

Payment Device SDK for Windows Integration Guide

For Software Version 3.12 (Myrddin)

Document Version: 2.8
Date: 22nd June 2023



TABLE OF CONTENTS

TABLE OF CONTENTS	1
1. How to Use This Guide	8
Caution	8
Key Point or Important Concept	8
Helpful Hint	8
Reference Material	8
Computer Text	8
References and Hyperlinks	8
Definitions	8
2. Introduction - Welcome to the Payment Device SDK for Windows	9
Payment Device SDK Software	10
3. What You Will Need	11
Payment Gateway Account	11
Payment Device SDK	11
Sandbox/Operating Environment	12
4. Transaction Quick Start	13
Transaction Quick Start Procedure	13
Ready the SDK Components	13
Configure	13
Execute	14
Troubleshoot	14
5. Integration Quick Start	15
Transaction Diagram	15
6. Configure and Install	16
Ready the SDK Components	16
Configure The Server	16

Configure The Server Data Storage	18
Configure The Server for SSL	18
Configure The Server for use with a Web Proxy	18
Install and Run The Server	19
Run ChipDNA Server as a Console Application	19
Run ChipDNA Server as a Windows Service	19
Configure and Run The Sample Client Applications	20
The Client CLI	20
Configure The Server for Production Processing	21
Terminal Management System (TMS)	21
7. Messaging System	22
Standard Transaction Messaging	23
Auto-confirm Transaction Messaging	24
8. Payment Methods	25
Initialization	25
ConnectAndConfigure	25
Start Transaction	26
Confirm Transaction	28
Void Transaction	29
Continue Signature Verification	30
Linked Refund Transaction	31
Terminate Transaction	32
Set Idle Message	33
9. Payment Events	34
Connect and Configure	34
Configuration Update	34
Transaction Update	34

Card Notification	35
Card Details	35
Signature Verification Requested	36
Transaction Finished	37
10. Utility Methods	40
Request TMS Update	40
Get Status	40
Get Transaction Information	42
Get Version	43
Get Merchant Data	44
11. Utility Events	46
Payment Device Availability Change	46
Tms Update	46
Request Queue Run Completed	47
12. Glossary of Terms	48
13. Troubleshooting & Support	49
Appendix 1. PIN pad Device ID Examples	50
Appendix 2. Processing of Transactions	53
Step 1 - Authorization	53
Step 2 - Settlement	53
Step 3 - Funding	54
Appendix 3. Supported PIN pads and Software Versions	55
Appendix 4. Firewall Configuration	58
Appendix 6. Supported PIN pads and Supported Features	60
Appendix 7. Supported PIN pads and Transaction Update Event Parameters	61

Document History

Document Version	Software Version	Date yyyy-mm-dd	Summary of Changes
1.18	Release 3.02 (Houdini)	2021-06-01	<p>Added Appendix 6 with a table containing each supported PIN pad and each corresponding supported feature.</p> <p>Added Appendix 7 with a table containing each supported PIN pad and each corresponding supported Transaction Update Event parameter value.</p>
1.19	Release 3.03 (Oasis)	2021-08-02	<p>Added support for the Ingenico Lane/3000, iPP350, iSelf, iSelf LE and iUC285 RAM devices for use in the UK and Europe and the Miura M020 device for use in the UK, Europe and US.</p> <p>Updated Table 5 - Added UK and Europe to the region column for Verifone Ux300 and UxFMTA devices.</p> <p>Updated Table 4 - EMV Contact (Chip) and EMV Contactless (Chip) are now Ready for the Ux300 and UxFMTA devices.</p>

1.20	Release 3.04 (Ultron)	2021-09-28	<p>Updated Table 7 and Table 8 – Miura M020 and M021 devices now support PAN Key Entry. Ingenico iSelf, iSelf LE and iUC285 RBA devices now support ApplicationSelectionStarted and AmountConfirmationStarted transaction update events and no longer support Card Removal Enforced. The iSelf and iSelf LE also support the PinEntryStarted event and the new MagstripeAccountSelectionStarted event. The Ingenico Lane/3000, Lane/5000 and Lane/7000 UPP devices now support all transaction update events.</p> <p>Clarified that macOS is not supported in Introduction - Welcome to the Payment Device SDK for Windows and Sandbox/Operating Environment.</p>
2.0	Release 3.05 (Lapis Lazuli)	2022-03-07	<p>Updated document template.</p> <p>SDK rebranded from 'ChipDNA SDK' to Payment Device SDK. Amendments made throughout the document as appropriate.</p> <p>Added values to the VOID_REASON parameter in Void Transaction.</p> <p>Updated Table 5 - Added 23k6 (23.52.6) for all Ingenico RBA devices and 6.8.2.21 for all Verifone VIPA devices.</p> <p>Updated Table 8 - MagstripeAccountSelectionStarted event is now supported with the Miura M020, Verifone Ux300 and Verifone UxFMTA devices.</p>

2.1	Release 3.06 (Nostromo)	2022-04-25	<p>Added Run Request Queue method and Request Queue Run Completed event.</p> <p>Clarified the Server's data storage requirements in Configure the Server Data Storage.</p>
2.2	Release 3.06 (Nostromo)	2022-06-17	<p>Updated Table 5 - Added 7.82.05 for Lane 3000/5000/7000 UPP devices.</p>
2.3	Release 3.07 (Prime)	2022-08-10	<p>Addition of battery status information in <code>PAYMENT_DEVICE_STATUS</code> response to Get Status.</p> <p>Error code returned by Transaction Finished renamed from <code>WhitelistedCardPresented</code> to <code>AllowlistedCardPresented</code>.</p> <p>Updated Table 5 - Added 2022 for Lane 3000 RAM devices.</p> <p>Updated Start Transaction parameters to include <code>CREDENTIAL_ON_FILE_FIRST_STORE</code> and <code>CREDENTIAL_ON_FILE_REASON</code></p>
2.4	Release 3.08 (Mastodon)	2022-09-26	<p>Added support for the Ingenico Self/4000 RAM device for use in the UK and Europe.</p>
2.5	Release 3.09 (Arcticus)	2022-11-07	<p>Removed support for the following devices:</p> <ul style="list-style-type: none"> • Ingenico iPP350 RAM • Ingenico iSC250 RBA • Ingenico iSelf RBA • VeriFone Mx925 XPI • VeriFone Mx915 XPI • VeriFone Vx820 XPI <p>Added Appendix 5 detailing how to enable Java client-side logging.</p>

			Added support for the Ingenico Self/4000 and Self/5000 UPP devices for use in the US.
2.6	Release 3.10 (Great Bear)	2023-01-23	<p>Updated Table 5 - Added 2238 for Self 4000 RAM devices.</p> <p>Added support for the Ingenico Self/2000 UPP device for use in the US.</p>
2.7	Release 3.11 (Lake Louise)	2023-03-13	<p>Added support for the Ingenico Self/2000 RAM device for use in the UK and Europe.</p> <p>Added Windows 11 as a supported operating system in Sandbox/Operating Environment</p> <p>Updated Table 5 - Added MPI 1-65 for Miura devices.</p>
2.8	Release 3.12 (Myrddin)	2023-06-22	Added support for the Ingenico Self/7000 with Self/8000 RAM device combination, for use in the UK and Europe.

1. How to Use This Guide

Throughout this guide you will notice a number of visual indicators and styles used to emphasize important information. These are explained below.

CAUTION



Critical information that must be obeyed to ensure success or avoid significant problems.

KEY POINT OR IMPORTANT CONCEPT



Key information that you should check you understand fully before continuing.

HELPFUL HINT



Hints and tips to help you act more effectively or avoid common mistakes.

REFERENCE MATERIAL



References to further information outside this guide, such as international standards and useful Internet addresses.

COMPUTER TEXT

Directories, filenames, commands, variables and the like are presented in-line like this:
C:\Payment Device SDK

...or in code blocks like this:

```
public ClientHelper(  
    String terminalID  
)
```

REFERENCES AND HYPERLINKS

Cross-references to other parts of this guide are clickable hyperlinks presented like this:
[How to Use This Guide](#)

References to Internet locations (URLs) or clickable hyperlinks are presented like this:
www.google.com

DEFINITIONS

Acronyms and terms which may be unfamiliar to you are listed and defined in the [Glossary of Terms](#), so you can look them up at any time.

2. Introduction - Welcome to the Payment Device SDK for Windows

What is the Payment Device SDK for? What are the benefits?

The SDK:

- Simplifies the use of EMV (EuroPay, MasterCard and Visa) chip cards in existing point-of-sale (POS) applications.
- Provides a high level of transaction security, through integration with our payment gateway and our end-to-end encryption technologies.
- Makes rapid deployment possible – it is easy to integrate, has been acquirer-tested and is pre-certified for certain combinations of PIN pads and acquirers.
- Requires little maintenance to ensure ongoing compliance and stability – it regularly obtains the latest configuration and PIN pad software updates automatically, via the Terminal Management System (TMS).

In summary, the SDK makes it as easy as possible for you to handle EMV transactions in your payment solution. For more background relating to the payment process see [Appendix 2. Processing of Transactions.](#)

What is the Payment Device SDK?

The SDK is a package of software and services from the payment gateway including:

- A software development kit (SDK) for Windows and Linux-based devices. macOS is not supported.
- Configuration and PIN pad software and updates, through integration with our TMS.
- Flexible integration choices for different operating environments through integration with a range of PIN pads



Pick PINpad



Select Processors



Integrate



Done!

What is the purpose of this guide?

This guide provides instructions to help you use the SDK to quickly and easily integrate EMV into your payment solution.

If you are unfamiliar with any of the acronyms or terms used in this guide, please see the [Glossary of Terms.](#)

Payment Device SDK Software

The SDK consists of two core components; the Server and the Client:

- The Server is the application that controls a PIN pad and communicates with the payment gateway.
- The Client is the application that makes requests for payment to the Server. It also receives feedback from the Server which may be routed as required and translated into messages for the user or operator.

3. What You Will Need

Payment Gateway Account

Register for a gateway account with your partner:

- 1) Once logged in, click on **Settings – Security Keys** and follow the on-screen instructions.
- 2) Your API Key will need to have 'API' source permissions only.



Keep the API Key details safe - you need them to configure the solution for processing transactions.

Payment Device SDK

The SDK is packaged and supplied as a .zip archive. The contents are described in [Table 1](#).

Table 1 – Contents of the SDK .zip archive

Folder Name	Description
ChipDNA API Documentation	The documentation for the Client Helper API (ChipDNA Framework .CHM reference files).
ChipDNA Client CLI	An example Client that sends payment requests to Server (Command Line).
ChipDNA Client CLI Source	The source code for an example Client that uses Server.
ChipDNA Client GUI	An example Client that sends payment requests to Server (Graphical).
ChipDNA Server	The application that controls the PIN pads and communicates with the payment gateway.



To view the content of any downloaded .CHM file you must first unblock it. Right-click on the file, click Properties → Unblock → OK. Then double-click the file to open and view as normal.



The Payment Device SDK was previously named 'ChipDNA SDK'

Sandbox/Operating Environment

Operating System:

- The Server will run on any device that supports the full version of Microsoft .NET Framework 4.7.2. All deployments should be using an operating system that is currently in vendor support, such as:
 - Windows 8.1
 - Windows 10
 - Windows 11
 - Windows Server 2012
 - Windows Server 2012 R2
- macOS is not supported.

Other software:

- A text editor, such as Windows Notepad or Notepad++.

Hardware:

- PIN pad: See [Appendix 4](#) for which PIN pads are supported by the Server.
- Chip card for test transactions.



If you do not already have a dedicated chip card for testing purposes you may use your own personal chip card - your details are safe in our PCI DSS Level 1 compliant environment and the sandbox environment cannot make real charges to your card.

4. Transaction Quick Start

This section explains how to quickly get the Server application communicating with an example Client so that transactions can be processed in a sandbox environment.

The Quick Start examples assume use of the following:

- PIN pad:
 - Type: Ingenico iPP320 RBA
 - Connection: USB (VCOM serial)
- Server/Client communication:
 - Protocol: TCP/IP
 - Port: 1869



For different options to suit your installation see [Configure and Install](#).



Firewall changes may be required so the Server can communicate with several internet services. Please see [Appendix 4. Firewall Configuration](#) for more details.

Transaction Quick Start Procedure

READY THE SDK COMPONENTS

These steps are required only once for each installation that you wish to run.

1. If you haven't already done so, create your API Key as described in [Payment Gateway Account](#).
2. Install the SDK by extracting the .zip archive to C:\Payment Device SDK.



You may use a different installation directory, but remember to substitute throughout the examples.

CONFIGURE

These steps are required only once for each installation that you wish to run.

- 1) Browse to C:\Payment Device SDK\ChipDNA Server.
- 2) Using your text editor, edit `chipdna.config.xml` at the following elements:
 - a) `ApplicationIdentifier="*****"`: Replace asterisks (*) with the Application Identifier or App Name.
 - b) `Id="*****"` in `PaymentDevice`: Replace asterisks with the ten-digit number that you will find on the screen of the PIN pad (after the "S/N:" prefix). For an example, see [Appendix 1. PIN pad Device ID Examples](#).
- 3) Check which RS-232 port the PIN pad is connected to. If it is not COM1 then at the element `Port` in `PaymentDevice` replace COM1 with the correct value.
- 4) Save the file and close the text editor.

EXECUTE

These steps are required each time you wish to run the Server and a sample Client.

- 1) Browse to `C:\Payment Device SDK\ChipDNA Server`.
- 2) Execute `ChipDNAServer.exe`.



When the Server runs it takes time to start up before it's ready to be used by a Client application. If run for the first time it must perform a TMS configuration update and each time its run it must connect to and initialize any configured PIN pads.

- 3) Using your text editor, edit `client.config.xml` and change the Terminal ID to the value obtained during registration. If the Server is listening on a port or protocol different to the default, then change the value as required.
- 4) Compile your preferred sample Client and then execute ensuring the configuration file `client.config.xml` is in the same directory.
- 5) The Client will present a list of the available commands. Input `P` for "Start Transaction" and then enter:
 - a) The amount in the minor value (for example: `123` for `1.23`). The currency is defined by the terminal configuration on our payment gateway.
 - b) The type of transaction or accept the default `sale` to process an Authorization transaction.
 - c) A unique reference (gateway Order ID) for the transaction or accept the auto-generated value.
- 6) At the PIN pad, you must now simulate the cardholder:



Take care when entering your PIN – if you enter it incorrectly 3 times your chip card may become locked just as it would in a real transaction attempt.

- a) Insert a chip card as prompted at the PIN pad. The test transaction will start.
 - b) Follow the prompts on the PIN pad until the test transaction is complete.
- 7) Check the sample Client console output to see the final status (either "Approved" or "Declined") along with the receipt details.
- 8) View the transaction details in your gateway reporting by logging in with your username and password.

TROUBLESHOOT

If you experience any difficulties we're here to help – please see [Troubleshooting & Support](#)

5. Integration Quick Start

In this section, we use a typical example to help you understand how to approach integration of the SDK into your payment solution. The majority of payments will be processed in the same way, as shown in the following Transaction Diagram.

Transaction Diagram

[Figure 1](#) shows the flow of a typical EMV transaction from your perspective as the integrator. For simplicity, the diagram assumes that the PIN is correct and no further authentication is required. There are of course other ways in which a transaction may be processed - these are detailed with suitable diagrams in the [Messaging System](#) section.

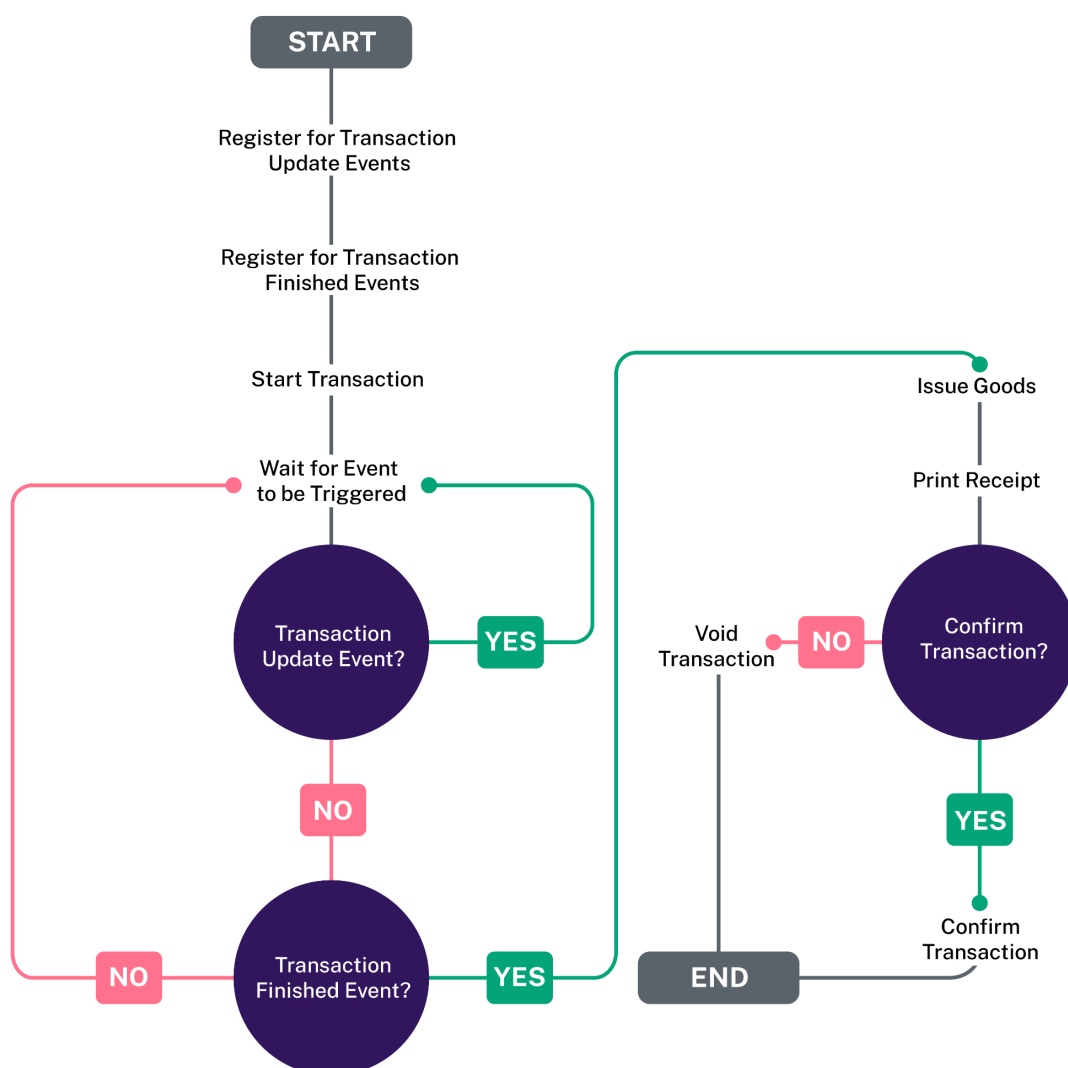


Figure 1 - Flow diagram for a typical EMV transaction.

6. Configure and Install

The [Transaction Quick Start](#) section uses a controlled example to demonstrate the simplicity of using the SDK. This section provides detailed information to help you get the SDK working in alternative environments and configurations.

Ready the SDK Components

If you have not already done so, register for a sandbox account and extract the SDK .zip archive as described in the [Ready the SDK Components](#) section.



Remember to substitute your installation directory in all examples if you are not using C:\Payment Device SDK.

Configure The Server

Before you can use the Server application you must modify the configuration XML file to suit your specific environment.

- 1) Browse to C:\Payment Device SDK\ChipDNA Server where you will find a sample configuration file (`chipdna.config.xml`) that you can adapt. Or for a completely fresh start, overwrite this with a copy of the blank template file also provided (`chipdna.config.xml.template`).
- 2) Using your text editor, edit `chipdna.config.xml` and change the values detailed in [Table 2](#) to match your environment. The path of the element is given using the XPath syntax.
- 3) When you have finished editing, save the file and close the text editor.

Table 2 – Server configuration options.

Path	Element	Value
/ChipDnaServer	ApplicationIdentifier	This must be a single word or acronym in UPPERCASE that uniquely identifies the integrating application. It should contain only the characters A-Z 0-9 and – and has a maximum length of 25. This value is used by the TMS platform to configure TMS properties specifically for an integrating application.
/ChipDnaServer	MachineName	This is a name used to identify the Server. This must pass the same validation as a DNS name and although the hostname of the machine can be used it does not have to be. The default for this value is <code>localhost</code> .
/ChipDnaServer/Terminals /Terminal/PaymentDevices /PaymentDevice	Id	Each PIN pad connected to the Server has a unique Device ID - generally an ID or serial number shown on the start-up screen or printed on the rear (for examples see Appendix 1. PIN pad Device ID Examples).

Path	Element	Value
/ChipDnaServer/Terminals /Terminal/PaymentDevices /PaymentDevice	DeviceActive	If you need to temporarily disable the configuration section for a device use one of the following to enable or disable a PIN pad for use with the Server: true = Enable. false = Disable.
/ChipDnaServer/Terminals /Terminal/PaymentDevices /PaymentDevice	Model	Use one of the following as appropriate to your hardware and region: Ingenico-iPP320-RBA Ingenico-iPP350-RBA Ingenico-iSelf-RAM Ingenico-iSelfLE-RBA Ingenico-iUC285-RAM Ingenico-iUC285-RBA Ingenico-Lane-3000-RAM Ingenico-Lane-3000-UPP Ingenico-Lane-5000-UPP Ingenico-Lane-7000-UPP Ingenico-Self-2000-RAM Ingenico-Self-2000-UPP Ingenico-Self-4000-RAM Ingenico-Self-4000-UPP Ingenico-Self-5000-UPP Ingenico-Self-7000-8000-RAM Miura-M020-MPI Miura-M021-MPI VeriFone-Ux300-VIPA VeriFone-UxFMTA-VIPA
/ChipDnaServer/Terminals /Terminal/PaymentDevices /PaymentDevice	Protocol	SERIAL
/ChipDnaServer/Terminals /Terminal/PaymentDevices /PaymentDevice	Port	This value must match the appropriate COM port, such as COM1. (See Configure for more on this).
/ChipDnaServer/Terminals /Terminal/PaymentDevices /PaymentDevice	Baudrate	This must match the baud rate defined in the configuration of the PIN pad. Please consult your PIN pad documentation for more information.
/ChipDnaServer/Terminals /Terminal/PaymentDevices /PaymentDevice	Standby Message	Enter a message to be displayed on the PIN pad when it is in standby/idle mode. The number of characters that can be displayed is device specific. [EOL] should be used to indicate line breaks in the display message.
/ChipDnaServer	Socket	Enter the IP address and port number (between 1024 and 65535) that ChipDNA Server is hosted on for communication. The two values should be concatenated using a colon. The default value is 127.0.0.1:1869.

CONFIGURE THE SERVER DATA STORAGE

The Server requires data storage for TMS configuration and transaction data. By default the Server will store data in AppData. The Server must have read/write access to AppData and AppData must persist.

The `chipdna.config.xml` file has an optional `FileStore` element which can be used to change the data storage location. The path to the storage location must be absolute. For example, to specify the path `C:\ProgramData\` the XML configuration file will contain the following:

```
<FileStore>
  <Location>C:\ProgramData\</Location>
</FileStore>
```

This additional configuration element should only be used if the integration requires a location different to the default.



The Server will not allow TMS updates or any payment methods to be performed if the disk space is below 10MB.



An alternative storage location must be used if the Server cannot access AppData or if AppData is frequently wiped as is the case when **UWF** is enabled.

CONFIGURE THE SERVER FOR SSL

The Server can be configured to use SSL to communicate with the Client(s). An additional element `CertificateHash` must be added in `chipdna.config.xml` and used to specify the Certificate Hash. For example:

```
<ChipDnaServer>
  <Socket>127.0.0.1:1869</Socket>
  <CertificateHash>
    72086bb5184ea4a4eef39b20e52566d87e44da6d
  </CertificateHash>
</ChipDnaServer>
```



The C++ client is dependent on OpenSSL. It is the integrator's responsibility to link and maintain the OpenSSL library and to ensure it is using the latest version. For Windows download the library from the OpenSSL website

CONFIGURE THE SERVER FOR USE WITH A WEB PROXY

The Server can be configured to use a web proxy to communicate with the payment gateway and TMS. An additional element `WebProxy` must be added in `chipdna.config.xml` and used to specify the IP address or hostname and port of the web proxy. If the web proxy requires authentication the username and password should also be specified in the `WebProxy` element. For example:

```
<ChipDnaServer>
  <WebProxy>
    <Connection>proxy.url.tld:3128</Connection>
    <Username>username</Username>
    <Password>password</Password>
  </WebProxy>
</ChipDnaServer>
```

Install and Run The Server

The Server can be executed as either a console application or installed as a Windows Service. Running it as a console application is useful during integration and debugging, however we would recommend installation as a Windows Service as it allows for automatic startup as a user that is not logged in. The following sections detail how to install and use the Server in either mode.

RUN CHIPDNA SERVER AS A CONSOLE APPLICATION

The following instructions detail how to run the Server as a console application. They assume that the Server has been configured as described in the [Configure The Server](#) section.

- 1) Browse to C:\Payment Device SDK\ChipDNA Server.
- 2) Execute ChipDNAServer.exe.

RUN CHIPDNA SERVER AS A WINDOWS SERVICE

The following instructions detail how to run the Server as a Windows Service. They assume that the Server has been configured as described in the [Configure The Server](#) section and the user has administrative privileges.

- 1) Browse to C:\Payment Device SDK\ChipDNA Server.
- 2) Using your text editor, edit service.install.bat and make sure that binpath shows the correct full path for ChipDNAServer.exe (in our examples this is C:\Payment Device SDK\ChipDNA Server\ChipDNAServer.exe).
- 3) Execute service.install.bat.



If for any reason the installation fails, use the command prompt to run the following command which will uninstall any previous Server Service installations: `sc delete CreditcallChipDNAServer`. Then try installing again.

- 4) By default, the CreditcallChipDNAServer service is installed to run under the 'Local System' account and to be started manually.



If you need the service to start automatically when the machine reboots, open the management console for Windows services and change the properties of the ChipDNA Server Service from Manual to Automatic.



The management console for Windows services can be found at Control Panel → Administrative Tools → Services.

- 5) If the ChipDNA Server Service is not running, open the management console for Windows services, select the ChipDNA Server Service and click 'Start'. Or, at the command prompt run the following command: `net start CreditcallChipDNAServer`.


Configure and Run The Sample Client Applications

A sample command line interface Client is provided. The following items detail how to configure and execute those applications.

THE CLIENT CLI

To configure and run the sample Client CLI:

- 1) Make sure that the Server is configured and running as detailed in [Install and Run The Server](#).
- 2) Browse to `C:\Payment Device SDK\ChipDNA Client CLI`.
- 3) Using your text editor, edit `client.config.xml` and change the Terminal ID to the value obtained during registration. If the Server is listening on a port or protocol different to the default, then change the value as required.
- 4) Run the Client CLI using `ChipDNAClient.exe`.
- 5) The Client CLI will present a list of the available commands. Input `P` for "Start Transaction" and then enter:
 - a) The amount in the minor value (for example: 123 for 1.23). The currency is defined by the terminal configuration on our payment gateway.
 - b) The type of transaction or accept the default `sale` to process a Sale transaction.
 - c) A unique reference (gateway Order ID) for the transaction or accept the auto-generated value.
- 6) At the PIN pad, you must now simulate the cardholder:

 Take care when entering your PIN – if you enter it incorrectly 3 times your chip card may become locked just as it would in a real transaction attempt.

 - a) Insert a chip card as prompted at the PIN pad. The test transaction will start.
 - b) Follow the prompts on the PIN pad until the test transaction is complete.
- 7) Check the Client CLI to see the final status (either "Approved" or "Declined").

Configure The Server for Production Processing

By default, the Server is configured to use the sandbox environment. This section details the configuration changes required for production processing. When a merchant is boarded on the payment gateway production platform they will need to generate an API Key under **Settings – Security Keys**. These values should be specified in the XML configuration file.

The XML configuration file also includes details for the TMS platform which must be edited to use the gateway's production platform. It should be modified to contain the following:

```
<Tms>
  <Servers>
    <Server use="tms">
      <Url>https://tms.cardeasexml.com</Url>
      <Timeout>45000</Timeout>
    </Server>
    <Server use="registration">
      <Url>https://live.cardeasexml.com/gw.cex</Url>
      <Timeout>45000</Timeout>
    </Server>
  </Servers>
</Tms>
```

Firewall changes may be required so the Server can communicate with several internet services. Again, these must be changed to use the production platform instead of the sandbox environment. Please see [Appendix 5. Firewall Configuration](#) for more details.

Terminal Management System (TMS)

The Server regularly contacts TMS in order to download additional configuration data, including EMV transaction settings, such as: TDOL, DDOL, Floor Limits, and Certificate Authority Public Keys. The TMS also provides application configuration data, such as timeouts and external connectivity options. The TMS can also provide firmware upgrades for PIN pads as required by certification, processors and schemes.

The Server automatically connects to the Terminal Management System at regular time intervals, for example every 24 hours (this process does not affect processing transactions). If after a TMS update a configuration update of the PIN pads is required the Server must interrupt processing transactions to allow the PIN pad update to happen. This will only happen during a specified time slot which is also configurable in TMS. The default value for this is 3am.

The Server stores the configuration data downloaded from TMS on the local disk.

7. Messaging System

The following sections discuss the messaging between the Client Helper and the POS application. In a transaction, the POS software initiates a transaction with the Client Helper and it updates the POS software on the progress of the transaction:

- **Standard transaction** – In a standard Chip transaction, the POS software initiates a transaction with the Client Helper and it updates the POS software on the progress of the transaction.
- **Auto-confirm transaction** – In an auto-confirm transaction, the POS software initiates a transaction with the Client Helper and it updates the POS software on the progress of the transaction. The installation will Confirm or Void the transaction and, if required, request signature verification before sending the TransactionFinished event. Errors can occur which require the POS to Confirm or Void

After calling the `StartTransaction()` method, the Client application may listen for the following events:

- `TransactionUpdate` – Fired throughout the transaction to update the POS software on the progress of the transaction.
- `CardNotification` – Fired when there is a change in the card status such as card provided or removed.
- `CardDetails` – Fired when card details are available.
- `SignatureVerificationRequested` – Fired when signature verification is required during an auto-confirm transaction.
- `TransactionFinished` – Fired when the transaction is completed on the PIN pad.

The installation does not perform signature verification during a transaction. If the card verification method of signature is required for an approved transaction then this is indicated in the `TransactionFinished` event. It is then up to the POS software to verify the cardholder's signature. If the signature is verified the transaction should be confirmed but if signature verification fails then the transaction should be voided.

The Client application may also listen for the `PaymentDeviceAvailabilityChange` event which is fired when a PIN pad is connected or disconnected. The Client application may receive this event at any time, not only when a transaction is in progress.

The following sections describe the messaging process for each of these flows in greater detail.

For complete details on all of the methods and properties of the Client Helper, see reference file `C:\Payment Device SDK\ChipDNA API Documentation\ChipDNA Framework.chm`.

Standard Transaction Messaging

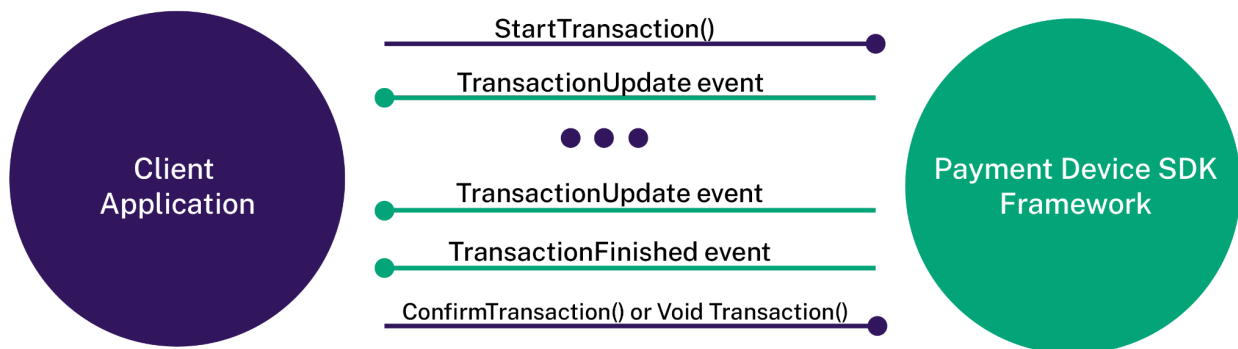


Figure 2 - Standard transaction messaging process.

The standard transaction messaging process follows these steps:

- 1) The Client application calls `StartTransaction()` (see [Start Transaction](#)) with the amount of the transaction and a unique reference which identifies the transaction to the Client application. It should also specify the type of transaction; Sale (Authorization) or Refund.
- 2) As the customer progresses through the transaction, the Client Helper sends `TransactionUpdate` events (see [Transaction Update](#)) describing the progress of the transaction, including EMV commands and data communication.
- 3) `CardNotification` and `CardDetails` events will be fired during a transaction when the data they provide is available.
- 4) When the customer has completed the transaction, the Client Helper sends the `TransactionFinished` event (see [Transaction Finished](#)).
 - a) If the transaction was successful, the `TransactionFinished` event contains the transaction information required to print a receipt and, if approved, whether signature verification is required.
 - b) If the transaction was not successful, the `TransactionFinished` event contains error information.
- 5) In order to be funded for an approved transaction, the Client application calls the `ConfirmTransaction()` method (see [Confirm Transaction](#)). If signature verification or goods issue fails for an approved transaction then the Client application calls the `VoidTransaction()` method (see [Void Transaction](#)). Declined or terminated transactions do not require the Client to call `ConfirmTransaction()` or `VoidTransaction()`. If a transaction was approved online but the final transaction result is declined or terminated then the Server will handle the void automatically before returning the result to the Client.



After authorization you must CONFIRM - otherwise the transaction will not be settled and the merchant will not receive funds.



The Server may decline a transaction after an authorization and attempt to send a reversal automatically. If this fails for any reason, a `VoidRequestFailed` error is sent in the

TransactionFinished event. The integration must VOID this transaction.

Auto-confirm Transaction Messaging

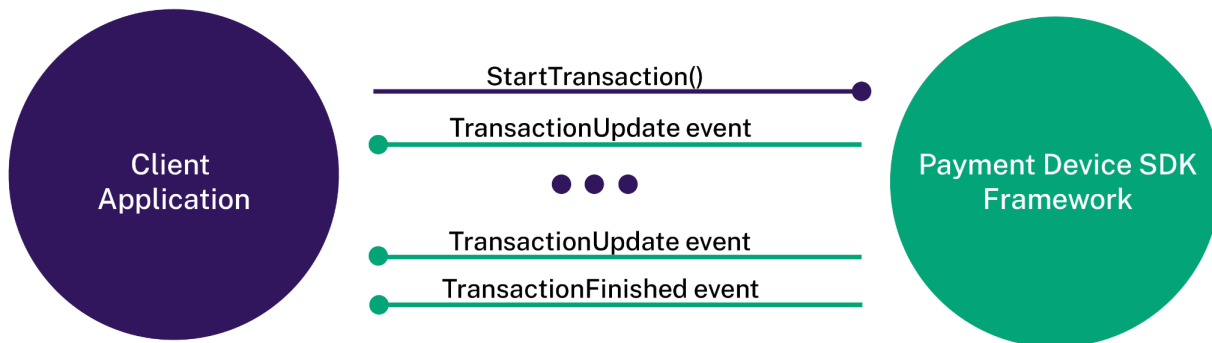


Figure 3 - Auto-confirm transaction messaging process.

The auto-confirm transaction messaging process follows these steps:

- 1) The Client application calls `StartTransaction()` (see [Start Transaction](#)) with the amount of the transaction, a unique reference which identifies the transaction to the Client application and auto-confirm set to true. It should also specify the type of transaction; Sale (Authorization) or Refund.
- 2) As the customer progresses through the transaction, the Client Helper sends `TransactionUpdate` events (see [Transaction Update](#)) describing the progress of the transaction, including EMV commands and data communication.
- 3) `CardNotification` and `CardDetails` events will be fired during a transaction when the data they provide is available.
- 4) `SignatureVerificationRequested` event may be fired during the transaction, before the `TransactionFinished` event (see [Signature Verification Requested](#)).
- 5) When the customer has completed the transaction, the Client Helper sends the `TransactionFinished` event (see [Transaction Finished](#)).
 - a) If the transaction was successful, the `TransactionFinished` event contains the transaction information required to print a receipt and, if approved, whether signature verification is required.
 - b) If the transaction was not successful, the `TransactionFinished` event contains error information.



If the Server fails to confirm a transaction for any reason, a `ConfirmRequestFailed` error is sent in the `TransactionFinished` event. The integration must **CONFIRM this transaction otherwise the transaction will not be settled and the merchant will not receive funds. This transaction can be **VOIDED** if required.**



If the Server fails to void a transaction for any reason, a `VoidRequestFailed` error is sent in the `TransactionFinished` event. The integration must **VOID this transaction.**

8. Payment Methods

Initialization

Your application interacts with the Server using a `ClientHelper` object instance. If multiple Client applications are running on a POS Server, each Client application must initialize its own instance of `ClientHelper`. Before beginning the messaging process, you must initialize a new `ClientHelper` instance using the `ClientHelper` constructor.

```
public ClientHelper(  
    string serverAddress, int serverPort, string serverSslHostName,  
    string apiKey  
)
```

This method takes the following parameters:

Parameter	Description/Value
apiKey	The API Key value you created after logging into your gateway account under Settings – Security Keys .
serverAddress	String. The IP address or hostname of the Server.
serverPort	Integer. The port number of the Server.

This method returns a `ClientHelper` object instance that you will use to interact with the Server.

ConnectAndConfigure

Call `ConnectAndConfigure()` to connect to the PINpad and configure the Server to be ready for transactions.

```
public Response ConnectAndConfigure (  
    List<Parameters> parameters  
)
```

`ConnectAndConfigure()` does not take any parameters. If `Response` does not contain errors, the Server will send updates via the `ConnectAndConfigure` event and will fire a `ConfigurationUpdate` event once the process is complete.

`ConnectAndConfigure()` returns a `Response` object instance containing a parameter list with the following name-value pairs:

Parameter	Description/Value	Presence
ERRORS	Error codes in comma separated values format.	On error

Start Transaction

Call `StartTransaction` to initiate a transaction using the Server:

```
public Response StartTransaction(  
    List<Parameters> parameters  
)
```

This can be used to start a sale, refund or account verification transaction.

`StartTransaction()` takes the following parameter name-value pairs:

Name	Description/Value	Presence
AMOUNT	String. Value of the transaction in the minor units of the currency used (e.g., pence or cents). For example, submit 123 for 1.23. The currency is defined by the terminal configuration on our payment gateway.	For Sale and Refund transactions only
AMOUNT_TYPE	Actual or Estimate. An amount specified as Actual will not be expected to change between Auth and Confirm (Capture).	Always
REFERENCE	Printable ASCII characters excluding the following characters "<> :*/\.". Maximum length of 50 characters and may not contain leading or trailing spaces. This is used as a transaction identifier for the Client application and must be a value unique to the transaction. It is also stored as an 'Order ID' in the payment gateway reporting system.	Always
BATCH_REFERENCE	Printable ASCII characters with a maximum length of 50 characters and may not contain leading or trailing spaces. This can be used to group transactions together.	As needed
TRANSACTION_TYPE	Sale or Refund. A Sale transaction functions like an 'Authorization' in the gateway system. It must be confirmed (captured) to settle.	Always
PAN_KEY_ENTRY	Requests a PAN key entry transaction is started. The card details are keyed into the PIN pad and should only be used for a Card Not Present transaction. This value should be set to True or False.	As needed

Name	Description/Value	Presence
CARDHOLDER_ADDRESS	Unicode characters excluding ASCII control characters. Maximum length of 330 characters and may not contain leading or trailing spaces. This is used to provide the cardholder address for PAN key entry transactions.	As needed for PAN key entry transactions only
CARDHOLDER_ZIPCODE	Unicode characters excluding ASCII control characters. Maximum length of 16 characters and may not contain leading or trailing spaces. This is used to provide the cardholder's ZIP code for PAN key entry transactions.	As needed for PAN key entry transactions only
TIPPING_SUPPORT	Default, None, OnDevice, EndOfDay or Both may be used to enable or disable the tipping method for a sale transaction. If not present or Default the configuration from TMS will be used. OnDevice, EndOfDay or Both may only be used if these tipping methods are enabled in TMS.	As needed for Sale transactions only (excluding PAN key entry)
CUSTOMER_VAULT_COMMAND	add-customer	As needed.
CUSTOMER_VAULT_ID	Up to 36 character string, ex: 12345ABCDE	As needed for Customer Vault Commands only
MERCHANT_DEFINED_FIELD_01 MERCHANT_DEFINED_FIELD_02 ... MERCHANT_DEFINED_FIELD_19 MERCHANT_DEFINED_FIELD_20	Up to 255 characters	As needed
BILLING_ADDRESS_1 BILLING_ADDRESS_2 BILLING_CITY BILLING_STATE BILLING_POSTAL_CODE BILLING_ZIP_CODE BILLING_COUNTRY BILLING_EMAIL_ADDRESS BILLING_PHONE_NUMBER	Up to 255 characters	As needed
PO_NUMBER	Printable ASCII characters excluding the following characters "<> :*/\.". Maximum length of 50 characters and may not contain leading or trailing spaces	As needed
TAX_AMOUNT	See AMOUNT	As needed
AUTO_CONFIRM	True or False..	As needed

Name	Description/Value	Presence
CREDENTIAL_ON_FILE_FIRST_STORE	True or False.	As needed for Sale transactions only
CREDENTIAL_ON_FILE_REASON	Unscheduled, Installment, Incremental, Resubmission, DelayedCharge, ReAuth or NoShow should be used to indicate the reason for a credential on file first store transaction	As needed for Sale transactions when CREDENTIAL_ON_FILE_FIRST_STORE is True



As mentioned previously the `REFERENCE` parameter is used as a transaction identifier for the Client application. It is used locally by the Server to find the transaction when performing a confirm, void or linked refund operation. It is also used in gateway reporting as the unique Platform ID value which can be used to look up the transactions. Therefore, the value supplied must be unique to each transaction.

`StartTransaction()` returns a `Response` object instance containing a parameter list with the following name-value pairs:

Name	Description/Value	Presence
ERRORS	Error codes in comma separated values format.	On error

Confirm Transaction

To finalize a transaction after it has been approved so that the transaction will be settled, you must call `ConfirmTransaction()`. If the data in the [TransactionFinished](#) event indicated that a signature was required, calling `ConfirmTransaction()` is also a confirmation that the signature passed verification.

You do not have to call `ConfirmTransaction()` immediately after the transaction has been approved. For example, you may want to authorize multiple cards for a single purchase, in which case you would call `ConfirmTransaction()` for each approval after all transactions have been authorized.

```
public Response ConfirmTransaction(
    List<Parameters> parameters
)
```

`ConfirmTransaction()` takes the following parameter name-value pairs:

Name	Description/Value	Presence
REFERENCE	The reference of the transaction to confirm. Printable ASCII characters excluding the following characters "<> :*/\.". Maximum length of 50 characters and may not contain leading or trailing spaces. This should be the reference provided by the TransactionFinished event.	Always
AMOUNT	String value of the amount in the minor units of the currency used (e.g., pence or cents) if different from authorized amount. For example, submit 123 for 1.23. The currency is defined by the terminal configuration on our payment gateway.	As needed

`ConfirmTransaction()` returns a `Response` object instance containing a parameter list with the following name-value pairs:

Name	Description/Value	Presence
TRANSACTION_RESULT	Either Approved or Declined	Always
ERRORS	Error codes in comma separated values format.	On Declined
RECEIPT_DATA	Elements required for receipting in XML format.	On Approved

If the transaction result is declined, it is the responsibility of the integrating application to retry until it is approved.

Void Transaction

To void a transaction before settlement, but after it has been approved or confirmed so that funding does not take place, call `VoidTransaction()`. If the data in the `TransactionFinished` event indicates that signature was required, and the signature verification fails then `VoidTransaction()` must be called.

```
public Response VoidTransaction(
    List<Parameters> parameters
)
```

`VoidTransaction()` takes the following parameter name-value pairs:

Name	Description/Value	Presence
REFERENCE	The reference of the transaction to void. Printable ASCII characters excluding the following characters "<> :*/\.". Maximum length of 50 characters and may not contain leading or trailing spaces. This should be the reference provided by the TransactionFinished event.	Always

Name	Description/Value	Presence
VOID_REASON	Value must be one of predefined Void Reasons. Currently supported values: SignatureDeclined TransactionFailure PrintFailure FulfillmentFailure StorageFailure	As needed

`VoidTransaction()` returns a `Response` object instance containing a parameter list with the following name-value pairs:

Name	Description/Value	Presence
TRANSACTION_RESULT	Either Approved or Declined	Always
ERRORS	Error codes in comma separated values format.	On Declined
RECEIPT_DATA	Elements required for receipting in XML format.	On Approved

If the transaction result is declined, it is the responsibility of the integrating application to retry until it is approved.

Continue Signature Verification

`ContinueSignatureVerification()` must be sent after a `SignatureVerificationRequested` event in order to verify the signature and finish the transaction. The Server will finalize the transaction and send the `transactionFinished` event after receiving this command. This command and its associated event are only used during auto-confirm transactions.

```
public Response ContinueSignatureVerification(
    List<Parameters> parameters
)
```

`ContinueSignatureVerification()` takes the following parameter name-value pairs:

Name	Description/Value	Presence
SIGNATURE_VERIFICATION_RESULT	True or False	Always

`ContinueSignatureVerification()` returns a `Response` object instance containing a parameter list with the following name-value pairs:

Name	Description/Value	Presence
ERRORS	Error codes in comma separated values format.	On error

Linked Refund Transaction

The Client can call `LinkedRefundTransaction()` to refund all or part of a previously approved and confirmed transaction.

```
public Response LinkedRefundTransaction(  
    List<Parameters> parameters  
)
```

`LinkedRefundTransaction()` takes the following parameter name-value pairs:

Name	Description/Value	Presence
AMOUNT	String value of the transaction in the minor units of the currency used (e.g., pence or cents). For example, submit 123 for 1.23. The currency is defined by the terminal configuration on our payment gateway.	Always
REFERENCE	Printable ASCII characters excluding the following characters "<> :*/\ ". Maximum length of 50 characters and may not contain leading or trailing spaces. This is used as a transaction identifier for the Client application and must be a value unique to the transaction.	Always
SALE_REFERENCE	The reference of the transaction to refund. Printable ASCII characters excluding the following characters "<> :*/\ ". Maximum length of 50 characters and may not contain leading or trailing spaces. This should be the reference provided by the <code>TransactionFinished</code> event.	Always
SALE_DATE_TIME	The date and time of the original transaction. This is in the format <code>yyyyMMddHHmmss</code> but only matches what is specified for example <code>yyyyMMdd</code> , <code>yyyyMMddHH</code> etc. A minimum of the date <code>yyyyMMdd</code> must be specified. Although an optional parameter supplying this improves the performance of the retrieval of the original sale transaction.	As needed

`LinkedRefundTransaction()` returns a `Response` object instance containing a parameter list with the following name-value pairs:

Name	Description/Value	Presence
TRANSACTION_RESULT	Either Approved or Declined	On completed transaction, except when terminated
RECEIPT_DATA	Elements required for receipting in XML format.	On completed transaction, except when terminated
TRANSACTION_ID	Numeric Gateway transaction ID.	On completed transaction, except when terminated
ERRORS	Error codes in comma separated values format.	On Declined or termination

Terminate Transaction

To cancel a transaction in progress, call `TerminateTransaction()`. If the transaction has finished before `TerminateTransaction()` is called and the result is Approved then `VoidTransaction()` can be used to cancel the transaction.

```
public Response TerminateTransaction(
    List<Parameters> parameters
)
```

`TerminateTransaction()` takes the following parameter name-value pairs:

Name	Description/Value	Presence
TERMINATE_DISPLAY_MESSAGE	Text to display on PIN pad screen. [EOL] should be used to indicate line breaks in the display message.	As needed
TERMINATE_REASON	Value must be one of predefined Terminate Reasons Currently supported values: EPOSTerminated	As needed

`TerminateTransaction()` returns a `Response` object instance containing a parameter list with the following name-value pairs:

Name	Description/Value	Presence
ERRORS	Error codes in comma separated values format.	On error

Set Idle Message

Call `SetIdleMessage()` to set the message display on PIN pad screens when they are in idle state.

```
public Response SetIdleMessage(  
    List<Parameters> parameters  
)
```

`SetIdleMessage()` takes the following parameter name-value pairs:

Name	Description/Value	Presence
IDLE_MESSAGE	Text to display on PIN pad screen. [EOL] should be used to indicate line breaks in the display message.	Always

`SetIdleMessage()` returns a `Response` object instance containing a parameter list with the following name-value pairs:

Name	Description/Value	Presence
ERRORS	Error codes in comma separated values format.	On error



Most devices display an additional idle message which is configured via TMS.

9. Payment Events

Connect and Configure

The Client Helper fires the `connectAndConfigure` event after `connectAndConfigure` is called and once the configuration process is complete.

```
public event EventHandler<EventParameters> ConnectAndConfigureEvent
```

The `EventParameters` contains a parameter list with the following name-value pairs:

Name	Description/Value	Presence
CONNECT_AND_CONFIGURE_RESULT	Will be one of the predefined values: Success Failure	Always
ERRORS	Comma separated list of error codes.	On error

Configuration Update

The Client Helper fires the `configurationUpdate` event after `connectAndConfigure` is called and provides updates on the current state of the process.

```
public event EventHandler<EventParameters> ConfigurationUpdateEvent
```

The `EventParameters` contains a parameter list with the following name-value pairs:

Name	Description/Value	Presence
CONFIGURATION_UPDATE	Will be one of the predefined values: ConnectAndConfigureStarted Registering	Always

Transaction Update

The Client Helper fires `TransactionUpdate` events as the customer progresses through the transaction. Each transaction update event describes the action that triggered the event, including EMV commands and data communication. The POS application can react to each event as needed, such as to update a display.

```
public event EventHandler<EventParameters> TransactionUpdateEvent
```

The `EventParameters` contains a parameter list with the following name-value pairs:

Name	Description/Value	Presence
UPDATE	Will be one of the predefined values: Unknown ApplicationSelectionStarted AmountConfirmationStarted CardRequested CardRemovalRequested MagstripeAccountSelectionStarted OnlineAuthRequested OnlineAuthCompleted PanKeyEntryStarted PanKeyEntryCompleted PinEntryStarted TippingRequested TransactionStarted VoiceReferralCompleted ZipCodeRequested	Always
PAYMENT_DEVICE_MODEL	The model of the PIN pad raising the event.	Always
PAYMENT_DEVICE_IDENTIFIER	The identifier of the PIN pad raising the event.	Always

Card Notification

When `StartTransaction()` has been successfully called, the Client Helper fires `CardNotification` events about card availability.

```
public event EventHandler <EventParameters> CardNotificationEvent
```

The `EventParameters` contains a parameter list with the following name-value pairs:

Name	Description/Value	Presence
NOTIFICATION	Will be one of the predefined values: Inserted Tapped Swiped Removed	Always
PAYMENT_DEVICE_MODEL	The model of the PIN pad raising the event.	Always
PAYMENT_DEVICE_IDENTIFIER	The identifier of the PIN pad raising the event.	Always

Card Details

When `StartTransaction()` has been successfully called, the Client Helper fires `CardDetails` events when the details are available. When `GetCardDetails()` has been

successfully called, the Client Helper fires a `CardDetails` event when the process is finished (instead of a `TransactionFinished` event).

```
public event EventHandler <EventParameters> CardDetailsEvent
```

The `EventParameters` contains a parameter list with the following name-value pairs:

Name	Description/Value	Presence
TRACK2_CLEAR_TEXT	Full Track 2 Data where end-to-end encryption is enabled and the card is allowlisted.	As needed
TRACK2_MASKED	Track 2 Data masked except for first 6 and last 4 digits of PAN, expiry date and service code.	As needed
PAN_CLEAR_TEXT	Full PAN where end-to-end encryption is not enabled. This has a maximum length of 19 digits.	As needed
PAN_MASKED	PAN masked except for first 6 and last 4 digits.	As needed
EXPIRY_DATE	The expiry date of the card in the format YYMM.	As needed
CARD_HASH_COLLECTION	Collection of card hashes available in XML format. Each hash is a unique reference that can be used to identify a card without using the PAN. The source of each hash indicates where it was generated for example the payment gateway or the PIN pad.	As needed

Signature Verification Requested

During an auto-confirm transaction, the Client Helper will fire this event before sending the `TransactionFinished` event if signature verification is required.

```
public event EventHandler <EventParameters>  
SignatureVerificationRequestedEvent
```

The `EventParameters` contains a parameter list with the following name-value pairs. The transaction will be Approved or Declined and a receipt can be issued.

Name	Description/Value	Presence
TRANSACTION_RESULT	Approved or Declined.	On completion post <code>CardDetails</code> event
RECEIPT_DATA	Elements required for receipting in XML format.	On completion post <code>CardDetails</code> event

When this event is received the `ContinueSignatureVerification()` command must be sent in order to finish the transaction. `TerminateTransaction()` is also valid at this stage and will decline the transaction as if the signature was rejected.

Transaction Finished

In any of the three messaging sequences, when `StartTransaction()` has been successfully called, the Client Helper fires a `TransactionFinished` event when the transaction is finished.

```
public event EventHandler<EventParameters> TransactionFinishedEvent
```

If the transaction is finished after the `CardDetails` event, the `EventParameters` contains a parameter list with the following name-value pairs. The transaction will be `Approved` or `Declined` and a receipt can be issued for the completed transaction. If the transaction is finished before the `CardDetails` event, the `EventParameters` contains only the `Errors` parameter indicating the transaction has been terminated. The `Errors` parameter contains a list of error codes detailing the errors that prevented the transaction from being completed. See the enumeration in the ChipDNA Framework reference CHM file for a complete list of error messages.

Name	Description/Value	Presence
TRANSACTION_RESULT	Approved or Declined.	On completion post <code>CardDetails</code> event
RECEIPT_DATA	Elements required for receipting in XML format.	On completion post <code>CardDetails</code> event
REFERENCE	Printable ASCII characters excluding the following characters "<> :*/\.". Maximum length of 50 characters and may not contain leading or trailing spaces.	On completion post <code>CardDetails</code> event
CARDEASE_REFERENCE	ASCII 36 characters in the format of a GUID. Unique transaction generated by the payment gateway.	On completion post <code>CardDetails</code> event if authorization was submitted online
CARD_HASH	Unique token generated by the payment gateway that can be used to identify a card without using the PAN.	On completion post <code>CardDetails</code> event if authorization was submitted online
CARD_REFERENCE	Unique reference generated by the payment gateway that can be used to identify a card without using the PAN.	On completion post <code>CardDetails</code> event if authorization was submitted online

Name	Description/Value	Presence
AUTH_DATE_TIME	Date and Time for the transaction in the format yyyyMMddHHmmss.	On completion post CardDetails event
TOTAL_AMOUNT	Total amount for the transaction in minor units.	On completion post CardDetails event
PAN_MASKED	The obfuscated Primary Account Number showing only the first 6 (if available) and last 4 digits.	On completion post CardDetails event
EXPIRY_DATE	The expiry date of the card in the format YYMM.	On completion post CardDetails event
SIGNATURE_VERIFICATION_REQUIRED	True or False.	On completion post CardDetails event
SIGNATURE_CAPTURED	True or False.	On completion post CardDetails event
SIGNATURE_IMAGE	Base 64 encoded raw image data.	If signature was captured on PIN pad
SIGNATURE_IMAGE_MEDIA_TYPE	The media type of the raw image data once decoded.	If signature was captured on PIN pad
CARD_HASH_COLLECTION	Collection of card hashes available in XML format. Each hash is a unique reference that can be used to identify a card without using the PAN. The source of each hash indicates where it was generated, for example the payment gateway or the PIN pad.	On completion post CardDetails event if authorization was submitted online
TRANSACTION_ID	Numeric Gateway transaction ID	On completion post CardDetails event
CUSTOMER_VAULT_ID	Up to 36 character string, ex: 12345ABCDE	As needed on completion post CardDetails event
CARD_HOLDER_FIRST_NAME	Card holder first name retrieved from the card.	On completion post OnlineAuthCompleted TransactionUpdate event

Name	Description/Value	Presence
CARD_HOLDER_LAST_NAME	Card holder last name retrieved from the card.	On completion post OnlineAuthCompleted TransactionUpdate event
ERRORS	Error codes in comma separated values format.	On decline or completion pre CardDetails event



When a transaction using an allowlisted card is finished, the **ERRORS** parameter includes an error code named **AllowlistedCardPresented**. In versions of ChipDNA Server prior to 3.07, this error code was named **WhitelistedCardPresented**.

When a transaction is completed the receipt data is returned in the parameter **RECEIPT_DATA**. This is returned in XML so a helper method **GetReceiptDataFromXml()** is provided to extract this data into a **ReceiptData** object which can be used by the integrator to generate receipts. For each receipt entry the following is provided:

- 1) **ID** – used to identify each entry.
- 2) **Label** – recommended label that should appear on receipt.
- 3) **Value** – the value for this entry.
- 4) **Type** – either **Mandatory**, **Optional** or **Debug** according to guidelines regarding receipt content. All **Mandatory** items must be shown on the receipt.
- 5) **Priority** – order in which items should appear on the receipt according to best practices. The priority is a guideline only and the order may be rearranged.

10. Utility Methods

Request TMS Update

Call `RequestTmsUpdate()` to request the Server connects to TMS to perform an update.

```
public Response RequestTmsUpdate(  
    List<Parameters> parameters  
)
```

`RequestTmsUpdate()` takes the following parameter name-value pairs:

Name	Description/Value	Presence
TMS_REQUEST_TYPE	TmsConfiguration (default).	As needed
UPDATE_TYPE	Either Partial (default) or Full.	As needed
CONFIGURATION_UPDATE_ SCHEDULE	Either Immediate (default) or MaintenanceTime.	As needed

`RequestTmsUpdate()` returns a `Response` object instance containing a parameter list with the following name-value pairs:

Name	Description/Value	Presence
ERRORS	Error codes in comma separated values format.	On error



It is recommended that all integrations expose the TMS update mechanism. This allows updated TMS properties to be downloaded immediately without the need to wait for a scheduled update.

Get Status

Call `GetStatus()` to get the current status of different components of the Server in a single call. Individual statuses can be requested by passing the parameter keys into `GetStatus()`. All statuses will be returned if `parameters` is empty.

```
public Response GetStatus(  
    List<Parameters> parameters  
)
```

`GetStatus()` takes the following parameter name-value pairs:

Name	Description/Value	Presence
CHIPDNA_STATUS	CHIPDNA_STATUS	As needed
VERSION_INFORMATION	VERSION_INFORMATION	As needed
PAYMENT_PLATFORM_STATUS	PAYMENT_PLATFORM_STATUS	As needed

Name	Description/Value	Presence
PAYMENT_DEVICE_STATUS	PAYMENT_DEVICE_STATUS	As needed
REQUEST_QUEUE_STATUS	REQUEST_QUEUE_STATUS	As needed
TMS_STATUS	TMS_STATUS	As needed

`GetStatus()` returns a `Response` object instance containing a parameter list with the following name-value pairs:

Name	Description/Value	Presence
ERRORS	Error codes in comma separated values format.	On error
CHIPDNA_STATUS	The transaction processing status of the Server in XML format. This contains a flag to indicate whether a transaction is currently being processed either <code>True</code> or <code>False</code> and whether an issue exists that may prevent transactions from being processed. The latter will be one of the predefined values: Uninitialized None NoPinPadsAvailable NoPinPadsConfigured EncryptionCertRequired EncryptionCertInvalid	On no error
VERSION_INFORMATION	Information obtained using <code>GetVersion()</code> in XML format.	On no error
PAYMENT_PLATFORM_STATUS	The status of the payment gateway in XML format. This contains the machine's local date and time, the local date and time according to the payment platform and whether the Server is able to connect to the payment gateway either <code>Unavailable</code> or <code>Available</code> .	On no error
PAYMENT_DEVICE_STATUS	The status of each PIN pad configured with the Server for this Client in XML format. For each PIN pad this includes the configured Device ID and model, the current configuration state (either <code>NotConfigured</code> , <code>ConfigurationInProgress</code> , <code>FirmwareUpdateInProgress</code> or <code>Configured</code>), whether it is processing a transaction and if it is available along with the availability error and information (as described for the	On no error

Name	Description/Value	Presence
	PaymentDeviceAvailabilityChangeEvent event).	
	For Miura devices, information about the battery status is also included: BatteryPercentage, BatteryChargingStatus (one of Not Charging Charging Fully Charged), BatteryStatusUpdateDateTime, BatteryStatusUpdateDateTimeFormat (currently dd/MM/yyyy HH:mm:ss).	
REQUEST_QUEUE_STATUS	The status of the queue of requests to be sent to the payment gateway in XML format. This includes the number of credit, credit confirm, credit void, debit, debit confirm and debit void requests still to be processed.	On no error
TMS_STATUS	The status of TMS configuration in XML format. This includes the date and time the last update was performed and the number of days until the next one is required.	On no error

Get Transaction Information

Call `GetTransactionInformation()` to get the current information corresponding to the specified transaction.

```
public Response GetTransactionInformation(
    List<Parameters> parameters
)
```

`GetTransactionInformation()` takes the following parameter name-value pairs:

Name	Description/Value	Presence
REFERENCE	The reference of the transaction. Printable ASCII characters excluding the following characters "<> :*/\. Maximum length of 50 characters and may not contain leading or trailing spaces. This should be the reference provided by the TransactionFinished event.	Always

`GetTransactionInformation()` returns a `Response` object instance containing a parameter list with the following name-value pairs:

Name	Description/Value	Presence
ERRORS	Error codes in comma separated values format.	On error
REFERENCE	The reference of the transaction. Printable ASCII characters excluding the following characters "<> : *?/\ ". Maximum length of 50 characters and may not contain leading or trailing spaces.	On no error
TRANSACTION_RESULT	Approved or Declined	On no error
TRANSACTION_STATE	Either Uncommitted, Committed or Voided	On no error
TRANSACTION_DATE_TIME	The date and time of the transaction in the format yyyyMMddHHmmss.	On no error
CARDEASE_REFERENCE_STAGE_1	ASCII 36 characters in the format of a GUID. Unique transaction generated by the payment gateway.	Present when authorization has been submitted online
CARDEASE_REFERENCE_STAGE_2	ASCII 36 characters in the format of a GUID. Unique transaction generated by the payment gateway.	Present when confirm or void has been submitted online
CARD_HASH	Unique token generated by the payment gateway that can be used to identify a card without using the PAN.	Present when transaction has been submitted online
CARD_REFERENCE	Unique reference generated by the payment gateway that can be used to identify a card without using the PAN.	Present when transaction has been submitted online
TRANSACTION_ID	Numeric Gateway transaction ID.	Always
CUSTOMER_VAULT_ID	Up to 36 character string, ex: 12345ABCDE	If available

Get Version

Call `GetVersion()` to get the version data of the Server.

```
public Response GetVersion()
```

`GetVersion()` returns a `Response` object instance containing a parameter list with the following name-value pairs:

Name	Description/Value	Presence
CHIPDNA_VERSION	Build version of the Server.	Always
CHIPDNA_RELEASE_NAME	Release name of the Server.	Always
CHIPDNA_APP_NAME	Application name of the Server.	Always

Get Merchant Data

Call `GetMerchantData()` to get the current information corresponding to the configured merchant accounts. This information includes, the currencies supported, transaction types supported and the merchant's name and number.

```
public Response GetMerchantData()
```

`GetMerchantData()` returns a `Response` object instance containing a parameter list with the following name-value pairs:

Name	Description/Value	Presence
ERRORS	Error codes in comma separated values format.	On error
MERCHANT_DATA	The current merchant account information in XML format.	On no error

Run Request Queue

Call `RunRequestQueue()` to immediately run the request queue and optionally re-run failed transactions after a specified date. The request queue automatically runs periodically and processes any offline transactions that are queued for upload to the payment gateway. However, the automatic request queue process will not upload failed transactions, whereas `runRequestQueue()` will. This command can help ensure stored transactions, such as deferred authorizations, are uploaded more quickly and reliably when the SDK is operating in an environment with an intermittent internet connection.

The result of every request queue process, whether run automatically or via this method, is reported back to the integrating application via `RequestQueueRunCompletedEvent`.

```
public Response RunRequestQueue(
    List<Parameters> parameters
)
```

`RunRequestQueue()` takes the following parameter name-value pairs:

Name	Description/Value	Presence
REQUEST_QUEUE_TYPE	The type of request queue to run. Values can be Pending, Failed or PendingAndFailed. Defaults to Pending if not sent.	As needed.
RUN_QUEUE_FAILED_TRANSACTIONS_FROM_DATE	The date failed transactions should be processed from. The Server will run all failed transactions from this date until the present-day.	Always if REQUEST_QUEUE_TYPE is Failed or PendingAndFailed.



Failed transactions should be processed with caution. While the SDK includes this method to process these, using it may go against acquirer / processor and scheme rules.

RunRequestQueue () returns a Response object instance containing a parameter list with the following name-value pairs:

Name	Description/Value	Presence
ERRORS	Error codes in comma separated values format.	On error

11. Utility Events

Payment Device Availability Change

The Client Helper fires a `PaymentDeviceAvailabilityChange` event when there is a change in the availability of a PIN pad to process transactions. A PIN pad may not be available to process transactions for a number of reasons, for example it has been disconnected or the actual Device ID does not match the Device ID specified in the configuration file. If a PIN pad is not available further information relating to the error will be returned when possible, for example if there is a Device ID mismatch the actual Device ID will be returned.

```
public event EventHandler<EventParameters> PaymentDeviceConnectionEvent
```

The `EventParameters` contains a parameter list with the following name-value pairs:

Name	Description/Value	Presence
PAYMENT_DEVICE_MODEL	The model of the PIN pad raising the event.	Always
PAYMENT_DEVICE_IDENTIFIER	The identifier of the PIN pad that is raising the event.	Always
IS_AVAILABLE	True or False.	Always
AVAILABILITY_ERROR	Will be one of the predefined values: None CommsLink DeviceIdMismatch InvalidFirmwareVersion DeviceNotConfigured	Always
AVAILABILITY_ERROR_INFORMATION	Further details relating to the AVAILABILITY_ERROR in XML format.	Always

Tms Update

When `RequestTmsUpdate()` has been successfully called, the Client Helper fires a `TmsUpdate` event when the update is finished with the result of the request.

```
public event EventHandler<EventParameters> TmsUpdateEvent
```

The `EventParameters` contains a parameter list with the following name-value pairs:

Name	Description/Value	Presence
TMS_REQUEST_TYPE	The type of update that was requested.	Always
TMS_UPDATE_RESULT	The result of the request is either Success or Failed.	Always
ERRORS	Comma separated list of error codes.	If the update failed

Request Queue Run Completed

The Client Helper fires a `RequestQueueRunCompletedEvent` after the request queue is processed. Whether that is done automatically by the Server or after a call to `RunRequestQueue()`.

```
public event EventHandler<EventParameters> RequestQueueRunCompletedEvent
```

The `EventParameters` contains a parameter list with the following name-value pairs:

Name	Description/Value	Presence
REQUEST_QUEUE_TYPE	The type of request queue run. Values can be Pending, Failed or PendingAndFailed.	Always
REQUEST_QUEUE_REPORT	Data collected while running the request queue in XML format. This can be deserialized via helper methods in the Client.	Always
ERRORS	Comma separated list of error codes.	If the request queue encountered an error

12. Glossary of Terms

Term	Explanation
Cardholder	The customer that is trying to pay with a chip card.
Chip card	<p>'Smart' payment cards which include an integrated circuit microchip. Also known as:</p> <ul style="list-style-type: none">• 'ICC' or 'IC card', from Integrated Circuit.• 'Magstripe', in reference to the magnetic strip card type which actually pre-dates the integrated microchip type.• 'Chip and PIN', from the brand name adopted by the banking industries in the United Kingdom and Ireland for the rollout of the EMV smart card payment system.• 'EMV card', from EuroPay, MasterCard and Visa (see below).
CLI	Command Line Interface.
EMV	See 'EuroPay, MasterCard and Visa'.
EuroPay, MasterCard and Visa	A global standard for inter-operation of chip cards and chip card capable PIN pads, for authenticating credit and debit card transactions.
GUI	Graphical User Interface.
Payment Gateway	The hosted interface that provides payment processing services via the certified integrations with processors.
PCI DSS	Payment Card Industry Data Security Standard. An information security standard for organizations that store, process and transmit chip card data.
PIN pad	A transaction terminal device which includes a key pad for the user to enter a Personal Identification Number (PIN) for authentication. The SDK works with both attended and unattended PIN pads.
POS	Point of Sale. The point at which a customer makes a payment to the merchant.
SDK	Software Development Kit.
Terminal Management System	The hosted Terminal Management System that provides configuration and software information to deployed installations.
TMS	See 'Terminal Management System'.

13. Troubleshooting & Support

For assistance please send an email to your payment gateway provider with the following information:

- 1) Description of the problem you're experiencing.
- 2) Attach configuration and log files from your SDK installation directory:
 - a) Server configuration file. For example:
`C:\Payment Device SDK\ChipDNA Server\chipdna.config.xml`
 - b) Client GUI configuration file. For example:
`C:\Payment Device SDK\ChipDNA Client GUI\ChipDNAClientGUI.exe.config`
 - c) Server log file. For example:
`C:\Payment Device SDK\ChipDNA Server\logs\ChipDNAServer.log`



The Server creates a daily log but does not delete any of these files automatically. It is the responsibility of the integrator to periodically delete log files if storage is an issue.

Appendix 1. PIN pad Device ID Examples

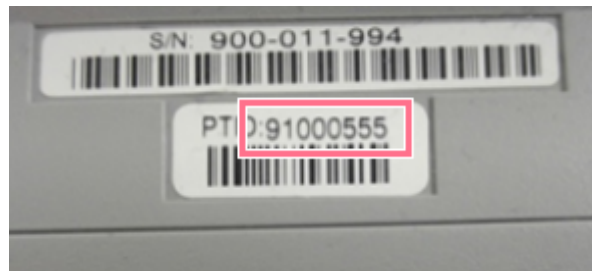


Figure 4 - Device ID on a VeriFone Ux300 and UxFTMA VIPA PIN pad.

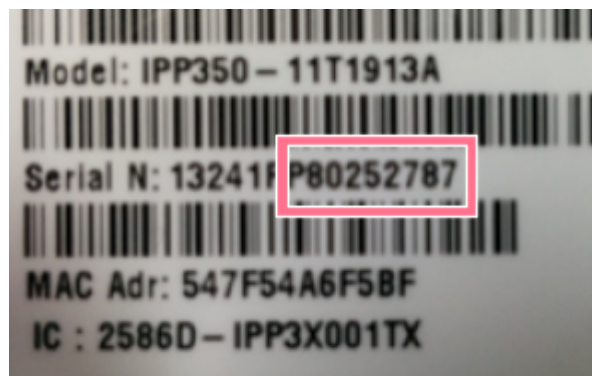


Figure 5 – Device ID on an Ingenico iPP350 RBA PIN pad.

The Device ID used in the configuration file must match the ID returned by the firmware on the device. For the Ingenico iPP350 RBA, this is the last eight digits of the serial number prepended with two leading zeros. 0080252787 should be used for the device in the image above.



Figure 6 – Device ID on an Ingenico iPP320 RBA PIN pad.



Figure 7 – Device ID on a Miura M020/M021 PIN pad.

The Device ID used in the configuration file must match the ID returned by the firmware on the device. For the Miura PIN pads, this is the serial number with the first zero and hyphen omitted. 17001271 should be used for the device in the image above.

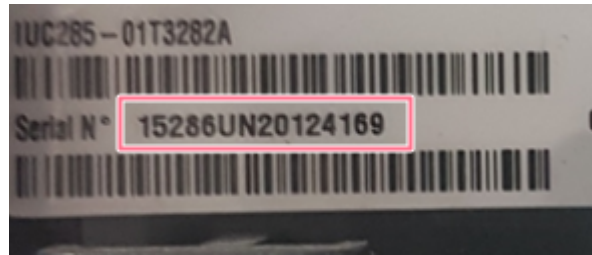


Figure 8 - Device ID on an Ingenico iUC285 RBA PIN pad.



Figure 9 - Device ID on an Ingenico iUC285 RAM PIN pad.



Figure 10 – Device ID on an Ingenico Lane/3000 RAM PIN pad.

The Device ID used in the configuration file must match the ID returned by the firmware on the device. For the Ingenico Lane/3000 PIN pads, this is the last nine digits of the serial number shown in the red box with an extra zero prepended. For example for the device in the above image the Device ID 0003287225 should be used.

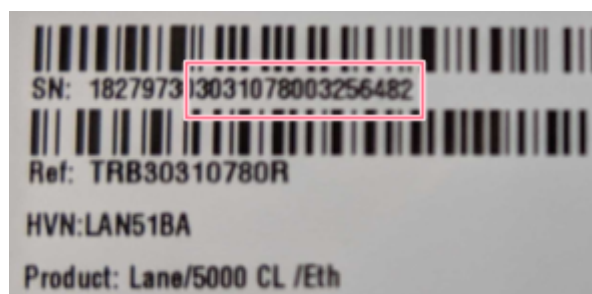


Figure 11 – Device ID on Ingenico Lane/3000, Lane/5000, Lane/7000, Self/2000, Self/4000 and Self/5000 UPP PIN pads.

The Device ID used in the configuration file must match the ID returned by the firmware on the device. For the Ingenico Lane UPP PIN pads, this is the last 16 digits of the serial number shown in the pink box. 3031078003256482 should be used for the device in the image above. The serial number can also be viewed on the device screen by pressing 0 four times.



Figure 12 – Device ID on Ingenico Self/4000 RAM PIN pad.

The Device ID used in the configuration file must match the ID returned by the firmware on the device. For the Ingenico Self/4000 RAM PIN pad, this is the entire number shown on the screen or the last eight digits of the serial number printed on the underside of the device with two zeros prepended. 0023895621 should be used for the device in the images above.



Figure 13 – Device ID on Ingenico Self/2000 RAM PIN pad.

The Device ID used in the configuration file must match the ID returned by the firmware on the device. For the Ingenico Self/2000 RAM PIN pad, this is the last eight digits of the serial number printed on the underside of the device with two zeros prepended. 0025471019 should be used for the device in the image above.



Figure 14 – Device ID on Ingenico Self/7000-8000 RAM device combination.

The Device ID used in the configuration file must match the ID returned by the firmware on the device. For the Ingenico Self/7000-8000 RAM device combination, this is the last eight digits of the serial number printed on the underside of the **Self/7000** unit with two zeros prepended. 0026514737 should be used for the device in the image above.

Appendix 2. Processing of Transactions

Step 1 - Authorization

After the payment gateway sends the transaction information to the payment processor the processor transmits the transaction amount and customer information for verification and authorization. The processor then returns a response that indicates whether the transaction was approved or declined. A hold is put on that amount in the cardholder's account; however, the amount is not transferred until the settlement process described in the next section.

Each time the merchant performs a transaction the payment gateway authorizes the transaction and stores the transaction information for settlement in a batch.

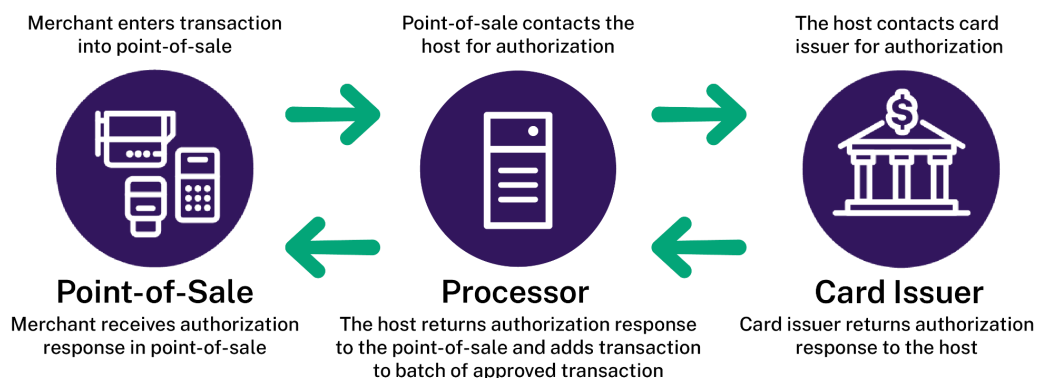


Figure 14 - Credit card transaction authorization process.

Step 2 - Settlement

In order to receive payment for the transactions submitted, the payment gateway will perform settlement on the merchant's behalf by sending the batch of transactions to the processor.



Figure 15 - Manual settlement process.

Step 3 - Funding

At a predefined time, the payment processor processes the settlements the payment gateway has sent. It creates a settlement batch for each connected financial institution and then transmits the batch to the appropriate entity. Upon processing the settlement batch, funds are routed to the merchant's account for deposit.

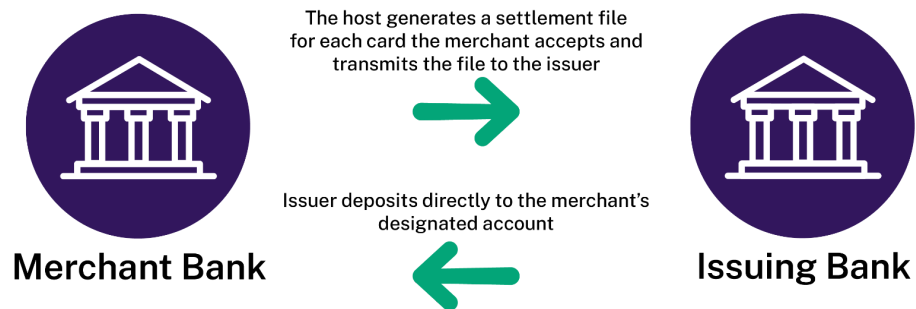


Figure 16 - Funding process.

Appendix 3. Supported PIN pads and Software Versions

The Server supports a variety of different PIN pads and communication protocols for those PIN pads. [Table 4](#) details the currently supported PIN pads and [Table 5](#) details the software versions.



Please contact your payment gateway provider to purchase an approved PINpad. PINpads obtained outside of approved channels cannot be supported.

Table 4 - Supported PIN pads.

Manufacturer	Device	Region	EMV Contact (Chip)	EMV Contactless (Chip)	EMV EZ Contact (Magstripe)
Ingenico	iPP320 (RBA)	US	Ready	Ready	Ready
Ingenico	iPP350 (RBA)	US	Ready	Ready	Ready
Ingenico	iUN/iSelf (RAM)	UK/Europe	Ready	Ready	N/A
Ingenico	iUN/iSelf LE (RBA)	US	Ready	Ready	Ready
Ingenico	iUC285 (RBA)	US	Ready	Ready	Ready
Ingenico	iUC285 (RAM)	UK/Europe	Ready	Ready	N/A
Ingenico	Lane/3000 (RAM)	UK/Europe	Ready	Ready	N/A
Ingenico	Lane/3000 (UPP)	US	Ready	Ready	Ready
Ingenico	Lane/5000 (UPP)	US	Ready	Ready	Ready
Ingenico	Lane/7000 (UPP)	US	Ready	Ready	Ready
Ingenico	Self/2000 (RAM)	UK/Europe	N/A	Ready	N/A
Ingenico	Self/2000 (UPP)	US	N/A	Ready	N/A
Ingenico	Self/4000 (RAM)	UK/Europe	Ready	Ready	N/A
Ingenico	Self/4000 (UPP)	US	Ready	Ready	N/A
Ingenico	Self/5000 (UPP)	US	Ready	Ready	N/A

Manufacturer	Device	Region	EMV Contact (Chip)	EMV Contactless (Chip)	EMV EZ Contact (Magstripe)
Ingenico	Self/7000-8000 (RAM)	UK/Europe	Ready	Ready	N/A
Miura	M020	UK/Europe/US	Ready	Ready	Ready
VeriFone	Ux300 (VIPA)	UK/Europe/US	Ready	Ready	N/A
VeriFone	UxFMTA (VIPA)	UK/Europe/US	Ready	Ready	N/A

Table 5 - Supported software versions for PIN pads.

Manufacturer	Device	Region	Version
Ingenico	iPP320 (RBA)	US	23k6 (23.52.6)
Ingenico	iPP350 (RBA)	US	23k6 (23.52.6)
Ingenico	iUN/iSelf (RAM)	UK/Europe	2129
Ingenico	iUN/iSelf LE (RBA)	US	23k6 (23.52.6)
Ingenico	iUC285 (RBA)	US	23k6 (23.52.6)
Ingenico	iUC285 (RAM)	UK/Europe	2129
Ingenico	Lane/3000 (RAM)	UK/Europe	2022
Ingenico	Lane/3000 (UPP)	US	7.82.05
Ingenico	Lane/5000 (UPP)	US	7.82.05
Ingenico	Lane/7000 (UPP)	US	7.82.05
Ingenico	Self/2000 (RAM)	UK/Europe	2238
Ingenico	Self/2000 (UPP)	US	7.83.19
Ingenico	Self/4000 (RAM)	UK/Europe	2238
Ingenico	Self/4000 (UPP)	US	7.83.15
Ingenico	Self/5000 (UPP)	US	7.83.15
Ingenico	Self/7000-8000 (RAM)	UK/Europe	2238
Miura	M020	UK/Europe/US	1-65
VeriFone	Ux300 (VIPA)	UK/Europe/US	6.8.2.21
VeriFone	UxFMTA (VIPA)	UK/Europe/US	6.8.2.21

Appendix 4. Firewall Configuration

The Server communicates with several internet services. It must be able to communicate with Direct Connect (<https://live.cardeasexml.com>) and TMS (<https://tms.cardeasexml.com>) on the payment gateway. This may require changes to your firewall configuration.

The port number the Server is hosted on is configured in the configuration XML file (see [Configure and Install](#) for more details) and the default value is 1869.

The table below lists the IP addresses and port numbers for each external service used by the Server.

Table 6 - IP addresses and port numbers for external services.

Service	Platform	IP addresses	Ports
Direct Connect	Live	91.197.92.250	443
		91.197.93.250	
		91.197.93.251	
		91.197.94.250	
		91.197.94.252	
		74.120.0.250	
		74.120.1.250	
		74.120.1.251	
		74.120.2.250	
		74.120.2.252	
	Test	91.197.92.230	443
		91.197.93.230	
		91.197.94.203	
		91.197.95.230	
TMS	Live	91.197.92.239	443
		91.197.93.239	
		91.197.94.239	
		74.120.0.239	
		74.120.1.239	
		74.120.2.239	
	Test	91.197.92.219	443
		91.197.93.219	
		91.197.94.219	
		91.197.95.219	

Appendix 5. Configuring client-side Logging with the Java Client

The Java client library (`ChipDnaClientLib.jar`) supports logging using Log4j. This feature is disabled by default; to enable it, call `setLoggerEnabled()` and ensure that dependencies `log4j-api-2.17.1.jar` and `log4j-core-2.17.1.jar` are available in the classpath. If you wish to confirm that logging is enabled, call `isLoggerEnabled()`.

By default, Log4j outputs errors to the console only. For convenience we have provided a ready-made Log4j configuration file (`client.config.log4j2.xml`) that will redirect output to a file (`logs/ChipDNAClient.log`). To use, the config file must be available in the application working directory before logging is enabled.

Appendix 6. Supported PIN pads and Supported Features

The SDK supports a variety of different PIN pads and features for those PIN pads. [Table 7](#) details the currently supported PIN pads and the features each PIN pad supports.

Table 7 – Supported PIN pads and each supported feature.

Manufacturer	Device	Region	Card Allowlisting	Card Removal Enforced	PAN Keyed Entry	On-Device Tipping	Voice Referral	Partial Approval	Deferred Authorization
Ingenico	iPP320 (RBA)	US	Yes	Yes	Yes	Yes	Yes	Yes	Yes
Ingenico	iPP350 (RBA)	US	Yes	Yes	Yes	Yes	Yes	Yes	Yes
Ingenico	iUN/iSelf (RAM)	UK/Europe	Yes	Yes	No	No	No	No	Yes
Ingenico	iUN/iSelf LE (RBA)	US	Yes	No	No	No	No	Yes	Yes
Ingenico	iUC285 (RBA)	US	Yes	No	No	No	No	Yes	Yes
Ingenico	iUC285 (RAM)	UK/Europe	Yes	Yes	No	No	No	No	Yes
Ingenico	Lane/3000 (RAM)	UK/Europe	Yes	Yes	Yes	Yes	Yes	No	Yes
Ingenico	Lane/3000 (UPP)	US	Yes	No	Yes	Yes	Yes	Yes	Yes
Ingenico	Lane/5000 (UPP)	US	Yes	No	Yes	Yes	Yes	Yes	Yes
Ingenico	Lane/7000 (UPP)	US	Yes	No	Yes	Yes	Yes	Yes	Yes
Ingenico	Self/2000 (RAM)	UK/Europe	Yes	No	No	No	No	No	Yes
Ingenico	Self/2000 (UPP)	US	Yes	No	No	No	No	No	Yes
Ingenico	Self/4000 (RAM)	UK/Europe	Yes	Yes	No	No	No	No	Yes
Ingenico	Self/4000 (UPP)	US	Yes	No	No	No	No	No	Yes
Ingenico	Self/5000 (UPP)	US	Yes	No	No	No	No	No	Yes
Ingenico	Self/7000-8000 (RAM)	UK/Europe	Yes	Yes	No	No	No	No	Yes
Miura	M020	UK/Europe/US	Yes	No	Yes	Yes	Yes	Yes	Yes
VeriFone	Ux300 (VIPA)	UK/Europe/US	No	Yes	No	No	No	Yes	Yes
VeriFone	UxFMTA (VIPA)	UK/Europe/US	No	Yes	No	No	No	Yes	Yes

Appendix 7. Supported PIN pads and Transaction Update Event Parameters

Some of the transaction update event parameter values apply to all PIN pads and other transaction update event parameter values apply to particular PIN pads based upon supported features. [Table 8](#) details the currently supported PIN pads and the transaction update event parameter values that each PIN pad supports.

Table 8 – Supported PIN pads and supported transaction update parameter values.

Manufacturer	Device	Region	ApplicationSelectionStarted	AmountConfirmationStarted	CardRequested	CardRemovalRequested	MagstripeAccountSelectionStarted	OnlineAuthRequested	OnlineAuthCompleted	PanKeyEntryStarted	PanKeyEntryCompleted	PinEntryStarted	TippingRequested	VoiceReferralCompleted	ZipCodeRequested
Ingenico	iPP320 (RBA)	US	No	No	Yes	Yes	No	Yes	Yes	Yes	Yes	No	Yes	Yes	Yes
Ingenico	iPP350 (RBA)	US	No	No	Yes	Yes	No	Yes	Yes	Yes	Yes	No	Yes	Yes	Yes
Ingenico	iUN/iSelf (RAM)	UK/Europe	Yes	Yes	Yes	Yes	No	Yes	Yes	No	No	Yes	No	No	No
Ingenico	iUN/iSelf LE(RBA)	US	Yes	Yes	Yes	Yes	Yes	Yes	Yes	No	No	Yes	No	No	Yes
Ingenico	iUC285 (RBA)	US	Yes	Yes	Yes	Yes	No	Yes	Yes	No	No	No	No	No	No
Ingenico	iUC285 (RAM)	UK/Europe	No	Yes	Yes	Yes	No	Yes	Yes	No	No	No	No	No	No

Manufacturer	Device	Region	ApplicationSelectionStarted	AmountConfirmationStarted	CardRequested	CardRemovalRequested	MagstripeAccountSelectionStarted	OnlineAuthRequested	OnlineAuthCompleted	PanKeyEntryStarted	PanKeyEntryCompleted	PinEntryStarted	TippingRequested	VoiceReferralCompleted	ZipCodeRequested
Ingenico	Lane/3000 (RAM)	UK/Europe	Yes	Yes	Yes	Yes	No	Yes	Yes	Yes	Yes	Yes	Yes	Yes	No
Ingenico	Lane/3000 (UPP)	US	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
Ingenico	Lane/5000 (UPP)	US	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
Ingenico	Lane/7000 (UPP)	US	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
Ingenico	Self/2000 (RAM)	UK/Europe	No	No	Yes	No	No	Yes	Yes	No	No	Yes	No	No	No
Ingenico	Self/2000 (UPP)	US	No	No	Yes	No	No	Yes	Yes	No	No	Yes	No	No	No
Ingenico	Self/4000 (RAM)	UK/Europe	Yes	Yes	Yes	Yes	No	Yes	Yes	No	No	Yes	No	No	No
Ingenico	Self/4000 (UPP)	US	Yes	Yes	Yes	Yes	Yes	Yes	Yes	No	No	Yes	No	No	No
Ingenico	Self/5000 (UPP)	US	Yes	Yes	Yes	Yes	Yes	Yes	Yes	No	No	Yes	No	No	No
Ingenico	Self/7000-8000 (RAM)	UK/Europe	Yes	Yes	Yes	Yes	No	Yes	Yes	No	No	Yes	No	No	No
Miura	M020	UK/Europe/US	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	No

Manufacturer	Device	Region	ApplicationSelectionStarted	AmountConfirmationStarted	CardRequested	CardRemovalRequested	MagstripeAccountSelectionStarted	OnlineAuthRequested	OnlineAuthCompleted	PanKeyEntryStarted	PanKeyEntryCompleted	PinEntryStarted	TippingRequested	VoiceReferralCompleted	ZipCodeRequested
VeriFone	Ux300 (VIPA)	UK/Europe/US	Yes	Yes	Yes	Yes	Yes	Yes	Yes	No	No	Yes	No	No	No
VeriFone	UxFMTA (VIPA)	UK/Europe/US	Yes	Yes	Yes	Yes	Yes	Yes	Yes	No	No	Yes	No	No	No