EVO® Snap

Upgrade Guide

CommerceDriver™ for *IOS*®
Version 2.27 to 2.30

CommerceDriver™ Upgrade Guide

EVO Snap* has released the v2.30 version of CommerceDriver™. This guide is designed to assist partners in the migration process from integrations using v2.27 to v2.30 of CommerceDriver™.

Partners who were previously using the 2.27 libraries and who have been directed by their Snap* Solutions Engineer to update their projects to version 2.30 will need to upgrade to the 2.30 version of CommerceDriver™.

Enhancements & New Features

Terminal Service Management (TSM)

All CommerceDriver™ integrators upgrading to version 2.30 are required to support the new TSM features in order to support updates to the Ingenico line of terminals, including the iCMP, iPP320, and iPP350 terminals.

Terminals will receive updates periodically, and if an update is not applied to a terminal by the associated deadline date, the terminal will be unable to transact until the update is installed.

For more information on the new TSM feature, please see the TSM User Guide.

Tokenized Transactions

Tokenization is the process of using a token to run what would typically be a "card only" transaction. The EVO Snap* Platform generates a unique token associated with a customer's card that can be used instead of the customer's actual card to process a transaction. For more information on the new Tokenization feature, please see the Tokenization section below.

API Changes

EVOCommerceDriverAPI

Creating the Object

The construction of the `EVOCommerceDriverAPI` in v2.27 required that an instance of `EVOPlatformConfiguration` be passed in. This requirement has been removed in v2.30, and the preferred way to create the object is using the `initWithServiceKey:applicationProfileId:` method.

Version	Code Change
2.27	`- (instancetype) initWithPlatformConfig:(EVOPlatformConfiguration *)configuration;`
2.30	`- (instancetype)initWithServiceKey:(NSString *)serviceKey applicationProfileId:(NSString *)applicationProfileId;`

IsAuthorized

A typo was fixed in the `isAuthorized` property in the 2.30 version.

Version	Code Change
2.27	`- (BOOL) isAthorized;`
2.30	`- (BOOL)isAuthorized;`

Deprecated Methods

The following methods have been deprecated and should no longer be used:

Login User

Method Type	Code Change
Deprecated	`- (void)loginUser:(NSString *)username password:(NSString *)password DEPRECATED_MSG_ATTRIBUTE("Please use loginUser:password:completion");`
New	`- (void)loginUser:(NSString *)username password:(NSString *)password completion:(void (^)(BOOL success, EVOIdentityLoginState state, NSString *message))completion;`

Forgot Password

Method Type	Code Change
Deprecated	`- (void)forgotPassword:(NSString *)username DEPRECATED_MSG_ATTRIBUTE("Please use forgotPassword:completion");`
New	`- (void)forgotPassword:(NSString *)username completion:(void (^)(EVOForgotPasswordState state, EVOResetPasswordData *data, NSString *message))completion;`

Change Password

Method Type	Code Change
Deprecated	`- (void)changePasswordForUsername:(NSString *)username oldPassword:(NSString *)oldPassword newPassword:(NSString *)newPassword confirmPassword:(NSString *)confirmPassword DEPRECATED_MSG_ATTRIBUTE("Please use changePasswordForUsername:oldPassword:newPassword:confirmPassword:completion");`
New	`- (void)changePasswordForUsername:(NSString *)username oldPassword:(NSString *)oldPassword newPassword:(NSString *)newPassword confirmPassword:(NSString *)confirmPassword completion:(void (^)(EVOChangePasswordState state, NSString *message))completion;`

Security Questions

In v2.27, it was required to send the username and password when calling the methods related to security questions. In v2.30, this requirement has been removed, and thus the names of the methods related to security questions have changed. When calling the new methods, they are called for the currently authenticated user.

Define Security Questions

Version	Code Change
2.27	`- (void) securityQuestionsDefinedForUsername:(NSString*)username password:(NSString*)password completion:(void (^)(BOOL isDefined))completion;`
2.30	`- (void)securityQuestionsDefined:(void (^)(BOOL isDefined))completion;`

Get User Security Questions

Version	Code Change
2.27	`- (void) getUserSecurityQuestionsForUsername:(NSString*)username password:(NSString*)password completion:(void (^)(EVOSecurityQuestionsResponse *questionsResponse))completion;`
2.30	`- (void)getUserSecurityQuestions:(void (^)(EVOSecurityQuestionsResponse *questionsResponse))completion;`

Get All Security Questions

Version	Code Change
2.27	`- (void) getAllSecurityQuestionsForUsername:(NSString*)username password:(NSString*)password completion:(void (^)(EVOSecurityQuestionsResponse *questionsResponse))completion;`
2.30	`- (void)getAllSecurityQuestions:(void (^)(EVOSecurityQuestionsResponse *questionsResponse))completion;`

Merchant Profile Changes

Version 2.30 added the following methods related to a merchant's profile:

Method	Description
`- (NSArray<EVOMerchantProfile *> *) getMerchantProfiles;`	This method retrieves an array of merchant profiles associated with the authenticated user.
`- (BOOL) selectMerchantProfile:(NSString *) profileId;`	Use this method to specify the merchant profile to use when processing transactions. The `profileId` should match an `EVOMerchantProfile` in the array returned from `getMerchantProfiles`. Will return 'Yes' if the profile was successfully selected.
`- (EVOMerchantProfile *) getSelectedMerchantProfile;`	Retrieves the currently selected merchant profile. Returns the selected `EVOMerchantProfile` object.
`- (void) setDefaultMerchantProfileId:(NSString *)profileId;`	Specifies the merchant profile to use by default. The `profileId` is the `EVOMerchantProfile` selected upon login. **Note**: if there is only one merchant profile associated with an account, it will automatically
`- (NSString *) getDefaultMerchantProfileId;`	Returns the `profileId` for the default merchant profile.

EVOPOSTTransactionRequest

Deprecated Methods

The following methods have been deprecated and should no longer be used:

```
    + (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 DEPRECATED_MSG_ATTRIBUTE("Please use createAuthorize,
createAuthorizeAndCapture, or createReturnUnlinked method");
```

New Methods

```
    + (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;`
```

```
    + (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;
```

```
    + (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;
```

EVOPOSTransactionRequestDelegate

A new method was added, which is called when running an Amex MSR transaction which requires the CVV code from the back of the card. ***This new method is required and must be implemented.***

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

Tokenization

Tokenization is the process of using a token to run what would typically be a card only transaction. The EVO Snap* platform generates a unique token associated with a customer's card that can be used instead of the customer's actual card to process a transaction.

How to Run a Tokenized Transaction

In order to run a tokenized transaction, the `PaymentAccountDataToken` property must be populated with a valid payment token in the `TransactionRequest.CardData` field. If the `PaymentAccountDataToken` is populated, then CommerceDriver™ will automatically run the token and no card will be needed to process the transaction.

The transaction types that can use payment tokens are listed below:

* Authorize
* Authorize and Capture
* Return Unlinked

Where to Find the Payment Token

In the response of any transaction that uses a card, there will be a field called `PaymentAccountDataToken`. The value in this field is specific to the card that was used in that transaction and can be used to populate `TransactionRequest.CardData.PaymentAccountData` in order to run another transaction for that card.

Transactions that can return a `PaymentAccountDataToken` are listed below:

* Authorize
* Authorize and Capture
* Return Unlinked
* Verify