Date: 11 Mar 2025

MLKIT BARCODE/QR SCANNER

VERSION: 2.1.0

# **OVERVIEW**

The MLKit Barcode/QR Code Scanner component can scan and decode barcodes. Under the hood, the component contains NFIs that use the Google MLKit Barcode API. The component also supports continuous scanning.

## **Use case:**

 1. An event organizer app, where a user needs to continuously scan barcodes on passes and allow entry to the attendees.

 2. A calendar app that adds events to a user's calendar using the QR Code of the event

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

95%

## **Features**

1. The component utilizes Google MLKit Barcode SDK for decoding barcodes and QR codes.
2. It supports the following Barcode types:

Android:(<https://developers.google.com/ml-kit/vision/barcode-scanning/android>)

iOS:( <https://developers.google.com/ml-kit/reference/ios/mlkitbarcodescanning/api/reference/Enums/MLKBarcodeFormat> )

# **GETTING STARTED**

## **A. Prerequisites**

 Before you start using the MLKit Barcode /QR code Scanner component, ensure the following:

 • [HCL Foundry](https://manage.hclvoltmx.com/)

 • Volt MX Iris

## **Platforms Supported**

#### Mobile

##### iOS

##### Android

#### Tablets

## **Importing the app**

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

##  **To import the Google MLKit Barcode/QR code Scanner component, do the following:**

## Open your app project in Volt MX Iris.

* 1. In the Project Explorer, click the **Templates** tab.



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



* 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)

## **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 Barcode/QR code Scanner work in real time.

# **References**

## **Dynamic Usage**

 You can also add an **MLKit Barcode/QR code 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.

/\* Creating Google MLKit Barcode/QR code Scanner component instance \*/

var MLKitBCScanner = new com.voltmx.MLKitBCScanner ({

 "height": "100%",

 "id": "MLKitBCScanner",

 "isVisible": true,

 "left": "0dp",

 "masterType": constants.MASTER\_TYPE\_USERWIDGET,

 "isModalContainer": false,

 "skin": "slFbox",

 "top": "0dp",

 "width": "100%",

 "zIndex": 1,

 "overrides": {

 "MLKitBCScanner": {

 "right": "viz.val\_cleared",

 "bottom": "viz.val\_cleared",

 "minWidth": "viz.val\_cleared",

 "minHeight": "viz.val\_cleared",

 "maxWidth": "viz.val\_cleared",

 "maxHeight": "viz.val\_cleared",

 "centerX": "viz.val\_cleared",

 "centerY": "viz.val\_cleared"

 }

 }

 }, {

 "overrides": {}

 }, {

 "overrides": {}

});

//property to set the enableContinuousScan

MLKitBCScanner.enableContinuousScan = false;

 /\*Adding the Google MLKit Barcode/QR code Scanner component to a Form\*/

this.view.add(MLKitBCScanner);

if(this.\_isiOS) {

 this.view.MLKitBCScanner.codeFormats = [this.MLKBarcodeFormatAll, this.MLKBarcodeFormatQRCode];

 }

 else if(this.\_isAndroid) {

 this.view.MLKitBCScanner.codeFormats = [this.FORMAT\_ALL\_FORMATS, this.FORMAT\_QR\_CODE];

 }

 this.view.MLKitBCScanner.addResultCallback(this.callback);

 },

callback: function(data) {

 if(data.error.length !== 0) {

 voltmx.print("Error while scanning the codes");

 alert("Error while scanning the codes");

 return;

 }else{

 var result = delete data.error;

 alert(JSON.stringify(data));

 }

 this.view.MLKitBCScanner.stopSession();

 },

 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.

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”: "Your Description"

 

1. **Save** the file.

**Configuring Deployment Target**

1. From the left navigation menu, click **Project Settings**.
2. In the **Project Settings** window, go to **Native** → **iPhone/iPad**.
3. Under **Target Versions**, from the **iOS Version** list, select **11.0** or higher.
4. Set the application launch mode to **Both** and select both **Portrait** and **Landscape** for supported orientations.



After you configure the native settings, you can Build and Run your app to see the Google MLKit Barcode/QR Code Scanner component work in real time.

**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 **CAMERA** permission to **true**.

To set a permission to **true**, select the permission from the left panel, and then click  **Add**.


4. Switch to the **Gradle Entries** tab.

1. Add below Camera permission flag at Project Settings/Android Mobile/Tablet/Manifest Permissions, Tags and Gradle Build Entries/Tags/Child tag entries under <manifest> tag/

<uses-feature android:name="android.hardware.camera.any" />

1. In the **build.gradle entries to Suffix** box, type the given code based on the version of the component.

dependencies {

// CameraX

implementation "androidx.camera:camera-camera2:1.0.0-beta11"

implementation "androidx.camera:camera-lifecycle:1.0.0-beta11"

implementation "androidx.camera:camera-view:1.0.0-alpha18"

// MLKit

 implementation 'com.google.mlkit:barcode-scanning:17.3.0'

 implementation 'org.jetbrains:annotations:15.0'

// used for 32 bit support

android { defaultConfig { renderscriptSupportModeEnabled false}}}

**NOTE:**

* If you do not add the gradle entries to your project, the app crashes.
* You can update the **Play Services** version in the later builds of your app.

After you configure the native settings, you can Build and Run your app to see the Barcode/QR Code Scanner component work in real time.

**General Properties**

**1. Code Formats (codeFormats)**

| **Description:** | Specifies the code formats to allow. |
| --- | --- |
| **Syntax**: | codeFormats |
| **Type:** | Array of Constants |
| **Read/Write:** |  Read + Write |
| **Example:** | if(this.\_isiOS) { this.view.MLKitBCScanner.codeFormats=[this.MLKBarcodeFormatAll, this.MLKBarcodeFormatQRCode]; }else if(this.\_isAndroid) { this.view.MLKitBCScanner.codeFormats=[this.FORMAT\_ALL\_FORMATS, this.FORMAT\_QR\_CODE]; } |
| **Remarks:** | * On Iris, you can set this property from a drop-down list.
* The default value for the property is NA.
 |
| **Values:** |  Android:(<https://developers.google.com/ml-kit/vision/barcode-scanning/android>)iOS:( <https://developers.google.com/ml-kit/reference/ios/mlkitbarcodescanning/api/reference/Enums/MLKBarcodeFormat>) |

1. **Enable Continuous Scanning (enableContinuousScan)**

| **Description:** | Specifies whether the camera should remain open after scanning a code. |
| --- | --- |
| **Syntax**: | enableContinuousScan |
| **Type:** | Boolean |
| **Read/Write:** |  Read + Write |
| **Example:** | this.view.MLKitBCScanner.enableContinuousScan = true; |
| **Remarks:** | If you want the camera to scan codes continuously, set the Enable Continuous Scan property to true. If the Enable Continuous Scanning property is false, then the component closes the camera after scanning a code. * The default value for the property is “true”.
* For dynamic component import, it's mandatory to set this property.
 |

## **APIs**

### **stopSession**

|  **Description:** | Stops the scanning. |
| --- | --- |
|   **Syntax:** | stopSession () |
|  **Parameters:** | None |
|  **Return Value:** |  None |
|  **Remarks:**  | Call this API after successful scan to avoid the continuous scan.And call this API when the app moves to the background.  |
|  **Example:** | this.view.MLKitBCScanner.stopSession(); |

### **restartSession**

|  **Description:** | Restarts the stopped scanning. |
| --- | --- |
|   **Syntax:** | restartSession () |
|  **Parameters:** | None |
|  **Return Value:** |  None |
|  **Remarks:**  | Call this API to restart the scan after the stopSession() invocation.And in case the session stopped in the background. call this API when the app moves to the foreground to resume the scanning.  |
|  **Example:** | this.view.MLKitBCScanner.restartSession(); |

### **enableFlash**

|  **Description: Turns on/off the device flashlight** |
| --- |
|   **Syntax:** enableFlash() |
|  **Parameters:** None |
|  **Return Value:** None |
| **Remarks:** Call this API to turn on the flashlight |
|  **Example**: this.view. MLKitBCScanner.enableFlash(); |
| **zoomIn**

|  **Description:** |  Increase zoom by 1 step. |
| --- | --- |
|   **Syntax:** | zoomIn() |
|  **Parameters:** | None |
|  **Return Value:** |  None |
| **Remarks:** | Call this API to Zoom In |
|  **Example:** | this.view.MLKitBCScanner.zoomIn(); |

 |  |
|  **zoomOut**

|  **Description:** |  decrease zoom by 1 step. |
| --- | --- |
|   **Syntax:** | zoomOut() |
|  **Parameters:** | None |
|  **Return Value:** |  None |
| **Remarks:** | Call this API to Zoom Out |
|  **Example:** | this.view.MLKitBCScanner.zoomOut(); |

 |  |
|  |  |
|  **isFlashSupports**

|  **Description:** |  Returns status about the Flashlight support in the device. |
| --- | --- |
|  **Syntax:** | isFlashSupports() |
|  **Parameters:** | None |
|  **Return Value:** | Boolean: True/False |
| **Remarks:** | Call this API to get the Flashlight support status |
|  **Example:** | this.view.MLKitBCScanner.isFlashSuuports(); |

 |  |
|  **isCameraPermissionAllowed**

|  **Description:** |  Returns status about the Camera permissions allowed or not. |
| --- | --- |
|   **Syntax:** | isCameraPermissionAllowed () |
|  **Parameters:** | None |
|  **Return Value:** | Boolean: True/False |
| **Remarks:** | Call this API to get the Camera Permissions status |
|  **Example:** | this.view.MLKitBCScanner.isCameraPermissionAllowed(); |

 |  |

|  **setupCustomRectOfInterest**

|  **Description:** |  Adds the Rectangle/Square image on top of the Camera Screen. |
| --- | --- |
|   **Syntax:** | setupCustomRectOfInterest (frame, image) |
|  **Parameters:** | frame; is array of integers with the defined index [centerx, centery, width, height]Image; Parameter is for rectangle/Square image which should place in resources/mobile/native/iphone/mainbundle for iOS, resources/mobile/native/android/Drawable folders for Android passing empty parameter will take the default image. |
|  **Return Value:** | None |
| **Remarks:** | Call this API to add a Rectangle/Square image on top of the Camera. And it should be Called every time when the session restarts. And this function must be invoked afterThe component initialization completion callback; addInitCompleteCallback. |
|  **Example:** | this.view.MLKitBCScanner.setupCustomRectOfInterest([0, 0, 250, 250], ""); |

 |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |

|  **removeRectofInterest**

|  **Description:** |  Removes the added Rectangle/Square image from the Camera Screen. |
| --- | --- |
|   **Syntax:** | removeRectofInterest () |
|  **Parameters:** | None |
|  **Return Value:** | None |
| **Remarks:** | Call this API to remove the added Rectangle/Square image from the Camera. And it should be called every time in the result Callback to remove from the screen After a successful scan. |
|  **Example:** | this.view.MLKitBCScanner.removeRectofInterest(); |

 |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |

### **startSession**

|  **Description:** | Starts the scanning. |
| --- | --- |
|   **Syntax:** | startSession () |
|  **Parameters:** | None |
|  **Return Value:** |  None |
|  **Remarks:**  | Call this API to do a scan.And call this API when the app moves to the background.  |
|  **Example:** | this.view.MLKitBCScanner.startSession(); |

### **xi. setupRectOfInterest**

|  **Description:** |  Adds the Rectangle image from the Camera Screen. |
| --- | --- |
|   **Syntax**: | setupRectOfInterest () |
|  **Parameters**: | None |
|  **Return Value:** | None |
|  |  |
|  **Example:** | this.view.MLKitBCScanner.setupRectOfInterest(); |

###  **xii. setupSquareOfInterest**

|  **Description:** |  Adds the Square image from the Camera Screen. |
| --- | --- |
|   **Syntax**: | setupSquareOfInterest () |
|  **Parameters**: | None |
|  **Return Value:** | None |
|  |  |
|  **Example:** | this.view.MLKitBCScanner.setupSquareOfInterest(); |

## **Events**

### **1. addResultCallback**

| **Description:** | Invoked after the component scans and decodes a barcode or a QR code. |
| --- | --- |
|   **Syntax**: | addResultCallback |
| **Parameters**:  |  data[JSON] : The data that is decoded from the barcode or QR code. |
|  **Remarks:** | If you want the camera to scan codes continuously, do not call the stopSession() in addResultCallback. |
|  **Example:** | onNavigate: function(context) { this.view.MLKitBCScanner.addResultCallback(this.callback); }, callback: function(data) { if(data.error.length !== 0) { voltmx.print("Error while scanning the codes"); alert("Error while scanning the codes"); return; } else{  var result = delete data.error; alert(JSON.stringify(data)); }  this.view.MLKitBCScanner.stopSession(); }, |

### **2. addInitCompleteCallback**

| **Description:** | Invoked after the component completes the initialization. This addInitCompleteCallback should be invoked in the onNavigate function. |
| --- | --- |
|   **Syntax**: | addInitCompleteCallback |
| **Parameters**:  |  function(): a call back function |
|  **Remarks:** | setupCustomRectOfInterest invocation must be in this callback only. |
|  **Example:** | onNavigate: function(context) { this.view.MLKitBCScanner.addInitCompleteCallback(this.initComplete); },initComplete: function() { // first time adding the square of interest after scan initialization completes.  // First Parameter is for the rectangle frame: array indexes are respected following: 0-center x, 1-center y, 2-width, 3-height  // Second Parameter is for rectangle image which should be placed in resources/mobile/native/iphone/mainbundle for iOS, resources/mobile/native/android/Drawable folders for Android passing empty parameter will take default image from the NFI.  this.view.MLKitBCScanner.setupCustomRectOfInterest([0, 0, 250, 250], "");  // Hide/Unhide flash button based on the flash light support this.view.btnFlash.setVisibility(this.view.MLKitBCScanner.isFlashSupports() && this.view.MLKitBCScanner.isCameraPermissionAllowed()); }, |

# **REVISION HISTORY**

Asset version 2.1.0:

## **Limitations**

1. This component supports only 64-bit devices

**B. Known Issue**

-N.A.-