Barcode Scanning with a Smartphone - iOS (v2.0.x)

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 smartphone 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 aimingis 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 theLED flash of the device as a light source to project an aiming/targeting pattern. The advantage to these types of aimers isthat an on-screen preview is no longer required (since the mobile device can now project an aimer pattern similar to apurpose-built scanner). One limitation of passive aimers, though, is that since the mobile device flash is being used forthe 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 capabilitie.
- 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 (typically 1280x720) for barcode 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 control show 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.