Cognex Mobile Barcode SDK for Android (v2.0.x)

Introduction

Android (operating system)

Android is a mobile operating system developed by Google, based on the Linux kernel and designed primarily for touchscreen mobile devices such as smartphones and tablets. Android's user interface is mainly based on direct manipulation, using touch gestures that loosely correspond to real-world actions, such as swiping, tapping and pinching, to manipulate on-screen objects, along with a virtual keyboard for text input. In addition to touchscreen devices, Google has further developed Android TV for televisions, Android Auto for cars, and Android Wear for wrist watches, each with a specialized user interface. Variants of Android are also used on notebooks, game consoles, digital cameras, and other electronics.

Android's source code is released by Google under an open source license, although most Android devices ultimately ship with a combination of free and open source and proprietary software, including proprietary software required for accessing Google services. Android is popular with technology companies that require a ready-made, low-cost and customizable operating system for high-tech devices. Its open nature has encouraged a large community of developers and enthusiasts to use the open-source code as a foundation for community-driven projects, which deliver updates to older devices, add new features for advanced users or bring Android to devices originally shipped with other operating systems. The extensive variation of hardware in Android devices causes significant delays for software upgrades, with new versions of the operating system and security patches typically taking months before reaching consumers, or sometimes not at all. The success of Android has made it a target for patent and copyright litigation as part of the so-called "smartphone wars" between technology companies.

Applications ("apps"), which extend the functionality of devices, are written using the Android software development kit (SDK) and, often, the Java programming language. Java may be combined with C/C++, together with a choice of non-default runtimes that allow better C++ support. The Go programming language is also supported, although with a limited set of application programming interfaces (API). In May 2017, Google announced support for Android app development in the Kotlin programming language.

The SDK includes a comprehensive set of development tools, including a debugger, software libraries, a handset emulator based on QEMU, documentation, sample code, and tutorials. Initially, Google's supported integrated development environment (IDE) was Eclipse using the Android Development Tools (ADT) plugin; in December 2014, Google released Android Studio, based on IntelliJ IDEA, as its primary IDE for Android application development. Other development tools are available, including a native development kit (NDK) for applications or extensions in C or C++, Google App Inventor, a visual environment for novice programmers, and various cross platform mobile web applications

frameworks. In January 2014, Google unveiled an framework based on Apache Cordova for porting ChromeHTML 5 web applications to Android, wrapped in a native application shell.

Overview

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 Manatee Works Barcode Scanning SDK, the cmbSDK allows developers 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.

Legal Notices

The following are registered trademarks of Cognex Corporation:

Cognex, 2DMAX, Advantage, AlignPlus, Assemblyplus, Check it with Checker, Checker, Cognex Vision for Industry, Cognex VSOC, CVL, DataMan, DisplayInspect, DVT, EasyBuilder, Hotbars, IDMax, In-Sight, Laser Killer, MVS-8000, OmniView, PatFind, PatFlex, PatInspect, PatMax, PatQuick, SensorView, SmartView, SmartAdvisor, SmartLearn, UltraLight, Vision Solutions, VisionPro, VisionView

The following are trademarks of Cognex Corporation:

The Cognex logo, 1DMax, 3D-Locate, 3DMax, BGAII, CheckPoint, Cognex VSoC, CVC-1000, FFD, iLearn, In-Sight (design insignia with cross-hairs), In-Sight 2000, InspectEdge, Inspection Designer, MVS, NotchMax, OCRMax, PatMax RedLine, ProofRead, SmartSync, ProfilePlus, SmartDisplay, SmartSystem, SMD4, VisiFlex, Xpand

Other product and company trademarks identified herein are the trademarks of their respective owners.

Barcode Scanning with an MX Mobile Terminal

The SDK supports Cognex's line of MX mobile terminals, including the MX-1000 and MX-1502 devices. You can get a detailed description of these models at the official website of Cognex (https://www.cognex.com). Some of the relevant features of these devices regarding the cmbSDK usage are the following:

- **Hardware trigger** : MX mobile terminals include two built-intr iggers for barcode scanning,and support an optional pistol grip with trigger.
- **Illumination and aiming** : MX mobile terminals have built-in illumination and aiming.
- **Stored configurations** : An MX mobile terminal can be preconfigured using Cognex'sDataMan Setup Toolfor Windows, the Quick Setup mobile application, or the SDK itself. The MX mobile terminal can save and restore multiple configuration sets.
- **High-capacity battery** : Each MX mobile terminal has an integrated battery that not only powers the MX scanning engine, but also provides power to the mobile device. The optional pistol grip includes a second battery that doubles the MX's power capacity.

The following features of the MX platform combine to make application development with them straightforward.

- **Ease of setup** : MX mobile terminals come preconfigured to provide an exceptional out-of-the-box experience.In many cases, it is not even necessary to change the defaults of the device. Also, since the MX mobile terminals have saved configurations that can be distributed to all your devices, setup is usually not necessary at the application level. Nevertheless, it is often desirable to put the device in a "known" state when the barcode scanning application starts, so the cmbSDK provides methods to restore the device defaults as well as to control individual settings.

- Having illumination and aimer, there is no need to have a live preview on the smartphone’sscreen like traditional, mobile barcode scanning applications often do. MX mobile terminals do not even support a “live- stream” decoding mode.

Debugging on MX Mobile Terminal

The MX Mobile Terminals connect to your device via the device's usb or lightning port. This means that the port will be occupied while your application is running. There are other ways to debug your application and we will discuss how to debug via wifi below.

Debugging on Android:

This will work with Android Studio, Eclipse, Xamarin or anything else that can run android applications. First connect your android device via USB and make sure you can run and debug your application using the USB cable. Open adb from the android tools, it is what we will be using to have the device connected without using the usb cable.

1. Type "**adb tcpip 5555**" to set the device's port to 5555
2. Get the device's ip address by typing "**adb shell ip -f inet addr show wlan0**" or find it manually in your settings menu.
3. Type "**adb connect device_ip:5555**" to connect to your device. You will get a message saying: connected to if it was connected successfully.
4. Disconnect the USB cable and proceed to run or debug your app on your android device as if it was connected via cable. **Note**: After you plug MX Mobile Terminal in your Mobile Device, wifi connection might be lost. In this case just repeat step 3 once again (type "**adb connect device_ip:5555**").
5. When you're done, type "**adb -s device_ip:5555 usb**" to switch your device back to USB connection mode.

WARNING: leaving the wireless debugging option enabled is dangerous, anyone in your network can connect to your device in debug, even if you are in data network. Do it only when connected to a trusted Wi-Fi and remember to disconnect it (using step #5) when done!

Barcode Scanning with a Smartphone - Android

Barcode Scanning with a Smartphone

It is important to recognize that there are several fundamental differences in the capabilities of smartphones (and tablets) as barcode scanning devices. These differences result in a user experience different from purpose-built scanners, impacting the design of the mobile barcode scanning application.

These differences and the general impact they have on your application are the following:

- A smart phone does not have a dedicated hardware trigger. Without a hardware trigger, the application program itself is generally responsible for initiating the scanning process, which results in accessing the built-in camera, displaying a preview screen if required, and analyzing captured frames from the video stream for barcodes.
- A smartphone (unless otherwise configured) does not have an aimer. Generally, the application program provides a live-stream camera preview on the mobile device screen, thereby allowing the user to see what the camera sees and can then position the device over the barcode.
- Mobile device orientation may need to be considered. Most users hold and use a mobile device primarily in a portrait orientation and for barcode scanning. Having the camera in this orientation is generally sufficient. However, most mobile device cameras have a higher resolution along their landscape orientation. When scanning very long or dense barcodes, reorienting the device to landscape can be beneficial and even necessary to decode these barcodes.
- Image analysis and barcode decoding is performed in software on the mobile device which can be a CPU intensive task. For this reason (and others discussed later), it is highly recommended to only enable the symbologies and features of the SDK your application will need, not everything the cmbSDK is capable of.

The cmbSDK has been specifically engineered to make these differences as transparent as possible to the application developer and the user. By following a few simple guidelines, it is possible to develop applications that work and behave the same, whether using an MX-1000 mobile terminal, or just the built-in camera of the device.

Mobile Device Triggering

Without a hardware trigger, mobile devices must use alternative methods to initiate barcode scanning. There are three common paradigms used:

- **Application or workflow driven trigger**: In this paradigm, it is the application code itself, or the business logic/workflow of the application that starts the scanning process. In other words, the user of the application has reached a point where a barcode needs to be scanned, so the application invokes the scanning module. In simple programming terms, this is akin to calling a function like "startScanner()".
- **Virtual trigger**: This is where the application program provides a button on the screen whereby the user can use to start/stop the scanning process. Depending on the application design, the user may be required to press and hold the virtual button to keep the scanner running. This method is similar to the workflow driven method as the button from the user interface is merely being used to invoke the scanning module.
- **Simulated trigger**: For this method, one of the buttons on the mobile device, typically the volume-down button,is used to simulate a hardware trigger. When the user presses and holds this button, the scanner starts/stops just like when a trigger is pulled on a purpose-built scanner. This method is not commonly used as users find it

non- intuitive and inconvenient to use the volume key in this fashion.

The cmbSDK supports all three of these methods, any one of which (or multiple) can be used in an application.

Mobile Device Aiming

As previously discussed, unlike like purpose-built scanners, mobile devices do not have a built-in aimer. Barcode aiming is generally accomplished by providing a live-stream preview from the camera on the mobile device display: the user can then reposition the device until the barcode presents in the field of view and is decoded.

This task is greatly simplified with the cmbSDK as it provides a built-in preview control that can be displayed full-screen, partial screen, and in either portrait or landscape orientation.

The cmbSDK also supports "passive" aimers: devices that attach to the mobile device or mobile device case that use the LED flash of the device as a light source to project an aiming/targeting pattern. The advantage to these types of aimers is that an on-screen preview is no longer required (since the mobile device can now project an aimer pattern similar to a purpose-built scanner). One limitation of passive aimers, though, is that since the mobile device flash is being used for the aimer, using the LED flash for general scanning illumination is not available.

Mobile Device Orientation

Mobile devices support developing applications for either portrait orientation, landscape orientation, or auto-rotation between the two. The cmbSDK fully supports all three options for both the presentation of the barcode preview as well as the scan direction. As mentioned previously, most barcodes can be scanned by a mobile device regardless of the orientation of the application and/or mobile device.

In some circumstances, though, using landscape orientation may be advantageous or even necessary. Mobile cameras have a higher resolution along the "height" of the image in portrait mode. For example, a common resolution used is 1280x720. When scanning barcodes in portrait mode, this means that 720 pixels of data are available for barcode decoding along the horizontal axis. If scanning a particularly long or dense barcode (e.g. a large PDF417), using the landscape orientation provides 1280 pixels on the horizontal scan line. Orientation makes little to no difference when scanning "square" barcodes like QR, Data Matrix, and MaxiCode.

Mobile Device Performance

Today's smartphones and tablets have significant computing power. With multi-core CPUs and even dedicated image processors, they provide an ideal platform for cost-effective

and efficient barcode decoding. As powerful as these devices are, developers are still advised to consider optimizing their barcode scanning applications. While the SDK has been optimized specifically for mobile environments, image analysis and barcode decoding is still a CPU intensive activity: and since these processes must share the mobile device CPU with the operating system, services, and other applications, developers should limit their applications to only using the features of the SDK that satisfy their needs.

Application optimizations include the following:

- Only enable decoding for the barcode types the application needs to scan.The cmbSDK supports the decoding of almost 40 different barcode types and subtypes, and while you can enable all of these, it can negatively impact performance as well as introduce unwanted side effects
- The more symbologies enabled, the slower the performance. This can lead to sluggish decoding and the degradation of the overall performance of the mobile device, leaving the user with an inaccurate impression of the SDK's capabilities.
- False reads are possible. This is particularly possible when some of the weaker symbologies, like Code 25, are enabled without proper consideration and configuration of other, more advanced features like minimum code length and barcode location. These features help mitigate false reads with the weak symbologies, but at a cost of degraded performance (and again, are not intended to all be turned on and used at the same time).
- Using an optimal camera resolution. By default, the cmbSDK uses HD images (typically1280x720)forbarcode decoding. This resolution is sufficient for all but the very smallest or dense of barcodes. As the application developer, you can use a higher resolution (full HD), but keep in mind that these images are significantly larger, so they will require more time to analyze and decode.
- Using an appropriate decoder effort level.The SDK has a user-configurable effort-level that controls how aggressively the SDK performs image analysis. Like most other settings, the SDK uses a default value (level 2) that is sufficient for almost all barcodes. Using a higher level can result in better decoding of poorer quality barcodes, but at the price of slower performance.

For these reasons, when the cmbSDK is initialized for use with the built-in camera of the mobile device, no barcode symbologies are enabled by default: the application must explicitly enable the symbologies it needs. As most barcode scanning applications only truly need to scan a handful of symbologies, this behavior steers the developer to using the SDK in an efficient manner.

Enabling symbologies is a very simple process, which is explained later in this document.

cmbSDK for Android

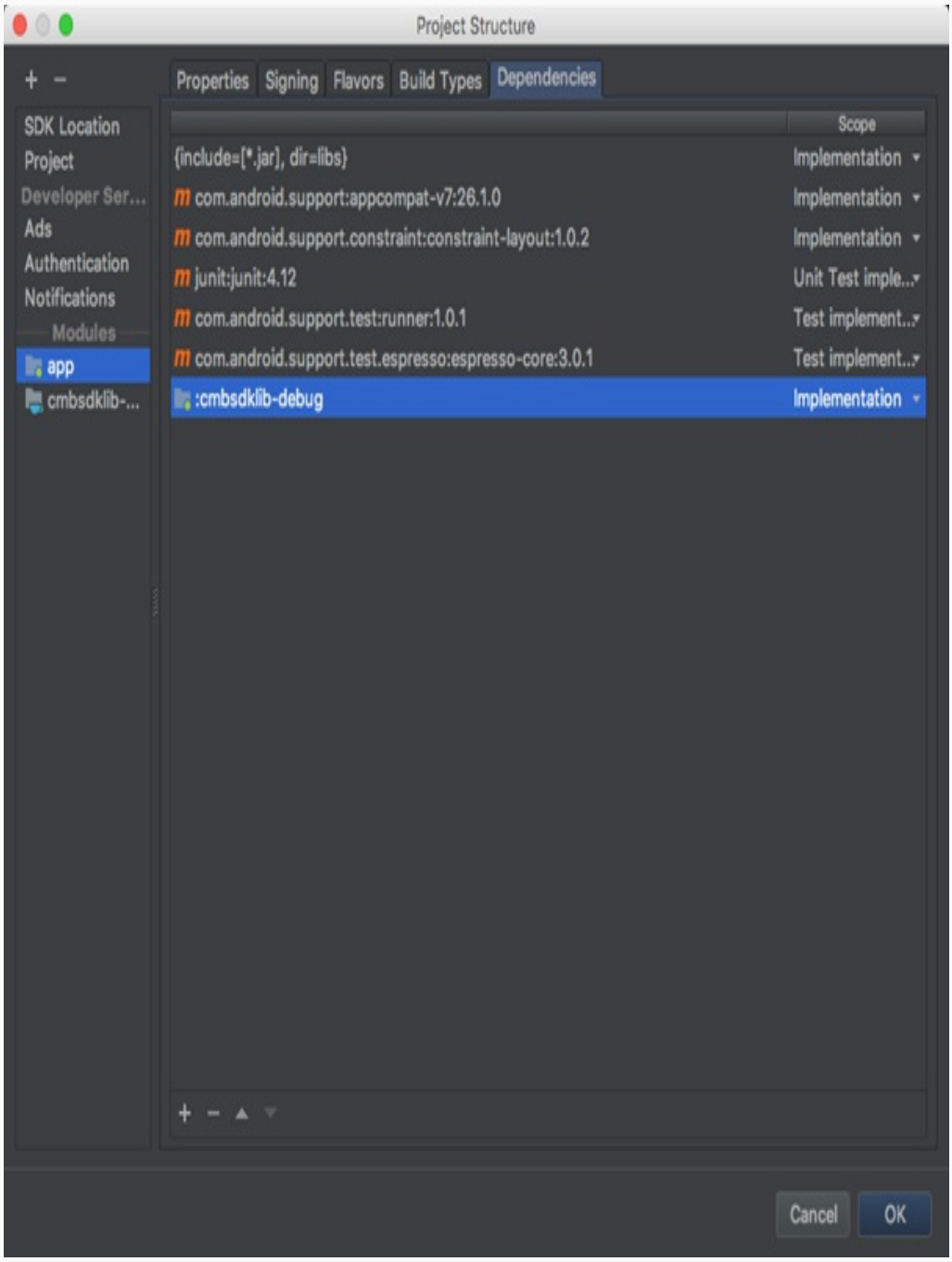
Getting Started

Note: cmbSDK is compatible with Android Studio.

Perform the following steps to install the Android cmbSDK:

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

    - Right click your app module, select **New > Module > Import .JAR/.AAR Package**, and click **Next**.
    - Browse the SDK .AAR file in the File name field and click **Finish**.
    - After the new module is available, right click your app module again, select **Open Module Settings**, and choose **Dependencies** tab.
    - Click the **+** sign on the bottom of the dialog and select **3 Module dependency**.
    - Select cmbsdklib from the popup and click **OK**.
    - The cmbsdklib module should be available now under the **Dependencies** tab.



3. To communicate with MX readers, install MX Connect app from Play Store on your mobile device.

Note: Please uninstall all other applications that connect to the MX reader with previous versions of the SDK. Update the QuickSetup application to latest version.

Licensing the SDK

If you plan to use the cmbSDK to do mobile scanning with a smartphone or tablet (with no MX mobile terminal), then the SDK requires the installation of a license key. Without a

license key, the SDK will still operate, although scanned results will be obfuscated (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, add the following line in your application's AndroidManifest.xml file, under the application tag:

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

Next, put your key in place of YOUR_MX_MOBILE_LICENSE.

```
<application
android:allowBackup="true"
android:icon="@drawable/ic_launcher"
android:label="@string/app_name"
android:theme="@style/AppTheme" >
<activity
android:name=".ScannerActivity"
android:label="@string/app_name" >
<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>
```

Migrating from a Previous DataMan SDK for MX Readers

Previous SDK versions accessed the MX mobile terminal via direct USB Device or USB Accessory connection. These methods are now deprecated, and you should connect to an MX mobile terminal using the MXConnect application. (See Step 4 .)

The cmbSDK provides an easy factory method, DataManSystem.createDataManSystemForMXDevice() to create a DataManSystem for the MX mobile terminal over the MXConnect application.

Please remove any DataManSystem.createDataManSystemOverUsb() and

DataManSystem.createDataManSystemOverUsbAccessory() methods from your project, and also remove the USB_ DEVICE-ATTACHED and USB_ACCESSORY_ATTACHED Intent filters and meta-data from the AndroidManifest.xml file.

You can also delete the USB and accessory descriptor xml files from the XML folder. You can also use and migrate to Barcode SDK to access the MX mobile terminal.

Writing a Mobile Application

The cmbSDK has been designed to provide a high-level, abstract interface for supported scanning devices. This includes not only the MX series of mobile terminals, but also for applications that intend to use the mobile device camera as the imaging device. The intricacies of communicating with and managing these devices is encapsulated within the SDK itself: leaving the application to just connect to the device of choice, then using it.

The primary interface between your application and a supported barcode scanning device is the ReaderDevice class. This class represents the abstraction layer to the device itself, handling all communication as well as any necessary hardware management (e.g. for smartphone scanning).

Perform the following steps to use the cmbSDK:

1. Initialize a Reader Device for the type of device you want to use (MX reader or camera reader).

2. Connect the Reader Device.

3. Configure the reader (if necessary).

4. Start scanning.

Initialization, connection, and configuration generally need to be performed only once in your application except for the following cases:

- An MX reader may become disconnected (times out from disuse,dead battery, etc.). A method has been provided to handle this case, and is discussed in a later section.
- Your application has been designed to allow the user to change devices.The cmbSDK is explicitly designed to support this: your application simply disconnects from the current device and establishes a new connection to a different device. The provided sample application has been written to explicitly demonstrate this capability.

Setting up your Application to use the Cognex Mobile Barcode SDK for Android

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

1. Import the following package members:

```
import com.cognex.dataman.sdk.CameraMode;
import com.cognex.dataman.sdk.ConnectionState;
import com.cognex.dataman.sdk.PreviewOption;
import com.cognex.mobile.barcode.sdk.ReadResult;
import com.cognex.mobile.barcode.sdk.ReadResults;
import com.cognex.mobile.barcode.sdk.ReaderDevice;
import com.cognex.mobile.barcode.sdk.ReaderDevice.Availability;
import com.cognex.mobile.barcode.sdk.ReaderDevice.OnConnectionCompletedListener;
import com.cognex.mobile.barcode.sdk.ReaderDevice.ReaderDeviceListener;
import com.cognex.mobile.barcode.sdk.ReaderDevice.Symbology;
```

2. Provide needed UI elements:

- **ViewGroup** container (like **RelativeLayout**) for the camera preview
- **ImageView** nested inside the**ViewGroup** container with matching size as its parent for showing the last frame of a preview or scanning session:

The below examples use the following:

```
private RelativeLayout rlPreviewContainer;
private ImageView ivPreview;
```

3. The following interfaces are necessary to monitor the connection state of the reader and receive information about the read code:

```
public class ScannerActivity extends Activity implements OnConnectionCompletedListener, ReaderDevice
Listener {
....
@Override
public void onConnectionCompleted(ReaderDevice reader, Throwable error) {
if (error != null) {
// READER DISCONNECTED (ERROR OCCURED)
} }
@Override
public void onConnectionStateChanged(ReaderDevice reader) {
if (reader.getConnectionState() == ConnectionState.Connected) { // READER CONNECTED
} else if (reader.getConnectionState() == ConnectionState.Disconnected) { // READER DISCONNECTED
} }
@Override
public void onReadResultReceived(ReaderDevice reader, ReadResults results) {
```

```
if (results.getCount() > 0) {
ReadResult result = results.getResultAt(0);
// USE String symbologyName; String code; Bitmap frame; VARIABLES IN YOUR APPLICATION
if (result.isGoodRead()) {
String symbologyName;
String code = result.getReadString();
Symbology symbology = result.getSymbology(); if (symbology != null) {
symbologyName = symbology.getName();
tvSymbology.setText(symbologyName); } else {
tvSymbology.setText("UNKNOWN SYMBOLOGY"); }
tvCode.setText(code); } else {
tvSymbology.setText("NO READ");
tvCode.setText(""); }
Bitmap frame = result.getImage();
ivPreview.setImageBitmap(frame); }
// READY TO SCAN AGAIN }
@Override
public void onAvailabilityChanged(ReaderDevice reader) {
if (reader.getAvailability() == Availability.AVAILABLE) { // READER DEVICE IS AVAILABLE AND CAN BE C
ONNECTED
} else {
// DISCONNECT DEVICE
} }
.... }
```

4. Instantiate a ReaderDevice object.

The cmbSDK provides two different reader class initializers: one for scanning using an MX mobile terminal (like the MX- 1000 or MX-1502) and another for scanning using the built-in camera of the mobile device (Android Phones, Android Tablets, etc.).

Using the MX Reader

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

```
boolean listeningForUSB = false;

ScannerActivity.readerDevice = ReaderDevice.getMXDevice(ScannerActivity.this); if (!listeningForUSB)
 {

readerDevice.startAvailabilityListening();
listeningForUSB = 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 (implemented in Step 3 above):

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

Using the Camera Reader

Barcode scanning with the built-in camera of the mobile device can be more complex than with an MX mobile terminal. The cmbSDK supports several configurations to provide the maximum flexibility. This includes the support of optional, external aimers/illumination, as well as the ability to customize the appearance of the live-stream preview.

To scan barcodes using the built-in camera of the 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

The Context parameter simply 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 following values:

- **NO_AIMER**: This 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 when the mobile device does not have an aiming accessory.

- **PASSIVE_AIMER**: This initializes the reader to usea passive aimer, which is an accessory that is attached to the mobile device or mobile device case that uses the built-in LED flash of the mobile device as a light source for projecting an aiming pattern. In this mode, no live-stream preview is presented on the device screen, since an aiming pattern will be projected.

- **FRONT_CAMERA**: This initializes the reader to use the mobile front facing camera of the device, if available (not all mobile devices have a front camera). This is an unusual, but possible configuration. Most front facing cameras do not have auto focus and illumination, and provide significantly lower resolution images. This option should be used with care. In this mode, illumination is not available.

All of the above modes provide the following default settings for the reader:

- The rea rcamera is used.

- The zoom feature is available and a buttonto control it is visible on the live-stream preview (if displayed).
- The simulated hardware trigger is disabled.
- When start Scanning( ) is called,the decoding process is started. (See PreviewOption.PAUSED below for more details).

Based on the selected mode, the following additional options and behaviors are set:

- **NO_AIMER** (NoAimer)
- The live-stream preview is displayed when the *startScanning()* method is called.
- 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 will be ignored.

- **PASSIVE_AIMER** (Passive Aimer)
- The live-stream preview will not be displayed when the *startScanning()* method is called.
- Illumination is not available, and the live-stream preview will not have an illumination button.
- If commands are sent to the reader for illumination control, they will be ignored, since it is assumed in this mode that the built-in LED of the mobile device is being used for the aimer.

- **FRONT_CAMERA** (FrontCamera)
- The live-stream preview is displayed when the startScanning() method is called.
- The front camera is used.
- Illumination is not available and the live-stream preview will not have an illumination button.
- If commands are sent to the reader for aimer or illumination control, they will be ignored.

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*. Multiple options can be specified by OR-ing them when passing the parameter. The available options are:

- **DEFAULTS**: Use this option to accept all defaults set by the CameraMode.
- **NO_ZOOM_BUTTON**: This hides the zoom button on the live-stream preview, preventing a user from adjusting the mobile device camera's zoom.
- **NO_ILLUMINATION_BUTTON**: This hides the illumination button on the live-stream preview, preventing a user from toggling the illumination.
- **HARDWARE_TRIGGER**: This enables a simulated hardware trigger (the volume down button) for starting scanning on the mobile device. This button only starts scanning when pressed. Iit does not need to be held like a more traditional purpose-built scanner's trigger. Pressing the button a second time does not stop the scanning

process.

- **PAUSED**: If using a live-stream preview, when this option is set, the preview will be displayed when the *startScanning()* method is called, but the reader will not started decoding (i.e. looking for barcodes) until the user presses the on-screen scanning button to actually start the scanning process.
- **ALWAYS_SHOW**: This option forces a live-stream preview to be displayed, even if an aiming mode has been selected (e.g. CameraMode == PASSIVE_AIMER)

The last parameter of the ViewGroup type specifies the container for the live-stream preview.

Examples

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

```
ScannerActivity.readerDevice = ReaderDevice.getPhoneCameraDevice( ScannerActivity.this,
CameraMode.NO_AIMER,
PreviewOption.NO_ZOOM_BUTTON | PreviewOption.PAUSED, rlPreviewContainer);
```

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

The viewfinder in the above image has an active scanning surface, a result of having set active symbologies. For more details on this topic, see Configuring the Reader Device).

Requesting Camera permission for Phone Camera Scanner

From Android 6.0 and above you need to request permission from the user to access the phone camera.

If the phone 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);
```

Please note, that you need to implement *ActivityCompat.OnRequestPermissionResultCallback* interface in your Activity to catch user permission result.
To handle user response in *onRequestPermissionResult(...)*, you may use the following code to retry connecting to the PhoneCamera.

```
@Override
public void onRequestPermissionsResult(int requestCode, @NonNull String[] permissions, @NonNull int[
] grantResults) {
    if (requestCode == REQUEST_PERMISSION_CODE) {
        if (grantResults.length > 0 && grantResults[0] == PackageManager.PERMISSION_GRANTED) {
            if (readerDevice != null && readerDevice.getConnectionState() != ConnectionState.Connect
ed)
                readerDevice.connect(ScannerActivity.this);
        } else {
            if (ActivityCompat.shouldShowRequestPermissionRationale(((ScannerActivity) this), Manife
st.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 String[]
{Manifest.permission.CAMERA}, REQUEST_PERMISSION_CODE);
                            }
                        })
                        .setNegativeButton("Cancel", null);
                AlertDialog dialog = builder.create();
                dialog.show();
            }
        }
    }
}
```

Connecting to the Device

Before connecting, set the ReaderDeviceListener object, in order to receive events:

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

See Step 3 for details.

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

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

After initializing the ReaderDevice and setting a listener method to handle responses from the reader, the connect method can be invoked, which takes a OnConnectionCompletedListener (see Step 3 for details) as parameter:

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

If everything was done correctly, the following listener methods will be called with the new ReaderDevice status information. The onConnectionCompleted method (passed as parameter of connect) will also be invoked as the connection process completes, providing a Throwable object, if there was a connection error.

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

Activate scanning

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

You can stop scanning with the following:

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

The onReadResultReceived listener method (see Step 3) will be invoked as a barcode was decoded by the

reader, or the scanning process has finished.

Configuring the Reader Device

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

Like in the case of initializing the Reader Device, there are some differences between using an MX reader and the camera reader for scanning. These differences are detailed in the following sections.

MX Mobile Terminals

The MX family of mobile terminals provides sophisticated device configuration and management, including saved configurations on the device itself. In general, these devices come from Cognex preconfigured for an exceptional out-of- the-box experience with most symbologies and features ready to use.

When custom reconfiguration is desired, this is typically done using either the DataMan Setup Tool, or the DataMan Quick Setup as these tools can be used to distribute saved configurations easily to multiple devices, thereby greatly simplifying configuration management.

However, it is still possible (and sometimes desirable) for the mobile application itself to configure the MX device:

- You can have multiple scanning applications, each of which requires a different set of device settings.
- You may simply want to guarantee that the certain options are in a “known” state and not rely on the fact that the device has been preconfigured correctly.

Built-in Camera

Much like an MX mobile terminal, the cmbSDK employs a default set of options for barcode reading with the built-in camera of the mobile device, providing a good out-of-box experience. However, there are two important differences to keep in mind:

- The cmbSDK does not implement saved configurations for the camera reader. This means that every time an application that uses the camera reader starts, it starts with its defaults.

- The cmbSDK does not enable any *symbologies* by default: you as the application programmer must enable all barcode symbologies your application needs to scan. By

requiring the application program to explicitly enable only the symbologies it needs, the most optimal scanning performance can be achieved. This concept was more thoroughly discussed in the Overview section.

Enabling Symbologies

Individual symbologies can be enabled using the following method of the Reader Device object:

```
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);
```

The same method can also be used to turn symbologies off:

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

You can implement the method for OnSymbologiesListener:

```
@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 connection between
the readerDevice and MX device
*/
} else {
```

```
// Success }
}
```

Resetting the Configuration

The cmbSDK includes a method for resetting the device to its default settings. In the case of an MX mobile terminal, this is the configuration saved by default (not the factory defaults), while in the case of the built-in camera, these are the defaults identified in Appendix B, where no symbologies will be enabled. This method is the following:

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

Being an async method, you can monitor its completion using OnResetConfigListener interface passed as an optional parameter to the method.

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

Illumination Control

If your reader device is equipped with illumination lights (e.g. LEDs), you can control whether they are ON or OFF when scanning starts using the following method of your Reader Device object:

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

Optionally, you can implement the interface method for OnLightsListener, which is the second parameter of the method.

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

Keep in mind that not all devices and device modes supported by the cmbSDK allow for illumination control. For example, if using the built-in camera in passive aimer mode, illumination is not available since the LED is being used for aiming.

Advanced Configuration

Every Cognex scanning device implements DataMan Control Commands (DMCC), a method for configuring and controlling the device. Virtually 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.

Appendix A includes the complete DMCC reference for use with the camera reader. DMCC commands for other supported devices (e.g. the MX-1000) are included with the documentation of that particular device.
Appendix B provides the default values for the camera reader's configuration settings as related to the corresponding DMCC setting.
The following examples show different DMCC commands being 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(); }
} );
}
```

Scanning Barcodes

With a properly configured reader, you are now ready to scan barcodes. This can be done by calling the startScanning method from your Reader Device object.

What happens next is based on the type of Reader Device and how it has been configured, but in general:

- If using an MXreader, the user can now press a trigger button on the device to turn the scanner on and read a barcode.
- If using the camera reader, the 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.

Scanning stops under one of the following conditions:

- The reader found and decoded a barcode.
- The user released the trigger or pressed the stop button on the live-stream preview screen.
- The camera reader timed out with out finding a barcode.
- The application itself calls the *stopScanning()* method.

When a barcode is decoded successfully (the first case), you will receive a ReadResults iterable result collection object in ReaderDevice listener method, already implemented in Step 3.

```
@Override
public void onReadResultReceived(ReaderDevice reader, ReadResults results) {
 if (results.getCount() > 0) {
  ReadResult result = results.getResultAt(0);

  // USE String symbologyName; String code; Bitmap frame; VARIABLES IN YOUR APPLICATION

  if (result.isGoodRead()) {
   String symbologyName;
   String code = result.getReadString();
   Symbology symbology = result.getSymbology();

   if (symbology != null) {
```

```
     symbologyName = symbology.getName();
     tvSymbology.setText(symbologyName);
    } else {
     tvSymbology.setText("UNKNOWN SYMBOLOGY");
    }

    tvCode.setText(code);

   } else {
    tvSymbology.setText("NO READ");
    tvCode.setText("");
   }

   Bitmap frame = result.getImage();
   ivPreview.setImageBitmap(frame);
  }
// READY TO SCAN AGAIN
}
```

In the above example, ivPreview is an ImageView used to display an image of the barcode that was scanned, and tvCode is a TextView used to show the result from the barcode. You can also use the BOOL from result.isGoodRead() to check whether the scan was successful or not.

Working with Results

When a barcode is successfully read, a ReadResult object is created and returned by the onReadResultReceived method. In case of having multiple barcodes successfully read on a single image/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 has 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. This usually happens when scanning is canceled or timed out.

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

To see an example on how the image and SVG graphics are used and displayed in parallel, refer to the sample applications provided in the SDK package.

```
Not all supported devices provide SVG graphics.
```

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. Here's an example how you can use a XML parser to extract the raw bytes from the XML property.

```
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

By default, the image and SVG results are disabled, which means that when scanning, the ReadResults will 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

There might be cases when a device disconnects due to low battery condition or manual cable disconnection. These cases 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 the (re)connection is not possible. You should always check the getAvailability() method of the ReaderDevice object before trying to call the connect() method.

Appendix A - DMCC for the Camera Reader

Appendix A - 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.

GET/SET	COMMAND	PARAMETER(S)	DESCRIPTION	
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.	
GET/SET	CODABAR.CODESIZE	ON min max OFF min max	Accepts any length Codabar. Sets min/max length of accepted Codabar.	X X
GET/SET	C11.CHKCHAR	ON | OFF	Turns Code 11 check digit on/off.	X
GET/SET	C11.CHKCHAR-OPTION	1 2	Requires single checksum. Requires double checksum.	X X
GET/SET	C11.CODESIZE	ON min max OFF min max	Accepts any length Code 11. Sets min/max length of accepted Code 11.	X X
GET/SET	C25.CODESIZE	ON min max OFF min max	Accepts any length Code 25. Sets min/max length of accepted Code 25.	X X

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	X
GET/SET	C93.CODESIZE	ON min max OFF min max	Accepts any length Code 93. Sets min/max length of accepted Code 93.	
	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.	
GET/SET	DATA.RESULT-TYPE	0 1 2 4 8	Specifies results to be returned (sum of multiple values): None Text string result (default) XML results XML stats 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.RSS14	ON | OFF	Turns the DataBar RSS14 symbology on/off.	X
GET/SET	DATABAR.RSS14STACK	ON | OFF	Turns the DataBar RSS14 Stacked symbology on/off.	X
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	Command	Parameter(s)	Description	
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.	X

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.	X
	DEVICE.DEFAULT		Resets 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, the 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, the 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). The default is 3.	
GET/SET	I2O5.CHKCHAR	ON | OFF	Turns Interleaved 2 of 5 check digit on/off.	
GET/SET	I205.CODESIZE	ON min max OFF min max	Accepts any length Interleaved 2 of 5. Sets min/max length of accepted Interleaved 2 of 5.	X X
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).	
GET/SET	LIGHT.AIMER-TIMEOUT	0-600	Timeout in seconds for an aimer. This value is always overridden by DECODER.MAX-SCAN- TIMEOUT.	

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)	X X
GET/SET	MSI.CODESIZE	ON min max OFF min max	Accepts any length MSI Plessey. Sets min/max length of accepted MSI Plessey.	X X
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.	X
GET/SET	SYMBOL.C128	ON | OFF	Turns the Code 128 symbology on/off.	

GET/SET	Command	Parameter(s)	Description	

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.	X
GET/SET	SYMBOL.DATAMATRIX	ON | OFF	Turns the Data Matrix symbology on/off.	
GET/SET	SYMBOL.DATABAR	ON | OFF	Turns the DataBar Expanded and Limited symbologies on/off.	
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.	X
GET/SET	SYMBOL.INVERTED	ON | OFF	Turns the Inverted symbology (Code 25 variant) on/off.	X
GET/SET	SYMBOL.ITF14	ON | OFF	Turns the ITF-14 symbology (Code 25 variant) on/off.	X
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.	X

GET/SET	SYMBOL.MAXICODE	ON | OFF	Turns the MaxiCode symbology on/off.	X
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	Not supported Not supported Manual (default) Not supported Not supported Continuous	
GET/SET	UPC-EAN.EAN13	ON | OFF	Turns the EAN-13 symbology on/off.	X
GET/SET	UPC-EAN.EAN8	ON | OFF	Turns the EAN-8 symbology on/off.	X
GET/SET	UPC-EAN.UPC-A	ON | OFF	Turns the UPC-A symbology on/off.	X

GET/SET	UPC-EAN.UPC-E	ON | OFF	Turns the UPC-E symbology on/off.	X
GET/SET	UPC-EAN.UPCE1	ON | OFF	Turns the UPC-E1 symbology on/off.	
GET/SET	UPCE-AN.SUPPLEMENT	0 1-4	Turns off UPC supplemental codes. Turns on UPC supplemental codes.	

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	

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	

Precautions

Precautions

Observe these precautions when installing the Cognex product, to reduce the risk of injury or equipment damage:

- To reduce the risk of damage or malfunction due to over-voltage, line noise, electrostatic discharge (ESD), power surges, or other irregularities in the power supply, route all cables and wires away from high-voltage power sources.
- Changes or modifications not expressly approved by the party responsible for regulatory compliance could void the user's authority to operate the equipment.
- Cable shielding can be degraded or cables can be damaged or wear out more quickly if a service loop or bend radius is tighter than 10X the cable diameter. The bend radius must begin at least six inches from the connector.
- This device should be used in accordance with the instructions in this manual.
- All specifications are for reference purpose only and may be changed without notice.