

EV[®] Snap

Commerce Driver™

Windows® Quick-Start Guide 1.0

- Understanding EMV® Certification..... 2
 - What is EMV? 2
 - How Does it Work? 2
 - Becoming EMV Compliant 2
 - Level 1 – Hardware/Terminal Certification..... 2
 - Level 2 – Kernel Certification 2
 - Level 3 – Payment Application Certification..... 3
- EVO Snap* Commerce Driver™ 4
 - How it Works 4
 - Commerce Driver™ Specifications 5
 - Authentication Methods 5
 - System Requirements 5
 - Networks 5
 - Managed API Calls..... 5
 - Pre-Certified Hardware 5
- Commerce Driver™ Integration..... 6
 - Getting Started 6
 - Installation 6
 - Framework Integration 6
 - API Integration..... 6
 - Swiper Integration 7
 - Request Examples 9
- Reference Documents 10

Understanding EMV® Certification

What is EMV?

EMV, named after the organizations that created the technology standard – Europay, MasterCard and Visa – is a technical standard for the interaction between chip-based “smart cards” and approved payment devices. The standard is now managed by EMVCo, a consortium with control split equally among American Express, JCB, Discover, MasterCard, UnionPay and Visa.

The purpose of the EMV Specifications is to facilitate the worldwide interoperability and acceptance of secure payment transactions. During a card-present payment transaction, payment data – securely stored on a microchip – can either be embedded in a traditional plastic card or mobile device.

How Does it Work?

EMV devices are able to read data stored on a chip within the card. By using chips as an active part of the payment transaction, EMV cards and devices help prevent credit card fraud from stolen account numbers, cloned payment cards and other security and fraud threats that exist today.

Each chip-based card is embedded with encrypted data. During the transaction authorization process, the encrypted data in the card is used to verify the card's authenticity. Strong cryptographic functions are used to authenticate the card and cardholder to ensure validity.

Becoming EMV Compliant

The EMV standards define the interaction at the physical, electrical, data and application levels between EMV cards and EMV card processing devices. There are three levels of EMV certification:



Level 1 – Hardware/Terminal Certification

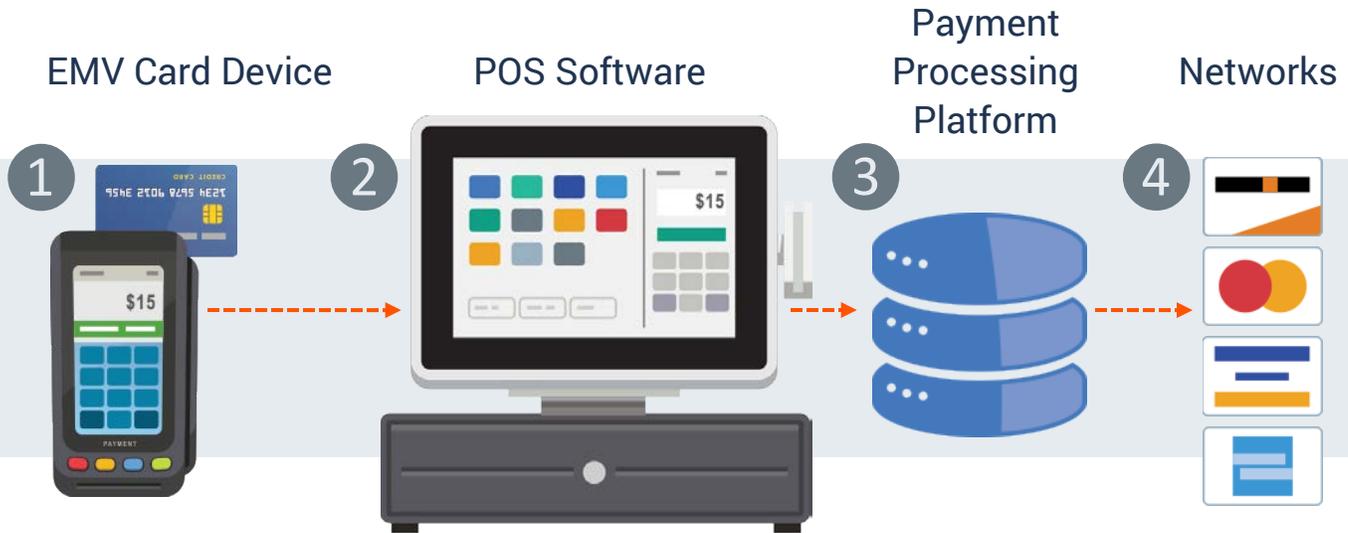
Level 1 certification covers the physical interface between the card acceptance terminal and the EMV card. *Responsible Party: Terminal Vendor*

Level 2 – Kernel Certification

Level 2 certification covers the software interface between the card acceptance terminal and the chip card. *Responsible Party: Terminal Vendor*

Level 3 – Payment Application Certification

Level 3 certification covers the software interface between a POS application and the card acceptance terminal. **Responsible Party: Software Vendor**



Level 3 certification - also called end-to-end (E2E) or network certification - tests each unique EMV path to the networks. The testing flow is as follows:

- 1 Level 1 and 2 certified card device, 2 the POS software, any middleware or gateway in use, 3 the processor, and finally out to 4 the card brands.

This process must be completed individually for:

1. Each device the POS application is using
2. Each version of software
3. Each processor
4. Each network

The cost and complexity of managing certification to each unique EMV path can quickly become overwhelming. There is a better way. You can create EMV approved Windows® based Point of Sale (POS) applications in a snap with the Commerce Driver™ from EVO Snap*.

EVO Snap* Commerce Driver™



The Commerce Driver™ makes implementing EMV technology faster and easier by combining technical, operational and strategic components into a fully integrated, tested and production-ready solution.

Meet all EMV Level 3 compliance requirements and instantly enable PCI-compliant transactions with end-to-end encryption using the Commerce Driver from EVO Snap*.

How it Works

Like a printer driver, the pre-certified Commerce Driver™ SDK installs alongside your software application - adding EMV transaction processing to your Windows®-based POS systems. The Commerce Driver facilitates all transactional communication with the EVO Payments global processing platforms and approved hardware devices to isolate payment data and keep it separate from the software application.



Advantages of the Commerce Driver™:

1. Easy to install...similar to a printer driver
2. Pre-certified...decreasing your time to market
3. Reduces PCI Compliance scope and liability for merchants
4. Provides Point to Point Encryption for all transactions
5. EVO maintained...no ongoing maintenance costs

Commerce Driver™ Specifications

Authentication Methods

- * Chip & PIN
- * Chip & Signature

System Requirements

- * Windows 7, 8, 8.1 & Vista

Networks

- * Bluetooth
- * USB
- * Ethernet
- * Wireless

Managed API Calls

- * Login
- * Authorize
- * Capture
- * Authorize & Capture w/Duplicate Transaction Override
- * Resubmit
- * Transaction Lookup
- * Void w/Forced Void
- * Batch Capture w/Forced Void
- * Local Settings File w/Authentication Parameters

Pre-Certified Hardware

	Payment Support				Security and Compliance		Device Connectivity			Platform Support			Univ SDK
	Smart Chip Capable (EMV)	MagStripe Capable	NFC Capable	PIN Capable	P2PE	Tokenization	USB	BlueTooth	Audio Jack	Apple Support	Android Support	Windows Mobile Support	Global Support
Ingenico													
iPP320	●	●	●	●	●	●	●					●	●
iPP350	●	●	●	●	●	●	●					●	●
iCMP	●	●	●	●	●	●		●		●	●	●	●
EVO													
IT m-50		●			●	●			●	●	●	●	
IT m-100	●	●	●	●	●	●	●		●	●	●	●	

Commerce Driver™ Integration

Getting Started

Getting started with the Commerce Driver™ is a snap. Once you've selected your desired Platform, Network and Hardware, the setup is similar to a direct Web Services integration. However, Commerce Driver™ will need to be hosted locally.

1. Create transaction data objects in your POS.
2. Pass the transaction data to the Commerce Driver™.
3. Commerce Driver™ initiates commands in the terminal and gathers the tender/EMV data and sends it to the platform.
4. EVO sends a return response to the Commerce Driver™ with details for the receipt.

Installation

1. Download the Commerce Driver™ SDK(s) appropriate for your selected device manufacturer(s).

Framework Integration

1. Unzip the SDK and add references to the following libraries in your project:
 - EVOSnapSDK
 - emvswipeapi-pcwid-1.0.0
 - Naudio
 - Newtonsoft.Json (included or can be installed from Nuget)
 - System
 - System.Runtime.Serialization
 - System.Configuration
 - EVOSnapSDK.BBPOS.dll
 - EVOSnapSDK.Ingenico.dll
 - RBA_SDK_CS.dll

API Integration

1. **Configure the API** by using the following configuration file:
 - a. Add the "App.config" file to your project.
 - i. Create/Add the following keys under the <appSettings> tag:

```
<!-- prod URL points to "https://api.cip.goevo.com/2.1.23/REST"
else "https://api.cipcert.goevo.com/2.1.23/REST" -->
<add key="ProductionBuild" value="false" />
<add key="ApplicationProfileId" value="xxxx" />
<add key="ServiceId" value="xxxxxxxx" />
<add key="ServiceKey" value="xxxxxxxx" />
```

Swiper Integration

1. Import the Interfaces and Models namespaces:

```
using EVOSnapSDK.Interfaces;

using EVOSnapSDK.Models;

using EVOSnapSDK.Ingenico; // Use one or the other

using EVOSnapSDK.BBPOS;    // based on your hardware
```

2. Declare the service a (private) member:

```
private static AccountService service;
```

3. Instantiate the service:

```
service = new AccountService( serviceKey ); //Servicekey provided by EvoSnap
```

4. Call a **LoginUser** method with username and password to retrieve the gateway session:

```
Task<GatewaySession> gatewaySession = service.SignInAsync( m_username, m_password
);
```

5. Initialize the **Library** with tokens received from the login method:

```
service.Initialize( responseTokens.SessionToken, responseTokens.UserAccessToken );
```

6. Initialize **MyDeviceListener** interface:

```
var listener = new MyDeviceListener( gatewaySession );
```

7. Initialize DeviceController:

```
m_controller = new DeviceController( listener );
```

Note: If this workflow is initialized by EMV, both SerialNumber and EMVCardData are required. The library returns one value at a time so tracking the values locally is recommended to ensure both are present before processing transactions. Please see the SDK package a for sample implementation.

- 8. Build the BankcardTransaction or BankcardTransactionPro objects using models/schemas (schemas.evosnap.com.CWS.v2._0) integrated into the EVOSnapSDK library provided by EVO. Please refer to the SDK package for examples.

Request Examples

1. Creating TransactionRequest Example:

```
public static TransactionRequest MakeTransaction( CardData card )
{
    return new TransactionRequest
    {
        TransactionData = new
        cws.Transactions.Bankcard.Pro.BankcardTransactionDataPro
        {
            Amount = 1,
            CustomerPresent =
            cws.Transactions.Bankcard.CustomerPresent.Present,
            EmployeeId = "1",
            LaneId = "1",
            OrderNumber = "123456",
            SignatureCaptured = false,
            Reference = "11222432",
            TransactionCode =
            cws.Transactions.Bankcard.TransactionCode.NotSet,
            TransactionDateTime = DateTime.UtcNow,
            CurrencyCode = cws.Transactions.TypeISOCurrencyCodeA3.USD,
            EntryMode = cws.Transactions.EntryMode.Track2DataFromMSR
        },
        Card = card
    };
}
```

2. Authorization Example:

```
Task<TransactionResponse> response = paymentService.AuthorizeAsync( transaction
);
```

3. Capture Example:

```
Task<BankcardCaptureReponsePro> captureResponse = paymentService.CaptureAsync(
bankcardCapture );
```

4. Authorize and Capture Example:

```
Task<TransactionResponse> response = paymentService.AuthorizeAndCaptureAsync(
transaction );
```

5. Query Transaction Details Example:

```
TransactionService transactionService = new TransactionService(gatewaySession);

QueryTransactionsDetail queryTransactionsDetail = new QueryTransactionsDetail();

queryTransactionsDetail.PagingParameters = new PagingParameters
{
    Page = 1,
    PageSize = 10
};

queryTransactionsDetail.QueryTransactionsParameters = new
QueryTransactionsParameters
{
    CaptureDateRange = new DateRange() { StartDateTime = DateTime.UtcNow.AddDays( 90
), EndDateTime = DateTime.UtcNow }, QueryType = QueryType.OR };

Task<List<TransactionDetail>> response = service.GetTransactionDetailsAsync(
queryTransactionsDetail );

// where TransactionDetails is of type
// schemas.evosnap.com.CWS.v2._0.DataServices.TMS.TransactionDetails
```

Reference Documents

For additional assistance, please visit the EVO Snap* Support site at <http://www.evosnap.com/support/> for more information.

Sample code available at <http://github.com/evo-snap/>