EVO Snap

CommerceDriver™

Quick-Start Guide for *iOS*®

EVO CommerceDriver™

Adding EMV transaction processing to your POS system is easy with the pre-certified EVO CommerceDriver™ SDK. The pre-certified CommerceDriver™ SDK installs alongside your software application to add EMV transaction processing to your POS system. CommerceDriver™ facilitates all transactional communication with the EVO Payments International global processing platforms and approved hardware devices to isolate payment data and keep it separate from the software application.

CommerceDriver™ is designed to support multiple terminal manufacturers while retaining a common API. At startup, CommerceDriver™ detects the supported terminal manufacturer(s)/models for processing Authorize, Authorize & Capture, and Return transactions.

How It Works

1. Create transaction data objects in your POS.
2. Pass the transaction data to CommerceDriver™.
3. CommerceDriver™ initiates terminal commands and gathers tender/EMV data to send to the EVO Snap* Platform.
4. The EVO Snap* Platform returns a response to CommerceDriver™ with receipt details.

Version Details

* CommerceDriver™ - v2.29.0
* Supports EVO Snap* v2.1.29 Platform calls
* Supported Terminals:
    - EVO ITM-100 via Audio Jack
    - Ingenico iCMP via Bluetooth
    - Ingenico iPP320/350 via Ethernet
    - Magtek iDynamo via Lightning Connector
    - Magtek uDynamo via Audio Jack

Compatibility

* CommerceDriver™ Framework – iOS 8.0 & Higher using Objective-C
* Sample Code, Projects, & Guides – Created using xCode 10.1 & iOS 9+

Integration

To get started with CommerceDriver™, select your Platform, Network, and Hardware. The setup is similar to a direct Web Services integration, but CommerceDriver™ must be hosted locally.

1. Drag and drop the framework files provided by your EVO Snap* Support Engineer into the Embedded Binaries section of your iOS project target.

2. Add the Import statement to the classes using the CommerceDriver™ framework.

```
#import <EVOCommerceDriver/EVOCommerceDriver.h>
```

3. Initialize the `EVOCommerceDriverAPI` object with your Service Key and Application Profile ID.

```
    NSString * sericeKey = @"<YOUR SERVICE KEY>";
    NSString * applicationProfileId = @"<YOUR APPLICATION PROFILE ID>";

    [[EVOCommerceDriverAPI alloc] initWithServiceKey:serviceKey
applicationProfileId:applicationProfileId];
```

4. Optionally, set the CommerceDriver™ logging level.

```
    [commerceDriverAPI setLogLevel:EVOLogLevelDebug];
```

Authentication

After initializing your instance of `EVOCommerceDriverAPI` with the `EVOPlatformConfiguration` settings, you are required to authenticate to the platform with your Username and Password.

1. Log into the Platform by calling the `loginUser:password:completion` method `EVOCommerceDriverAPI`.

```
[commerceDriverAPI loginUser:username password:password completion:^(BOOL success,
EVOIdentityLoginState state, NSString *message) {
        //The successful flag can be used to determine if login succeeded.
        if (success ) {
            NSLog(@"Logged in successfully with message: %@", message);
        } else {
            ///If login did not succeed, then check the state property to determine the next action.
            switch (state) {
                case EVOIdentityLoginStateSuccessMessage:
                    /// Logged in successfully
                    /// Continue normally.

                    break;
                case EVOIdentityLoginStateInvalidCredentialsMessage:
                    ///Login with the supplied credentials failed.
                    ///Prompt the user to try again.
                    break;
                case EVOIdentityLoginStateRequiredFieldsMessage:
                    ///There was a validation error with the data passed to the login call.
                    ///Display the error message to the user and let them retry.
                    NSLog(@"login message: %@", message);
                    break;
                case EVOIdentityLoginStatePasswordChangeRequired:
                    ///Indicates that the user must change their password before proceeding.

                    break;
                case EVOIdentityLoginStateAccountLocked:
                    ///Indicates that the account is locked and the user should be directed to  perform
a forgot password to unock.
                    break;
                case EVOIdentityLoginStateAccountLockedAdmin:
                    //The account has been locked by the EVO Snap service.  It can only be unlocked by
contacting support.
                    NSLog(@"login message: %@", message);
                    break;
                case EVOIdentityLoginStateServiceErrorMessage:
                    ///The service returned an error message.
                    ///Display the error message to the user.

                    NSLog(@"login message: %@", message);

                    break;

                default:
                    break;
            }
        }
```

Terminal Setup

CommerceDriver™ supports multiple terminal manufacturer families through individual frameworks. Choose the terminal(s) your organization would like to support by including the related framework, create the associated `EVOTerminal` object and add it to the `EVOCommerceDriverAPI` object.

A minimum of one terminal is required to perform the following activities:

* Authorize
* Authorize & Capture
* Return Unlinked

Supported Terminals

CommerceDriver™ for iOS currently supports the following devices.

* **EVO ITM-100** – requires the `EVOIntegratedTerminals.framework` library
* **Ingenico iCMP** & **iPP320/350** – require the `EVOIngenicoTerminals.framework` library
* **Magtek iDynamo/uDynamo** – require the `EVOMagtekTerminals.framework` library.

Terminal Integration

To **Setup** your device:

1. Drag and drop the EVO CommerceDriver™ framework files provided by EVO Snap* Support Engineer into the Embedded Binaries section of your iOS project target.

2. For the **EVO *ITM-100*** device, add the following Import statements to the classes using the `EVOIntegratedTerminals.framework`.

```
#import <EVOCommerceDriver/EVOCommerceDriver.h>
#import <EVOIntegratedTerminals/EVOIntegratedTerminals.h>
```

***Note**: the ITM100 also requires you grant access to the audio jack in your application. In your Info.plist file add "Privacy - Microphone Usage Description" key. The string value can be any string, e.g. "ITM100 Access to audio jack".*

3. For the **Ingenico** library, add the following Import statements to the classes using the `EVOIngenicoTerminals.framework`.

```
#import <EVOCommerceDriver/EVOCommerceDriver.h>
#import <EVOIngenicoTerminals/EVOIngenicoTerminals.h>
```

***Note**: when using the ICMP device, the following changes must be made to your XCode project:*

- *Add the `ExternalAccessory.Framework` to your project*
- *To your info.plist, add the "Supported external accessory protocols" array key to your project and an item to that array with the value "com.ingenico.easypayemv.spm-transaction"*

4. For the **Magtek** library, add the following Import statements to the classes using the `EVOMagtekTerminals.framework`.

```
    #import <EVOCommerceDriver/EVOCommerceDriver.h>
    #import <EVOMagtekTerminals/EVOMagtekTerminals.h>
```

Note: When using the iDynamo device, the following changes must be made to your XCode project:

- *Add the `ExternalAccessory.Framework`*
- *In your `info.plist` add the 'Supported external accessory protocols' array key and then add an item to the array with the value 'com.ingenico.easypayemv.spm-transaction'*

For the uDynamo, grant access to the audio jack in your application and in the `info.plist`, add the 'Privacy – Microphone Usage Description' key with any string value, (e.g.: 'uDynamo access to audio jack').

Registering Terminals

To **Register** your device for support:

1. Create the related terminal object and add the object to the `EVOCommerceDriverAPI`.

EVO ITM-100

Sample – Create an ITM-100 object using the audio jack interface.

```
//Create an ITM-100 terminal using the audio jack.
    EVOITM100Terminal * itm100 = [EVOITM100Terminal createTerminalWithIdentifier:@"ITM100"];

    //Add that terminal to the CommerceDriver object.
    [commerceDriverAPI addTerminal:itm100];

    //Tell CommerceDriver which device you want to use.
    //Note:  When dealing with one terminal, you do not need to make this call as
CommerceDriver will use the device automatically.
    [commerceDriverAPI selectTerminal:itm100];
```

***Note**: Only one ITM-100 device can be added to the commerceDriverAPI.*

Ingenico iCMP

Sample – Create an iCMP Terminal w/First Available Paired Device

```
//You first need a reference to your configured EVOCommerceDriverAPI object.
    EVOCommerceDriverAPI *commerceDriverAPI = [self getCommerceDriverObject];

    //Create an Ingenico ICMP terminal using the first available terminal that is paired
with your iOS Device.
    //The Identifier parameter is you own unique identifier for the terminal.
    EVOTerminal * icmp = [EVOIngenicoICMPTerminal createTerminalWithIdentifier:@"Paired-
ICMP"];


    //Add that terminal to the CommerceDriver object.
    [commerceDriverAPI addTerminal:icmp];

    //Tell CommerceDriver which device you want to use.
    //Note:  When dealing with one terminal, you do not need to make this call as
CommerceDriver will use the device automatically.
    [commerceDriverAPI selectTerminal:icmp];
```

Sample – Create an iCMP Object Referencing a Specific iCMP Device

```
    //Get a reference to your configured EVOCommerceDriverAPI object.
    EVOCommerceDriverAPI *commerceDriverAPI = [self getCommerceDriverObject];

    //Create an Ingenico ICMP terminal using the first available terminal that is paired
with your iOS Device.
    //The Identifier parameter is you own unique identifier for the terminal.
    EVOTerminal * icmp = [EVOIngenicoICMPTerminal  createTerminalWithAccessoryName:@"ICM122"
serialNumber:@"20552624" identifier:@"20552624"];


    //Add that terminal to the CommerceDriver object.
    [commerceDriverAPI addTerminal:icmp];

    //Tell CommerceDriver which device you want to use.
    //Note:  When dealing with one terminal, you do not need to make this call as
CommerceDriver will use the device automatically.
    [commerceDriverAPI selectTerminal:icmp];
```

Ingenico iPP320/350

Sample – Create an iPP320/350 Object Using an Ethernet Connection

```
//Create an IPP320 or IPP350 device using the ethernet interface.
    //Note: You will need to use the IPAddress and port specific to your IPP3XX device.
    EVOTerminal * ipp3xx = [EVOIngenicoIPP3XXTerminal
createIPP320TerminalWithIPAddress:@"192.168.1.147" port:@"12000" identifier:@"IPP350"];

    //Add that terminal to the CommerceDriver object.
    [commerceDriverAPI addTerminal:ipp3xx];

    //Tell CommerceDriver which device you want to use.
    //Note:  When dealing with one terminal, you do not need to make this call as
CommerceDriver will use the device automatically.
    [commerceDriverAPI selectTerminal:ipp3xx];
```

Magtek iDynamo

Sample – Create an iDynamo Object Using a Lightning Connection

```
//Create an Magtek iDynamo terminal using the lightning bolt port.
EVOTerminal * iDynamo = [EVOIDynamoTerminal createTerminalWithIdentifier:@"iDynamo"];

//Add that terminal to the CommerceDriver object.
[commerceDriverAPI addTerminal:iDynamo];

//Tell CommerceDriver which device you want to use.
//Note:  When dealing with one terminal, you do not need to make this call as CommerceDriver
will use the device automatically.
[commerceDriverAPI selectTerminal:iDynamo];
```

Magtek uDynamo

Sample – Create a uDynamo Object Using the Audio Jack

```
//Create an Magtek uDynamo terminal using the audio jack.
    EVOTerminal * uDynamo = [EVOUDynamoTerminal createTerminalWithIdentifier:@"uDynamo"];

    //Add that terminal to the CommerceDriver object.
    [commerceDriverAPI addTerminal:uDynamo];

    //Tell CommerceDriver which device you want to use.
    //Note:  When dealing with one terminal, you do not need to make this call as
CommerceDriver will use the device automatically.
    [commerceDriverAPI selectTerminal:uDynamo];
```

Transaction Processing

Two transaction sets can be processed using CommerceDriver™:

1. Terminal Required Transactions
   * Authorize
   * Authorize and Capture
   * Return Unlinked
2. No Terminal Required Transactions
   * Undo
   * Capture
   * Return by ID

Creating a POS Transaction Request

To **Start** a transaction:

1. Create an `EVOPOSTTransactionRequest`:

   ***Note**: Use the 'create' factory methods to create various transaction request types.*

```
    /** Use this factory method to create an EVOPOSOperationAuthorizeAndCapture with a
tenderType of EVOPOSTenderTypeCredit.*/
    + (instancetype)createAuthorizeAndCaptureRequestAmount:(NSDecimalNumber *)amount
employeeId:(NSString *)employeeId laneId:(NSString *)laneId orderNumber:(NSString
*)orderNumber reference:(NSString *)reference tipAmount:(NSDecimalNumber *)tipAmount
cashbackAmount:(NSDecimalNumber *)cashbackAmount overrideApDupe:(BOOL)overrideApDupe;

    /** Use this factory method to create an EVOPOSOperationAuthorizeAndCapture with a
specified tenderType.*/
    + (instancetype)createAuthorizeAndCaptureRequestAmount:(NSDecimalNumber *)amount
employeeId:(NSString *)employeeId laneId:(NSString *)laneId orderNumber:(NSString
*)orderNumber reference:(NSString *)reference tipAmount:(NSDecimalNumber *)tipAmount
cashbackAmount:(NSDecimalNumber *)cashbackAmount overrideApDupe:(BOOL)overrideApDupe
tenderType:(EVOPOSTenderType)tenderType;

    /** Use this factory method to create an EVOPOSOperationAuthorize.*/
    + (instancetype)createAuthorizeRequestAmount:(NSDecimalNumber *)amount
employeeId:(NSString *)employeeId laneId:(NSString *)laneId orderNumber:(NSString
*)orderNumber reference:(NSString *)reference tipAmount:(NSDecimalNumber *)tipAmount
cashbackAmount:(NSDecimalNumber *)cashbackAmount overrideApDupe:(BOOL)overrideApDupe;

    /** Use this factory method to create an EVOPOSOperationReturnUnlinked.*/
    + (instancetype)createReturnUnlinkedRequestAmount:(NSDecimalNumber *)amount
employeeId:(NSString *)employeeId laneId:(NSString *)laneId orderNumber:(NSString
*)orderNumber reference:(NSString *)reference tipAmount:(NSDecimalNumber *)tipAmount
cashbackAmount:(NSDecimalNumber *)cashbackAmount overrideApDupe:(BOOL)overrideApDupe;

    /* Use this factory method to create an Undo Request */
    + (instancetype) createUndoRequestTransactionID:(NSString *)transactionID;

    /* Use this factory method to create a Capture Request without a tip.*/
    + (instancetype) createCaptureRequestTransactionID:(NSString *)transactionID
amount:(NSDecimalNumber *)amount;

    /* Use this factory method to create a Capture request with a tip. */
    + (instancetype) createCaptureRequestTransactionID:(NSString *)transactionID
amount:(NSDecimalNumber *)amount tipAmount:(NSDecimalNumber *)tipAmount;

    /* Use this factory method to create a Return with a TransactionID */
    + (instancetype) createReturnRequestTransactionID:(NSString *)transactionID
amount:(NSDecimalNumber *)amount;

    /** Use this factory method to create a resubmit request.*/
    + (instancetype) createResubmitRequestTransactionId:(NSString *)transactionID
amount:(NSDecimalNumber *)amount tipAmount:(NSDecimalNumber *)tipAmount;
```

2. Once the POS Request object is created, call the `processTransactionRequest:` method from the `EVOCommerceDriverAPI` object:

```
    [commerceDriverAPI processTransactionRequest:authAndCaptureRequest];
```

To **Cancel** a Request:

1. Call `cancelAsyncProcess:`

```
[commerceDriverAPI cancelAsyncProcess:authAndCaptureRequest];
```

POS Transaction Request Delegate

The delegate of the `EVOPOSTransactionRequest` must adopt the `EVOPOSTransactionRequestDelegate` protocol. The delegate is used to communicate transaction and terminal statuses. It also uses this delegate to request data that is needed from the POS operator or the customer during a transaction.

After creating an `EVOPOSTransactionRequest`, set the delegate property to a class that implements the `EVOPOSTransactionRequestDelegate` protocol.

The protocols require implementation of the following methods:

1.
```
-(void)request:(EVOPOSTransactionRequest *)request failedToStartWithErrors:(NSDictionary *)errors ;
```
Called when a transaction can not be started. This method is called when there are problems connecting to the terminal or the transaction data passed do not meet basic validation tests. Check the errors dictionary for the specific reason for the failure.

2.
```
-(void)request:(EVOPOSTransactionRequest *)request cardReaderStatusUpdate:
(EVOCardReaderState)status;
```
Notifies the delegate of the the current type of card read that is in progress. This method is only called for terminals without a display. Upon receiving this method call, the POS must display the proper instructions to the customer to present their card.

3.
```
-(void)request:(EVOPOSTransactionRequest *)request selectApplication:(NSArray *)applicationList
completion:(void(^)(int arrayIndex))completion;
```
The delegate will receive this method call under two conditions:

   * The terminal does not have a display.
   * The card used has multiple payment applications.

   Upon receiving this method call, a interface must be shown that displays each item in the `applicationList` and provides the customer with the ability to select one of the applications to use for payment. After the user makes a seleaction, call the completion block passing the array index of the selected application.

4.
```
-(void)request:(EVOPOSTransactionRequest *)request confirmCardRemoved:(void(^)())completion;
```

   On terminals without a display, this delegate method is called when there is a problem reading an EMV card and the card reader needs to restart. After receiving this method call, the POS operator should be prompted to confirm that the card has been removed. Once that card is removed, call the completion block and transaction processing will continue.

5.
```
-(void)request:(EVOPOSTransactionRequest *)request confirmTransactionAmount:(NSDecimalNumber
*)amount completion:(void(^)(BOOL amountConfirmed))completion;
```

   On terminals without a display, the amount confirmation must happen on the POS. Upon receiving this delegate method, display a UI showing the amount and two options to either accpet or reject the amount. If the amount is accepted, call the completion block with `amountConfirmed` = YES. If the amount is rejected, call the completion block with `amountConfirmed` = NO.

6.
```
-(void)getSignatureForRequest:(EVOPOSTransactionRequest *)request
withResponse:(EVOTransactionResponse *)response completion:(void(^)(BOOL
signatureAccepted))completion;
```

   Called when validation of a signature is needed.

7.
```
-(void)request:(EVOPOSTransactionRequest *)request completedWithResponse:(EVOTransactionResponse
*)response;
```

   Called upon completion of a transaction.

8.
```
-(void)request:(EVOPOSTransactionRequest *)request getCVV:(void (^)(NSString *cvvCode))completion;
```

   Called when running an Amex MSR transaction which requires the CVV code from the back of the card.

Frameworks

CommerceDriver™ for iOS consists of the following frameworks:

* `EVOCommerceDriver.framework` - The core framework providing all CommerceDriver™ functionality. This framework is required.

* `EVOIntegratedTerminals.framework` - This framework provides the terminal implementation for all EVO payment terminals supported by CommerceDriver™.
* `EVOIngenicoTerminals.framework` - This framework provides the terminal implementation for all Ingenico payment terminals supported by CommerceDriver™.
* `EVOMagtekTerminals.framework` - This framework provides the terminal implementation for all Magtek payment terminals supported by CommerceDriver™.

Reference Information

For additional information, please visit the EVO Snap* Support site at http://www.evosnap.com/support/ or contact your EVO Technical Support representative.