable to recognize some components (e.g., cardholder name, expiry), the app will not properly handle these cases in all scenarios. The accuracy of the card details is dependent on the quality of the captured image.
* It may not handle scenarios where multiple cards are captured in a single scan or where the user captures multiple scans in sequence.
* Different card issuers may use different fonts and sizes, leading to incorrect text recognition, especially for card numbers or names with non-standard formatting.
* The camera should launch in the same orientation as the device (portrait or landscape) and remain locked to that orientation during capture.

## **B. Known Issue**

* NA