EVO® Snap

CommerceDriver™

Quick-Start Guide for *Windows*®

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™ - v2.34.2
* EVOSnap* Web Services - v2.1.34 (Platform calls)
* Certified Terminals:
    - Ingenico ICMP via Serial USB
    - Ingenico iPP320/iPP350 via Serial USB
    - BBPOS Chipper BT via Bluetooth and Serial USB

Compatibility

* CommerceDriver™ Framework – Windows® 7+
* Visual Studio 2015+
* .Net 4.6

Integration

To get started with CommerceDriver™, select your Platform, Network and Hardware. The setup is similar to a direct Web Services integration.

The following is an example for how to get set up using CommerceDriver™. For a more in depth example please refer to the Sample Application solution.

1. Download the CommerceDriver™ SDK based on Terminal Manufacturer.

2. Uncompress the archive into a temporary folder.

3. Copy the following files into the folders associated with your solution:
   * *EvoSnap.CommerceDriver.Common.dll*
   * *EvoSnap.CwsLibrary.dll*
   * *Newtonsoft.Json.dll*

4. Add the assembly references to the solution for the following files:
   * *EvoSnap.CommerceDriver.Common.dll*
   * *EvoSnap.CwsLibrary.dll*

5. Place the assemblies below into an associated \bin\debug output folder. (The assemblies above depend on the following assemblies.) For intergrators using the Ingenico termials, the following assemblies are required:
   * *EvoSnap.CommerceDriver.Ingenico.dll*
   * *RBA_SDK.dll*
   * *RBA_SDK_CS.dll*

   For integrators using BBPOS terminals, the following assemblies are required:
   * *EvoSnap.CommerceDriver.IntegratedTerminals.dll*
   * *BBDeviceClassicDesktop.dll*
   * *wisepadapi-pcwd-3.8.0.dll*

6. Use the (*EvoSnap.CommerceDriver.Common.Controllers) CommerceDriverController* class to create an instance and wire up the default event handlers.

```
// Instantiate the controller and define the logging info
Controller = new CommerceDriverController(sampleSeriveKey, sampleAppProfileId);
Controller.LogInstanceName = "TC001";
Controller.LogInstanceID = 1;
Controller.LogLevel = LogLevel.Trace;

// Wire up general event handlers
Controller.Log += Controller_Log;
```

```
Controller.Notification += Controller_Notification;
Controller.GenerateReceipt += Controller_GenerateReceipt;
Controller.ConfirmSignature += Controller_ConfirmSignature;
Controller.ServiceInvoked += Controller_ServiceInvoked;
Controller.Completed += Controller_Completed;
Controller.RetrieveCvv += Controller_RetrieveCvv;
Controller.ManualCardEntry += Controller_ManualKeyedEntry;

// Create an instance which contains the default password event handlers
Handlers = new DefaultEventHandlers(Controller);

// Wire up the password specific event handlers using the default ones
Controller.IdentityLogin += Handlers.Default_IdentityLogin;
Controller.ChangePassword += Handlers.Default_ChangePassword;
Controller.PasswordReset += Handlers.Default_PasswordReset;
Controller.ForgotPassword += Handlers.Default_ForgotPassword;
Controller.AssignQuestions += Handlers.Default_AssignQuestions;

Controller.AccountNotification += Handlers.Default_AccountNotification;
```

Authentication

Call *Login( )* or *LoginAsync( )* in the CommerceDriverController using a username and password:

```
GatewaySession session = CommerceDriver.Login("UserName", "Password");
```

Terminal Service Management (TSM)

Added in version 2.30 of CommerceDriver™, all CommerceDriver™ integrators 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.

Initialize Terminal

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 `InitializeSelectedTerminal()` method to determine if updates are available.

Code Snippets

To initialize a terminal in C#, call the `InitializeSelectedTerminal()` method in the `CommerceDriverController`. This method runs asynchronously and returns an `InitializeTerminalResult`, which contains an instance of `TerminalUpdate`.

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

```
try
            {
                //The InitializeSelectedTerminal() method initializes the SelectedTerminal, and returns
true in the InitializeTerminalResult.IsInitialized if it was successful.
                InitializeTerminalResult result = await Controller.InitializeSelectedTerminal();

                if (result.IsInitialized)
                {
                    string message = $"Terminal {Controller.SelectedTerminal.Name} successfully
initialized.";

                    //If the InitializeTerminalResult.HasUpdates is true then the user should be informed
that there are updates available for their terminal.
                    if (result.TerminalUpdate.HasUpdates && result.TerminalUpdate.UpdateDeadline != null)
                    {
                        message = message + $" Updates are available and must be applied by
{((DateTime)result.TerminalUpdate.UpdateDeadline):dddd, MMMM dd yyyy}";
                    }

                    MessageBox.Show(message,"Terminal IsInitialized",
MessageBoxButtons.OK,MessageBoxIcon.Information);
                }
                else
                {
                    MessageBox.Show("Unable to initialize terminal.", "Terminal Not Initialized",
MessageBoxButtons.OK, MessageBoxIcon.Warning);
                }
            }
            catch (Exception ex)
            {
                MessageBox.Show("Error Initializing Terminal: " + ex.Message, "Initalize Terminal Error",
MessageBoxButtons.OK, MessageBoxIcon.Error);
            }
```

EVOInitializeTerminalResult

```
    public class InitializeTerminalResult
    {
        #region Constructor
        public InitializeTerminalResult()
        {
            IsInitialized = false;
            Error = null;
            TerminalUpdate = new TerminalUpdate();
        }
        #endregion
        #region Properties
        public TerminalUpdate TerminalUpdate { get; internal set; }
        public ErrorInfo Error { get; internal set; }
        public bool IsInitialized { get; internal set; }
        #endregion
    }
```

EVOTerminalUpdate

```
public class TerminalUpdate
{
    #region Properties

    public bool HasUpdates { get; }

    public DateTime? UpdateDeadline { get; }

    #endregion

}
```

ErrorInfo

```
public class ErrorInfo
{
    #region Constructors

    public ErrorInfo()
    {
        Code = ErrorCode.UNK;
        Message = string.Empty;
        Exception = null;
    }

    #endregion

    #region Properties

    public ErrorCode Code { get; set; }

    public Exception Exception { get; set; }

    public string Message { get; set; }

    #endregion
}
```

If the `HasUpdates` property of the `TerminalUpdate` 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™.

Card Information Required Transactions

* Authorize
* Authorize and Capture
* Return Unlinked

No Terminal Required Transactions

* Undo
* Capture
* Return by ID

1. Compose a request object to authorize and capture a transaction for $10.00.

```
AuthorizeCaptureOperationRequest request = new AuthorizeCaptureOperationRequest();
request.Amount = 10.00m;
request.EmployeeId = "1234";
request.LaneId = "1";
request.OrderNumber = "7724";
request.Reference = "98106";
request.TipAmount = 0;
request.CashbackAmount = 0;
request.OverrideApDupe = false;
```

2. Invoke the Request.

```
await Controller.ProcessAsync(request);
```

Note: At first invocation, the user is asked to login via the IdentityLogin event. All dialogs presented are from DefaultEventHandlers instance.

3. The Completed event contains the processing results. (Requires successful login and completed transaction processing.)

```
private void Controller_Completed(object sender, CompletedEventArgs e)
{
    switch (e.OperationResponse.Type)
    {
        case OperationType.AuthorizeCapture:
            AuthorizeCaptureOperationResponse response = e.OperationResponse as
AuthorizeCaptureOperationResponse;
```

```
            // response.Request -- Contains the original request
            // response.Responses -- All communication with EvoSnap web services
            // response.TransactionResult -- Approved, Declined etc
            // response.TransactionResponse - Transaction response detail
            break;
    }
}
```

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 response.

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, falling back to initiate a contact transaction, or prompting to tap the card again.

4. 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).

Verify

Verify is a transaction operation added to the CommerceDriver™ which can be used to validate a card. The Verify operation will cause the terminal to prompt for a MSR card swipe.

CommerceDriverController Method Call

```
Task ProcessAsync(VerifyOperationRequest request)
```

VerifyOperationRequest

```
public class VerifyOperationRequest: TransactionRequest
    {
        #region Constructors

        public VerifyOperationRequest()

        public VerifyOperationRequest(TransactionRequest request) : base(request)

        #endregion

        #region Properties

        public ProcessOperation Operation { get; }

        #endregion

        #region Methods

        public void AddTransactionRequest(TransactionRequest request)

        public string ToString()

        #endregion
    }
```

Manual Keyed Entry

Manual Keyed Entry is the process of running what would typically be a card present transaction without a payment terminal by manually entering the card information in order to process a transaction.

How to Run a Manual Keyed Entry Transaction

In order to run a manual keyed entry transaction the `KeyedEntry` property must be set to true in the `CardTransactionOperationRequest`. If the `KeyedEntry` property is set to true then the `ManualCardEntry` event will be fired and the `Pan`, `CvvCode`, and `ExpirationDate` properties in the `ManualCardEntryEventArgs` will need to be populated by the user. If the `Pan`, `CvvCode`, or `ExpirationDate` are not populated properly then the transaction will be cancelled; otherwise, the transaction will go on to be processed as a regular card present transaction.

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`. 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

Event Handlers

To simplify the implementation process, event handlers are included in the SDK. EVO Snap* highly recommends using the default event handlers for the initial connection to the Platform Services to ensure the password change dialogs are in place. A successful password change is required for the user account to process requests.

The sample code above utilizes the default event handlers, but custom dialogs can be created.

For additional information, please refer to the CommerceDriver™ *Technical Reference Guide for Windows®* or the Sample Application.

Core Assemblies

- **EvoSnap.CommerceDriver.Common.dll** – An assembly containing common code and data models as well as the main CommerceDriverController class.
- **Newtonsoft.Json.dll** - An assembly containing code required for serializing/de-serializing JSON models for REST request and responses.

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.