Honeywell® Captuvo SDK Getting Started

Setting up Xcode Environment

The Captuvo SDK contains two files that must be included in a project for it to be successfully used. The *Captuvo.h* header file and the static library *libCaptuvoSDK.a.* It is recommended that a folder be created in your project and both of these files copied into it.

In order to include the header file, simply drag and drop into your current project.

Xcode will then prompt you with options for adding the file. You must select each target that will be built using the SDK. You will also have the option of copying the file into the destination. If you have already copied it into the project as mentioned before, this does not need to be done.

The static library also must be added to the project. In order to add the static library, select your project file from the file navigator. With the project selected, choose the desired target and scroll down to the *Linked Frameworks and Library* section. It is possible to add the library by either dragging and dropping it here or using the add library button. Once the library is added it should be visible in the list of libraries and in the file navigator. It is also required that the *ExternalAccessory.framework* be added here as well when using the Captuvo SDK. It can be added by pressing the add button and finding it in the list of Apple provided frameworks.

With both files added, it should now be possible to build a project using the library. The *libCaptuvoSDK.a* is compiled as a fat binary and is compatible with both armv7/arm64 to support iOS 6.0 or above (iOS device) and i386 (simulator).

Initializing and Using the SDK

The Captuvo SDK uses a singleton data pattern. In order to initialize the SDK, you simply have to access it. By requesting a shared object the SDK will either return the current valid SDK object or create a new one. The following code is an example of how the SDK should be used.

[[Captuvo sharedCaptuvoDevice] getCaptuvoSerialNumber]

In this example, a *sharedCaptuvoDevice* (the shared object for the SDK) is obtained and then the SDK method *getCaptuvoSerialNumber* is called. The Captuvo object may be stored locally but is not required because there is very little overhead in requesting the *sharedCaptucoDevice.*

The Captuvo SDK loosely follows the delegate data pattern for returning requested data. In the normal delegate pattern an object (A) typically will become the delegate for another object (B) when *A* is composed of *B,* and *A* and *B* have a one to one relationship. The SDK differs in this pattern since the calling objects only have a loose association with the SDK object and there is a many to one relationship between the single SDK object and the many users of it. The SDK allows multiple objects to register to be a delegate. This is done with the following method:

-(void) addCaptuvoDelegate:(id<CaptuvoEventsProtocol>) delegate

The object must implement the *CaptuvoEventsProtocol.* Once an object is added as delegate it will then be notified of events via the methods in the *CaptuvoEventsProtocol.* The object is only required to implement the protocol methods that are of interest to that object.

It is best to remove an object from the delegate list when it is about to be unallocated or is no longer interested in the protocol events. However this is not required and the SDK will cleanup delegates that have become unallocated. To remove an object as a delegate the follow method can be used:

-(void) removeCaptuvoDelegate:(id<CaptuvoEventsProtocol>) delegate

The Captuvo SDK is not thread safe and may have unexpected results if used concurrently on multiple threads.

Working with the Decoder

The decoder is the subsystem that is responsible for scanning and processing barcodes. To work with this subsystem it must be activated from the SDK. To activate the decoder the following method is used:

-(ProtocolConnectionStatus) startDecoderHardware

This method will return a *ProtocolConnectionStatus* enumeration that should be checked to ensure there were no errors starting the hardware. Once the decoder is activated it can then be used and configured. If the decoder is not activated then any decoder method will have no effect and will simply return. It is recommended to shut down the decoder when it is not being used to conserve battery power. To shut it down the following method is used:

-(void) stopDecoderHardware

Once the decoder is activated it can begin to scan barcodes by pressing the side trigger. In this simple example the object(s) that are interested in getting the results of the scan would implement the *CaptuvoEventsProtocol,* and the following method:

-(void) decoderDataReceived:(NSString) data*

This method will be called with the data that was read from the barcode. Every object that is a delegate of the SDK and has this method implemented, will receive the string data. See the SDK documentation for information on advance decoder features.

-(void) decoderRawDataReceived:(NSData) data*

This method will be called with the data that was read from the barcode. Every object that is a delegate of the SDK and has this method implemented, will receive the binary data. See the SDK documentation for information on advance decoder features.

Working with the MSR

The MSR (mag strip reader) is the subsystem for reading swiped cards and is available on select Captuvo products. To work with this subsystem it must be activated. To activate the MSR the following method is used:

-(ProtocolConnectionStatus) startMSRHardware

This method will return a *ProtocolConnectionStatus* enumeration that should be checked to ensure there were no errors starting the hardware. Once the MSR is activated it can then be used and configured. If the MSR is not activated then any MSR method will have no effect and will simply return. It is recommended to shutdown the MSR when it is not being used to conserve battery power. To shut it down the following method is used, note that it takes about a second until the MSR can be started again:

The developer who wants to use the MSR reader in their application, Must specifically enable MSR reader after start MSR hardware.

-(void) enableMSRReader

This method places the MSR reader in an un-enable state. The MSR reader default is un-enable state. This method is called the MSR reader will be active and will return data to its delegate when a card is swiped through the MSR reader.

By default there has been no MSR track selected. Developer must use the below method to configure track selection of MSR data capture. If track selection has been made for MSR reader, the MSR will not be able to read data from the swipe.

-(void) setMSRTrackSelection:(TrackSelection) selection.

This method is used to set what tracks the MSR will read when the card is swiped. If any of the required tracks fail to read for any reason, no data for any track will be sent. The TrackSelection enum type are as follows.

TrackSelectionAnyTrack,	/**< Any Track */
TrackSelectionRequire1,	/**< Require Track 1 Only */
TrackSelectionRequire2,	/**< Require Track 2 Only */
TrackSelectionRequire1and2,	/**< Require Track 1 & Track 2 */
TrackSelectionRequire3,	/**< Require Track 3 Only */
TrackSelectionRequire1and3,	/**< Require Track 1 & Track 3 */
TrackSelectionRequire2and3,	/**< Require Track 2 & Track 3 */
TrackSelectionRequireAllTracks,	/**< Require All Three Tracks */
TrackSelectionAnyTrack1or2,	/**< Any Track 1 & 2 */
TrackSelectionAnyTrack2or3,	/**< Any Track 2 & 3 */
TrackSelectionUndefined	/**< Error state */

-(void) stopMSRHardware

Cards can be swiped once the MSR is active. For an object to receive the swiped data they must implement the *CaptuvoEventsProtocol,* and one of the following methods:

-(void) msrStringDataReceived:(NSString) data validData:(BOOL) status*

-(void) msrRawDataReceived:(NSData) data validData:(BOOL) status*

The first will provide the data from the card as a string and the second will provide the data as raw bytes. See the SDK documentation for information on advanced MSR features.

Power Management

Since the Decoder and at a lesser degree the MSR, use the most power it is best to shut down both when they are not needed by the user. Both the decoder and MSR will continue to run when the screen locks on the iOS devices either by the user pressing the lock button or by a time out. It is best to use the iOS notification *UIApplicationWillResignActiveNotification* to shutdown these systems in order to conserve battery life. The hardware is shutdown by calling stopMSRHardware and stopDecoderHardware.

The SDK provides methods for getting the battery status and the remaining battery life of the Captuvo.

The following SDK method provides the charge/battery life status for Captuvo for iPhone5/5C/5S, Captuvo for iPod touch5, Captuvo for iPad mini 1/2:

-(ChargeStatus) getChargeStatus

The Charge status will either be: not charging, charging, fully charged, or undefined. The status will only be undefined for a brief period at start up and in error conditions. It is also possible to use a *CaptuvoEventsProtocol* delegate method to get notified of a change in this status. That method is listed below:

-(void) pmChargeStatusChange:(ChargeStatus) newChargeStatus

The SDK is capable of reporting the remaining battery life in granularities of 1/4th's. The method to request this data is as follows:

-(BatteryStatus) getBatteryStatus

The Battery status will be one of the following: 0 of 4 (empty), 1 of 4 (1 bar), 2 of 4 (2 bars), 3 of 4 (3 bars), 4 of 4 (full or 4 bars), or external source connected. It is also possible to be notified of changes to this status by using the *CaptuvoEventsProtocol* delegate method:

The following SDK method provides the charge/battery life status for Captuvo for iPhone 6 and iPhone 6 Plus:

-(void) queryBatteryDetailInfo

The Captuvo for iPhone 6 and iPhone 6 Plus battery has capability to provide more granular battery detail. Battery information that can be accessed is as follows:

- Valid data: -provides if the Captuvo for iPhone6/6Plus battery is valid or not (YES/NO).
- Percentage: -provides the current the Captuvo for iPhone6/6Plus battery percentage (%).
- Remaining Capacity: -provides the the Captuvo for iPhone6/6Plus battery current remaining capacity (mAh)
- Total Capacity: -provides the the Captuvo for iPhone6/6Plus battery total capacity (mAh).
- Temperature: -provides the the Captuvo for iPhone6/6Plus battery temperure use Kelvin (K).
- Voltage: -provides the the Captuvo for iPhone6/6Plus battery current voltage (V).

Callback delegate method is the following:

-(void) responseBatteryDetailInformation:(cupertinoBatteryDetailInfo) batteryInfo;*

-(void) pmBatteryStatusChange:(BatteryStatus) newBatteryStatus

The SDK also provides two warning delegate methods that **should** be implemented somewhere in every app. One is a low battery warning and the second is a dead battery shutdown. The low battery warning *CaptuvoEventsProtocol* delegate method is listed below:

-(void) pmLowBatteryWarning

Once a low battery warning is issued the app should notify the user. At this point the sled will continue to operate but the remaining battery life is very low. A depleted battery shutdown will follow if the battery is not connected to an external power source. The following is the *CaptuvoEventsProtocol* delegate method for a shutdown.

-(void) pmLowBatteryShutdown

Once a shutdown is issued both the decoder and MSR are shut down and are unable to be restarted. If they are updated up again they will return the following: *ProtocolConnectionStatusUnableToConnectIncompatiableSledFirmware* status.

Troubleshooting

The following points are guidelines/markers, which the developer is advised to follow while consuming functionality from Captuvo. These guidelines have been laid down to prevent starting of decoder hardware while iOS device is in a sleep state.

Behavior: It was observed that for some applications such as Honeywell Price, the decoder remained active while the application was left in the foreground and device was left idle for some time (~10mins). This resulted in heavy battery drainage as well as unintended scanning activity. It was also observed that this behavior was limited to the scenarios where the application was left in the foreground and the device was left to be idle.

Diagnosis: This behavior was found to be more prominent in versions of iOS 7 and above. A key observation was made in iOS 8 which supports any kind of background activity. It was observed that in the idle/deep sleep state, Captuvo connect notifications were received sporadically from the sled via iOS. This notification would be passed on to the application (note: the application would be the last active application, before device goes in deep sleep). On receiving these notifications, if an application has invoked startDecoderHardware method in delegated methods, the result would be the starting of decoder hardware.

Guidelines: Even though guidelines have been laid out after testing ways to prevent aforementioned behavior on iOS 7 & above, developer is advised to adhere by them while using any kind of iOS version.

1) Until and unless absolutely required, one should avoid adding UIBackgroundMode key in info.plist file with value set to external-accessory.

2) Application Delegate: All hardware components should be stopped whenever application goes in background or is about to terminate. If need be, start decoder hardware in *applicationDidBecomeActive*. Until and unless required captuvoConnected and disconnected delegate methods should not be implemented in the application delegate file.

3) A developer wishing to start the decoder hardware immediately inside captuvoConnected, should put a check if the application is in active state or not. Similarly if decoder functionality is not required, it should be stopped in captuvoDisconnected and delegate should also be removed in this method.

4) In a multi screen application, decoder hardware should only be started on the screen

where its functionality is required in its view delegate *viewDidAppear*. One should also check before starting the decoder, which if it’s already running it shouldn't be started.

5) Other important thing to remember is to remove delegates of Captuvo notifications whenever view disappears and unloads. On the other hand, delegate can be added in the *viewWillAppear* delegate method, so that it would receive timely notifications from Captuvo.

All above guidelines are advised to be followed in a tandem, based on whether or not the application is multi-screen or single screen OR whether or not it’s a requirement to start decoder hardware the moment Captuvo is connected.

Please see the additional SDK documentation for information on advanced features.