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*™ - V2.0.27
- Supports EVOSnap* v2.1.27 Platform calls
- Supported Terminal – Ingenico ICMP via Bluetooth

Compatibility

- *CommerceDriver*™ Framework – iOS 8.0 & Higher using Objective-C
- Sample Code, Projects & Guides – Created using xCode 8 & 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. Create an `EVOPlatformConfiguration` specifying your service related information and an `EVOCommerceDriverAPI` object to utilize the configuration.

```
    NSString * serviceId = @"<YOUR SERVICE ID>";
    NSString * sericeKey = @"<YOUR SERVICE KEY>";
    NSString * applicationProfile = @"<YOUR APPLICATION PROFILE ID>";
    NSString * hostDescription = @"ANY STRING YOU WANT";
    NSString * merchantProfileID = @"<OPTIONAL: MERCHANT PROFILE ID>";

    EVOPlatformConfiguration * config = [[EVOPlatformConfiguration alloc ] initWithServiceID:serviceId serviceKey:sericeKey
applicationProfileId:applicationProfile hostDescription:hostDescription merchantProfileId:merchantProfileID];

    EVOCommerceDriverAPI * commerceDriverAPI = [[EVOCommerceDriverAPI alloc ] initWithPlatformConfig:config];
```

4. Set the *CommerceDriver*™logging level. (Optional)

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

Authentication

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

1. Log into the Platformby calling the loginUser:password: method EVOCommerceDriverAPI.

```
[commerceDriverAPI loginUser:username password:password];
```

2. Listen For the results notification communicated from the `EVOIdentityLoginEvent`.

```
[[NSNotificationCenter defaultCenter] addObserver:self selector:@selector(onIdentityLoginEvent:) name:EVOIdentityLoginEvent
object:nil];
```

3. Process the login response using the example notification handler below.

```
    -(void)onIdentityLoginEvent:(NSNotification*)notification {

        //Gets the EVO Snap login results object.
        EVOIdentityLoginEventArgs *loginResult = [notification userInfo][EVOEventArgumentsKey];

        //The successful flag can be used to determine if login succeeded.
        if (loginResult.successful ) {
            NSLog(@"Logged in successfully with message: %@", loginResult.message);
        } else {
            ///If login did not succeed, then check the state property to determine the next action.
            switch (loginResult.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: %@", loginResult.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
unlock.
                    break;
                case EVOIdentityLoginStateAccountLockedAdmin:
                    //The account has been locked by the EVO Snap service.  It can only be unlocked by contacting support.
                    NSLog(@"login message: %@", loginResult.message);
                    break;
                case EVOIdentityLoginStateServiceErrorMessage:
                    ///The service returned an error message.
                    ///Display the error message to the user.

                    NSLog(@"login message: %@", loginResult.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
* AuthorizeAndCapture
* ReturnUnlinked

CommerceDriver™ for iOS currently supports the *Ingenico ICMP* device. The library for this device is `EVOIngenicoTerminals.framework` version 1.0.0.

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 Ingenico library, add the following Import statements to the classes using the *CommerceDriver*™ framework.

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

To **Register** your device for support:

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

Sample A – 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 B – 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];
```

Transaction Processing

Two transaction sets can be processed using *CommerceDriver*™.

Terminal Required Transactions

* Authorize
* Authorize and Capture
* Return Unlinked

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 a request of operation type EVOPOSOperationAuthorizeAndCapture
,EVOPOSOperationAuthorize, EVOPOSOperationReturnUnlinked.  Any other EVOPosOperation value will produce an exception.
Requests created with this factory method will require a terminal to proceed with the request */

    + (instancetype) createTerminalRequestWithOperation:(EVOPOSOperation)operation amount:(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;
```

   The default initializer can also be used to create a request. For additional information, please refer to the *CommerceDriver™ Apple Doc.*

```
  - (instancetype)initWithOperation:(EVOPOSOperation)operation
        amount:(NSDecimalNumber *)amount employeeId:(NSString *)employeeId laneId:(NSString *)laneId orderNumber:(NSString
*)orderNumber reference:(NSString *) reference tipAmount:(NSDecimalNumber *)tipAmount cashbackAmount:(NSDecimalNumber
*)cashbackAmount overrideApDupe:(BOOL)overrideApDupe;
```

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

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

To **Cancel** a Request:

1. Call `cancelAsynchPRocess`.

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

To **Request** a POS Delegate:

1. The `EVOPOSTransactionRequest` uses the `EVOPOSTransactionRequestDelegate` protocol to communicate transaction status. After creating an `EVOPOSTransactionRequest` set the delegate property to the class that implements the `EVOPOSTransactionRequestDelegate`.

```
    EVOPOSTransactionRequest * authAndCaptureRequest = [EVOPOSTransactionRequest
createTerminalRequestWithOperation:EVOPOSOperationAuthorizeAndCapture amount:[NSDecimalNumber numberWithInt:5]
employeeId:@"EE-ID1" laneId:@"LN-01" orderNumber:@"ORDER-01"
        reference:@"REF-01" tipAmount:[NSDecimalNumber zero] cashbackAmount:[NSDecimalNumber zero] overrideApDupe:YES];

    authAndCaptureRequest.delegate = self;
```

2. The following delegate methods are required.

```
    /// Called when validation of a signature is needed.
    ///
    /// This method will block the completion of a transaction until the
    /// signature is approved or declined.
    ///
    /// @param request - The original POS Request
    /// @param response - The EVO Platform response to that original request.
    /// @see EVOTransactionResponse
    /// @param completion -  You must call the completion block with the
    /// outcome of signature verification. Return YES to approve the signature
    /// and NO to reject it.
    + (void)getSignatureForRequest:(EVOPOSTransactionRequest *)request
        withResponse:(EVOTransactionResponse *)response completion:(void(^)(BOOL signatureAccepted))completion;

     /// Called when a transaction can not be started.
     /// @param request - The original POS Request
     /// @param errors - Check this dictionary for the errors encountered
     /// starting the Transaction Request.
    + (void) request:(EVOPOSTransactionRequest *)request
        failedToStartWithErrors:(NSDictionary *)errors ;


     /// Called upon completion of a transaction
     /// @param request  The original request sent
     /// @param response The outcome of the request
     /// @see EVOTransactionResponse
    + (void) request:(EVOPOSTransactionRequest *)request completedWithResponse:
        (EVOTransactionResponse *)response;
```

Frameworks

CommerceDriver™ for iOS consists of the following frameworks

* EVOCommerceDriver.framework - The core framework that provides all *CommerceDriver*™ functionality. This framework is required.
* EVOIngenicoTerminals.framework - This framework provides the terminal implementation for all Ingenico 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.