App Starters (v2.7.x)

Xamarin Forms Shopping Cart

Getting Started

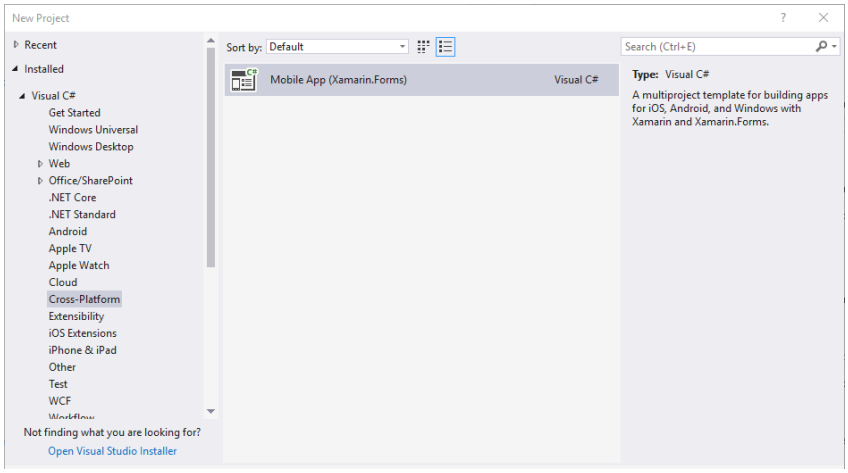
To start developing Xamarin application first you need to install Visual Studio or Xamarin Studio and make sure to include all necessary Xamarin components. For this example, we will use Visual Studio. On this link you can read step by step how to download and install Visual Studio for Xamarin applications.

In this section we have basic explanation how this demo project is developed. If you need more detailed explanation and more information about the objects and methods please check this link

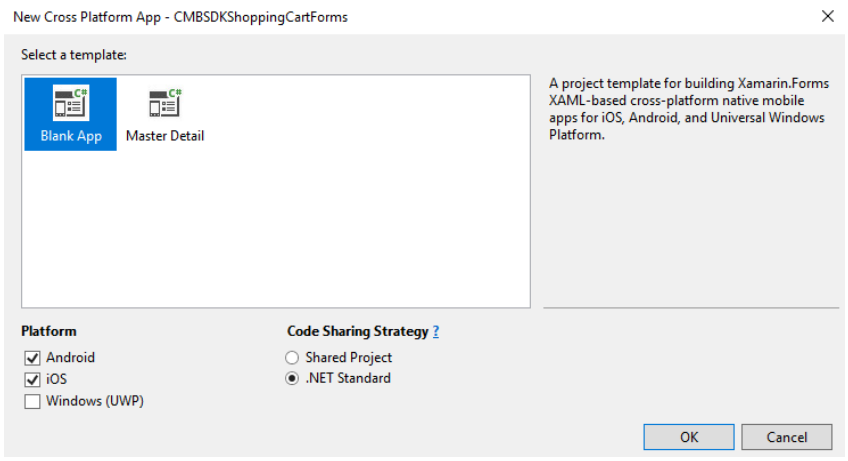
Once you finish with installation open Visual Studio and follow these steps:

1. Go to **File -> New -> Project.**

2. Create **Mobile App (Xamarin.Forms).**



3. Select **Blank App** template for **Android** and **iOS** platform and **.NET Standard** code sharing strategy.



This solution contains three projects:

1.Portable Class Libraries (PCL) Project

A Portable Class Library (PCL) is a special type of project that can be used across disparate CLI platforms such as Xamarin.iOS and Xamarin.Android, as well as WPF, Universal Windows Platform, and Xbox. The library can only utilize a subset of the complete .NET framework, limited by the platforms being targeted.

2. Android Platform-Specific Application Project

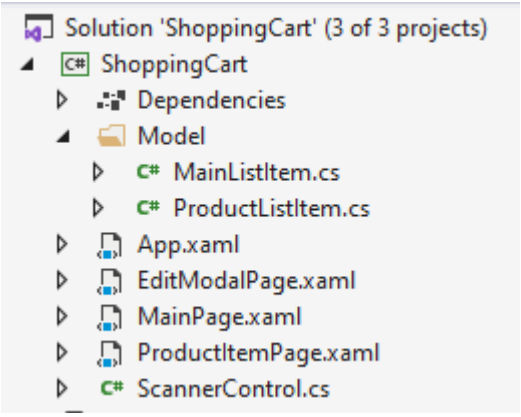
Android platform-specific project must reference the assemblies required to bind Xamarin.Android platform SDK, as well as the Core, shared code project.

3. iOS Platform-Specific Application Project

iOS platform-specific project must reference the assemblies required to bind Xamarin.iOS platform SDK, as well as the Core, shared code project.

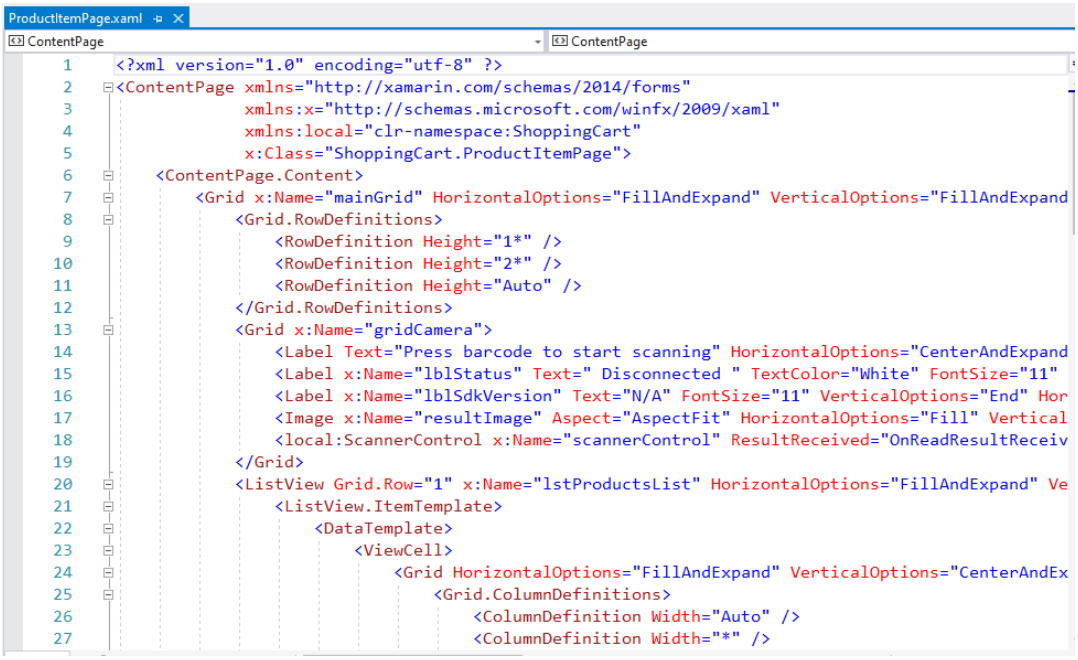
Portable Class Libraries (PCL) Project

In the portable project we have **MainPage** where all main lists are presented, **ProductItemPage** where are shown all products for a specific list, **EditModalPage** is a custom popup to edit list/product name, **MainListItem** and **ProductListItem** models that present list items for lists/products, and **ScannerControl** custom View control.



ScannerControl is a class that inherits from Xamarin.Forms.View control and we add some additional custom properties and event handlers. Later we will implement custom renderer for this class in platform-specific projects.

Here we are using **ScannerControl** in **ProductItemPage** to add new or edit an existing items.



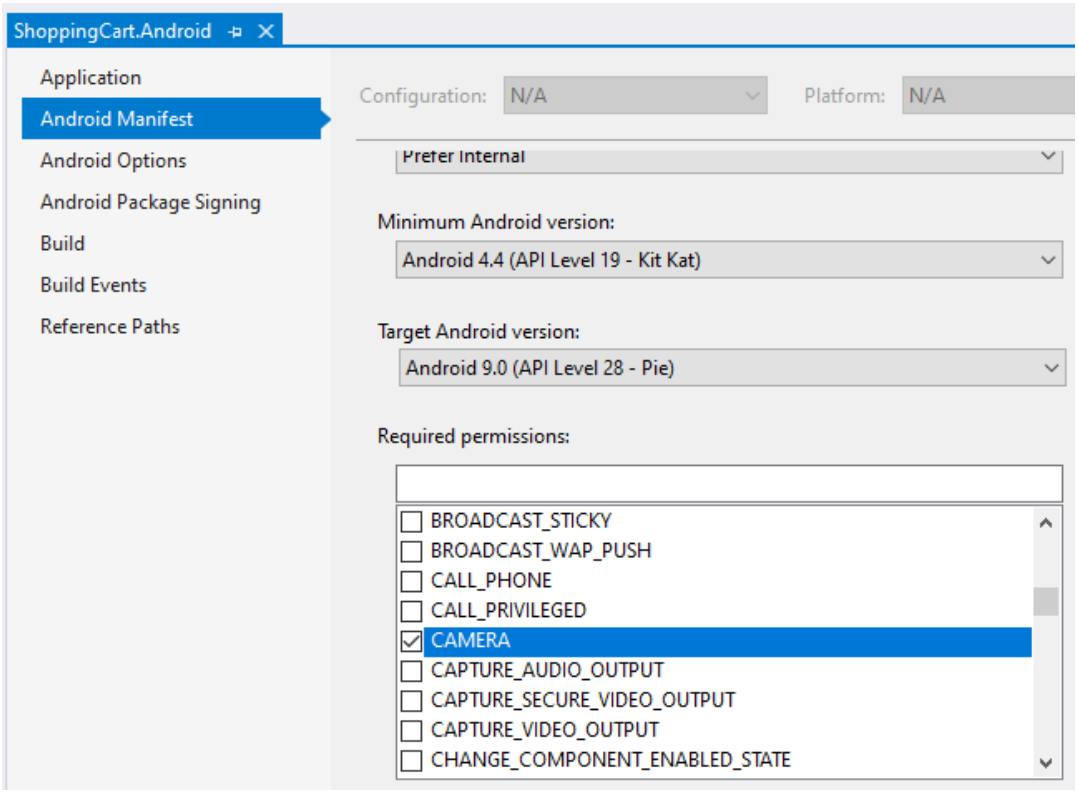
In code behind first we need to initialize, connect and configure **scannerControl** in order to start scanning process

With **scannerControl.StartScanning()** we start the scanning process

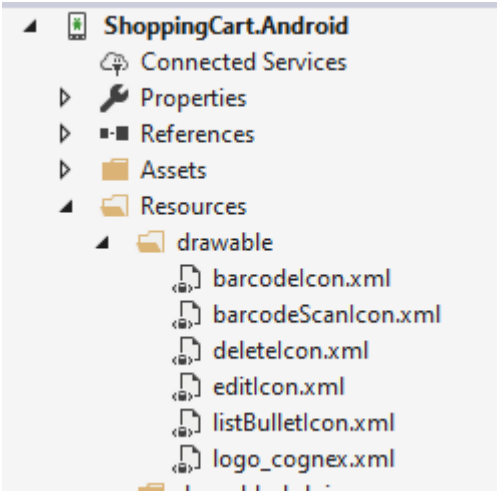
Android Platform-Specific Project

In this project we will set all settings that we need for android platform (min android version, target android version, app name, package name ..), require permissions that we need, add resources for android platform , create custom renderers for android platform , etc..

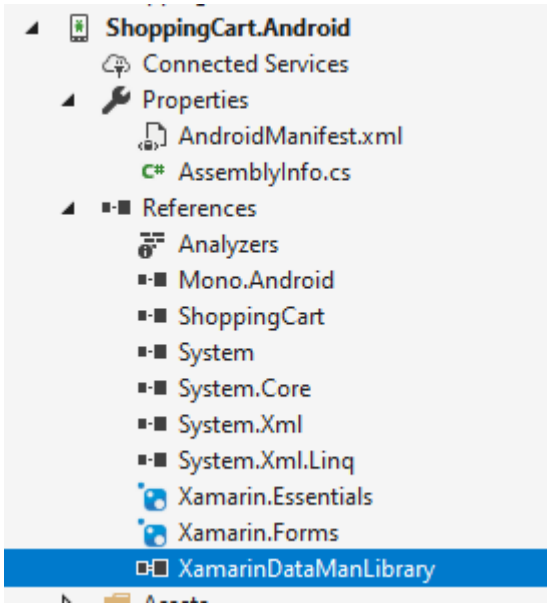
First we will check **Camera** as required permission for this application.



Next will add resources(that we use in portable project) in drawable folder.



Now we need to reference **XamarinDataManLibrary.dll** in order to use the cmbSDK.



*At the time when this document is written, there is a bug for Xamarin Forms with navigation bar icon on the android platform, that's why in this project we have a small modification on **Toolbar.axml** and there is a custom renderer for Navigation Page(**CustomNavigationRenderer**).*

Here we will create custom renderer for ScannerControl (PCL custom control) for Android platform. You don't need to edit this class. Use the same one in your project

```
[assembly: Xamarin.Forms.ExportRenderer(typeof(ShoppingCart.ScannerControl), typeof(ShoppingCart.Droid.ScannerControl))]
namespace ShoppingCart.Droid
{
    public class ScannerControl : ViewRenderer<ShoppingCart.ScannerControl, RelativeLayout>, IOnConnectionCompletedListener
        IReaderDeviceListener, IOnSymbologyListener
    {
        private RelativeLayout rlMainContainer;

        private ReaderDevice readerDevice;
        private bool availabilityListenerStarted = false;

        private static Bitmap svgBitmap;

        public ScannerControl(Context context) : base(context)
        {
        }

        protected override void OnElementChanged(ElementChangedEventArgs<ShoppingCart.ScannerControl> e)
        {
            base.OnElementChanged(e);

            if (e.OldElement != null || Element == null)
            {
                return;
            }

            rlMainContainer = new RelativeLayout(Context)
            {
                LayoutParameters = new RelativeLayout.LayoutParams(RelativeLayout.LayoutParams.MatchParent, RelativeLayout.
            };

...
```

iOS Platform-Specific Project

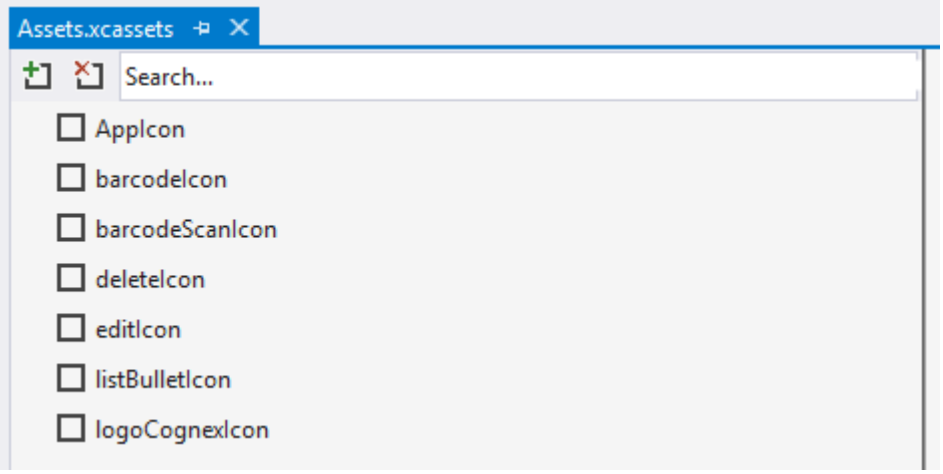
In this project, we will set all settings that we need for the ios platform (deployment target, app name, bundle identifier ..), require permissions that we need, add assets, create custom renderers for the ios platform, etc...

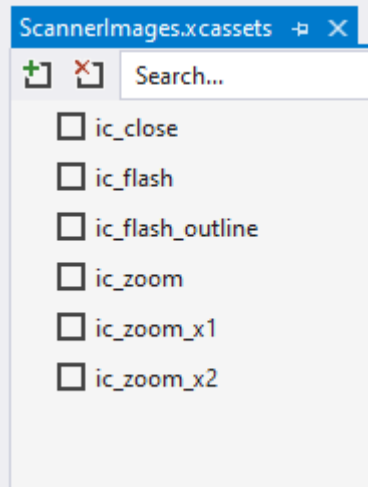
Open **Info.plist** file with some text editor and add the following lines:

```
<key>NSCameraUsageDescription</key>
<string>Camera used for scanning</string>
```

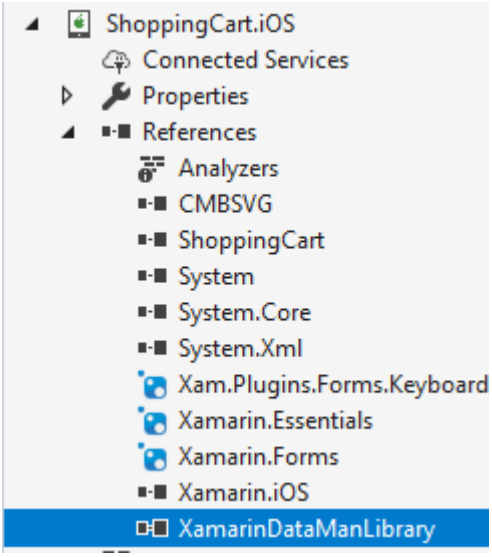
NSCameraUsageDescription key is for camera permission.

Next will add new icons in the Assets catalog and will create one more asset catalog named **ScannerImages** and add icons in that catalog





Now we need to reference **XamarinDataManLibrary.dll** in order to use the cmbSDK.



ScannerControl class is custom renderer for **ScannerControl(PCL custom control)** for iOS platform.

```
[assembly: Xamarin.Forms.ExportRenderer(typeof(ShoppingCart.ScannerControl), typeof(ShoppingCart.iOS.ScannerControl))]
namespace ShoppingCart.iOS
{
    public class ScannerControl : ViewRenderer<ShoppingCart.ScannerControl, UIView>, ICMBReaderDeviceDelegate
    {
        private UIView container;

        private CMBReaderDevice readerDevice;
        private CDMCameraMode cameraMode = CDMCameraMode.NoAimer;

        private UIAlertController connectingAlert;

        private NSObject didBecomeActiveObserver;

        protected override void OnElementChanged(ElementChangedEventArgs<ShoppingCart.ScannerControl> e)
        {
            base.OnElementChanged(e);

            if (e.OldElement != null || Element == null)
            {
                return;
            }

            container = new UIView();

            if (Control == null)
                SetNativeControl(container);


...
```

Licensing the SDK

If you plan to use the **cmbSDK** to do mobile scanning with a smartphone or a tablet (without the MX mobile terminal), the SDK requires the installation of a license key. Without a license key, the SDK will still operate, although scanned results will be blurred (the SDK will randomly replace characters in the scan result with an asterisk character).

Contact your Cognex Sales Representative for information on how to obtain a license key including trial licenses which can be used for 30 days to evaluate the SDK.

After obtaining your license key there are two ways to add your license key to an application.

For **Android Platform** open your manifest file and add this meta tag inside the application tag

```
<meta-data android:name="MX_MOBILE_LICENSE" android:value="YOUR_MX_MOBILE_LICENSE" />
```

or you can register your SDK directly from code when you create camera scanner

```
// Create a scanner device
        private void CreateScannerDevice()
        {
            //*********************************************************************************************
            // Create a camera scanner
            //
            // NOTE: SDK requires a license key. Refer to
            //       the SDK's documentation on obtaining a license key as well as the methods for
            //       passing the key to the SDK (in this example, we're relying on an entry in
            //       plist.info and androidmanifest.xml--also sdk key can be passed
            //       as a parameter in this (GetPhoneCameraDevice) constructor).
            //*********************************************************************************************
            scannerControl.GetPhoneCameraDevice(ScannerCameraMode.NoAimer, ScannerPreviewOption.Defaults, false, "SDK_KEY")

            // Connect to device
            scannerControl.Connect();
        }
```

For **iOS Platform** open Info.plist file and add this key

```
<key>MX_MOBILE_LICENSE</key>
<string>Your license key</string>
```

or you can register your SDK directly from code when you create camera scanner

```
// Create a scanner device
        private void CreateScannerDevice()
        {
            //*********************************************************************************************
            // Create a camera scanner
            //
            // NOTE: SDK requires a license key. Refer to
            //       the SDK's documentation on obtaining a license key as well as the methods for
            //       passing the key to the SDK (in this example, we're relying on an entry in
            //       plist.info and androidmanifest.xml--also sdk key can be passed
            //       as a parameter in this (GetPhoneCameraDevice) constructor).
            //*********************************************************************************************
            scannerControl.GetPhoneCameraDevice(ScannerCameraMode.NoAimer, ScannerPreviewOption.Defaults, false, "SDK_KEY")

            // Connect to device
            scannerControl.Connect();
        }
```

Ionic Demo App Using CMBSDK

Introduction

This demo uses the cordova plugin built to work with the Cognex Mobile Barcode SDK.

The Cognex Mobile Barcode SDK (cmbSDK) is a simple, yet powerful tool for developing mobile barcode scanning applications.

Based on Cognex's flagship DataMan technology and the Cognex Barcode Scanning SDK, the *cmbSDK* provides developers with a powerful tool to create barcode scanning applications for the entire range of mobile scanning devices: from smartphones and tablets to the MX line of high-performance, industrial barcode scanners.

By adhering to a few simple guidelines, developers can write applications that will work with any supported MX mobile terminal or smartphone with little or no conditional code. The SDK achieves this by abstracting the device through a "reader" connection layer: once the application establishes its connection with the desired reader, a single, unified API is used to configure and interface with the device.

The SDK provides two basic readers: an “MX reader” for barcode scanning with devices like the MX-1000 and MX-1502, and a “camera reader” for barcode scanning using the built-in camera of the mobile device.

Here is an example of how one would go about integrating the *cmbSDK* within an Ionic mobile app.

Ionic with Capacitor

Ionic is used for hybrid, cross-platform mobile app development utilizing HTML and JavaScript to present the user interface, while Capacitor makes it easy to build modern web apps that run natively on iOS, Android, and the Web.

This provides the developer with reusable code that can be executed across platforms. It pairs with Capacitor and PhoneGap/Cordova plugins to enable native access to each platform. The objective of this exercise is to try to show how the Cognex Mobile Barcode Scanner SDK may be used in a real-life app that is built using the Ionic cross-platform framework.

Simple steps to start

You should first install and configure the necessary build tools for developing Ionic apps with capacitor:

- Node.js
- npm
- Ionic CLI
- Capacitor

Build the demo application

You may now download our sample app

1. Download this (zip download - ionic-cmbsdk-demo-app)
2. Unzip it to your preferred location and CD to that directory
3. Install the dependencies for this project (our project depends on several Capacitor plugins. Several other "default" plugins are installed when first creating a clean Ionic app). To install all dependencies, execute:

```
npm install
```

4. When finished add our Capacitor/Cordova plugin published on npm

```
npm install cmbsdk-cordova
```

5. Add the platform (android for example)

```
ionic capacitor add android
```

6. Run the app

```
ionic capacitor run android
```

6.1 If we want to run with live reload

```
ionic capacitor run android -l --external
```

7. If you are building for iOS

```
ionic capacitor add ios
```

```
ionic capacitor build ios
```

Once building is done XCode will be opened.

7.1 Set your signing profile

7.2 If you plan to use built-in camera for scanning go to your project's **Info.plist** file and add the **Privacy - Camera Usage Description** or NSCameraUsageDescription to display a message about how your application uses the camera of the user's mobile device

```
<key>NSCameraUsageDescription</key>
<string>Required for Scanning</string>
```

7.3 If you are using cmbSDK with an MX Mobile Terminal, you also need to add **Supported external accessory protocols** or *UISupportedExternalAccessoryProtocols* to *com.cognex.dmcc* in your project's **Info.plist**. (You have to create an MFi Request in such case before publishing your app in Apple App Store. You can read further details in Getting your MX Mobile Terminal Enabled App into the App Store section.)

```
<key>UISupportedExternalAccessoryProtocols</key>
<string>com.cognex.dmcc</string>
```

7.4 Run the app from the XCode or with CLI command

```
ionic capacitor run ios
```

Description of the solution

Probably the first place to visit, is the cmbscanner.ts provider which is located at

```
/project_folder/src/providers/cmbscanner/cmbscanner.ts
```

This is where we configure all the various cmbSDK settings.

As a general rule when working with the cmbSDK we want to enable symbologies or change output defaults in the global **setConnectionStateDidChangeOfReaderCallback.** Once we receive a connectionState = connected, we can do the magic with the symbologies that we want enabled or whatever other setting that we aim to use.

> For a full list of all the API methods that we can use please refer to : https://cmbdn.cognex.com/knowledge/cordova-plugin-for-cmbsdk/api-methods.

```
cmbScanner.setConnectionStateDidChangeOfReaderCallback((connectionState: number) => {

      this.events.publishConnectionChange(connectionState)

      if (connectionState == cmbScanner.CONSTANTS.CONNECTION_STATE_CONNECTED) {

        //image results should be set after we have a connection to the READER
        cmbScanner.enableImage(this.settings.enableImage);
        cmbScanner.enableImageGraphics(this.settings.enableImageGraphics);

        this.settings.enabledSymbols.forEach((v) => {
          this.allDone.push(cmbScanner.setSymbologyEnabled(v, true).then((symbResult: any) => {
            return symbResult;
          }));
        });

        Promise.all(this.allDone).then(results => {
          //when all is said and done
          this.isDone = true; //set the isDone flag
        });

        this.connected = true;

        cmbScanner.setTriggerType(this.settings.triggerType);
      }
      else if (connectionState == cmbScanner.CONSTANTS.CONNECTION_STATE_DISCONNECTED) {
        this.connected = false;
      }
    });
```

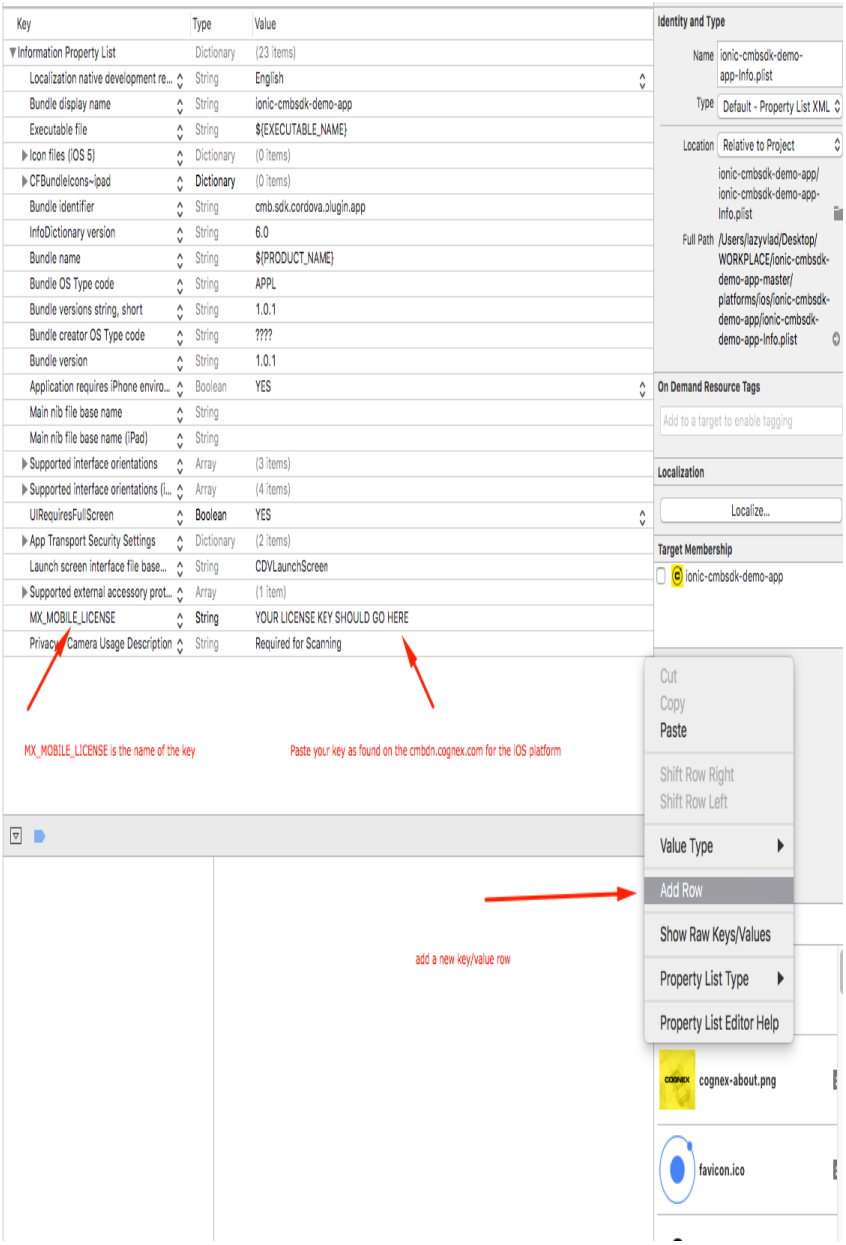
If you have a valid license key, you'd want to use it **before** loadScanner().

```
cmbScanner.registerSDK("key"); //you can add your license key here
```

Alternatively, you can add the key directly in the manifest file (for android) or in the plist file for ios:

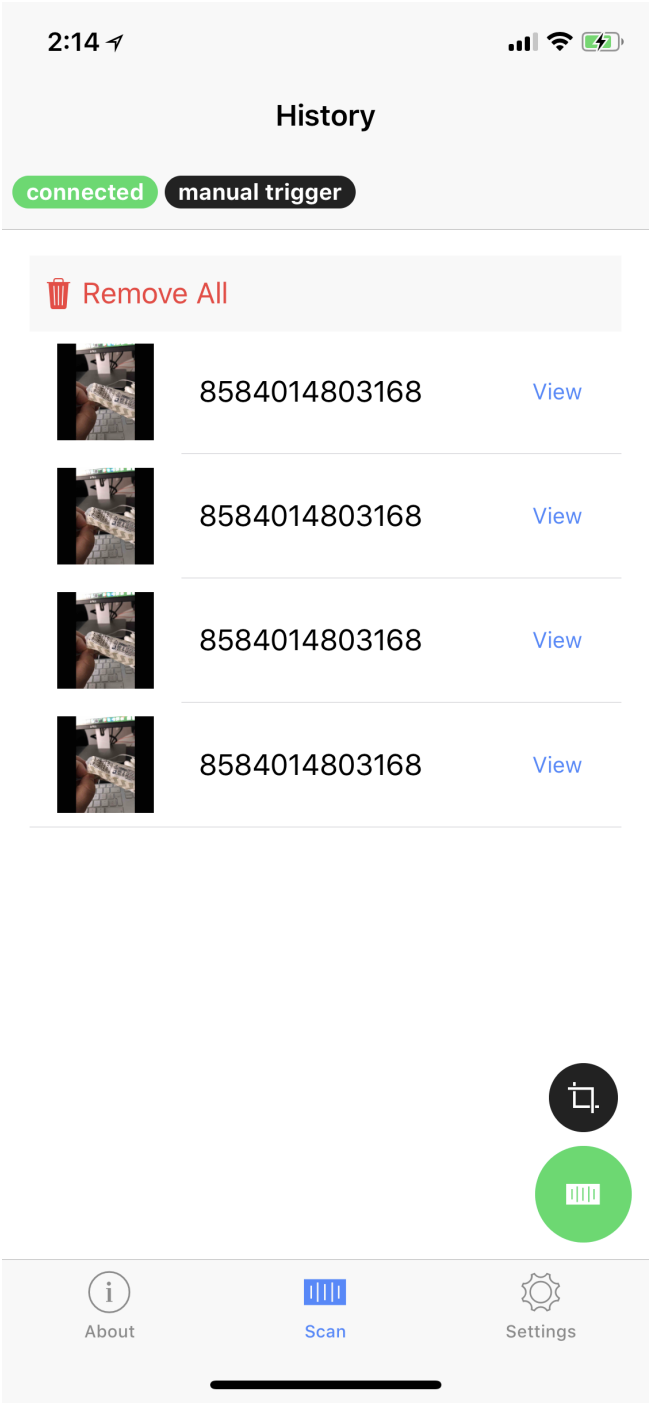
```
<application
  android:hardwareAccelerated="true"
  android:icon="@mipmap/icon"
  android:label="ionic-cmbsdk-demo-app"
  android:supportsRtl="true">
  <activity
    android:name="MainActivity"
    android:configChanges="orientation|keyboardHidden|keyboard|screenSize|locale"
    android:label="@string/launcher_name"
    android:launchMode="singleTop"
    android:theme="@android:style/Theme.DeviceDefault.NoActionBar"
    android:windowSoftInputMode="adjustResize">
    <intent-filter android:label="@string/app_name">
      <action android:name="android.intent.action.MAIN" />
      <category android:name="android.intent.category.LAUNCHER" />
    </intent-filter>
  </activity>
  <meta-data
    android:name="MX_MOBILE_LICENSE"
    android:value="key" />
</application>
```

Or for the IOS platform



App features

The app has two modes of connection. The Camera Connector which will use the camera phone as a scanner, and the MX connector, which will use the MX device as a scanner



It also has two modes of trigger. You can either scan a code and get the result back, or stay in continuous mode and continue scanning while the scanned items are added to a list.

This example also uses the Capacitor storage service. We want our settings to be persistent, but don't want to overcomplicate things, that's why the Capacitor storage service is perfect for our task.

With the use of storage, we can have our scanned results saved and reused (or checked) at a later date.