Cognex Mobile Barcode SDK for Android (v2.4.x)

Overview

Cognex Mobile Barcode SDK (cmbSDK) is a tool for developing mobile barcode scanning applications. CmbSDK is based on Cognex's DataMan technology and the Manatee Works Barcode Scanning SDK and it allows you to create barcode scanning applications for mobile devices. Mobile devices used for barcode scanning range from smartphones to the MX Series industrial barcode readers. CmbSDK abstracts the device through a *ReaderDevice* connection layer. Once the application establishes its connection with the reader, a single, unified API is used as interface to configure the device, eliminating the need to write too much conditional code.

CmbSDK provides two basic *ReaderDevice* connection layers:

- *MX reader* for barcode scanning with devices like the MX-1000 and MX-1502
- *Camera reader* for barcode scanning with the built-in camera of the mobile device

Barcode Scanning with an MX Mobile Terminal

The cmbSDK supports Cognex’s MX Series Mobile Terminals and some of their features using cmbSDK are the following:

- **Hardware trigger**: MX Mobile Terminals include two built-in triggers for barcode scanning. They also support a pistol grip with trigger that is an optional accessory.
- **Illumination and aiming**: MX Mobile Terminals have built-in illumination and aiming, making it unnecessary to have a live preview on the smartphone's screen.
- **Configurations**: You can export and import configuration sets to MX Mobile Terminals using Cognex’s DataMan Setup Tool for Windows, the Quick Setup mobile application or cmbSDK. You can have multiple scanning applications, each of which requires a different set of device settings.
- **High-capacity battery**: MX Mobile Terminals have an integrated battery that powers the MX scanning engine and the mobile device. The optional pistol grip includes a second battery that doubles the power capacity of the MX Mobile Terminal.

Debugging on MX Mobile Terminal

Normally you connect your mobile device (phone or tablet) to your PC via the USB or lightning port to start debugging. If an MX Mobile Terminal is attached to your mobile device via the USB or lightning port while your application is running, you need to debug

your application via Wi-Fi.

Debugging on Android:

To debug using Android Studio, connect your Android device via USB to your PC and make sure you can run and debug your application using the USB cable.
To Connect your Android device to Wi-Fi, make sure that Android Tools are installed beside your IDE.

1. Type "**adb tcpip 5555**" to set the device's port to 5555 in the terminal.
2. Get the mobile device's IP address by typing "**adb shell ip -f inet addr show wlan0**" or find it manually in the settings menu of your mobile device.
3. Type "**adb connect device_ip:5555**" to connect to your mobile device. This prompts a message if it is connected successfully.
4. Disconnect the USB cable from your mobile device.
5. Connect your mobile device to the MX Mobile Terminal and proceed to debug your app as if it was connected via cable.

   > **Note**: After you connect your mobile device to the MX Mobile Terminal, Wi-Fi connection might be lost. If the Wi-Fi connection is lost, repeat step 3.

6. When you are done, type "**adb -s device_ip:5555 usb**" to switch your device back to USB connection mode.

> **CAUTION**: Leaving the wireless debugging option enabled is not recommended as anyone in your network can connect to your device in debug, even if you are in data network. Do it only when you are connected to a trusted WiFi and disconnect when you are done.

Barcode Scanning with a Smartphone

Barcode Scanning with a Smartphone or Tablet

The differences in the capabilities of smartphones as barcode scanning devices result in a user experience different from purpose-built scanners, impacting the design of the mobile barcode scanning application. By following a few simple guidelines, you can develop applications with the cmbSDK that work the same way when using an MX Mobile Terminal or the built-in camera of a mobile device.

- To initiate barcode scanning without a dedicated hardware trigger, see Mobile Device Triggering.
- To aim for barcode scanning with a smartphone that does not have an aimer, see Mobile Device Aiming.

- To choose the most suitable orientation for barcode scanning, see Mobile Device Orientation.
- To reduce the CPU usage of the mobile device when it performs image analysis and barcode decoding, see Optimizing Mobile Device Performance.

CmbSDK employs a default set of options for barcode reading with the built-in camera of the mobile device. However, cmbSDK does not implement saved configurations for the camera reader. This means that every time an application starts that uses the camera reader, it starts with the default settings of the camera reader. For a list of the default settings, see the Appendix.

Mobile Device Triggering

Without a hardware trigger, mobile devices must use alternative methods to initiate barcode scanning. The cmbSDK supports three methods to trigger barcode scanning:

- **Application or workflow driven trigger:** The application code or the business logic/workflow of the application invokes the scanning module. In simple programming terms, it is calling a function like *startScanner()*.
- **Virtual trigger:** To start or stop the scanning process the application provides a button on the screen. Depending on the application design, you need to press and hold the virtual button to keep the scanner running, this invokes the scanning module.
- **Simulated trigger:** Press one of the volume-down buttons to start or stop the scanning process just like when you pull a trigger on a purpose-built scanner.

Mobile Device Aiming

The built-in camera provides a live-stream preview on the display of the mobile device for barcode aiming. Reposition the mobile device until the barcode appears in the field of view of the built-in camera and the application decodes it. CmbSDK provides a built-in preview control that can be displayed in partial or full screen, and in either portrait or landscape orientation.

The cmbSDK also supports passive aimers: devices attached to the mobile device or its case that use the LED flash of the device as a light source to project an aiming or targeting pattern. The mobile device can project an aimer pattern similar to a purpose-built scanner so live-preview is not needed. However, by using the LED flash as an aimer, general scanning illumination is not available.

Mobile Device Orientation

The cmbSDK supports portrait orientation, landscape orientation and auto-rotation for both the presentation of the barcode preview and the scan direction. Mobile devices can scan most barcodes regardless of the orientation of the application and/or the mobile device.

PORTRAIT OR LANDSCAPE	PORTRAIT ONLY	
Most barcodes can be scanned in either portrait or landscape orientation.	Most well defined and moderately sized barcodes can be scanned in a portrait orientation, which is the most natural way to hold the mobile device. Example: QR, Data Matrix, Maxicode.	Long, barco landsc

Optimizing Mobile Device Performance

Mobile devices are an ideal platform for barcode decoding. The cmbSDK is optimized for mobile environment, but image analysis and barcode decoding is still a CPU intensive activity. Since these processes share the mobile device's CPU with the mobile operating system (OS), services, and other applications, these processes optimize your barcode scanning application and limit it to only using the features of the cmbSDK that they need.

To optimize your application:

- Enable decoding only for the barcode types the application needs to scan. The cmbSDK supports the decoding of almost 40 different barcode types and subtypes, enabling all results in low performance and unexpected errors.
- Do not enable certain symbologies and/or advanced features at the same time.
- Optimize your camera resolution. By default, the cmbSDK uses HD images for barcode decoding.
- Use an appropriate decoder effort level. The cmbSDK has a configurable effort level that controls how aggressively it performs image analysis. The cmbSDK uses a default value (level 2) that is sufficient for most barcodes. Using a higher level can result in better decoding of poorer quality barcodes, resulting in slower performance.

No barcode symbologies are enabled by default, when the cmbSDK is initialized for use with the mobile device's built-in camera.

Using cmbSDK

Installing cmbSDK

Installing the Android cmbSDK

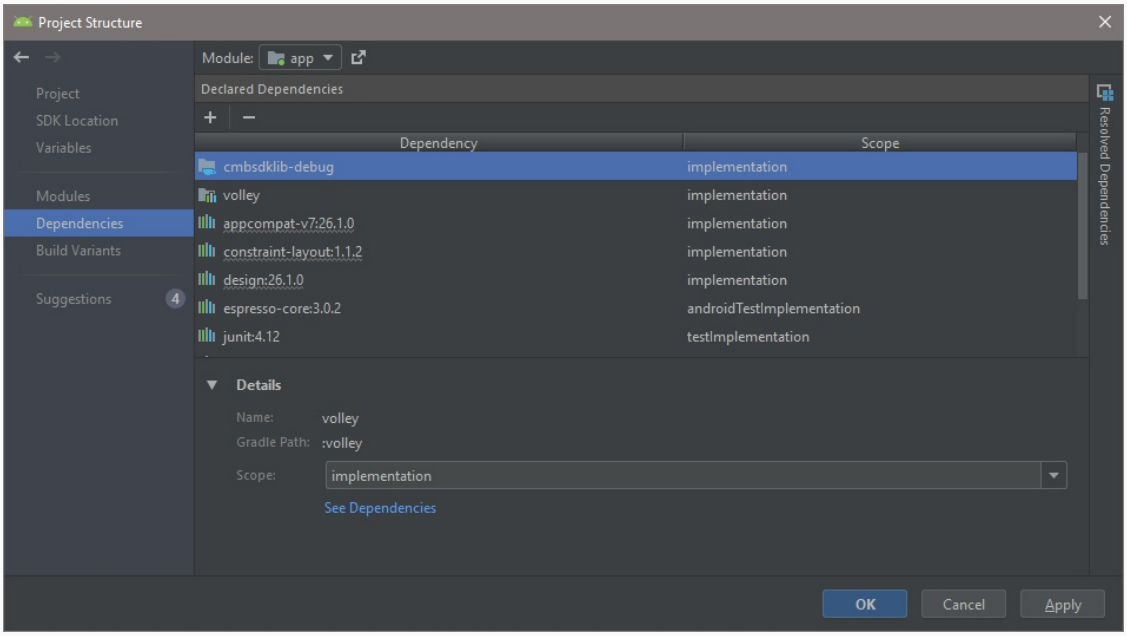
Note: cmbSDK is compatible with Android Studio.

1. Download the Cognex Mobile Barcode SDK for Android from the Cognex Mobile Barcode Scanner Solutions page.
2. Start Android Studio and add the SDK AAR file as a module to your project:

    1. Right click your app module, select *New* > *Module* > *Import .JAR/.AAR Package*, and click **Next**.

    2. Browse the cmbSDK .AAR file in the File name field, and click **Finish**.

3. After the new module is available, right click your app module, select the **Open Module Settings**, and choose the **Dependencies** tab.



4. Click the **+** sign at the top of the Declared Dependencies dialog box and select the **3 Module dependency**.
5. Select *cmbsdklib* from the popup window and click **OK**, making the cmbsdklib module available under the **Dependencies** tab.

6. Install the MX Connect application from the Play Store to communicate with MX mobile terminals.

Installing the iOS cmbSDK:

1. Install the latest XCode for iOS Development.
2. Download the Cognex Mobile Barcode SDK for iOS.

Licensing cmbSDK

To use cmbSDK for barcode scanning with a mobile device without an MX mobile terminal, you need to install a license key. If the license key is missing, asterisks will appear instead of scanned results.

Contact your Cognex Sales Representative for information on how to obtain a license key, including 30-day trial licenses.

Android:

1. After obtaining your license key, add the following line in the AndroidManifest.xml file of your application under the application tag:

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

2. Replace YOUR_MX_MOBILE_LICENSE with your license key.

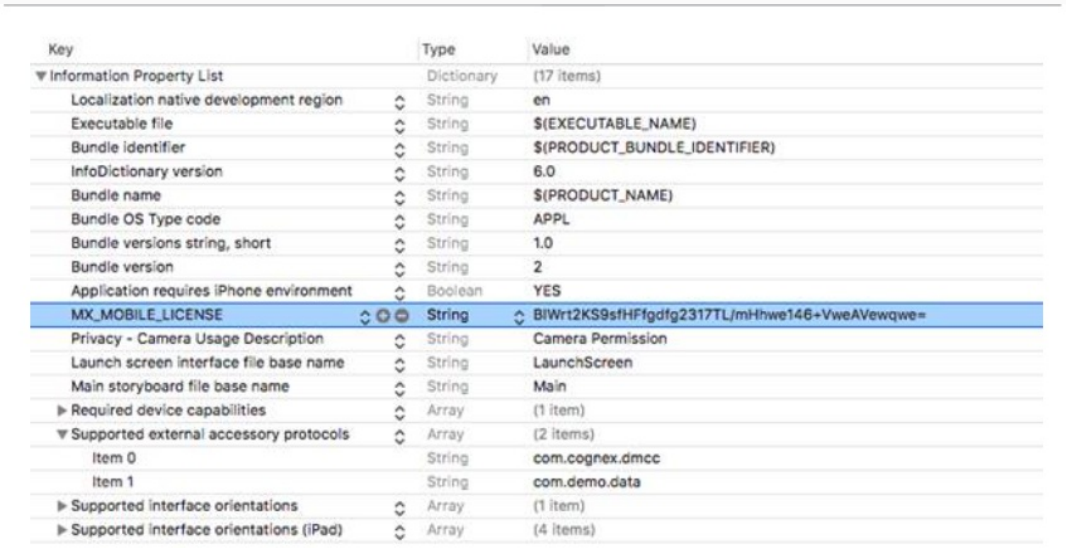
```
<application
        android:allowBackup="true"
        android:icon="@mipmap/ic_launcher"
        android:label="@string/app_name"
        android:roundIcon="@mipmap/ic_launcher"
        android:supportsRtl="true"
        android:theme="@style/AppTheme">
        <activity android:name=".ScannerActivity" android:configChanges="orientation|screenSize
">
            <intent-filter>
                <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="g/9ytJzcja+sxt4DTEDxR4hp6sZh9bmL97vUx+EE9uY=" />

</application>
```

You can also add the license key by copying the text below when you create new instance from *ReaderDevice*.

```
case PhoneCamera:
          readerDevice = ReaderDevice.getPhoneCameraDevice(this, param_cameraMode,
             PreviewOption.DEFAULTS, null, "SDK_KEY");
```

iOS:

After obtaining your license key, add it as a String in your application's Info.plist file under the key *MX_MOBILE_LICENSE*.



Key	Type	Value
▼ Information Property List	Dictionary	(17 items)
Localization native development region	String	en
Executable file	String	$(EXECUTABLE_NAME)
Bundle identifier	String	$(PRODUCT_BUNDLE_IDENTIFIER)
InfoDictionary version	String	6.0
Bundle name	String	$(PRODUCT_NAME)
Bundle OS Type code	String	APPL
Bundle versions string, short	String	1.0
Bundle version	String	2
Application requires iPhone environment	Boolean	YES
MX_MOBILE_LICENSE	String	BlWrt2KS9sfHFfgdfg2317TL/mHhwe146+VweAVewqwe=
Privacy - Camera Usage Description	String	Camera Permission
Launch screen interface file base name	String	LaunchScreen
Main storyboard file base name	String	Main
▶ Required device capabilities	Array	(1 item)
▼ Supported external accessory protocols	Array	(2 items)
Item 0	String	com.cognex.dmcc
Item 1	String	com.demo.data
▶ Supported interface orientations	Array	(1 item)
▶ Supported interface orientations (iPad)	Array	(4 items)

Migrating from a DataMan SDK for MX Readers to cmbSDK

1. Install the MX Connect application from the Play store. This app enables your mobile phone to seamlessly connect to Cognex MX readers.

2. Install cmbSDK to your project.

3. Use *DataManSystem.createDataManSystemForMXDevice()* factory method to create a DatamanSystem object.

4. Remove:

    - All *DataManSystem.createDataManSystemOverUsb()* methods from your project.
    - All *DataManSystem.createDataManSystemOverUsbAccessory()* methods from your project.
    - USB_ DEVICE-ATTACHED and USB_ACCESSORY_ATTACHED Intent filters and meta-data from the AndroidManifest.xml file.
    - USB and accessory descriptor xml files from the XML folder.

Writing a Mobile Application

CmbSDK provides a high-level, abstract interface for supported scanning devices: the MX mobile terminals and the camera of the mobile phone.

The primary interface between your application and the barcode scanning device is the *ReaderDevice* class. The *ReaderDevice* class represents an abstraction layer to the

device, handling all communication and necessary hardware management, such as scanning with a smartphone.

Perform the following steps to use cmbSDK:

1. Create an instance from the *ReaderDevice* class with the type of scanning device you want to use (*MX reader* or *camera reader*).

2. Connect to the *ReaderDevice* instance you created.

3. Configure the *ReaderDevice* instance, if necessary.

4. Start scanning.

Initialization and connection need to be performed only once in your application.

- MX mobile terminals need to be reconfigured if they become disconnected due to for example timing out or drained battery. To avoid this, you can save the configuration.
- Your application can use both an MX mobile terminal and camera scanning. In this case you have to establish a new connection to a different device after disconnecting from the current device. You can check our sample app for demonstration to see how it works.

Setting up an Application to Use cmbSDK for Android copy

Perform the following steps to set up and start using cmbSDK:

1. Import the following package members, or just the classes you use:

```
import com.cognex.dataman.sdk.*
import com.cognex.mobile.barcode.sdk.*
```

2. Build your UI according to your needs, but considering the following aspects:

    - You have to decide if you you want to show partial or a full screen (that is the default) camera preview. You need a **ViewGroup** container to use partial preview, for example **RelativeLayout.** No additional container is needed for full screen preview.

Our sample app (that you can find in the cmbSDK bundle) is using full screen preview. To change the sample app to use partial view, add the following **RelativeLayout** at the end of **ConstraintLayout** in **activity_scanner.xml** file. Use this layout ad a **ViewGroup** parameter in reader device constructor (**getPhoneCameraDevice**) when reader device is initialized.

```
<RelativeLayout
   android:id="@+id/rlPreviewContainer"
   android:layout_width="match_parent"
   android:layout_height="200dp"
   app:layout_constraintTop_toTopOf="parent"
   app:layout_constraintStart_toStartOf="parent"
   app:layout_constraintStart_toEndOf="parent" />
```

- To display the last scanned image, an **ImageView** container is needed.
- To display the scanned result as a text, a **TextView** is needed.

3. Set up the following interfaces to monitor the connection state of the reader and receive information about the read code:

```
public class ScannerActivity extends AppCompatActivity implements
        OnConnectionCompletedListener, ReaderDeviceListener,
        ActivityCompat.OnRequestPermissionsResultCallback {
....

// The connect method has completed, here you can see whether there was an error with establish
ing the connection or not
    @Override
    public void onConnectionCompleted(ReaderDevice readerDevice, Throwable error) {
        // If we have valid connection error param will be null,
        // otherwise here is error that inform us about issue that we have while connecting to
reader device
        if (error != null) {

            // ask for Camera Permission if necessary
            if (error instanceof CameraPermissionException)
                ActivityCompat.requestPermissions(((ScannerActivity) this), new String[]{Manife
st.permission.CAMERA}, REQUEST_PERMISSION_CODE);

            updateUIByConnectionState();
        }
    }

// This is called when a connection with the self.readerDevice has been changed.
    // The readerDevice is usable only in the "ConnectionState.Connected" state
    @Override
    public void onConnectionStateChanged(ReaderDevice reader) {
        clearResult();
        if (reader.getConnectionState() == ConnectionState.Connected) {
            // We just connected, so now configure the device how we want it
            configureReaderDevice();
```

```
        }

        isScanning = false;
        updateUIByConnectionState();
    }

// This is called after scanning has completed, either by detecting a barcode, canceling the sc
an by using the on-screen button or a hardware trigger button, or if the scanning timed-out
    @Override
    public void onReadResultReceived(ReaderDevice readerDevice, ReadResults results) {
        clearResult();

        if (results.getSubResults() != null && results.getSubResults().size() > 0) {
            for (ReadResult subResult : results.getSubResults()) {
                createResultItem(subResult);
            }
        } else if (results.getCount() > 0) {
            createResultItem(results.getResultAt(0));
        }

        isScanning = false;
        btnScan.setText("START SCANNING");
        resultListAdapter.notifyDataSetChanged();
    }

// This is called when a MX-1xxx device has became available (USB cable was plugged, or MX devi
ce was turned on),
    // or when a MX-1xxx that was previously available has become unavailable (USB cable was un
plugged, turned off due to inactivity or battery drained)
    @Override
    public void onAvailabilityChanged(ReaderDevice reader) {
        if (reader.getAvailability() == Availability.AVAILABLE) {
            connectToReaderDevice();
        } else if (reader.getAvailability() == Availability.UNAVAILABLE) {
            AlertDialog.Builder alert = new AlertDialog.Builder(this);
            alert
                    .setTitle("Device became unavailable")
                    .setPositiveButton("OK", null)
                    .create()
                    .show();
        }
    }
```

4. Instantiate a *ReaderDevice* object.

Using the MX Reader

Initialize a Reader Device object for MX readers using the following factory method:

```
case MX:
      readerDevice = ReaderDevice.getMXDevice(this);

      //Listen when a MX device has became available/unavailable
```

```
        if (!availabilityListenerStarted) {
            readerDevice.startAvailabilityListening();
            availabilityListenerStarted = true;
        }
```

The availability of the MX mobile terminal can change when the device turns on or off, or if the USB cable gets connected or disconnected. You can handle those changes using the following *ReaderDeviceListener* interface method:

```
public void onAvailabilityChanged(ReaderDevice reader);
```

Using the Camera Reader

You are recommended to use an MX mobile terminal to scan barcodes. However, cmbSDK also supports using the built-in camera of a mobile device. This includes the support of optional external aimers or illumination, and the customization of the live-stream preview's appearance.

To scan barcodes using the built-in camera of a mobile device, initialize the *ReaderDevice* object using the *getPhoneCameraDevice* static method. The camera reader has several options when initialized. The following parameters are required:

- *Context*
- *CameraMode*
- *PreviewOption*
- *ViewGroup*
- *RegistrationKey*
- *CustomData*

The *Context* parameter provides a reference to the activity you are currently in.

The *CameraMode* parameter is of type CameraMode defined in **CameraMode.java** and it accepts one of the values listed in the following table.

These modes provide the following default settings for the reader:

- The zoom feature is available and a button to control it is visible on the live-stream preview (if displayed).
- The simulated hardware trigger (volume control buttons) is disabled.
- When *startScanning()* is called, the decoding process is started.

Based on the selected mode, additional illumination options and behaviors are set, also listed in the table.

VALUE	DESCRIPTION	ILLUMINATION	LIVE-STREAM PREVIEW
NO_AIMER	Initializes the reader to use a live-stream preview on the mobile device screen so the user can position the barcode within the camera's field of view for detection and decoding. Use this mode if the mobile device does not have an aiming accessory.	Illumination is available and a button to control it is visible on the live-stream preview. If commands are sent to the reader for aimer control, they are ignored.	Displayed
PASSIVE_AIMER	Initializes the reader to use a passive aimer. No live-stream preview is available on the device screen in this mode, since an aiming pattern is projected.	Illumination is not available, and the live-stream preview does not have an illumination button. If commands are sent to the reader for illumination control, they are ignored because it is assumed in this mode that the built-in LED of the mobile device is being used for the aimer.	Not Displayed
	Initializes the reader to use the front camera of the mobile device, if available. Use this configuration with	The front camera is used. Illumination is not available and the live-stream	

FRONT_CAMERA	care because most front facing cameras do not have auto focus and illumination, and provide significantly lower resolution images. Illumination is not available in this mode.	preview does not have an illumination button. If commands are sent to the reader for aimer or illumination control, they are ignored.	Displayed

The *PreviewOption* parameter is of type *PreviewOption* defined in **PreviewOption.java**, and is used to change the reader's default values or override defaults derived from the selected *CameraMode*. You can specify the following options:

VALUE	DESCRIPTION
DEFAULTS	Accept all defaults set by the CameraMode.
NO_ZOOM_BUTTON	Hides the zoom button on the live-stream preview, preventing the user from adjusting the zoom of the mobile device camera.
NO_ILLUMINATION_BUTTON	Hides the illumination button on the live-stream preview, preventing the user from toggling the illumination.
HARDWARE_TRIGGER	Enables a simulated hardware trigger (the volume down button) for starting scanning on the mobile device. This button only starts scanning when pressed, it does not need to be held like a purpose-built scanner's trigger, and pressing it a second time does not stop the scanning process.
PAUSED	If using a live-stream preview, the preview is displayed when the *startScanning()* method is called, but the reader does not start decoding until the user presses the on-screen button to start the scanning process.

ALWAYS_SHOW	Forces a live-stream preview to be displayed even if an aiming mode is selected (for example CameraMode == PASSIVE_AIMER).
HIGH_RESOLUTION	Uses the device camera in higher resolution, changing the default 1280x720 resolution to 1920x1080 on devices that support it, and to the default resolution on devices that do not support it. This can help with scanning small barcodes, but increases the decoding time as there is more data to process in each frame.
HIGH_FRAME_RATE	Uses the device's camera in 60 FPS instead of the default 30 FPS to provide a smoother camera preview.
SHOW_CLOSE_BUTTON	Show close button in partial view.

The *ViewGroup (optional)* parameter specifies the container for the live-stream preview. If the parameter is left **null**, a full screen preview is used.

The *RegistrationKey (optional)* parameter is used to license your SDK with license key that you have

The CustomData *(optional)* parameter is used for custom tracking

Example

Create a reader with no aimer, no zoom button, and using a soft trigger:

```
readerDevice = ReaderDevice.getPhoneCameraDevice(this, CameraMode.NO_AIMER, PreviewOption.NO_ZOOM_BU
TTON | PreviewOption.PAUSED);
```

This starts a preview with the scanner paused and a soft trigger button to toggle scanning. After pressing the soft trigger button, the expected preview look is this:

The viewfinder in the image has an active scanning surface as a result of having set active symbologies. For more details, see Enabling Symbologies.

Requesting Camera Permission for Phone Camera Scanner

From Android 6.0 and above you need to request permission from the user to access the built-in camera of the mobile device.

If the camera cannot be opened due to permission issues, the *onConnectionCompleted(readerDevice, error)* callback contains a *CameraPermissionException* in the error parameter. You can check for this exception type with the *instanceof* operator and request permission within the Activity.

```
if (error instanceof CameraPermissionException)
      ActivityCompat.requestPermissions(((ScannerActivity) this), new String[]{Manifest.permission.C
AMERA}, REQUEST_PERMISSION_CODE);
```

You need to implement the *ActivityCompat.OnRequestPermissionResultCallback* interface in your Activity to catch the user permission result.
To handle user response in *onRequestPermissionResult(...)*, you can use the following code to retry connecting to the phone camera:

```
@Override
    public void onRequestPermissionsResult(int requestCode, @NonNull String[] permissions, @NonNull
int[] grantResults) {
        // Check result from permission request. If it is allowed by the user, connect to readerDevi
ce
        if (requestCode == REQUEST_PERMISSION_CODE) {
            if (grantResults.length > 0 && grantResults[0] == PackageManager.PERMISSION_GRANTED) {
                if (readerDevice != null && readerDevice.getConnectionState() != ConnectionState.Con
nected)
                    readerDevice.connect(ScannerActivity.this);
            } else {
                if (ActivityCompat.shouldShowRequestPermissionRationale(((ScannerActivity) this), Ma
nifest.permission.CAMERA)) {
                    AlertDialog.Builder builder = new AlertDialog.Builder(this)
                            .setMessage("You need to allow access to the Camera")
                            .setPositiveButton("OK", new DialogInterface.OnClickListener() {
                                @Override
                                public void onClick(
                                        DialogInterface dialogInterface,
                                        int i) {
                                    ActivityCompat.requestPermissions(ScannerActivity.this, new Stri
ng[]{Manifest.permission.CAMERA},
                                            REQUEST_PERMISSION_CODE);
                                }
                            })
                            .setNegativeButton("Cancel", null);
                    AlertDialog dialog = builder.create();
                    dialog.show();
                }
            }
        }
    }
```

Connecting to the Reader Device

Before connecting, set the *ReaderDeviceListener* object to receive events:

```
readerDevice.setReaderDeviceListener(this);
```

For details, see step 3 in Setting up you application to-use the Cognex Mobile Barcode SDK for Android.

Additionally, you can enable sending the last triggered image and SVG from the reader:

```
readerDevice.enableImage(true);
readerDevice.enableImageGraphics(true);
```

Invoke the connect method after initializing the *ReaderDevice* and setting a listener method to handle responses from the reader. The connect method takes *OnConnectionCompletedListener* as parameter:

```
 //Make sure the device is turned ON and ready
readerDevice.connect(ScannerActivity.this);
```

The following listener methods are called with the new *ReaderDevice* status information:

```
public void onConnectionStateChanged(ReaderDevice reader);
public void onConnectionCompleted(ReaderDevice reader, Throwable err)
```

The *onConnectionCompleted* method passed as a parameter of connect is also invoked as the connection process completes. If there was a connection error, this method provides a *Throwable* object.

Scanning Barcodes

After connecting to the scanning device, you may need to change some of its settings. CmbSDK provides a set of high-level and device-independent APIs for setting and retrieving the current configuration of the device.

You can start scanning barcodes with a properly configured reader by calling the *startScanning* method from your *ReaderDevice* class:

```
readerDevice.startScanning();
```

- If using an MX mobile terminal, you can press a trigger button on the device to turn the scanner on and read a barcode.
- If using the camera reader, cmbSDK starts the camera, displays the configured live-stream preview, and begins analyzing the frames from the video stream, looking for a configured barcode symbology.

You can stop scanning with the following:

```
readerDevice.stopScanning();
```

Scanning stops under one of the following conditions:

- The reader found and decoded a barcode.
- You released the trigger or pressed the stop button on the live-stream preview screen.
- The camera reader timed out without finding a barcode.
- The application calls the *stopScanning()* method.

When a barcode is decoded successfully, you receive a ReadResults iterable result collection object in the *ReaderDevice* listener method. The *onReadResultReceived* listener method is invoked either because the reader decoded a barcode or the scanning process was complete.

Example

```
// This is called after scanning has completed, either by detecting a barcode, canceling the scan by
 using the on-screen button or a hardware trigger button, or if the scanning timed-out
    @Override
    public void onReadResultReceived(ReaderDevice readerDevice, ReadResults results) {
        clearResult();

        if (results.getSubResults() != null && results.getSubResults().size() > 0) {
            for (ReadResult subResult : results.getSubResults()) {
                createResultItem(subResult);
            }
        } else if (results.getCount() > 0) {
            createResultItem(results.getResultAt(0));
        }

        isScanning = false;
        btnScan.setText("START SCANNING");
        resultListAdapter.notifyDataSetChanged();
    }
```

Enabling Symbologies

CmbSDK does not enable any *symbologies* by default for barcode reading with the built-in camera of the mobile device. You must enable all barcode symbologies your application needs to scan to achieve optimal scanning performance. For more details, see

Optimizing Mobile Device Performance.

Individual symbologies can be enabled using the following method of the *ReaderDevice* class:

```
public void setSymbologyEnabled(final Symbology symbology, final boolean enable, final OnSymbologyLi
stener listener)
readerDevice.setSymbologyEnabled(Symbology.DATAMATRIX, true, null);
readerDevice.setSymbologyEnabled(Symbology.UPC_EAN, true, null);
```

All symbologies used for the symbology parameter in this method can be found in **ReaderDevice.java**.

Examples

```
 /* Enable QR scanning */
readerDevice.setSymbologyEnabled(Symbology.QR, true, null);
```

You can also use the same method to disable symbologies:

```
/ * Disable Code 25 scanning */ readerDevice.setSymbologyEnabled(Symbology.C25, false, null);
```

You can implement the method for *OnSymbologiesListener* to check the result of the symbology change:

```
@Override
public void onSymbologyEnabled(ReaderDevice reader, Symbology symbology, Boolean enabled, Throwable
error) {
if (error != null) {
/* Unsuccessful
probably the symbology is unsupported by the current device, or there is a problem with the connecti
on between the readerDevice and MX device */
} else {
// Success }
}
```

Illumination Control

If your reader device is equipped with illumination lights, you can control them: when scanning starts, you can turn them on or off. Use the following method of your Reader Device object:

```
readerDevice.setLightsOn(true, null);
```

You can implement the interface method for *OnLightsListener*, which is the second parameter of the method.

```
public class ScannerActivity extends AppCompatActivity implements .... OnLightsListener .... { ....
@Override
public void onLightsOnCompleted(ReaderDevice reader, Boolean on, Throwable error) {
if (error != null) { // Unsuccessful
} else {
// Success }
} }
```

Not all devices and device modes support illumination control.

Camera Zoom Settings

If the built-in camera of a mobile device is used as the reader device, you can configure zoom levels and how they are used. There are three zoom levels:

- normal: not zoomed (100%)
- level 1 zoom (150% on Android by default)
- level 2 zoom (300% on Android by default)

The *SET CAMERA.ZOOM-PERCENT [100-MAX] [100-MAX]* command is for configuring how far the two levels zoom in percentage. 100 is not zoomed and MAX (goes up to 1000) zooms as far as the device is capable of. The first argument is used for setting level 1 zoom, and the second for level 2 zoom.

You can check the current zoom setting with the *GET CAMERA.ZOOM-PERCENT* command, which returns two values: level 1 and level 2 zoom.

Example

```
readerDevice.getDataManSystem().sendCommand("SET CAMERA.ZOOM-PERCENT 250 500");
```

Note: The camera needs to be started within cmbSDK at least once to have a valid maximum zoom level. It means that if you set the zoom level to 1000 and the device can only go up to 600, the *GET CAMERA.ZOOM-PERCENT* command returns 1000 as long as camera is not opened, but it returns 600 afterwards.

GET/SET CAMERA.ZOOM 0-2 is another command that sets the zoom level or returns the actual setting. Possible values for the SET command are:

- 0 - normal (not zoomed)
- 1 - level 1 zoom
- 2 - level 2 zoom

You can call this command before or even during scanning, and the zoom goes up to the configured level. If scanning is finished, the value is reset to normal behavior (0).

Example

```
readerDevice.getDataManSystem().sendCommand("SET CAMERA.ZOOM 2");
```

Camera Overlay Customization

When using the mobile device's camera, cmbSDK allows you to see the camera preview inside a preview container or in full screen. This preview also contains a customizable overlay. The cmbSDK camera overlay features buttons for zooming, flashing and closing the scanner, and a progress bar indicating the scan timeout.

To use the legacy camera overlay originally used in cmbSDK v2.0.x and ManateeWorks SDK, use this property from MWOverlay before initializing the readerDevice:

```
MWOverlay.overlayMode = MWOverlay.OverlayMode.OM_LEGACY;
```

The customization of the legacy camera overlay is limited, so it is recommended to use the cmbSDK overlay.

When using the cmbSDK overlay:

1. Copy the layout files from the Resources/layout directory into your project and modify them. Use **cmb_scanner_partial_view.xml** if scanning is started inside a container (partial view), and use **cmb_scanner_view.xml** if scanning is started in full screen.

2. Modify the layout according to your needs. For example, you can change the sizes, positions or color of the views, remove views and add your own views, like an overlay image.

CmbSDK accesses the views it uses (zoom, flash, close buttons, the view used for drawing lines on the corners, and the progress bar) with the android:tag attribute. Do not change the android:tag attribute, otherwise cmbSDK cannot recognize the views and continues to function as if they are removed.

Both the cmbSDK and the legacy overlay allow you to change the images used on the zoom and flash buttons if your images have the same name as the names cmbSDK uses. You can find the images and names used in cmbSDK in the Resources/**drawable-mdpi** and **drawable-hdpi** directories. While the other resolutions are optional, these two directories must contain your images with the correct names so that cmbSDK displays the proper images.

Both the cmbSDK and the legacy overlay allow you to change the color and width of the rectangle that is displayed when a barcode is detected.

Example:

```
MWOverlay.locationLineColor = Color.YELLOW;
MWOverlay.locationLineWidth = 6;
```

Advanced Configuration using DataMan Control Commands

Cognex scanning devices implement DataMan Control Commands (DMCC) for configuring and controlling the device. Every feature of the device can be controlled using this text-based language. The API provides a method for sending DMCC commands to the device. Commands exist both for setting and querying configuration properties.

The Appendix includes the complete DMCC reference for the camera reader.

The DMCCs for MX mobile terminals and other supported devices can be found in their respective manuals available through Setup Tool.

The following examples show different DMCC sent to the device for more advanced configuration.

Examples

```
//Change the scan direction to omnidirectional
readerDevice.getDataManSystem().sendCommand("SET DECODER.1D-SYMBOLORIENTATION 0", ScannerActivity.th
is);
//Change live-stream preview's scanning timeout to 10 seconds
readerDevice.getDataManSystem().sendCommand("SET DECODER.MAX-SCAN-TIMEOUT 10", ScannerActivity.this);
```

You can also invoke DMCC query commands and receive their response in the *OnResponseReceivedListener.onResponseReceived()* method.

```
//Get the type of device connected readerDevice.getDataManSystem().sendCommand("GET DEVICE.NAME", ne
w OnResponseReceivedListener() {
@Override
public void onResponseReceived(DataManSystem dataManSystem, DmccResponse dmccResponse) {
if (dmccResponse.getError() != null) {
// Unsuccessful
Log.e("DMCC_ERR", "GET DEVICE.NAME failed",dmccResponse.getError());
} else {
// Success - Use the following result fields:
//int mResponseId = dmccResponse.getResponseId(); //String mPayLoad = dmccResponse.getPayLoad(); //b
yte[] mBinaryData = dmccResponse.getBinaryData(); }
} );
}
```

Resetting the Configuration

NOTE: This section includes resetting to CmbSDK defaults and does not include instruction on resetting to factory defaults.

CmbSDK includes a method for resetting the device to its default settings. In case of an MX mobile terminal, the default setting are the saved configurations. In case of a built-in camera, the default settings are the defaults identified in the Appendix, where no symbologies are enabled.

To reset the device, add:

```
readerDevice.resetConfig(null);
```

When using an MX mobile terminal, there are three states that we can distinguish:

- Factory defaults
- Saved configuration: when there were different configurations set on the device and CONFIG.SAVE DMCC was called.
- Session configuration: when you make changes on the saved configuration, the changes are valid until the MX Mobile Terminal is rebooted. If it is rebooted, it has the saved configuration state.

You can monitor the completion of this async method using the *OnResetConfigListener* interface, which is an optional parameter.

```
public class ScannerActivity extends Activity implements .... OnResetConfigListener .... { ....
@Override
public void onResetConfigCompleted(ReaderDevice reader, Throwable error) {
if (error != null) { // Unsuccessful
} else {
// Success }
}
```

Working with Results

When a barcode is successfully read, the *onReadResultReceived* method creates and returns a ReadResult object. In case of having multiple barcodes successfully read on a single image or frame, multiple *ReadResult* objects are returned in the *ReadResult* object.

The ReadResult class has properties describing the result of a barcode read:

- **is GoodRead()** (boolean): tells whether the read was successful or not
- **get ReadString()** (String): the decoded barcode as a string
- **get Image()** (Bitmap): the image/frame that the decoder processed
- **get ImageGraphics()** (String): the boundary path of the barcode as SVG data
- **get Xml()** (String): the raw XML that the decoder returned
- **get Symbology** (Symbology): the symbology type of the barcode. This enum is defined in ***ReaderDevice.java.***

When a scanning ends with no successful read, a *ReadResult* is returned with the *goodRead* property set to false.

To enable the image and imageGraphics properties being filled in the *ReadResult* object, set the corresponding *enableImage()* and/or *enableImageGraphics()* properties of the *ReaderDevice* object.

To access the raw bytes from the scanned barcode, you can use the XML property. The

bytes are stored as a Base64 String under the "full_string" tag. The example shows how you can use an XML parser to extract the raw bytes from the XML property.

Example

```
try {
    XmlPullParserFactory factory = XmlPullParserFactory.newInstance();
    factory.setNamespaceAware(true);
    XmlPullParser xpp = factory.newPullParser();

    String tag = "";

    // the raw bytes will be stored in this variable
    byte[] bytes;

    xpp.setInput(new StringReader(result.getXml()));
    int eventType = xpp.getEventType();
    while (eventType != XmlPullParser.END_DOCUMENT) {
        if (eventType == XmlPullParser.START_TAG) {
            tag = xpp.getName();
        }
        else if (eventType == XmlPullParser.TEXT && tag.equals("full_string")) {
            String base64String = xpp.getText();
            // Get the bytes from the base64 string here
            bytes = Base64.decode(base64String, Base64.DEFAULT);
            break;
        }
        else if (eventType == XmlPullParser.END_TAG && tag.equals("full_string")) {
            tag = "";
            break;
        }
        eventType = xpp.next();
    }
} catch (Exception e) {
    e.printStackTrace();
}
```

Image Results

The image and SVG results are disabled by default, which means that when scanning, the *ReadResults* do not contain any data in the corresponding properties.

To enable image results, invoke the *enableImage()* method from the *ReaderDevice* object:

```
readerDevice.enableImage(true);
```

To enable SVG results, invoke the *enableImageGraphics()* method on *ReaderDevice* object:

```
readerDevice. enableImageGraphics(true);
```

Handling Disconnects

If a device disconnects due to low battery condition or manual cable disconnection, it can be detected by the *onConnectionStateChanged()* method of the *ReaderDeviceListener* interface.

Note: The *onAvailabilityChanged()* method of *ReaderDeviceListener* is also called when the device becomes physically unavailable. It means that (re)connection is not possible. Always check the *getAvailability()* method of the *ReaderDevice* object before trying to call the *connect()* method.

Appendix - DMCC for the Camera Reader

Appendix - DMCC for the Camera Reader

The following table lists the various DMCC commands supported by the cmbSDK when using the built-in camera for barcode scanning.

Many of these commands are also supported by the MX mobile terminals. Commands that are unique to the camera reader are indicated as such with an X in the last column.

GET/SET	COMMAND	PARAMETER(S)	DESCRIPTION	DEF
GET/SET	BATTERY.CHARGE		Returns the current battery level of the device as a percentage.	

	BEEP		Plays the audible beep (tone).	
GET/SET	BEEP.GOOD	[0-3] [0-2]	Sets the number of beeps (0-3) and the beep tone/pitch (0- 2, for low, medium, high). For the built-in camera, only a single beep with no pitch control is supported. Thus, 0 1 turns the beep off, 1 1 turns the beep on.	1
GET/SET	CAMERA.ZOOM	0-2	The possible values for the SET command are: 0 - normal (un-zoomed), 1 - zoom at level 1, 2 - zoom at level 2. This zoom level is used during scanning. When scanning ends it reset to 0.	
GET/SET	CAMERA.ZOOM-PERCENT	[100-MAX] [100-MAX]	Sets/Returns **level 1** zoom (default 150% on Android, 200% on iOS), and **level 2** zoom (default 300% on Android, 400% on iOS). Note: The camera needs to be started at least once from sdk to have a proper value for max capable zoom (MAX)	
		ON min max	Accepts any length Codabar.	

GET/SET	CODABAR.CODESIZE	OFF min max	Sets min/max length of accepted Codabar.	
GET/SET	C11.CHKCHAR	ON | OFF	Turns Code 11 check digit on/off.	
GET/SET	C11.CHKCHAR-OPTION	1 2	Requires single checksum. Requires double checksum.	
GET/SET	C11.CODESIZE	ON min max OFF min max	Accepts any length Code 11. Sets min/max length of accepted Code 11.	
GET/SET	C25.CODESIZE	ON min max OFF min max	Accepts any length Code 25. Sets min/max length of accepted Code 25.	
GET/SET	C39.ASCII	ON | OFF	Turns Code 39 extended ASCII on/off.	
GET/SET	C39.CODESIZE	ON min max OFF min max	Accepts any length Code 39. Sets min/max length of accepted Code 39.	
GET/SET	C39.CHKCHAR	ON | OFF	Turns Code 39 check digit on/off	
GET/SET	C93.ASCII	ON | OFF	Turns Code 93 extended ASCII on/off	
GET/SET	C93.CODESIZE	ON min max OFF min max	Accepts any length Code 93. Sets min/max length of accepted Code 93.	
GET/SET	COM.DMCC-HEADER	0 1	Sets or gets the header option used	

			in Extended mode.	
GET/SET	COM.DMCC-RESPONSE	0 1	DMCC response format. 1: Extended.	0
	CONFIG.DEFAULT		Resets most of the camera API settings to default, except those noted as not resetting (see Appendix B). To reset all settings, use DEVICE.DEFAULT.	
	CONFIG.SAVE		Saves the current configuration to non-volatile memory (MX-1xxx only). Note that when an MX powers off or enters sleep mode, the last saved configuration is restored when the device wakes up.	
	CONFIG.RESTORE		Restores the saved configuration from non-volatile memory (MX-1xxx only).	
GET/SET	DATA.RESULT-TYPE	0 1 2 4 8	Specifies results to be returned (sum for multiple values): 0 - None 1 -Text string result (default) 2 - XML results 4 - XML stats 8 - Scan image (see IMAGE.* commands)	
GET/SET	DATABAR.EXPANDED	ON | OFF	Turns the DataBar Expanded	

			symbology on/off.	
GET/SET	DATABAR.LIMITED	ON | OFF	Turns the DataBar Limited symbology on/off.	
GET/SET	DATABAR.GROUP DATABAR.RSS14	ON | OFF	Turns the DataBar GROUP (before cmbSDK 2.4.1 known as RSS14) symbology on/off.	
GET/SET	DATABAR.RSS14STACK	ON | OFF	Turns the DataBar RSS14 Stacked symbology on/off. It is deprecated from cmbSDK v2.4.1, use DATABAR.GROUP instead.	
GET/SET	DECODER.1D-SYMBOLORIENTATION	0 1 2 3	Use omnidirectional scan orientation. Use horizontal and vertical scan orientation. Use vertical scan orientation. Use horizontal scan orientation.	
GET/SET	DECODER.EFFORT	1-5	Sets the effort level for image analysis/decoding. The default is 2. Do not use 4-5 for online scanning.	
GET/SET	DECODER.MAX-SCAN-TIMEOUT	1-120	Sets the timeout for the live-stream preview. When the timeout is reached, decoding is paused; the live-stream preview will remain on-screen.	

Returns the max

GET	DECODER.MAX-THREADS		number of CPU threads supported by the device.	
GET/SET	DECODER.THREADS-USED	[0-MAX]	Specify the max number of CPU threads that the scanner can use during the scanning process.	
	DEVICE.DEFAULT		Resets the device (including the camera API) settings to default (see Appendix B).	
GET	DEVICE.FIRMWARE-VER		Gets the device firmware version.	
GET	DEVICE.ID		Returns device ID assigned by Cognex to the scanning device. For a built-in camera, SDK returns 53.	
GET/SET	DEVICE.NAME		Returns the name assigned to the device. By default, this is “MX-“ plus the last 6 digits of DEVICE.SERIAL-NUMBER.	
GET	DEVICE.SERIAL-NUMBER		Returns the serial number of the device. For a built-in camera, the SDK assigns a pseudo-random number.	
GET	DEVICE.TYPE		Returns the device name assigned by Cognex to the scanning device. For a built-in	

			camera, SDK returns “MX-Mobile”.	
GET/SET	FOCUS.FOCUSTIME	0-10	Sets the camera’s auto-focus period (how often the camera should attempt to refocus).	
GET/SET	I2O5.CHKCHAR	ON | OFF	Turns Interleaved 2 of 5 check digit on/off.	
GET/SET	I2O5.CODESIZE	ON min max OFF min max	Accepts any length Interleaved 2 of 5. Sets min/max length of accepted Interleaved 2 of 5.	
GET/SET	IMAGE.FORMAT	0 1 2	Scanner returns image result in bitmap format. Scanner returns image result in JPEG format. Scanner returns image result in PNG format.	
GET/SET	IMAGE.QUALITY	10, 15, 20, ...90	Specifies JPEG image quality.	
GET/SET	IMAGE.SIZE	0 1 2 3	Scanner returns full size image. Scanner returns 1/4 size image. Scanner returns 1/16 size image. Scanner returns 1/62 size image.	
GET/SET	LIGHT.AIMER	0-1	Disables/enables the aimer (when the scanner starts).	D c 1:

GET/SET	LIGHT.AIMER-TIMEOUT	0-600	Aimer Timeout in seconds.	
GET/SET	LIGHT.INTERNAL-ENABLE	ON | OFF	Enables/disables illumination (when the scanner starts).	
GET/SET	MSI.CHKCHAR	ON | OFF	Turns MSI Plessey check digit on/off.	
GET/SET	MSI.CHKCHAR-OPTION	0 1 2 3 4 5	Use mod 10 checksum Use mod 10 mod 10 checksum Use mod 11 checksum (IBM algorithm) Use mod 11 mod 10 checksum (IBM algorithm) Use mod 11 checksum (NCR algorithm) Use mod 11 mod 10 checksum (NCR algorithm)	
GET/SET	MSI.CODESIZE	ON min max OFF min max	Accepts any length MSI Plessey. Sets min/max length of accepted MSI Plessey.	
GET/SET	SYMBOL.AZTECCODE	ON | OFF	Turns the Aztec Code symbology on/off.	
GET/SET	SYMBOL.CODABAR	ON | OFF	Turns the Codabar symbology on/off.	
GET/SET	SYMBOL.C11	ON | OFF	Turns the Code 11 symbology on/off.	
GET/SET	SYMBOL.C128	ON | OFF	Turns the Code 128 symbology on/off.	

GET/SET	SYMBOL.C25	ON | OFF	Turns the Code 25 symbology on/off (standard).
GET/SET	SYMBOL.C39	ON | OFF	Turns the Code 39 symbology on/off.
GET/SET	SYMBOL.C93	ON | OFF	Turns the Code 93 symbology on/off.
GET/SET	SYMBOL.COOP	ON | OFF	Turns the COOP symbology (Code 25 variant) on/off.
GET/SET	SYMBOL.DATAMATRIX	ON | OFF	Turns the Data Matrix symbology on/off.
GET/SET	SYMBOL.DATABAR	ON | OFF	Turns the DataBar symbologies on/off. Check also DATABAR:GROUP, DATABAR.LIMITED, DATABAR.EXPANDED to check which subtypes are read if Databar is turned on.
GET/SET	SYMBOL.DOTCODE	ON | OFF	Turns the DotCode symbology on/off.
GET/SET	SYMBOL.IATA	ON | OFF	Turns the IATA symbology (Code 25 variant) on/off.
GET/SET	SYMBOL.INVERTED	ON | OFF	Turns the Inverted symbology (Code 25 variant) on/off.
GET/SET	SYMBOL.ITF14	ON | OFF	Turns the ITF-14 symbology (Code 25 variant) on/off.
GET/SET	SYMBOL.UPC-EAN	ON | OFF	Turns the UPC-A, UPC-E, EAN-8, and EAN-13

			symbologies on/off.	
GET/SET	SYMBOL.MATRIX	ON | OFF	Turns the Matrix symbology (Code 25 variant) on/off.	
GET/SET	SYMBOL.MAXICODE	ON | OFF	Turns the MaxiCode symbology on/off.	
GET/SET	SYMBOL.MSI	ON | OFF	Turns the MSI Plessey symbology on/off.	
GET/SET	SYMBOL.PDF417	ON | OFF	Turns the PDF417 symbology on/off.	
GET/SET	SYMBOL.PLANET	ON | OFF	Turns the PLANET symbology on/off.	
GET/SET	SYMBOL.POSTNET	ON | OFF	Turns the POSTNET symbology on/off.	
GET/SET	SYMBOL.4STATE-IMB	ON | OFF	Turns the Intelligent Mail Barcode symbology on/off.	
GET/SET	SYMBOL.4STATE-RMC	ON | OFF	Turns the Royal Mail Code symbology on/off.	
GET/SET	SYMBOL.QR	ON | OFF	Turns the QR and MicroQR symbologies on/off.	
GET/SET	TRIGGER.TYPE	0 1 2 3 4 5	Single (not supported) Presentation (not supported) Manual (default) Burst (not supported) Self (not supported) Continuous	

GET/SET	UPC-EAN.EAN13	ON | OFF	Turns the EAN-13 symbology on/off.	
GET/SET	UPC-EAN.EAN8	ON | OFF	Turns the EAN-8 symbology on/off.	
GET/SET	UPC-EAN.UPC-A	ON | OFF	Turns the UPC-A symbology on/off.	
GET/SET	UPC-EAN.UPC-E	ON | OFF	Turns the UPC-E symbology on/off.	
GET/SET	UPC-EAN.UPCE1	ON | OFF	Turns the UPC-E1 symbology on/off.	
GET/SET	UPC-EAN.SUPPLEMENT	0 1 2 3 4	Turns off UPC supplemental codes (ignored) Turns on UPC supplemental codes (required) Required 2 digit supplemental. Required 5 digit supplemental. Not required.	
GET/SET	VIBRATION.GOOD	ON | OFF	Sets/gets whether to vibrate when a code is read (default is ON)	

Appendix B - Camera Reader Defaults

Appendix B - Camera Reader Defaults

The following table lists the defaults the SDK uses on startup for the camera reader.

> **Note:** At the low-level, the cmbSDK supported devices can perform two types of configuration resets: a device reset and a config reset. A device reset restores all

configuration properties to their saved defaults, while a config reset restores mostly the scanning settings, leaving communication settings alone. In the table below, those items that are only reset by a device reset are indicated.

Note: The Reader Device method resetConfig() performs a config reset. To perform a device reset, the DMCC command DEVICE.RESET would need to be issued.

SETTING	DEFAULT VALUE	DEVICE RESET ONLY?
BEEP.GOOD	1 1 (Turn beep on)	
C11.CHKCHAR	OFF	
C11.CHKCHAR-OPTION	1	
C39.ASCII	OFF	
C39.CHKCHAR	OFF	
C93.ASCII	OFF	
COM.DMCC-HEADER	1 (Include Result ID)	Y
COM.DMCC-RESPONSE	0 (Extended)	Y
DATA.RESULT-TYPE	1	Y
DECODER.1D-SYMBOLORIENTATION	1	

DECODER.EFFORT	2	
DECODER.MAX-SCAN-TIMEOUT	60	
DEVICE.NAME	“MX-“ + the last six digits of DEVICE.SERIAL-NUMBER	
Symbologies (SYMBOL.*)	OFF (all symbologies are disabled)	
Symbology sub-types (groups): DATABAR.EXPANDED DATABAR.LIMITED DATABAR.RSS14 DATABAR.RSS14STACK UPC-EAN.EAN13 UPC-EAN.EAN8 UPC-EAN.UPC-A UPC-EAN.UPC-E UPCE- AN.UPCE1	ON OFF OFF OFF ON ON ON ON OFF	
FOCUS.FOCUSTIME	3	
I2O5.CHKCHAR	OFF	
IMAGE.FORMAT	1 (JPEG)	
IMAGE.QUALITY	50	
IMAGE.SIZE	1 (1/4 size)	
LIGHT.AIMER	Default based on cameraMode: 0: NoAimer and FrontCamera 1: PassiveAimer and ActiveAimer	Y
LIGHT.AIMER-TIMEOUT	60	

LIGHT.INTERNAL-ENABLE	OFF	

Setting	Default Value	Device Reset Only?
Minimum/maximum code lengths	ON 4 40	
MSI.CHKCHAR	OFF	
MSI.CHKCHAR-OPTION	0	
TRIGGER.TYPE	2 (Manual)	
UPC-EAN.SUPPLEMENT	0	

Migration from mwSDK to cmbSDK

Why to migrate to cmbSDK

The Manatee Works Barcode Scanner SDK (MW SDK) has been fully integrated into the Cognex Mobile Barcode SDK (cmbSDK).

cmbSDK is backward compatible with the MW SDK and it adds a higher-level API to the scanning methods that utilize the camera of a smartphone or tablet. You can also continue to use the lower-level methods you have become familiar within the MW SDK. Your old MW SDK account and license key(s) remain the same. If you decide to use the higher-level API (from cmbSDK), then your app is supporting the Cognex MX Series mobile barcode readers, and MX Series mobile terminals too, with a single code base.

Remove mwSDK

To avoid conflicts between MW SDK and cbmSDK we need to remove old library (libBarcodeScannerLib.so) files for all architectures and mwbscanner.jar file:

1. Please open **libs** folder inside your project ({project_name}/app/libs) and remove **mwbscanner.jar** file if there is any.

2. Next open **jniLibs** folder inside your project ({project_name}/app/src/main/jniLibs) and remove all **libBarcodeScannerLib.so** files inside all sub folders (arm64-v8a, armeabi-v7a, x86, x86_64).

Change code to use cmbSDK

Next step is to add **cmbsdklib-release.aar** file and add it to your project as dependency. Please navigate to this url to check step by step how to integrate cmbSDK inside your project.

After that please remove all API's and methods that you are using from MW SDK, and follow our guide from here to see how to implement cmbSDK in your project.

Here are some of the main differences between coding in MW SDK and cmbSDK:

FUNCTIONALITY	CMBSDK	MW SDK
Initialization	**ReaderDevice** is instantiated.	**BarcodeScanner** object is used.
Connection	Need to connect and set necessary callback that will handle responses due to connection state changes, availability, result received.	There is no need to connect, only initialize the scanner with parameters according to your needs.
Configuration	Use DMCC commands or ReaderDevice API methods to configure reader device	Use BarcodeScanner API methods to configure scanner
		There are two different ways to start the scanning.

Start scanning	Use *startScanning*() of **ReaderDevice** object. Full screen mode: Use null for *previewContainer* parameter when instantiating **ReaderDevice** object. Partial view: first create the container and handle it to **ReaderDevice** object when creating it.	Full screen mode: New Activity must be started and configured to start the scanning session Partial view: Create fragment with desired size and put scanning activity in that fragment
Read result	Result object will be received in *onReadResultReceived* callback. The read result object contains a lot of details (e.g.: decoded barcode, symbology, image from last frame, SVG...)	Result will be received after frame is decoded in handleDecode function