CommerceDriver™

Quick-Start Guide for *Android*®

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 – Android API Level 17+ (4.2 JellyBean)

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.

Gradle

Core Library

The core library contains the *CommerceDriver*™ and all non-terminal supporting classes.

```
compile project(path: ':evo-core-module', configuration: 'evo-core-configuration')
```

Terminal Libraries

Each terminal model/manufacturer may require at least 1 additional library.

```
compile project(path: ':evo-terminal-1-path', configuration: 'evo-terminal-1-configuration')
compile project(path: ':evo-terminal-2-path', configuration: 'evo-terminal-2-configuration')
```

Connect a Terminal

Terminals connect through an audio jack or BlueTooth. Please refer to the *CommerceDriver*™ ***Android*** JavaDocs for supported terminal information and to ensure your terminal is properly connected prior to running a transaction.

Initialize the CommerceDriver™ Instance

```
// context should be your Application context. The Service Id and Application Profile Id will be provided
static final String APPLICATION_PROFILE_ID = "11111";
static final String SERVICE_KEY = "ABCD123";
static final String MERCHANT_PROFILE_ID = "EFGH456";
static final String ACME_WIDGET_TERMINAL_ID = "acme_terminal_1";
static final String USERNAME = "tom";
static final String PASSWORD = "foolery";

// CommerceDriver instance created via "Builder" pattern
// optionally add callbacks
commerceDriver = new CommerceDriver.Builder(getContext(), APPLICATION_PROFILE_ID, SERVICE_KEY)
```

CommerceDriver™ Events

```
// CommerceDriver instance created via "Builder" pattern
commerceDriver.addLogCallback(...); // commerce driver logs events (debug builds)
commerceDriver.setLogLevel(...); // level for commerce driver logs (debug builds)
commerceDriver.addEventCallback(...); // commerce driver events
commerceDriver.addTransactionSignatureCallback(...); // txn signature events
commerceDriver.addTransactionCompletedCallback(...); // txn completed events
commerceDriver.addTransactionAuthorizationCallback(...); // authorization (online) events
commerceDriver.addTransactionCancelledCallback(...); // txn cancellation events
commerceDriver.addTransactionErrorCallback(...); // txn error events
commerceDriver.addTransactionFallbackEventCallback(...); // txn fallback events
commerceDriver.addTransactionPinEventCallback(...); // txn pin events
commerceDriver.addPlatformHttpLoggingCallback(...); // platform http logging (debug builds)
commerceDriver.addPlatformEventCallback(...); // platform events
commerceDriver.addPlatformLogCallback(...); // platform log events (debug builds)
commerceDriver.setPlatformLogLevel(...); // level for platform logs (debug builds)
```

Authentication

Authenticate Your CommerceDriver™ Instance

```
// Call this method off of the UI thread using an AsyncTask, Handler, Thread, or similar.
loginResponse = commerceDriver.login(USERNAME, PASSWORD);
```

Validate that the login was successful

```
// Call this method off of the UI thread using an AsyncTask, Handler, Thread, or similar.
if (loginResponse.getSuccessResponse() !=null) {
    // perform post-login routines
}
```

Set Your Merchant Profile

```
// Set your active merchant profile (required to run transactions with a terminal)
commerceDriver.setActiveMerchantProfileId(MERCHANT_PROFILE_ID);
```

Terminal Setup

Make sure that the terminal you want to use is added to your project.

Add a Terminal

```
// in this example "AcmeWidgetsHardwareController" would come from a separate EVO library that supports the "AcmeWidget"
Brand/Model of EMV Terminals.
commerceDriver.addHardwareController(ACME_WIDGET_TERMINAL_ID, new AcmeHardwareController());
```

Set a Terminal as Active

```
// in this example "AcmeWidgetsHardwareController" would come from a separate EVO library that supports the "AcmeWidget"
Brand/Model of EMV Terminals.
commerceDriver.setActiveHardwareController(ACME_WIDGET_TERMINAL_ID);
```

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

Build a Transaction Request

```
// Create a PosTransactionRequest (construction uses "Builder" pattern so you can chain calls)
PosTransactionRequest request = new PosTransactionRequest.Builder(PosOperation.AUTHORIZE_AND_CAPTURE)
// amount, cashback, and tip amount available to set with an "AmountModifier"
.addModifier(myAmountModifier)
// employee id (SE will let you know if this is required) can be set with "EmployeeIdModifier"
.addModifier(myEIDModifier)
// order id (SE will let you know if this is required) can be set with "OrderIdModifier"
.addModifier(myOIDModifier)
// build the request
.build();
```

Start a Transaction

```
commerceDriver.startTransaction(request);
```

Libraries (.aar)

The *CommerceDriver*™ library (.arr) should have a separate Gradle module.

Terminal libraries have their own .aar and should also have their own Gradle modules.

Add the following CommerceDriver™ dependencies via Gradle:

```
compile 'com.squareup.retrofit2:retrofit:2.1.0'
compile 'com.squareup.retrofit2:converter-gson:2.1.0'
compile 'com.squareup.okhttp3:logging-interceptor:3.4.1'
compile 'com.squareup.okhttp3:okhttp:3.4.1'
compile 'com.google.code.gson:gson:2.7'
compile 'com.google.guava:guava:20.0'
compile 'com.google.android.gms:play-services-base:10.2.0'
```

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.