# CommerceDriver™

## Quick-Start Guide for *Windows*®

# 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

* *Commerce Driver*™ - v2.0.27
* EVOSnap* Web Services - v2.1.27 (Platform calls)
* Supported Terminal – Ingenico ICMP via Bluetooth

# Compatibility

* *CommerceDriver*™ Framework – Windows® 7+
* Visual Studio 2015
* .Net 4.5

# 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.

For our example, the following setup uses the EVOSnap* assembly for interacting with *Ingenico* terminals. The EvoSnap.CommerceDriver.Ingenico.dll assembly must be placed in the same folder as *CommerceDriver*™.

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.CommerceDriver.Ingenico.dll*
   * *Newtonsoft.Json.dll*
   * *RBA_SDK.dll*
   * *RBA_SDK_CS.dll*
4. Add the assembly references to the solution for the following files:
   * *EvoSnap.CommerceDriver.Common.dll*
5. Place the assemblies below into an associated \bin\debug output folder. (The assemblies above depend on the following assemblies.)
   * *RBA_SDK.dll*
   * *RBA_SDK_CS.dll*
6. Update the Configuration File settings to include the required terminal settings. e.g.: Interface Type or COM port values
7. 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();
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;

// 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

Update the Configuration File settings to include the UserName, Password, ServiceKey, ServiceID, ApplicationProfileID and VendorID supplied by EVOSnap*.

```
<evoSnap.commerceDriver>
  <platform>
    <accountService serviceKey="xxx" clientTimeout="70" promptDaysBeforeExpire="10">
      <credentials userName="xxx" password="xxx" />
      <defaults serviceID="xxx" merchantProfileID="" />
    </accountService>
    <paymentService applicationProfileID="xxx" vendorID="xxx" clientTimeout="70" />
    <transactionService clientTimeout="70" />
  </platform>
</evoSnap.commerceDriver>
```

Authentication can be perform manually via the *PerformIdentityLogin* method or by invoking the *ProcessAsync* method to process a transaction. In the sample code the *ProcessAsync* method is used.

# Terminal Setup

1. Initialize the CommerceDriverController.

```
try
{
    // Call the Initialize() method to load the manufacturer/terminal information
    Controller.Initialize();
}
catch (Exception ex)
{
    // An exception can be raised if a DLL or one of its dependancies are not found
    LaunchExceptionDialog(ex);
}
```

2. Identify the manufacturer and terminal model to use.

```
// Retrieve the Ingenico manufacturer definition
ManufacturerInfo manufacturer = Controller.Manufacturers.Find(m => m.Name == "Ingenico");

if (manufacturer != null)
{
    // Retrieve the iPP350 terminal definition
    string terminalName = manufacturer.TerminalNames.Find(t => t == "iPP350");

    if (terminalName != null)
    {
        try
        {
            Controller.SetTerminal(manufacturer.Name, terminalName, terminalName);
        }
        catch (Exception ex)
        {
            LaunchExceptionDialog(ex);
        }
    }
}
```

3. Initialize the terminal.

```
if (!Controller.IsTerminalReady)
{
    try
    {
        Controller.InitializeTerminal();
    }
    catch (Exception ex)
    {
      // An exception can be raised if a DLL or one of its dependancies are not found
    }
}
```

# 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

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;
    }
}
```

### Default Dialogs/Event Handlers

To simplify the implementation process, default dialogs and associated 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®*.

# 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.

# Supplementary Files

- **EvoSnap.CommerceDriver.TestClient.exe** – A Windows® desktop application used to create transactions. The file is dependent on *all* of the core assemblies.
- **EvoSnap.CommerceDriver.Extras.dll** – Sample assembly containing shared handlers, helpers and forms
- **EvoSnap.CommerceDriver.TestClient.exe.config** - A file containing the configuration settings for the *TestClient* application. *Note: To ensure the TestClient operates properly, use only the values obtained from EVOSnap* in the file.*

# 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.