Honeywell

SwiftDecoder™ SDK

For iOS®

Software Integration Manual

Disclaimer

Customer Support and Technical Assistance

For customer support, contact your local Honeywell Sales Representative or fill out the support form at sensing.honeywell.com/contact-support-form.

For our latest contact information, see sensing.honeywell.com/contact.

TABLE OF CONTENTS

CHAPTER

1 INTRODUCTION

The Software Integration Guide for SwiftDecoder™ SDK contains information on how to successfully integrate SwiftDecoder™ technology into your iOS™ application.

Purpose

This document describes the usage and application integration of the SwiftDecoder™ SDK. The purpose of this document is to outline the steps necessary to successfully integrate the SwiftDecoder SDK into an application, as well as communicate its proper use.

Scope

This Software Integration Guide contains information for the following software components of the SwiftDecoder™ SDK software solution.

- API Framework which embeds Library, Header Files, and Resources
- API License Key
- Required Core Frameworks and dylibs

Intended Audience

The intended audience is a software developer looking to understand the integration steps needed to integrate SwiftDecoder technology into an application.

Terms, Acronyms, & Abbreviations

Below are the terms, acronyms, and abbreviations used within this document. Additional project-specific terms can be found in the project glossary.

Term, Acronym, Abbreviation	Definition
API	Application Programming Interface
SDK	Software Development Kit

CHAPTER

2 DESIGN CONSIDERATIONS AND API UPDATES

The use of the SwiftDecoder™ Mobile SDK requires multiple application integration steps. This integration requires the user to add the API dynamic library, various header and Resource files to their project. In addition to this they must ensure to reference a variety of core cocoa frameworks listed later in this document.

Integrating SwiftDecoder.xcframework

Using Cocoapods

Add SwiftDecoder via Cocoapods:

- Create a podfile in the root of your project folder by running pod init command
- Add SwiftDecoder framework Podfile. For instance, add:

```
pod 'SwiftDecoder', :git => 'https://github.com/HON-IA-SD/
swiftdecoder-ios-core.git'
```

- If you want to install a specific version, you can do so by specifying the version explicitly. For instance:

```
pod 'SwiftDecoder', :git => 'https://github.com/HON-IA-SD/
swiftdecoder-ios-core.git', :tag => '5.8.14'
```

Note: *Since this is a private repo, add your github credentials in keychain.*

Using Swift Package Manager

Add SwiftDecoder via swift-package-manager:

- To include the SDK in your Xcode project you have to add a Package Dependency in the project's settings section and in the tab "Package Dependencies".
- Add the url of this repository: https://github.com/HON-IA-SD/swiftdecoder-ios-core

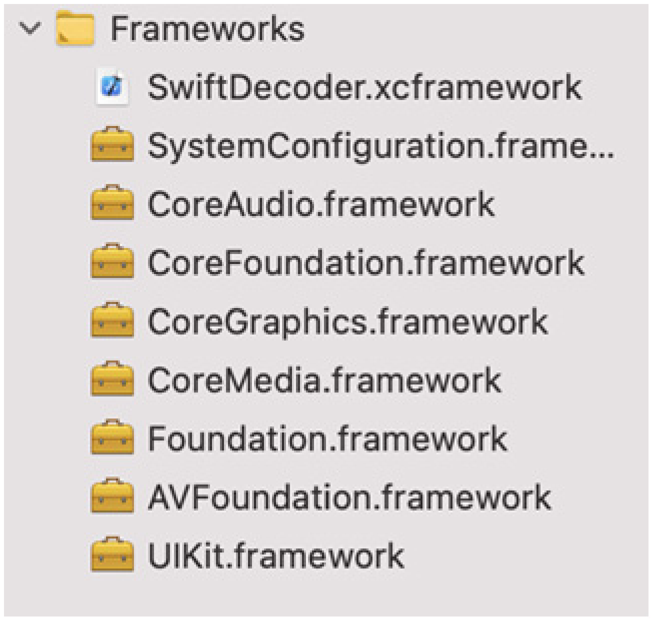
- Once you have added the package, you should see SwiftDecoder added to your Package Dependencies in Xcode's Project Navigator

Note: *Since this is a private repo, add your github credentials under Xcode settings -> accounts*

Manual Integration

API Framework

SwiftDecoder.xcframework is a framework that contains all the needed decoding related methods. This needs to be linked into your application. You must ensure that your application has a search path set to find this library.



Frameworks
SwiftDecoder.xcframework
SystemConfiguration.frame...
CoreAudio.framework
CoreFoundation.framework
CoreGraphics.framework
CoreMedia.framework
Foundation.framework
AVFoundation.framework
UIKit.framework

Header Files

There are several header files that are embedded within SwiftDecoder.xcframework that need to be referenced for various operations within the API. You must ensure that your application has a search path set to find these headers.

Integrating SwiftOCR.xcframework

Using Cocoapods

Add SwiftDecoder via Cocoapods:

- Create a podfile in the root of your project folder by running pod init command

- Add SwiftOCR and its dependency SwiftDecoder framework in Podfile.
  For instance, add:

```
pod 'SwiftOCR', :git => 'https://github.com/HON-IA-SD/
swiftdecoder-ios-swiftocr.git'
pod 'SwiftDecoder', :git => 'https://github.com/HON-IA-SD/
swiftdecoder-ios-core.git'
```

- If you want to install a specific version, you can do so by specifying the version explicitly. For instance:

```
pod 'SwiftOCR', :git => 'https://github.com/HON-IA-SD/
swiftdecoder-ios-swiftocr.git', :tag => '6.0.5'
```

Note: *Since this is a private repo, add your github credentials in keychain*

Using Swift Package Manager

Add SwiftDecoder via swift-package-manager:

- To include the SDK in your Xcode project you have to add a Package Dependency in the project's settings section and in the tab "Package Dependencies".
- Add the url of this repository:

`https://github.com/HON-IA-SD/swiftdecoder-ios-swiftocr` and select `main branch`

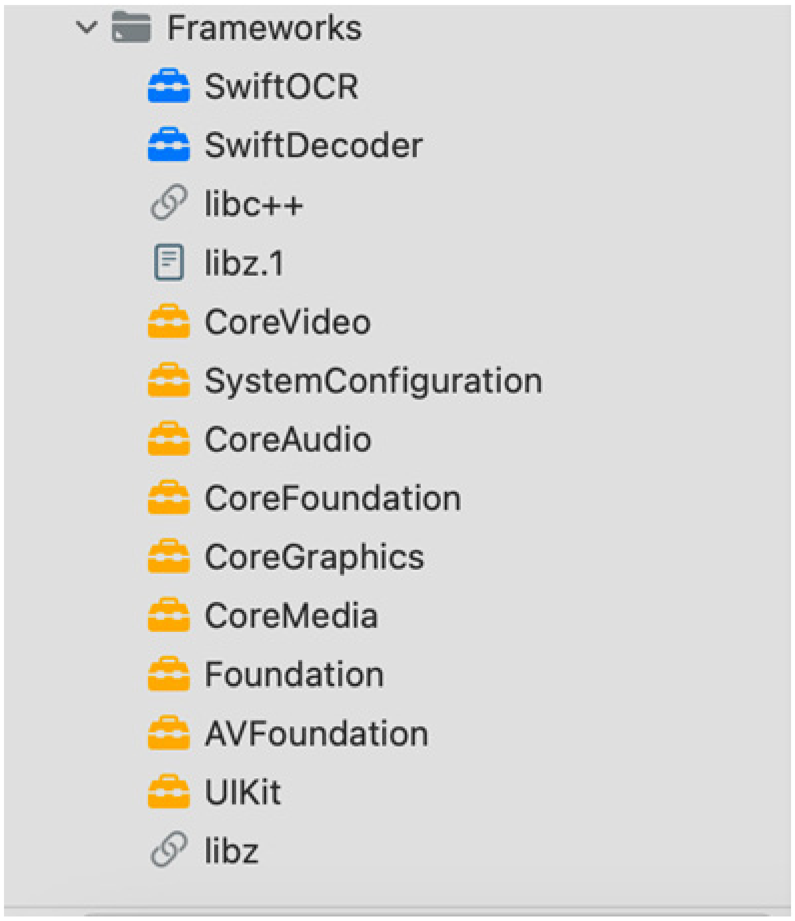
- Once you have added all the packages, you should see SwiftOCR, SwiftDecoder, Currency added to your Package Dependencies in Xcode's Project Navigator

Note: *Since this is a private repo, add your github credentials under Xcode settings -> accounts.*

Manual Integration

API Framework

SwiftOCR.xcframework is a framework that contains all the OCR related methods. This will need to be linked into your application. You must ensure that your application has a search path set to find this library. Also this xcframework is dependent of SwiftDecoder.xcframework



Frameworks
SwiftOCR
SwiftDecoder
libc++
libz.1
CoreVideo
SystemConfiguration
CoreAudio
CoreFoundation
CoreGraphics
CoreMedia
Foundation
AVFoundation
UIKit
libz

Header Files

There are several header files that is embedded within SwiftOCR.xcframework that will need to be referenced for various operations within the API. You must ensure that your application has a search path set to find these headers.

Integratin the DLEngine.xcframework

Using Cocoapods

To add DLEngine via Cocoapods:

- Create a podfile in the root of your project folder by running pod init command
- Add DLEngine and its dependency SwiftDecoder framework in Podfile.

For instance, add:

```
pod 'DLEngine', :git => 'https://github.com/HON-IA-SD/
swiftdecoder-ios-dl.git'
pod 'SwiftDecoder', :git => 'https://github.com/HON-IA-SD/
swiftdecoder-ios-core.git'
```

- If you want to install a specific version, you can do so by specifying the version explicitly.

For instance:

```
pod 'SwiftOCR', :git => 'https://github.com/HON-IA-SD/
swiftdecoder-ios-dlengine.git', :tag => '6.0.7'
```

Note: *Since this is a private repo, add your github credentials in keychain*

Using Swift Package Manager

Add SwiftDecoder via swift-package-manager:

- To include the SDK in your Xcode project you have to add a Package Dependency in the project's settings section and in the tab **Package Dependencies**.
- Add the url of this repository: https://github.com/HON-IA-SD/swiftdecoder-ios-dl and select main branch
- Once you have added all the packages, you should see DLEngine and SwiftDecoder, added to your Package Dependencies in Xcode's Project Navigator

Note: *Since this is a private repo, add your github credentials under **Xcode settings** -> **accounts**.*

Manual Integration

Manual integration is not recommended for DL so that all the internal dependencies get added properly.

Barcode Decoding Options

With SwiftDecoder, barcode scanning can be achieved several different ways.

- By calling scanBarcode() on an HSMDecoder instance
- By embedding an HSMDecodeComponent within your own activity
- By writing your own SwiftPlugin

scanBarcode

The first, and simplest, is by calling scanBarcode() on an HSMDecoder instance. This will launch the default barcode scanning activity (HSMCameraPreview) and will return the result to any listeners.

HSMDecodeComponent

Another option is to embed an HSMDecodeComponent within your own activity. An HSMDecodeComponent is the real-time camera preview frame layout that can be included in your own activity for greater control over the look and feel of the barcode scanning operation. This is the component that is used behind the scenes in the HSMCameraPreview activity.

SwiftPlugin

You can also write your own SwiftPlugin to modify the barcode scanning experience even further. This option allows you to register your plugin with the system and completely control the look and function of a barcode scanning operation. Any registered plugins will be run in both the default HSMCameraPreview activity as well as within any HSMDecodeComponent.

OCR Decoding Option (from 6.0)

This decoding option allows you to scan and decode any generic OCR text found on labels, receipts, forms, etc. This feature is available on SwiftOCR.xcframework as a microservice for each of the detection modes.

The 2 detection modes, TEMPLATE_OCR and OPEN_OCR, can be set using setSwiftOCRDetectionMode API and Scan needs to be performed using the core xcframework HSMDecodeComponent.

Drivers License Decoding Option (from 6.0)

This decoding option allows you to scan and decode any International Drivers license which returns decoded data in EZDL format.

This feature is available on DLEngine.xcframework as a microservice.

Handling Unique ID for Licensing

There are no major changes on unique ID handling front for IOS platform. Integrating application needs to make a note of below constraints for IOS platform.

Note: *The UDID depends on the certificate required for application signing. So if 2 applications are using different certificates for 2 different apps on same device the Unique ID will be different.*

It majorly depends on the company policy for certificate usage tied to your environment. In case 2, applications are signed using the same certificate and are run on the same device, both applications would have same Unique ID tied to licensing.

Image Resolution Configuration API (from 6.0)

From 6.0 release onwards the core SwiftDecoder.xcframework handling camera allows you to configure the image resolution to be used for decoding. Refer to the API document for further details on the API parameters and return values. Here is the quick reference of the API's.

```
(BOOL)setCameraResolution:(Resolution)resolution;
(NSArray*)getCameraResolution;
(NSString*)getStringForCameraResolution:(int)resolution;
(Resolution)getCameraResolutionForString:(NSString*)resolutionSt
r;
```

Augmented Reality Feature changes (from 6.0)

Augmented Reality feature improvements have been added on 6.0.

Apart from improvements, a new plugin called SwiftFindPlugin has been added which allows search and find functionality on barcodes. Existing integrators need to look for compilation issues linked with the below renaming of existing plugins.

- Existing AugmentedRealityPlugin renamed to PreviewSelectPlugin.
- PanaromicDecodePlugin renamed to BatchScanPlugin.
- PanaromicDecodeResultListener renamed to BatchScanResultListener.
- New API called getPreviewResult added on PreviewSelectPlugin.
- New API called getScannedBatchResult added on BatchScanPlugin.

iOS Minimum Supported Version

The minimum supported version of iOS is 13.

EZDL and Parser Library Integration (from 5.5)

Starting in the 5.5 release we have integrated EZDL as part of the SwiftDecoder SDK. We also have included various parsers inside the library for Motor Vehicle, Boarding Pass, and Passport/Visa/ID scanning.

- **MRZ** — Machine readable zone at the bottom of travel identification documents such as a Passport, Visa, or ID card
- **EZDL** — PDF417 on the back of driver's licenses in the United States and Canada
- **EZBP** — PDF417 on flight boarding passes
- **EZMV** — PDF417 on motor vehicle documents such as the Title and Registration.

API modification for EZDL

Please modify your application if using a separate EZDL library. The library is now included in SwiftDecoder.

1. **Swift Projects**: Remove the header file import ***#import "LicenseParser.h"*** if applicable.
2. **Objective-C Projects**: Remove ***#import "LicenseParser.h"*** and add ***#import <SwiftDecoder/LicenseParser.h>***
3. ***LicenseParser.parseRawData*** method does not require any changes.
4. ***+(BOOL)isLicenseEnabled*** is a new API to check if your license is authorized to use this micro service.

armv7 Support Deprecated (from 5.6)

Support for armv7 architecture has been removed.

CHAPTER
3 SDK COMPONENTS

Overview

This section provides details about the main SDK components.

API Framework

The API framework includes 3 different xcframeworks

- SwiftDecoder.xcframework is a framework that contains all the needed barcode decoding, licensing, and camera related methods.
- SwiftOCR.xcframework is a framework that contains all the needed OCR decoding related methods.
- DLEngine.xcframework is a framework that contains all the needed DL decoding related methods.

API License Key & Licensing Models

The API license key will be delivered to the licensee via e-mail. This will need to be passed to the ActivationManager activate method before the API will be fully functional. This license key should be kept safe and never given to anyone that does not require access to it. This should be the first decoding related operation your application performs. It should be noted that any settings made to HSMDecoder before activation has occurred will be overridden upon activation.

One-Time Remote Activation

In this license model ActivationManager activate must be called each time an application is first launched, but the API will only need to contact a remote licensing server once. Once a successful activation has occurred with the remote server, the API will no longer require internet access. However, the ActivationManager class also contains a method named deactivate which relinquishes the license back to the license server (requires internet connectivity). If this is called, the device will no longer decode barcodes until ActivationManager activate is called and the device can re-acquire a license from the remote license server. These two methods facilitate a floating licensing model should it be desired. This will only be supported if enabled by your license type.

Please ensure your application will have access to:

https://flex20004.flexnetoperations.com/ on TCP443

One-Time Local Server Activation

While one-time remote activation is the preferred licensing model, it may not be suitable for all use cases. If your use case will never have internet connectivity (not even once) then we provide a local license server instance that you can run on a dedicated Windows PC within your facility. You will be given a zip file containing a license server that your devices will need to reach to activate the SwiftDecoder software. The local server package will also include a IdentityClient.bin file. Like the one-time remote activation model, a device will only need connectivity to this server once. Within this zip file are instructions on how to install, run and provision this license server.

Refer to the API documentation in the SDK package for the function definition to pass a license key, server URL including port, and the contents of IdentityClient.bin (included in zip file).

HSMDecoder

This class is responsible for all decode related functions and configurations. HSM-Decoder is a singleton, meaning there only ever exists one instance which can be accessed anywhere within your application. The instance can be obtained by calling the static method [HSMDecoder.getInstance] and must be disposed of by calling the static method [HSMDecoder.disposeInstance] before the application is closed.

Objective-C Sample Code

```
- (void)viewDidLoad
{
    [super viewDidLoad];
    // intially set this launch type to -1

    launchType = -1;
    //init custom view controller containg the HSMDecodeComponent
    customViewController = [[MyDecodeComponent alloc] 
initWithNibName:@"MyDecodeComponent" bundle:nil];

    //activate the API with your license key (this is also available as an 
asynchronous method)
    //NOTE: any HSMDecoder settings made before activation will be defaulted once 
actvation has occurred!
    [self entitlementIDActivate:kEntitlementId];

}
-(void)loadView {
    [super loadView];
}
-(void)viewWillAppear:(BOOL)animated {
    [super viewWillAppear:animated];
}
#pragma mark - DecodeResult Listener Delegates

- (void) onHSMResult:(HSMDecodeResultArray*)barcodeData
{
    //update display on main thread
    [self performSelectorOnMainThread:@selector(updateDisplay:)
withObject:barcodeData waitUntilDone:YES];
}

#pragma mark - Activation Response Delegates

- (void) onActivationComplete:(ActivationResult)result
{
    NSString *msg = @"";

    if (result == SUCCESS) {
        msg = @"Device Activated!";
    } else {
        msg = [NSString stringWithFormat:@"Activation Failed. Error = %d",
result];
    }
    dispatch_async(dispatch_get_main_queue(), ^{

    // open a alert with an OK and cancel button
    UIAlertController *alert = [UIAlertController alertControllerWithTitle:@""
message:msg preferredStyle:UIAlertControllerStyleAlert];

    UIAlertAction *ok = [UIAlertAction actionWithTitle:@"OK"
style:UIAlertActionStyleDefault handler:^(UIAlertAction * _Nonnull action)
                {
                //get decoder singelton and configure as needed
                hsmDecoder = [HSMDecoder getInstance];
                [hsmDecoder enableSymbology:UPCA];
                [hsmDecoder enableSymbology:CODE128];
                [hsmDecoder enableSymbology:CODE39];
                [hsmDecoder enableSymbology:CODE93];
                [hsmDecoder enableSymbology:QR];
                [hsmDecoder enableSymbology:EAN13];

                [hsmDecoder setAimerColor:[UIColor redColor]];
                [hsmDecoder setOverlayTextColor:[UIColor whiteColor]];
                [hsmDecoder setOverlayText:@"Place barcode inside viewfinder"];
                if (result == SUCCESS) {
                    if (launchType == 0) {
                        [self onScanButtonPressed:nil];
                    } else if(launchType == 1) {
                        [self onButtonComponentPressed:nil];
                    }
                }

            }];
    [alert addAction:ok];
    [self presentViewController:alert animated:YES completion:nil];
    });

}

- (void)onDeactivationComplete:(ActivationResult)result {

}
```

```
#pragma mark - Action methods

- (IBAction)onScanButtonPressed:(id)sender
{
    launchType = 0;
    if ([ActivationManager isActivated]) {
        //enable default decoding functionality
        [hsmDecoder addOnResultListener:self];

        //launch the default barcode scanning view controller
        [hsmDecoder scanBarcode:self];
    } else {
        [self entitlementIDActivate:kEntitlementId];
    }
}


- (IBAction)onButtonComponentPressed:(id)sender
{
    launchType = 1;
    if ([ActivationManager isActivated]) {
        //disable default decoding functionality, as the view controller we are
about to show uses a custom Swift plugin instead of the default decoder
        [hsmDecoder removeOnResultListener:self];

        //launch a custom ViewController with an embedded HSMDecodeComponent
        [self presentViewController:customViewController animated:NO
completion:nil];
    } else {
        [self entitlementIDActivate:kEntitlementId];
    }
```

Swift Sample Code

```
override func viewDidLoad()
{
    super.viewDidLoad()
    // activate decoder
    self.entitlementIDActivate()
}

override func didReceiveMemoryWarning()
{
    super.didReceiveMemoryWarning()
    // Dispose of any resources that can be recreated.
}

// MARK: - Decode result listener
func onHSMResult(_ barcodeData: HSMDecodeResultArray!)
{
    DispatchQueue.main.async {
        // update display on main thread
        self.updateDisplay(barcodeData)
    }
}

// MARK: - Activation handler
func onActivationComplete(_ result: ActivationResult)
{
    var message : String
    if result == SUCCESS
    {
        message = "Device Activated!"
    } else
    {
        message = "Activation Failed. Error = \(result.rawValue)"
    }

    let alert: UIAlertController = UIAlertController.init(title: nil, message:
message, preferredStyle: UIAlertControllerStyle.alert)

    let okAction = UIAlertAction.init(title: "OK", style:
UIAlertActionStyle.default, handler: okActionHandler);
    alert.addAction(okAction)

    self.present(alert, animated: true,completion: nil)
}

// MARK: - IBAction methods
@IBAction func scanViewControllerButtonClicked(_ sender: Any)
{
    launchType = 0

    if ActivationManager.isActivated()
    {
        // enable default decoding functionality
        hsmDecoder?.add(on: self)
        // launch the default barcode scanning view controller
        hsmDecoder?.scanBarcode(self)
    } else
    {
        self.entitlementIDActivate()
    }
}

@IBAction func scanComponentButtonClicked(_ sender: Any)
{
    launchType = 1

    if ActivationManager.isActivated()
    {
        // disable default decoding functionality, as the view controller we are
about to show uses a custom Swift plugin instead of the default decoder
        hsmDecoder?.remove(on: self)

        // init custom view controller containg the HSMDecodeComponent
        let customViewController: MyDecodeComponent = MyDecodeComponent(nibName:
"MyDecodeComponent", bundle: nil)
        customViewController.modalPresentationStyle = .fullScreen;
        // launch a custom ViewController with an embedded HSMDecodeComponent
        self.present(customViewController, animated: true, completion: {})
    } else
    {
        self.entitlementIDActivate()
    }
}

// MARK: - Private methods

func entitlementIDActivate() {
    if ActivationManager.isActivated() == false
    {
        // activate the API with your licnese key
        ActivationManager.entitlementIdActivateAsync(kentitlementId, with: self)
    }
}
func onDeactivationComplete(_ result: ActivationResult) {
}

private func updateDisplay(_ barcodeData: HSMDecodeResultArray!)
{
   // get the first result in the list (there may be many)
   let firstResult: HSMDecodeResult = (barcodeData?.result(at: 0))!
   // update the screen with the last barcode data
   self.barcodeDataLabel.text = firstResult.barcodeData;
   self.symbologyLabel.text = "Symbology:
\(firstResult.symbology.description)";
   self.lengthLabel.text = "Length: \(firstResult.length)";
   self.decodeTimeLabel.text = "Decode Time: \(firstResult.decodeTime)";

   // display an image of the barcode
   self.barcodeImageView.image =
self.hsmDecoder?.getLastBarcodeImage(firstResult.bounds)
}

private func okActionHandler(action: UIAlertAction)
{
   hsmDecoder = HSMDecoder.getInstance()
   hsmDecoder?.enableSymbology(Int32(UPCA.rawValue))
   hsmDecoder?.enableSymbology(Int32(CODE128.rawValue))
   hsmDecoder?.enableSymbology(Int32(CODE39.rawValue))
   hsmDecoder?.enableSymbology(Int32(CODE93.rawValue))
   hsmDecoder?.enableSymbology(Int32(QR.rawValue))
   hsmDecoder?.enableSymbology(Int32(EAN13.rawValue))
   hsmDecoder?.setAimerColor(UIColor.red)
   hsmDecoder?.setOverlayText("Place barcode inside viewfinder")
   hsmDecoder?.setOverlayTextColor(UIColor.white)

   // check if decoder is activated
   if ActivationManager.isActivated()
   {
      if launchType == 0
      {
         self.scanViewControllerButtonClicked(scanViewControllerButton)
      }
      else if launchType == 1
      {
         self.scanComponentButtonClicked(scanComponentButton)
      }
   }
}
```

Required Core Frameworks and Dylibs

The following cocoa frameworks and dynamic library must be included in your application for the API to function:

- libz.dylib
- AudioToolbox.framework
- AVFoundation.framework
- CoreGraphics.framework
- CoreMedia.framework
- CoreVideo.framework

- Foundation.framework
- UIKit.framework

HSMDecodeComponent

An HSMDecodeComponent is a real-time camera preview frame layout that can be include in your own activity for greater control over the look and feel of the barcode scanning operation. This allows you to resize the camera preview as you see fit. This can be used as a replacement for the HSMCameraPreview activity that is launched by the [HSMDecoder scanBarcode] method. Using an HSMDecodeComponent embedded within your own view controller allows you to customize the look and feel of the barcode scanning operation.

Plugin Callbacks

```
//Called when the HSMCameraPreview or HSMDecodeComponent activity comes to the
foreground. This should be used to initialize the plugin on each scan attempt.
- (void)onStart;

//Called when the HSMCameraPreview or HSMDecodeComponent is no longer visible.
- (void)onStop;

//This is used to clean up plugin resources.
- (void)onDestroy;

//Called when a barcode(s) has been decoded in the image
- (void)onDecode:(HSMDecodeResultArray*)results;

//Called each time a frame cannot be successfully decoded
- (void)onDecodeFailed;

//Called each time an image is sent to the decoder
- (void)onImage:(unsigned char*)image Width:(int)width Height:(int)height;
```

Example

Please see the SDK Sample project in the SDK package for a plugin example.

Swift Plugins

A SwiftPlugin is a special “plugin” UIView that allows you to completely control the look and function of a barcode scanning operation. All plugins have their own UI that is rendered over the real-time camera preview. A SwiftPlugin must extend the SwiftPlugin base class, which contains many callback methods that are fired throughout the plugin life cycle.

A SwiftPlugin utilizes the observer pattern to notify any “observers” of a decode result (or any notification). Any observer of a SwiftPlugin must register itself as a result listener with the plugin. This is done by creating an interface that extends the PluginResultListener interface. A SwiftPlugin will be notified when a barcode is decoded, when the screen is drawn, when the screen is touched and many other times throughout the plugin lifecycle. This allows you to completely control the barcode scanning experience.

You can register your plugin with the system via the HSMDecoder registerPlugin() method where it will be run in both the default HSMCameraPreview activity, as well as within any HSMDecodeComponent.

Ready to Use Feature Plugins

The following are the set of ready to use feature plugins providing a specific set of functionality.

PreviewSelect Plugin

Used for rendering Augmented Reality Overlay over the detected barcodes within the preview. This plugin mainly allows a "Preview & Select" functionality along with capability to fetch the Preview Overlay count or Result data. The overlays on the preview allow the touch functionality to select one barcode. The application can register the DecodeResultListener call back using HSMDecoder::addResultListener API to have further action on the selected barcode.

BatchScan Plugin

Used for scanning a batch of barcodes, either predefined batch or batch count, until user intervention.

- Option one is with predefined batch limit which ensures that n number of bar codes are decoded before the results are returned to the business logic.
- Option two is to get the batch count would be the n number of barcodes until user intervention using API getScannedBatchResult or getScannedBatchCount.

SwiftFind Plugin

Used for rendering searching and finding barcodes within the preview through overlay indicators.

Windowing/Targeting Plugin

Used to dynamically set the decoding mode using a Window or Target. This plugin mainly allows configuring the "Windowing" /"Targeting" functionality. Setting of the windowing mode is possible through API setWindowMode on this plugin.

When set to Windowing Mode only barcodes within the given window will be decoded. The API enableTouchResizing on this plugin allows resizing of the window. Also it can be set using setWindow API.

When set to Targeting Mode the barcodes on which the target is aimed can be decoded. The size of the target can be set using API setTargetSize on this plugin.

The application can register DecodeResultListener call back using HSMDecoder::addResultListener API to have further action on the selected barcode. Use HSMDecoder::registerPlugin and HSMDecoder::unRegisterPlugin API to register/unregister any plugin on SwiftDecoder.

DLAgeVerificationPlugin

Used to verify a person's age by scanning the barcode on the back of a driver's license issued in the United States or Canada. This plugin allows SwiftDecoder to compare the age from the barcode record against a configured value and indicate if the license holder is above that age. SwiftDecoder displays a check-mark if the age of the ID holder is greater than or equal to the configured age or a cross mark if the age is less than the configured age. The application can register the DLResultListener to display the data from the barcode by tapping the check-mark or cross mark in the app.

PlugIn Details

This is the plugin used to set the age for verification.

```
DLAgeVerificationPlugin
```

Set Age Verification

The age limit can be passed with constructor of the plugin or using the following API:

```
-(void)setAgeForVerification:(int)age;
```

Register the Listener

This is the API used to register the DLResultListener to allow the information in the barcode to be displayed in SwiftDecoder when the user taps the check-mark or cross mark.

```
(void)setDLResultListener:(id<DLResultListener>)listener;
```

Stock Plugin

This plugin is an extension to Augmented Reality (AR) plugin. It is used to display the stock data returned by the customer as an overlay element.

Initialize Stock Plugin

```
//Create Instance of HSM Decoder
HSMDecoder* hsmDecoder;
hsmDecoder = [HSMDecoder getInstance];

//Create Instance of Stock Plugin
StockPlugin *stockPlugin;
stockPlugin = [[StockPlugin alloc] init];
```

Configuring the image to be used for display on Stock Plugin

```
[stockPlugin setStockIcon:[UIImage imageNamed:@"iconright"]];
```

Configuring Touch Listener

Argument passed to the below API implements -(void)onResultSelected:(NSDictionary *)arPluginResult for ARPluginTouchListener. Refer API documentation for more details

```
[stockPlugin addOnStockPluginTouchListener:self];
```

Register the plugin

This is the API used to register the plugin.

```
[hsmDecoder registerPlugin:stockPlugin];
```

OnStockResult

This is the API used to send the stock data associated with barcodes to display on the augmented reality user interface.

```
[stockPlugin onStockResult:data];
```

Result Format

```
[
  {
    "barcodeData1": [
      {
        "label1": "label1Value"
      },
      {
        "label2": ""
      },
      {
        "color": "#FF0000"
      }
    ]
  },
  {
    "barcodeData2": [
      {
        "label1": "label1Value"
      },
      {
        "label2": ""
      },
      {
        "color": "#FF0000"
      }
    ]
  }
]
```

Example

```
[
  {
    "12345": [
      {
        "Stock": "42"
      },
      {
        "date": 1651553823831
      },
      {
        "color": "#FF0000"
      }
    ]
  },
  {
    "1234567890": [
      {
        "Stock": "42"
      },
      {
        "date": 1651553823832
      },
      {
        "color": "#FF0000"
      }
    ]
  }

]
```

Freeze Frame Plugin

Freeze frame is a feature that is enabled in conjunction with preview and select mode. The feature allows the user to capture and image and provide AR overlay within the rendered screen.

Note: *Freeze Frame mode only works when in preview and select mode*

The following sequence is to be followed for using freeze frame with Preview and Select Mode

1. SDK provides an interface named freeze “**setFreezeMode(boolean mode)” on HSMDecodeComponent** to enable/disable the freeze mode.
2. This mode needs be enabled only if registered plugin is AugmentedRealityPlugin.
3. Once the integrating application enables freeze mode the decoding of frames is disabled at the back end. Hence the preview would still be running in freeze mode without decoding any of the frames.

4. On first double tap in the preview screen the captured image frame would be displayed in preview.
5. If the captured image has barcodes, the AR overlay would be rendered on the screen with bounding boxes around the barcodes.
6. User can select the required barcodes with single touch for getting the decoding result. The decoding result is returned as part of **onHSMDecodeResult** implemented by integrating application.
7. With double tap on frozen image the preview screen would continue to capture the video frames.
8. **FreezeFrameListener** call backs to be implemented by integrating application to get the notification for **onFreezeFrame**(first double tap) and **onUnFreezeFrame**(second double tap). Use API **registerFreezeFrameListener(FreezeFrameListener listener)** on HSMDecoder for registering the listener.
9. To know if currently freeze mode is enabled or not use API **isFreezeMode** exposed on HSMDecoder.
10. While changing the freeze mode dispose the old AR plugin first and then register a new AR plugin.
11. In case integrating application needs this feature in full screen mode then can use API **scanBarcodeInFreezeMode** exposed on HSMDecoder.

Note: *For more hands on to understand this feature check the demo using Honeywell Barcode Scanner app available in App store.*

Parsers

Parsers are separate features that can be enabled within your application to return the parsed data for the raw decoded data. These features are charged separately from the original standard features. This can be added on at any time. To decode motor vehicles, boarding passes, and driver’s license, please ensure that PDF417 barcode symbology is enabled.

Motor Vehicles (EZMV)

Easy Motor Vehicles feature allows to parse the raw decoded data from vehicle documents with PDF417 Barcode. The supported vehicle documents include Title(TD) and Registration(RG).

Refer to the MotorVehicleParser class documentation in the SDK package for more details on API arguments and return values. The MotorVehicleData class documentation describes the list of parsed data fields for each document type.

Sample code:

```
MotorVehicleData *ezmvdata = [MotorVehicleParser parseRawData:[barcodeData
resultAtIndex:0].barcodeByteData];
if(ezmvdata != NULL){
    //Add the application logic for using the returned parsed data object
}
```

Boarding Passes (EZBP)

Easy Boarding Pass feature allows to parse the raw decoded data from boarding passes with PDF417 symbology.

Refer to the BoardingPassParser class documentation in the SDK package for more details on API arguments and return values. The BoardingPassData class documentation describes the list of parsed data fields.

Sample code:

```
BoardingPassData *bcbpData = [BoardingPassParser parseRawData:[barcodeData
resultAtIndex:0].barcodeByteData];
if(bcbpData!= NULL){
    //Add the application logic for using the returned parsed data object
}
```

Driver's License (EZDL)

Easy Driver's License feature allows to parse the raw decoded data from Driver's Licenses with PDF417 symbology.

Refer to the LicenseParser class documentation in the SDK package for more details on API arguments and return values . The LicenseData class documentations describes the list of parsed data fields.

Sample Code:

```
LicenseData* licenseData = [LicenseParser parseRawData:[barcodeData
resultAtIndex:0].barcodeByteData];
if(licenseData != NULL){
    //Add the application logic for using the returned parsed data object
}
```

Machine Readable Zone (MRZ)

MRZ parser feature allows to parse the raw decoded data for OCR symbology with template passport enabled.

Refer the MRZParser class documentation in the SDK package for more details on API arguments and return values. The MRZData class documentations describes the list of parsed data fields.

Sample Code:

```
MRZData *mrzData = [MRZParser parseRawData:[barcodeData
resultAtIndex:0].barcodeData];

if(mrzData!= NULL){
    //Add the application logic for using the returned parsed data object
}
```

OCR-A and OCR-B Font Detection

Optical Character Recognition gives the ability to read OCRA and OCRB fonts. OCR can parse Price Field, ISBN, Passport, and MICR characters. Please refer to the OCR Programming User's Guide (OCR_Honeywell.pdf) document for more information. Custom template information can also be found in the OCR Programming User's Guide.

Sample Code:

```
switch (IOCRSelector) {
    case PASSPORT:
        if ([MRZParser isLicenseEnabled]) {
            [hsmDecoder addOnResultListener:self];
            [hsmDecoder setOCRActiveTemplate:OCR_PASSPORT];
        } else {
            [self testAlert:@"Error" withMsg:@"License not enabled for MRZ."
withStyle:UIAlertControllerStyleAlert];
        }
        break;
    case ISBN:
            [hsmDecoder addOnResultListener:self];
            [hsmDecoder setOCRActiveTemplate:OCR_ISBN];
        break;
    case PRICEFIELD:
            [hsmDecoder addOnResultListener:self];
            [hsmDecoder setOCRActiveTemplate:OCR_PRICE_FIELD];
        break;
    case MICR:
            [hsmDecoder addOnResultListener:self];
            [hsmDecoder setOCRActiveTemplate:OCR_MICR];
        break;
    case USER:
            hsmDecoder addOnResultListener:self];
            [hsmDecoder setOCRActiveTemplate:OCR_USER];
            NSString *userTemplateText = [NSString
stringWithFormat:@"%@",userOCRTemplate];
            [hsmDecoder setOCRUserTemplate:[userTemplateText UTF8String]];
            [self testAlert:@"Enter OCRUserTemplate" withMsg:@""
withStyle:UIAlertControllerStyleAlert];
            UIAlertController *alert = [UIAlertController
alertControllerWithTitle:@"Enter OCRUserTemplate" message:@""
preferredStyle:UIAlertControllerStyleAlert];
            [alert addTextFieldWithConfigurationHandler:^(UITextField * _Nonnull
userTemplate){
                userTemplate.text = userTemplateText;
            }];
```

```
          UIAlertAction *confirmAction = [UIAlertAction actionWithTitle:@"OK"
style:UIAlertActionStyleDefault handler:^(UIAlertAction * _Nonnull action) {
              proxyServerAddress = alert.textFields.firstObject.text;
              [hsmDecoder scanBarcode:self];
          }];
          [alert addAction:confirmAction];
          [self presentViewController:alert animated:YES completion:nil];
      break;
      }
      }
[hsmDecoder scanBarcode:self];
```

SwiftOCR

Swift OCR is a new separate xcframework from 6.0 release, dependent on the core xcframework mainly allowing OCR functionality.

Below is sample reference for enabling this feature.

Detection mode needs to be set prior to enabling the feature.

```
[[SwiftOCRDecoder getInstance] setSwiftOCRDetectionMode:TEMPLATE_OCR];
[[SwiftOCRDecoder getInstance] enableSwiftOCRFeature:YES];
```

Template OCR

Template OCR is a feature that allows a users to parse important OCR data on user supplied labels. This is done by passing the template information associated with the label to the SDK. This can be useful in a variety of manners when trying to cross reference information between a barcode and what is printed on the labels/tags. Once user provides a label or tag, Honeywell can generate custom templates provided they meet the following guidelines,

1. There must be a minimum of one barcode on the label
2. Only labels with English characters
3. Apart from English characters, ASCII characters are supported as well

An example of where you might want to use it would be when checking the pricing printed on the shelves with the user data base. By using Templat OCR, you can extract the barcode information as well as other information printed on the users tag for price checking. You can use this additional information for a multitude of functions, like price checking.

Interfaces

Before starting to use these interfaces, customers should have submitted their example labels to Honeywell SA along with details of the important OCR detection for Template generation. Please refer to the template generation workflow in below section for more details. Customers developing the APP need to store these QR code template files shared by Honeywell within the application package. Templates can be tested using the Honeywell Scanning Demo Application to verify the outputs.

The Interfaces can be grouped into the following categories:

- Configuring the templates
- Activating the template
- Getting the OCR result

Template Configurations

Templates can be configured one time during the initialization. A maximum of 40 templates can be configured by a customer application.

Interface	Input	Output
`addOCRTemplate`	bye[] (Android / NSData* (IOS) QRCode Template file data as an array	int - <templateID> : Template ID starting from 1 if template added successfully. -2 : If template was laready added and is a duplicate -3 : If template count goes more than 40 -1 : Any generic error in template addition
`removeOCRTemplateID`	int The ID of Template that was already added	int Input Template ID on SUCCESS of removal -1 ERROR while removing the template ID

Activating the template

Of the 40 templates, only one can be active at a time. Before setting the template ID as active there are 2 API's that need to be used. Follow the sequence below.

	Interface	Input	Output
1	`enableSymbology`	int Symbology.OCR	boolean True if enable SUCCESS False if enable ERROR
2	`setOCRActiveTemplate`	int ADVANCED_TEMPLATE as defined in OCRActiveTemplate	void

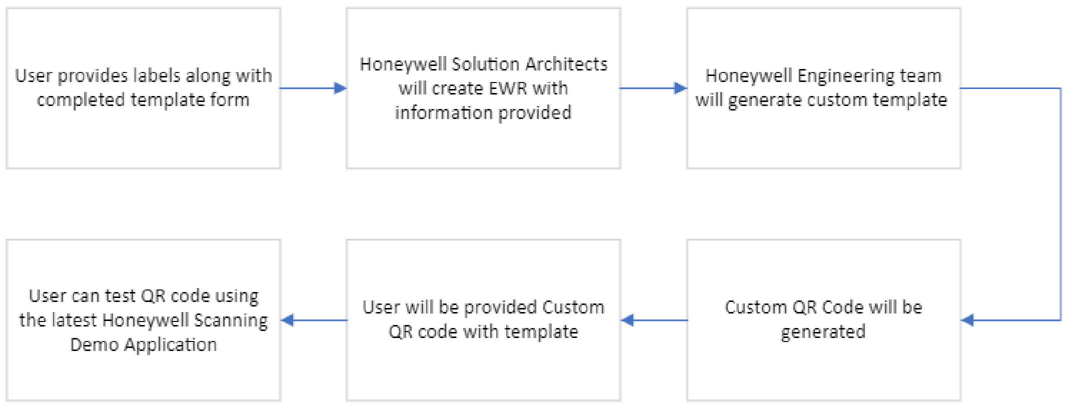
3	`setOCRActiveTemplateID`
	
	`Can be used at runtime to set the active template ID with above 2 APIs already set.`
		int
		
		The intially configured Template ID that needs to be active
			int
			Input template ID that was passed for setting active if SUCCESS
			-1 if ADVANCED_TEMPLATE is not set as an active template
			-2 If input Template ID is not a Valid ID

Getting the OCR Result

Since this OCR detection is referenced to a barcode within a given label, this protocol must be implemented by any class that wishes to receive SwiftOCR results. The result be available on callback onSwiftOCRTemplateResult.

```
@protocol SwiftOCRResultListener <PluginResultListener>
- (void) onSwiftOCRResult:(NSArray*)ocrData;
- (void) onSwiftOCRTemplateResult:(NSArray*)ocrData;
@end
```

Template Generation Process


User provides labels along with completed template form
Honeywell Solution Architects will create EWR with information provided
Honeywell Engineering team will generate custom template
Custom QR Code will be generated
User will be provided Custom QR code with template
User can test QR code using the latest Honeywell Scanning Demo Application

OPEN OCR

Open OCR detection mode allows to detect any general OCR text without any barcode reference as compared to template OCR.

There are 2 options for this mode:

- TARGETED_SINGLE_ROI
- FULL PREVIEW (needs to be set based on the requirement)

TARGETED_SINGLE_ROI

Provides a targeted window within the preview to scan any targeted text.

```
[[SwiftOCRDecoder getInstance] setSwiftOCRScanArea:TARGETED_SINGLE_ROI]
```

TARGETED_SINGLE_ROI does allow some more configuration of the targeted window size, orientation etc. refer the API document for more details.

FULL PREVIEW

Allows you to scan the text within the given preview window.

```
[[SwiftOCRDecoder getInstance] setSwiftOCRScanArea:FULL_PREVIEW]
```

This protocol must be implemented by any class that wishes to receive SwiftOCR results:

```
@protocol SwiftOCRResultListener <PluginResultListener>
- (void) onSwiftOCRResult:(NSArray*)ocrData;
- (void) onSwiftOCRTemplateResult:(NSArray*)ocrData;
@end
```

International DL Scan

The International DL scan feature allows you to scan the front of a driver's license which is mainly OCR for most of the countries except for the US which uses both barcode and OCR. The feature is available on DLEngine.xcframework.

Refer the SampleDLProject provided along with xcframework for quick start on the API's to be used for integrating this feature.

For enabling DL scan:

```
DLScanType scanType = FRONT_SCAN;
DLDecoderStatus status =   [[DLDecoder getInstance] setDLScanFeature:YES 
withScanType:scanType];
```

The protocol DriversLicenseResultListener must be implemented by any class that wishes to receive DL results.

```
@protocol DriversLicenseResultListener <PluginResultListener>
- (void) onDLDecodeResult:(LicenseData*)licenseData;
- (void) onDLDecodeError:(DLDecoderErrorCode) error;

@end
```

Honeywell
855 S. Mint St.
Charlotte, NC 28202

automation.honeywell.com