EVO® Snap

CommerceDriver™

Quick-Start Guide for *iOS*®

EVO CommerceDriver™

Adding credit and debit card 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 credit and debit card 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 process transactions using one of our multiple supported terminal manufacturers or terminal-not-present solutions while retaining a common easy to use API.

How It Works

1. Create transaction data objects in your POS.
2. Pass the transaction data to CommerceDriver™.
3. CommerceDriver™ gathers card data by initiating terminal commands or prompting the user in order to send to the EVO Snap* Platform.
4. The EVO Snap* Platform returns a response to CommerceDriver™ with receipt details.

Version Details

- CommerceDriver™ - v2.34.X
- Supports EVO Snap* v2.1.34 Platform calls
- Supported Terminals:
  - BBPOS Chipper BT via Bluetooth
  - BBPOS Chipper OTA 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 10.0 & Higher using Objective-C

* Sample Code, Projects, & Guides – Created using xCode 11.3 & iOS 10.2+

Integration

To get started with CommerceDriver™, select your Platform, Network, and Hardware.

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 as well as manual card entry in a terminal-not-present solution. 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.

Supported Terminals

CommerceDriver™ for iOS currently supports the following devices.

* **BBPOS Chipper OTA** & **BBPOS Chipper BT** - require the `EVOIntegratedTerminals.framework` library.
* **Magtek iDynamo/uDynamo** – require the `EVOMagtekTerminals.framework` li 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 **BBPOS Chipper OTA** device, add the following Import statements to the classes using the `EVOIntegratedTerminals.framework`.

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

__Note__: the Chipper OTA also requires the following changes to your XCode project:

* *Grant access to the audio jack in your application by adding the "NSMicrophoneUsageDescription " key in your Info.plist file. The string value can be any string, e.g. "Chipper OTA Access to audio jack".*
* *Add the UIBackgroundModes key with an item for "audio" to prevent the interruption of a transaction being processed.*

3. For the **BBPOS Chipper BT** device, add the following Import statements to the classes using the `EVOIntegratedTerminals.framework`.

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

***Note**: when using the Chipper BT device, the following changes must be made to your XCode project's info.plist:*

- *Add the NSBluetoothPeripheralUsageDescription key with a string similar to "Bluetooth is used to operate the credit card reader."*
- *Add the NSBluetoothAlwaysUsageDescription key with a string similar to "Bluetooth is used tooperate the credit card reader."*
- *Add the UIBackgroundModes key with an item for "bluetooth-central".*

4. 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"*

5. 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.magtek.idynamo".*

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`.

BBPOS Chipper OTA

Sample – Create an Chipper OTA object using the audio jack interface.

```
//Create an CHIPPER OTA terminal using the audio jack.
    EVOChipperOTATerminal * chipperOTA = [EVOChipperOTATerminal
createTerminalWithIdentifier:@"ChipperOTA"];

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

    //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:chipperOTA];
```

***Note**: Only one Chipper OTA device can be added to the commerceDriverAPI.*

BBPOS Chipper BT

Sample – Create a Chipper BT Object Using Bluetooth

```
//Create an Chipper BT terminal using Bluetooth
     EVOTerminal * chipperBTTerminal = [EVOChipperBTTerminal
createTerminalWithIdentifier:identifier];

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

    //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: chipperBTTerminal];
```

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];
```

Removing a Registered Terminal

In v2.33 a completion block was added to the removeTerminal method in EVOCommerceDriverAPI. Upon removing a terminal this completion block will run:

```
/**
 * Remove a registered terminal.
 *
 * @param terminal The instance of the terminal you would like to remove.
 * @param completion This block is called once the terminal is removed.
 */
- (void)removeTerminal:(EVOTerminal *)terminal completion:(void(^)())completion;
```

Terminal Service Management (TSM)

The `InitializeTerminal:` method of the Commerce Driver object now provides information if an update is available for the terminal currently in use. **Users must be signed on to their instance of CommerceDriver™ in order for the initialize terminal process to begin and for the terminal to begin checking for updates.** This sign on procedure can be found for each operating system in their respective Quick Start Guides. After performing the steps to authenticate and add a terminal, check the response from the `InitializeTerminal:` method to determine if updates are available.

Code Snippets

To initialize a terminal in Objective-C, call the `InitializeTerminal:` method in the `CommerceDriverAPI` method. This method runs asynchronously and returns an `EVOInitializeTerminalResult` object, which contains an instance of `EVOTerminalUpdate`.

Example code to call and handle the completion of the `InitializeTerminal:` method can be found below:

```
[self.commerceDriverAPI initializeTerminal:^(EVOInitializeTerminalResult *response) {

        EVOTerminalUpdate * updateResponse = response.updateResponse;

        if (updateResponse.hasUpdates) {
            NSString * title = @"Terminal has Updates";

            NSString * message = [NSString stringWithFormat:@"Please install the terminal update by
%@", updateResponse.updateDeadline] ;

            UIAlertController *alert = [UIAlertController alertControllerWithTitle:title
                message:message
                preferredStyle:UIAlertControllerStyleAlert];

            [alert addAction:[UIAlertAction actionWithTitle:NSLocalizedString(@"OK", @"")
                style:UIAlertActionStyleCancel
                handler:^(UIAlertAction *action){
                }]];

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

        NSString *message = [response description];

        NSLog(@"%@",message);
    }];
```

EVOInitializeTerminalResult

This class is returned from the `initializeTerminal:` method in `EVOCommerceDriverAPI`.

```
@interface EVOInitializeTerminalResult : NSObject

/**
 * True if the the call to `initializeTerminal:` method was
 * successful, false if it failed.
 *
 * If True, then check the `updateResponse` property to check
 * if the temrinal has updates available.
 *
 * If false, then check the `errorMessage` for the reason for
 * the failure.
 */
@property (nonatomic, readonly) BOOL isInitialized;

/**
 * The reason a call to `initializeTerminal:` failed.
 */
@property (nonatomic, readonly) NSError * error;

/**
 * The localizedDescription from the error property.
 *
 */
@property (nonatomic, readonly) NSString * errorMessage;

/**
 * `EVOTerminalUpdate` provides details if a terminal has pending
 * updates and the deadline to install those updates.
 *
 * See @EVOTerminalUpdate.
 *
 */
@property (nonatomic, readonly) EVOTerminalUpdate * updateResponse;
```

EVOTerminalUpdate

This class is used to provide details about updates that are pending for the payment terminal. It is returned from the `initializeTerminal:` method in `EVOCommerceDriverAPI`.

```
@interface EVOTerminalUpdate : NSObject

/**
 * If True, there are pending updates for the payment terminal.
 */
@property (nonatomic, readonly) BOOL hasUpdates;

/**
 * If an update is available, this will contain the date that
 * the update should be installed before.
 */
@property (nonatomic, readonly) NSDate * updateDeadline;
```

If the `hasUpdates:` property of the `EVOTerminalUpdate` object is true, call the `DownloadAndApplyUpdate:` method as described below before the terminal update deadline date.

IMPORTANT! If a terminal has not downloaded the available terminal updates by the associated deadline date, the terminal will be deactivated, preventing any future transactions.

For more information on downloading and applying terminal updates, or the TSM feature as a whole, please see the TSM User Guide.

Transaction Processing

Two transaction sets can be processed using CommerceDriver™:

1. Card information Required Transactions
   * Authorize
   * Authorize and Capture
   * Return Unlinked
   * Verify (Terminal Required)

2. No Card Information 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;

    /**Use this factory method to create an EVOPOSOperationVerify request.*/
    + (instancetype)createVerifyRequestEmployeeId:(NSString *)employeeId laneId:(NSString *)laneId
orderNumber:(NSString *)orderNumber reference:(NSString *)reference;
```

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. When working with a terminal without a display, such as with the Chipper BT, CommerceDriver™ will communicate various card states via the `request:cardReaderStatusUpdate:` method.

   It uses the following enumeration to represent state and actions that should be taken:

```
typedef NS_ENUM(NSUInteger, EVOCardReaderState) {

    ///MSR card swipe is enabled.
    EVOCardReaderStateSwipeCard,

    ///EMV card insert is enabled.
    EVOCardReaderStateInsertCard,

    ///Contactless card tap is enabled.
    EVOCardReaderStateTapCard,

    ///MSR swipe and Contactless are enabled.
    EVOCardReaderStateSwipeTapCard,

    ///MSR swipe, EMV insert, Contactless tap are all enabled.
    EVOCardReaderStateInsertSwipeTapCard,

    ///MSR swipe and EMV insert are enabled.
    EVOCardReaderStateInsertSwipeCard,

    ///EMV card insert and Contactless tap are enabled.
    EVOCardReaderStateInsertTapCard,

    ///The card must be removed.
    EVOCardReaderStateRemoveCard,

    ///The terminal did not read the magstripe on the card.
    EVOCardReaderStateErrorBadCardSwipe,

    ///The card read is an EMV card and should be inserted.
    EVOCardReaderStateChipCardSwipedPleaseInsert,
    ///The chip on the card cannot be read, please swipe the card.
    EVOCardReaderStateCannotReadChipPleaseSwipe,

    ///The chip on the card cannot be read, please try again.
    EVOCardReaderStateErrorBadCardInsert,
};
```

The following is a stub implementation of `request:cardReaderStatusUpdate:` with instructions on how to handle each state:

```
- (void) request:(EVOPOSTransactionRequest *)request
cardReaderStatusUpdate:(EVOCardReaderState)status {

    switch (status) {
        case EVOCardReaderStateInsertSwipeCard:
         //Show a messgae to @"Please insert or swipe card"
            break;

        case EVOCardReaderStateSwipeCard:
            //Show a message to @"Please swipe card";
            break;

        case EVOCardReaderStateInsertCard:
            //Show a message to @"Please insert card";
            break;

        case EVOCardReaderStateInsertSwipeTapCard:
            //Show a message to @"Please insert swipe or tap card";
            break;

        case EVOCardReaderStateTapCard:
            //Show a message to @"Please tap card";
            break;
        case EVOCardReaderStateRemoveCard:
            //Show a message to @"Please remove card";
            //Also see the 'request:confirmCardRemoved:' method.
            break;

        case EVOCardReaderStateSwipeTapCard:
            //Show a message to @"Please swipe or tap card";
            break;

        case EVOCardReaderStateInsertTapCard:
            //Show a message to @"Please insert or tap card";
            break;

        case EVOCardReaderStateCardRemoved:
            //Informational.  The card was removed from the terminal.
            break;

        case EVOCardReaderStateErrorBadCardSwipe:
            //Informational.  The card swiped was not read.
            //Possibly show this message @"Error bad card swipe.";
            break;

        case EVOCardReaderStateChipCardSwipedPleaseInsert:
            //Show a message that a cip card was swiped and it should
            // be inserted.
            //e.g. @"Chip card swiped, please insert.";
            break;

        case EVOCardReaderStateCannotReadChipPleaseSwipe:
            //Show a message that the Chip cannot be read and
            //it should be swiped.
            //e.g. @"Cannot read chip, please swipe.";
            break;
        case EVOCardReaderStateErrorBadCardInsert:
            //Show a message that the Chip cannot be read and
            //the customer should try again.
            //e.g. @"Error bad card insert, please try again.";
            break;
    }
}
```

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.

9.
```
-(void) request:(EVOPOSTransactionRequest *)request getManualCardEntry:(void (^)(NSString *,
NSString *, NSString *))completion;
```

Called when running a manual keyed entry transaction to obtain credit card data.

Strong Customer Authentication (SCA) – Contactless PIN

Strong Customer Authentication (SCA) is an overarching mandate that is aimed at increasing and adding security for potentially suspicious transactions, whether they be Card Present or Card Not Present transactions. This particular part of the SCA mandate focuses on EMV Contactless PIN, where additional security will be requested from customers initiating contactless transactions in the form of asking customers to enter their PIN in order to successfully process certain transactions. Payment service providers are exempted from the application of SCA, where the payer initiates a contactless electronic payment transaction, provided that both the following conditions are met:

- The individual amount of the contactless transaction does not exceed 50 EUR.
- The number of previous contactless transactions initiated since the last application of SCA does not exceed 150 EUR or 5 consecutive payment transactions.

SCA Contactless PIN workflow is supported for the European market.

Note: this process is handled entirely within CommerceDriver™ and, from a merchant perspective, no extra integration changes are needed. Refer to the Platform Integration Guide for more information about the specifics included in the Resubmit or Challenge Required

Workflow

The following steps outline the process flow for the EMV Contactless PIN flow for SCA.

1. Cardholder taps their card to initiate a contactless transaction.
2. The transaction request is sent to the issuing bank, and they will determine if the completion of a challenge is needed to complete the transaction, as determined by the logic set forth under the new SCA mandate.

3. CommerceDriver™ handles the issuing bank's response by asking the terminal to prompt for an online PIN or falling back to initiate a contact transaction.

If a PIN was required, CommerceDriver™ handles a resubmit to the issuing bank with the extra data needed to approve the contactless transaction (e.g. PIN and KSN).

Tokenized Transactions

Verification

A Verify transaction request can be used to create a token for use in future transactions. The process for running a Verify transaction request is the same process outlined in the Transaction Processing section. Please refer to that section for more details. Please note that Verify is not supported for manual keyed entry and requires a terminal.

1. Use the `createVerifyRequestEmployeeId:laneId:orderNumber:reference:` method of `EVOPOSTransactionRequest` to create a transaction request.
2. Then pass the transaction request to the `processTransactionRequest:` of the `EVOCommerceDriverAPI`. The terminal will start a swipe only authorization for $0.00.

Example

```
NSDecimalNumber * amount = [NSDecimalNumber decimalNumberWithString:@"10.00"];
    NSDecimalNumber * tipAmount = [NSDecimalNumber decimalNumberWithString:@"0.00"];
    NSDecimalNumber * cashbackAmount = [NSDecimalNumber decimalNumberWithString:@"0.00"];
    NSString * employeeId = @"Clerk-01";
    NSString * reference = @"Example payment";
    NSString * laneId = @"01";
    NSString * orderNumber = @"Example-1234";



    EVOPOSTransactionRequest * request = [EVOPOSTransactionRequest createVerifyRequestEmployeeId:employeeId
laneId:laneId orderNumber:orderNumber reference:reference];


    request.delegate = self;


    [commerceDriverAPI processTransactionRequest:request ];
```

Upon successful completion of the platform request, you will receive an `EVOTransanctionResponse` continaing the `paymentAccountDataToken` field. The `paymentAccountDataToken` is a secure, tokenized representation of the card used for the transaction. This token can be saved for later use in subsequent transaction requests.

Processing Transactions with a Token

Once you have a `paymentAccountDataToken`, you can process transactions without the need for a credit card payment terminal.

The steps for processing a payment are similar to the step outlined in the Transaction Processing section, but instead of using the `processTransactionRequest:` method, the `processTransactionRequest: paymentAccountDataToken:` method is used. This method will start a transacyion and go straight to online authorization.

Example

```
EVOCommerceDriverAPI * commerceDriverAPI = ...;


    NSString *accountDataToken = @"49947f6b-6fb4-4c7c-78a3-810de5c6a1f9127986b7-4b5e-47b1-6f88-d15c34712c0d";


    NSDecimalNumber * amount = [NSDecimalNumber decimalNumberWithString:@"10.00"];
    NSDecimalNumber * tipAmount = [NSDecimalNumber decimalNumberWithString:@"0.00"];
    NSDecimalNumber * cashbackAmount = [NSDecimalNumber decimalNumberWithString:@"0.00"];
    NSString * employeeId = @"Clerk-01";
    NSString * reference = @"Example payment";
    NSString * laneId = @"01";
    NSString * orderNumber = @"Example-1234";


    EVOPOSTransactionRequest * request = [EVOPOSTransactionRequest createAuthorizeRequestAmount:amount
employeeId:employeeId laneId:laneId orderNumber:orderNumber reference:reference tipAmount:tipAmount
cashbackAmount:cashbackAmount overrideApDupe:YES];


    request.delegate = self;


    [commerceDriverAPI processTransactionRequest:request paymentAccountDataToken:accountDataToken];
```

Manual Keyed Entry Transactions

The ablity to process transactions with keyed credit card data was added in v2.34 of CommerceDriver™. Keyed entry transactions can be used with transaction types:

* Authorize
* Authorize and Capture
* Return Unlinked

Keyed entry transactions are created the same way as outlined in the Transaction Processing section of this document. After creating a transaction request, set the `keyedEntry` field to `true` to indicate credit card data will be entered manually.

```
    NSDecimalNumber * amount = [NSDecimalNumber decimalNumberWithString:self.amountField.text];
    NSDecimalNumber * tipAmount = [NSDecimalNumber decimalNumberWithString:self.tipAmountField.text];
    NSDecimalNumber * cashbackAmount = [NSDecimalNumber decimalNumberWithString:self.cashbackAmountField.text];

    EVOPOSOperation operation = self.segmentedControlTransactionType.selectedSegmentIndex;
    EVOPOSTenderType tenderType = self.segmentedControlTenderType.selectedSegmentIndex;

    NSString * orderNumber;
    orderNumber = [[NSUUID UUID].UUIDString substringToIndex:6];

    EVOPOSTransactionRequest * request = [EVOPOSTransactionRequest createAuthorizeAndCaptureRequestAmount:amoun
t employeeId:@"1234" laneId:@"lane01" orderNumber:orderNumber reference:@"" tipAmount:tipAmount cashbackAmount:
cashbackAmount overrideApDupe:NO tenderType:tenderType];

    //Turn Keyed Entry on
    request.keyedEntry = YES;

    [commerceDriverAPI processTransactionRequest:request];
```

CommerceDriver™ calls the new delegate method `request:getManualCardEntry:` to gather the credit card data.

An example implemenation of `request:getManualCardEntry:` that uses a UIAlertViewController to prompt the user for the Pan, Expiration Date and CVV Data follows.

```
-(void) request:(EVOPOSTransactionRequest *)request getManualCardEntry:(void (^)(NSString *, NSString *, NSStri
ng *))completion {

    UIAlertController *alertController = [UIAlertController alertControllerWithTitle:@"Keyed Entry" message:@"P
lease enter your card." preferredStyle:UIAlertControllerStyleAlert];
    [alertController addTextFieldWithConfigurationHandler:^(UITextField * _Nonnull textField) {
        textField.placeholder = @"PAN";
        textField.secureTextEntry = NO;
    }];
    [alertController addTextFieldWithConfigurationHandler:^(UITextField * _Nonnull textField) {
        textField.placeholder = @"EXPIRATION (MMYY)";
        textField.secureTextEntry = NO;
    }];
    [alertController addTextFieldWithConfigurationHandler:^(UITextField * _Nonnull textField) {
        textField.placeholder = @"CVV Code";
        textField.secureTextEntry = NO;
    }];

    __weak __typeof__(alertController) weakController = alertController;
    UIAlertAction *confirmAction = [UIAlertAction actionWithTitle:@"OK" style:UIAlertActionStyleDefault handler
:^(UIAlertAction * _Nonnull action) {
        NSString * pan =  [[weakController textFields][0] text];
        NSString * expiration =  [[weakController textFields][1] text];
        NSString * cvvCode =  [[weakController textFields][2] text];

        completion(pan,expiration,cvvCode);

    }];
    [alertController addAction:confirmAction];
    UIAlertAction *cancelAction = [UIAlertAction actionWithTitle:@"Cancel" style:UIAlertActionStyleCancel handl
er:^(UIAlertAction * _Nonnull action) {
        [self.commerceDriverAPI cancelAsyncProcess:request];
    }];
    [alertController addAction:cancelAction];
    [self presentViewController:alertController animated:YES completion:nil];

}
```

Transaction requests complete as outlined in the Transaction Processing section.

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.