On Track Innovations Ltd. (OTI)

otiKiosk EMV SDK

Document version: 2.2

Package Version: 2.2.8

Kiosk core version:

Windows – 2.2.8.0

Linux – 2.2.8.0

Revision History

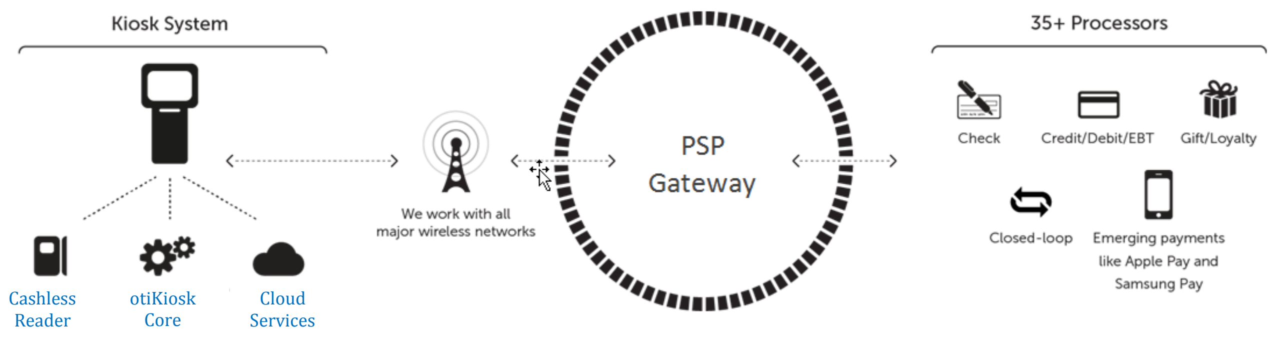
Version	Description	Date
1.5	Update Kiosk Core version, test application, SATURN reader firmware and configuration. Add section to describe the Configuration file Add method: ShowMessage Add Events: Void_completed SettlementFailed Add USB protocol (API for CONNECT 3000 USB)	24-May-2016
1.5A	Changed API example from StartTransaction to PreAuthorize Fix SystemStatusChanged return value from enum to int	25-May-2016
1.6	Added Linux platform support Document structure changes	21-June-2016
1.7	Updated TMS URL in the Configuration file section. Added a section about Log files and error handling. Windows platform - new Kiosk Core (v1.1.4.2) and Test application installation to solve: - Communication issues between the Kiosk core and the test application. - Kiosk core logging issues. Utilities update: - Updated reader config. File (BaseConfiguration_03_USA) - Updated DUKPT Utility (V 2.11.0.0)	17-July-2016
1.8	Windows platform - new Kiosk Core (v1.1.4.3) and Test application installation to solve Kiosk ID (Serial number) read errors Added a section about Getting the Kiosk ID	21-July-2016
1.8A	Updated the reader configuration files (BaseConfiguration_04_USA) Changed the structure of the Windows folder (added 'otiKiosk' subfolder) Updated the Windows installation process	22-July-2016
1.9	- Package for **Windows only** (Linux will be added at later stage) - Added Release Notes section - Kiosk API updates: • Updated Void Transaction function signature. • Updated Transport rules (port usage) • Updated TransactionComplete event (Channel field) • Added explanation to Authorization Details • Added Refund Transaction API	10-May-2017
2.01	- Added Linux platform support - EMV support - Added a section for Error handling and troubleshooting - New version of Saturn Management Utility (v2.15)	26-Dec-2017
2.1	- Added Reader Display Message Events - Updated Reader configuration files to support EMV CDA mode 1 - See SDK v2.1 Release Notes	26-Apr-2018
2.2	- Updated Message Sequences (POS can close a new Pre-Authorization session after starting a new one) - Updated Reader FW and cocnfiguration files (See Release Notes for details) - Added Card Token support - See SDK v2.2 Release Notes	18-Oct-2018

Table of Contents

1 General

The goal of this document is to define the scope and functionality for OTI Kiosk EMV solution.

1.1 System overview



The Kiosk solution (hereinafter otiKiosk), provides the Kiosk system developers a pre-certified EMV integration and remote management solution that allows them to easily and quickly develop their Kiosk software, while concentrating in the user interface and business aspects, and without dealing with the complexity of the EMV transactions processing.

otiKiosk will include the following elements

- Cashless reader hardware – SATURN 6500 TRIO or SATURN 6700 UNO
- otiKiosk Core – a PC based client application that provides the payment functionality for the Kiosk system integrator. It will provide the integration between the Kiosk software, Apriva gateway and the cashless reader.
- OTI TMS (Terminal Management Service) – a pre-integrated cloud service that is responsible for remote management of the terminals (Kiosk software & Reader firmware).

In addition – OTI supply documentation and sample code to support system integrators.

2 Technical details

2.1 otiKiosk system environment

The requirements for otiKiosk core are:

- **Windows platform:**
  - Windows 7
  - Software development framework: .NET v4.5
- **Linux platform:**
  - Ubuntu 14.04 LTS
  - Mono 4.4
- **Stable Ethernet communication**

2.2 otiKiosk functionality

2.2.1 Cashless payment functionality

- **Credit Sale:**
  - MagStripe, Contactless MSD, EMV Contact, EMV Contactless.
  - NO CVM only.
  - Including Debit Cards ran as a Credit Sale.
- **Reversals:**
  - Transactions can be canceled before the next transaction has been started. See more details about transaction cancelation options in the CancelTransaction or Void Transaction APIs.
  - Support for post sales RefundTransaction (for US PSP).
- **User interface –** support for changing the cashless reader display messages.

2.2.2 Terminal management (TMS) functionality

- **Cashless reader -** Firmware and Configuration update
- **otiKiosk core –** Firmware and Configuration update and status including:
  - Health (e.g. connection status)
  - Software versions
  - Log files

2.3 otiKiosk API

2.3.1 otiKiosk Core High Level API

otiKiosk will expose the following high-level API:

- Payment:
  - A. Pay transaction
  - B. Pre-Authorize
  - C. Confirm Transaction
  - D. Void Transaction
  - E. Cancel transaction
  - F. Get transaction clearing data
  - G. Refund Transaction (US PSP)
  - H. Get otiKiosk status
  - I. Write to log
  - J. Show message (on reader display)
- TMS:
  - A. Start update (update software or configuration of otiKiosk application and the Card Reader)
  - B. Get version (for otiKiosk application or the Card Reader)
- Events:
  - A. Transaction completed
  - B. UpdateIsPending (an update is pending for the otiKiosk application or the Card Reader)
  - C. System Status changed (active / not active and which component is off)
  - D. Update completed
  - E. Void completed
  - F. Settlement Failed
- Reader Display Messages Event:
  - A. Reader Messages Event

2.3.2 otiKiosk Core Detailed API

2.3.2.1 Payment

Pay Transaction

- **Description**: Execute a single step card transaction (as opposed to 2 step transaction done with the PreAuthorize and ConfirmTransaction commands). otiKiosk will wait for a card to be presented to the Reader, Read the card, and process it with the payment service if needed. It will then send a TransactionComplete event. The application can cancel the last transaction by calling the VoidTransaction method before another transaction has been performed.
- **Signature**: `bool` **`PayTransaction`** `(decimal amount, ushort currencyCode, uint timeoutSeconds)`

`amount`	amount to charge, in full units, e.g. for $1.5 transaction: `"params": {"amount": 1.5, "currencyCode": 840, "timeout": 60}`
`currencyCode`	ISO 4217 code. example: 840 means $US
`timeoutSeconds`	maximum number of seconds to wait. If no card arrives within this time – return Timeout status. The call may take longer including service communication.

Pre-Authorize

- **Description**: Pre-authorize a card with the payment processor.
  otiKiosk will wait for a card to be presented to the Reader, Read the card, and process it with the payment service if needed. It will then send a `TransactionComplete` event. To complete the transaction, the application must call ConfirmTransaction or VoidTransaction method.
- **Signature**: `bool` **`PreAuthorize`** `(decimal amount, ushort currencyCode, uint timeoutSeconds)`

`amount`	amount to charge, in full units, e.g. for $1.5 transaction: `"params": {"amount": 1.5, "currencyCode": 840, "timeout": 60}`
`currencyCode`	ISO 4217 code. example: 840 means $US
`timeoutSeconds`	maximum number of seconds to wait. If no card arrives within this time – return Timeout status. The call may take longer including service communication.

Confirm Transaction

- **Description**: A confirmation request is the second stage to an authorization request. It is used to confirm the authorization obtained so that settlement can take place.
- **Signature**: `bool` **`ConfirmTransaction`** `(string AthorizeCode, decimal amount, int productID)`

`AuthorizeCode`	Authorize number came from PSP to identify the transaction. The parameter is TransactionDbID that came with the TransactionComplate event.
`amount`	amount to charge, in full units, e.g. for $1.5 transaction: `"params": {"amount": 1.5, "currencyCode": 840, "timeout": 60}`
`productID`	– [optional] ID of the product selected by the consumer. This number will be saved in the server and available for reports.

Void Transaction

- **Description**: A Void Transaction request is used to void the last transaction that has been done in 2 cases:
    - A. Transaction has Pre-Authorized but not Confirmed (settled), in order to release funds that have been reserved against an account when the transaction will no longer take place.
    - B. Transaction has been done using PayTransaction but needs to be canceled (e.g. when product has not been vended)
- **Signature**:

```
A. CancelStatus VoidTransaction (string AthorizeCode)
B. enum CancelStatus {OK, NoTransaction, CannotCancel}
```

AuthorizeCode	Authorize number came from PSP to identify the transaction

CancelTransaction

- **Description**: cancel the last transaction started by PayTransaction or by Pre-Authorize:
    - A. If card was not presented yet, it will stop the Reader poll sequence (reader will stop waiting for a card).
    - B. If card was already presented, it will cancel the transaction by sending Void command. In such case, otiKiosk will send a TransactionComplete event.
- **Signature**:

```
A. CancelStatus CancelTransaction()
B. enum CancelStatus {OK, NoTransaction, CannotCancel}
```

Note: Cancel transaction will only work on the last transaction that was performed.

Refund Transaction

- **Description**: refunds a transaction. The result of the refund operation is returned with a TransactionComplete event.
- **Signature**: `bool RefundTransaction (string transaction_Referance, decimal amount)`

transaction_Referance	Transaction_Referance came from PSP to identify the transaction
amount	Amount to refund

NOTE: Limited to US PSP (Apriva).
This API works directly with the PSP and does not involve the payment reader or the card.
Refunds should be used with great care and on full responsibility of the user.
For this matter, OTI Kiosk Core serves as a transparent channel to the PSP and does not provide any protection measurements for wrong usage of this API.

Get Card Token

- **Description**: Poll for card and get card token.
  otiKiosk will wait for a card to be presented to the Reader, read the card, and calculate two card tokens:
    - A. **Strong token -** card token based on partial PAN, Exipry date and card holder name.
    - B. **Weak token -** card token based on partial PAN and Exipry date.
- It will then send a `CardTokenReceived` event with the card tokens and the last 4 digits of the PAN.
- If the card holder name is missing (not returned by the card), otiKiosk will send `CardTokenReceived` and provide only the weak token (cardToken_B). See example below.

- **Signature**: `bool GetCardToken (uint timeoutSeconds)`

`timeoutSeconds`	maximum number of seconds to wait. If no card arrives within this time – return Timeout status. The call may take longer including service communication.

Log

- **Description**: Allows the application to write log entries in otiKiosk's log. The log can be retrieved from the TMS (Terminal Management System).
- **Signature**:
  A. `bool log(byte level, string data)`

`level`	Level of the log entry. If the level argument is higher than the value of LOG_POS configuration parameter – then the data will not be written to the log. Possible values: 0: don't log, 1:Error, 3: Info, 5: Debug
`message`	Data to be logged, in the following format: `[Time] [Module] [Level] | [data]` Example: `23:59:59.999 POS 5 | data sent from POS`

Get Transaction Details

- **Description**: return data required for printing a receipt and maintain logs
- **Signature**: `AuthorizationDetails GetTransactionDetails()`

Get otiKiosk Status

- **Description**: return the otiKiosk status
- **Signature**:
  A. `AppStatus GetStatus()`
  B. `Enum AppStatus {NotReady, Ready, PaymentTransaction, Update}`

Show Message

- **Description**: Allows the application to show messages on OTI Saturn Reader display.
- **Signature**: `bool ShowMessage(string line1, line2)`

2.3.2.2 TMS (Terminal Management System)

Start Update (for otiKiosk core and the reader)

- **Description**: the POS calls this when it is ready to be temporarily disabled for an update, after receiving an `UpdateIsPending` event. It should always be followed by an `UpdateCompleted` event.
- **Signature**: `bool StartUpdate()`

Get version (for otiKiosk core and the reader)

- **Description**: called by the POS to get the version of components in otiKiosk system.
- **Signature**: `string GetVersion(SoftwareComponent softwareComponent)`

`SoftwareComponent`	`enum SoftwareComponent {otiKiosk, Reader}`

Get Kiosk ID

- **Description**: called by the POS to get the Kiosk identification number.
- **Signature**: `string GetKioskID()`

2.3.2.3 Events

Transaction complete

- **Description**: called by otiKiosk when transaction is complete with either success or failure. Contains the details required by the POS to print a receipt and to link transactions to sale events (e.g. in case of a dispute).
- **Signature**: `void TransactionComplete (Status, int errorCode, string errorDescription, AuthorizationDetails authorizationDetails`

Status	`enum Status{OK, Declined, Error, Timeout, Cancelled}`
errorCode	A numerical code describing the error if one occurs. See error codes table below.
errorDescription	A short technical description of the error – aimed to help developers and technical support people to identify the problem. Should not be presented to users.
Authorization Details	`class AuthorizationDetails {decimal amountAuthorized; string authorizationCode; string partialPan; string cardType; string TransactionDbID; string AuthID; string RRN; string Channel; string AID; stringTVR; stringIAD; stringTSI; stringARC; string TransactionTime; string Currency; string Application_Label; string TID; string CVM; boolean IsTransactionApproved` `See field explanation in the table below.`

- **Authorization Details:**

Field	Explanation
amountAuthorized	Amount approved for the transaction. May be lower than the original requested amount if it is a prepaid card.
authorizationCode	Apriva Transaction Identifier issued for approved transactions.
partialPan	Last 4 of the card account number.
cardType	Card issuer.
TransactionDbID	Apriva Transaction Identifier issued for approved transactions.
AuthID	Approval Code from the processor.
ReceiptID	Digital receipt ID in the Apriva System.
RRN	Processor reference number for approved transactions.
Channel	Data channel used for the transaction (MagStripe / Contactless / Contact)
AID	Application AID
TVR	Transaction Verification Results
IAD	Issuer Application Data
TSI	Transaction Status Indictor
ARC	Authorization Response Code
TransactionTime	The transaction time received from the PSP
Currency	Transaction currency
Application_Label	Application Label
TID	Terminal ID
CVM	Cardholder Verification (CVM)
IsTransactionApproved	Flag indicates whether the Transaction is Approved

- **Error codes:**

Error type	Error code
JsonParse	-32700
General	-1000
Terminate	-999
Timeout	-998
Payment	-997
ReaderCommunication	-996
Get Card token Not supported	-994
Positive code	Apriva errors

Update Is Pending

- **Description:** New software or Configuration is pending (otiKiosk core and the reader)
- **Signature**: **`Void`** **`UpdateIsPending`**`(SoftwareComponent component, SubComponent subComponent)`

`SoftwareComponent`	`enum SoftwareComponent {otiKiosk, Reader}`
`SubComponent`	`enum SubComponent (Software, Configuration)`

System Status Changed

- **Description**: send by otiKiosk when status has been changed (Active / Non Active).
- **Signature**: `void` **`SystemStatusChanged`** `(int component);`

`component`	`System error status where:`
	`System OK = 0,`
	`Network = 1,`
	`PaymentService = 2,`
	`TmsService = 4,`
	`Configuration = 8,`
	`Reader = 16.`
	`It can be a combination of those parameters.`

Update completed

- **Description**: sent by otiKiosk when an update is complete.
- **Signature**: `void` **`UpdateCompleted`** `(Status status, int errorCode, string errorDescription)`

`Status`	`enum Status{OK, Declined, Error}`
`errorCode`	0 means no error.
`errorDescription`	empty string ("") if no error. This string is meant in technical English and is meant to aid developers. It should not be shown to users. Instead – the POS should have a table translating `errorCode` values to user strings.

Void completed

- **Description**: sent by otiKiosk when a Void Transaction is completed.
- **Signature**: `void VoidCompleted (Status status, int errorCode, string errorDescription)`

Status	`enum Status{OK, Declined, Error}`
errorCode	0 means no error.
errorDescription	empty string ("") if no error. This string is meant in technical English and is meant to aid developers. It should not be shown to users. Instead – the POS should have a table translating `errorCode` values to user strings.

Settlement Failed

- **Description**: sent by otiKiosk when confirmation on Post authorization is failed in clearing.
- **Signature**: `void SettlementFailed (int errorCode, string errorDescription, AuthorizationDetails authorizationDetails`

errorCode	A numerical code describing the error if one occurs. See error codes table.
errorDescription	A short technical description of the error – aimed to help developers and technical support people to identify the problem. Should not be presented to users.
Authorization Details	`class AuthorizationDetails {decimal amountAuthorized; string authorizationCode; string partialPan; string cardType; string TransactionDbID; string AuthID; string string RRN; string Channel)`

Card Token Received

- **Description**: sent by otiKiosk when card token received after calling GetCardToken.
  otiKiosk tries to calculate two tokens: A (strong) and B (weak).
- If the card holder name is missing (not returned by the card), otiKiosk will send `CardTokenReceived` and provide only the weak token (cardToken_B). See example below.
- **Signature**: `void cardTokenReceived (Status status, int errorCode, string errorDescription, string cardToken_A, string cardToken_B, string partialPan)`

Status	`enum Status{OK, Declined, Error}`
errorCode	0 means no error. -994 In case it can't calculate the card token (e.g. when the token tags are missing in the reader configuration.)
errorDescription	empty string ("") if no error. This string is meant in technical English and is meant to aid developers. It should not be shown to users. Instead – the POS should have a table translating `errorCode` values to user strings.
cardToken_A	Strong card token, basedon partial PAN, Exipry date and Card holder name, in base 64 format.
cardToken_B	Weak card token, basedon partial PAN and Exipry date, in base 64 format.
partialPan	Last 4 of the card account number.

2.3.2.4 Reader Message Event

Reader Message Event

- **Description**: sent by otiKiosk when the display message on Saturn Reader is changed. This event can be used to synchronize the kiosk application with the Reader state.

Note: For this feature to work, the reader must be configured for sending External Display message events. See more details at the SDK Release Notes.

- **Signature**: `void ReaderMessageEvent (int index)`

index	The index of the message. See Message Index table below.

- **Message Index:**

Field	Explanation
	Main States
0x00	(Blank Screen)
0x01	xxx1 Present card
0x11	Welcome
0x23	Card read OK Remove card
0x04	Please remove card
0x0D	Approved sign
	Transaction Recovery States
0x05	Please present one card only
0x03	xxx1 Please present one card only
0x27	xxx2 Read error Please try again
	Stateless Messages
0x12	Thank You
0x15	Please Use Other Card
0x19	Please Wait
0x1A	International Card, Please Swipe!
	EMV Online Transaction Results States
0x24	Authorizing Please wait
0x06	Approved
0x09	Not Approved
	EMV Offline Transaction Results States
0x07	Approved
0x08	Not Approved
	EMV Transaction Error States
0x1E	Timed Out Please insert or swipe card
0x25	Communication Failure
0x26	Please insert or swipe card
0x0A	Terminated
0x0B	PIN Entry Required – see attendant
0x0C	See Phone

2.3.3 Transport Rules

2.3.3.1 Windows platform

- Commands are sent from POS to otiKiosk. Events are sent from otiKiosk to POS.
- otiKiosk listens on port 10000; the POS listens on port 10001 and 10002.
- **Port 10000** is used for **all incoming messages** (from POS to otiKiosk) and **port 10001** is used for **all outgoing messages** from otiKiosk, so the POS app should send all its messages on 10000 and receive all its messages on 10001.
- **Port 10002** is used for sending the Reader Message Index Events.

2.3.3.2 Linux platform

- Sending and receiving command/events using Linux Domain Sockets technology (IPC socket).
- Commands are sent from POS to otiKiosk. Events are sent from otiKiosk to POS.
- Reader Messages index event is send from otiKiosk to POS.
- POS send messages and receive responses on file: **pos_sockets**
- otiKiosk send events and receive responses on file: **kiosk_sockets**
- otiKiosk send the Reader Messages index on file: **events_sockets**
- The socket file paths are set in the otiKiosk configuration file:
  ```
  /home/<user>/.local/share/otiKiosk/otiKiosk.cfg
  ```
- The default file paths are:
  ```
  KIOSK_SOCKET_PATH=/home/<user>/.local/share/otiKiosk/kiosk_sockets
  HOST_SOCKET_PATH=/home/<user>/.local/share/otiKiosk/pos_sockets
  EVENTS_SOCKET_PATH=/home/<user>/.local/share/otiKiosk/events_sockets
  ```

Sending a message:

- Get a socket
  A. If a socket is already open on the target port – use it
  B. Otherwise - open a socket on the target port
- Send the message (JSON-RPC)
- Wait for the JSON-RPC response.
- Once data starts arriving – wait for a complete JSON to arrive (all parentheses closed), or a timeout of 1 second.

Receiving a message:

- Wait for socket opening on the target port. Allow multiple sockets.
- Once a socket is opened – do the following in a loop:
  A. Wait for incoming data.
  B. Once data start arriving – wait for a complete JSON to arrive (all parentheses closed), or a timeout of 1 second.
  C. Parse and process the incoming messages as JSON-RPC.
  D. Send a response (JSON-RPC).
- Be ready for a socket to be closed by the other end, or for additional messages to be sent.
- No need to send respond to Reader Message Index Events.

2.3.4 JSON Protocol Rules & Examples

- All commands and Events use the JSON-RPC 2.0 protocol with named parameters.
- All commands and Events require a response except the `log()` and `ReaderMessageEvent()` commands which does not require a response and hence has no id.

Examples

Pre - Authorize:

```
>> {"jsonrpc": "2.0", "method": "PreAuthorize", "params": {"amount":
1.50, "currencyCode": 840, "timeout": 60}, "id": 123}
<< {"jsonrpc": "2.0", "result": true, "id": 123}
```

Confirm Transaction

```
>> {"jsonrpc": "2.0", "method": "ConfirmTransaction", "params":
{"authorizationCode": 964774961, "amount": 1.00, "productID": 1},
""id"": 124}
<< {"jsonrpc": "2.0", "result": true, "id": 124}
```

Pay Transaction

```
>> {"jsonrpc": "2.0", "method": "PayTransaction", "params":
{"amount": 1.5, "currencyCode": 840, "timeout": 60}, "id"": 125}
<< {"jsonrpc": "2.0", "result": true, "id": 125}
```

Void Transaction

```
>> {"jsonrpc": "2.0", "method": "VoidTransaction", "params":
{"authorizationCode": "964774962"}, "id": 126}
<< {"jsonrpc": "2.0", "result": true, "id": 126}
```

Log

```
>> {"jsonrpc": "2.0", "method": "log", "params": {"logLevel": 3,
"message": "sample data to log"}}
```

Get Version

```
>> {"jsonrpc": "2.0", "method": "GetVersion", "params": {"component
": "Reader"}, "id": 127}
<< {"jsonrpc": "2.0", "result": "040506", "id": 127}
```

System Status Changed

```
<< {"jsonrpc": "2.0", "method": "SystemStatusChanged", "params":
{"component": 0 }, "id": 128}
>> {"jsonrpc": "2.0", "result": true, "id": 128}
```

Get Card Token

```
>> {"jsonrpc"": "2.0", "method": "GetCardToken", "params":
{"timeout": 60}, "id": 636}
<< {"jsonrpc": "2.0", "result": true, "id": 636}
```

Card Token Received

```
<<  {"jsonrpc": "2.0", "method": "CardTokenReceived", "params":
{"status": "OK", "errorCode": 0, "errorDescription": "",
"CardToken_A": "qBQMCrv32dnjL5m8do7vEURrpg==", "CardToken_B":
"uPx228KsBfAKytAMRudCEXbXdQ==", "PartialPAN": "1643"}, "id":
636854100438673020}"

>> {"jsonrpc": "2.0", "result": true, "id": 636854100438673020}
```

Card Token Received (Missing card holder name)

```
{"jsonrpc": "2.0", "method": "CardTokenReceived", "params":
{"status": "Error", "errorCode": 0, "errorDescription": "",
"CardToken_A": null, "CardToken_B": "uPx228KsBfAKytAMRudCEXbXdQ==",
"PartialPAN": "1643"}, "id": 636854106524752431}"

>> {"jsonrpc": "2.0", "result": true, "id": 636854106524752431}
```

Transaction Complete

```
<< {"jsonrpc": "2.0", "method": " TransactionComplete ", "params":
{"status": "OK", "errorCode": 0, "errorDescription": "Test Processor
Success(FAKE*2)", "authorizationDetails": {"amountAuthorized": 4.0,
"authorizationCode": "1308258281", "partialPan": "0102", "cardType":
"MasterCard", "TransactionDbID": "1308258281", "AuthID":
"3691151956", "ReceiptID": "195811285",  "RRN": "3691151956123",
"Channel": "Contactless", "AID": "", "TVR": "", "IAD": "", "TSI":
"", "ARC": "", "TransactionTime": "19/12/2017 17:59:18", "TID":
"****9550001320FB", "Application_Label": "MasterCard", "Currency":
"USD$", "CVM": "", "AmountRequested": 4,  "IsTransactionApproved":
true , "CardToken_A": "", "CardToken_B": ""}}, "id": 129}
>> {"jsonrpc": "2.0", "result": true, "id": 129}
```

Transaction Complete with Card Token

```
<< {"jsonrpc": "2.0", "method": " TransactionComplete ", "params":
{"status": "OK", "errorCode": 0, "errorDescription": "Test Processor
Success(FAKE*2)", "authorizationDetails": {"amountAuthorized": 4.0,
"authorizationCode": "1308258281", "partialPan": "1643", "cardType":
"MasterCard", "TransactionDbID": "1308258281", "AuthID":
"3691151956", "ReceiptID": "195811285",  "RRN": "3691151956123",
"Channel": "Contactless", "AID": "", "TVR": "", "IAD": "", "TSI":
"", "ARC": "", "TransactionTime": "19/12/2017 17:59:18", "TID":
"****9550001320FB", "Application_Label": "MasterCard", "Currency":
"USD$", "CVM": "", "AmountRequested": 4,  "IsTransactionApproved":
true , "CardToken_A": "qBQMCrv32dnjL5m8do7vEURrpg==", "CardToken_B":
"uPx228KsBfAKytAMRudCEXbXdQ=="}}, "id": 129}
>> {"jsonrpc": "2.0", "result": true, "id": 129}
```

ReaderMessageEvent

```
<< {"jsonrpc": "2.0", "method": "ReaderMessageEvent", "params": [1]}
```

2.3.5 Message Sequences

- When otiKiosk starts it sends `SystemStatusChanged()`
- When POS sends `PreAuthorize()`:
    - A. otiKiosk must respond (after some time) with `TransactionComplete()`.
    - B. The POS should then send either `ConfirmTransaction()` if the product was successfully delivered, or `VoidTransaction()` otherwise.
    - C. The POS may send `CancelTransaction()`; when it does – the `TransactionComplete()` event will be sent with code `Cancelled`.

Notes:

- The POS must close the PreAuthorize session (with Confirm or Void/Cancel), otherwise the transaction will remain open and will be reversed by the processor after some time, such operation may be involved with higher fees and therefore should be avoided.
- It's recommended that the POS will close PreAuthorize session before starting a new session but it's not mandatory. The kiosk core will accept confirm/void commands up to one hour from the Authorization time.
- Once `ConfirmTransaction()` has been acknowledged by the kiosk core, the POS must assume it will be sent to the PSP and should avoid re-sending of the same message. Retry confirm message can be sent only after `SettlementFailed` event was received and analyzed.
- When otiKiosk core sends `UpdateIsPending()` event, the POS application should wait until it is a suitable time to temporarily disable the system and then call `StartUpdate()`.
- If the POS doesn't call `StartUpdate()` within 10 minutes, otiKiosk will force the update. This mechanism serves as a watchdog mechanism in case the POS hangs up and needs to be updated.
- otiKiosk performs the update and responds with `UpdateCompleted()` event.

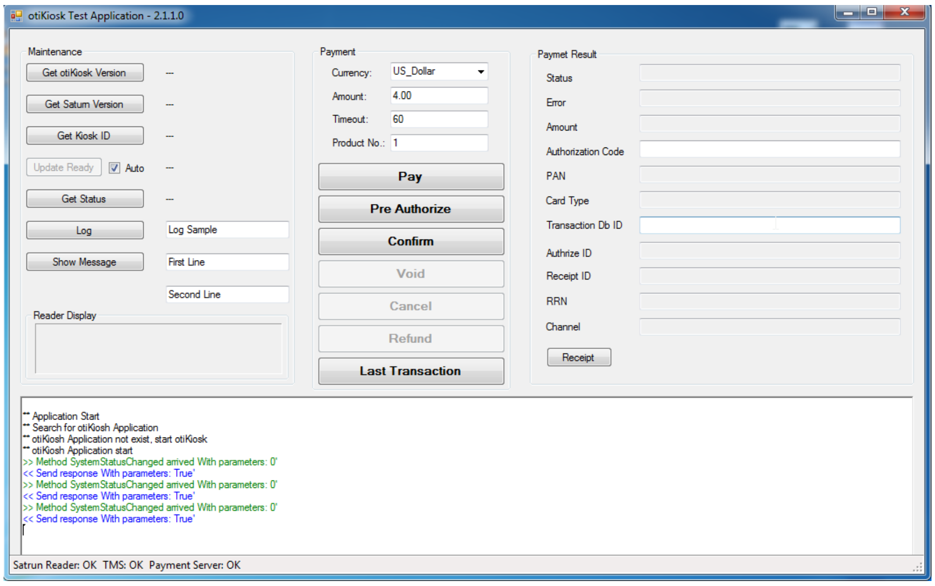
3 Installation and Test application

3.1 Installation on Windows

1. Install & run:
   a. Double click **otiKioskSetup.msi** and follow the installation instructions.
   b. Run the application from the Desktop shortcut:

      otiKiosk Test Application

   c. The Kiosk test application should open with the following screen:



Currency: US_Dollar
Amount: 4.00
Timeout: 60
Product No.: 1

Log: Log Sample
Show Message: First Line / Second Line

```
** Application Start
** Search for otiKiosh Application
** otiKiosh Application not exist, start otiKiosk
** otiKiosh Application start
>> Method SystemStatusChanged arrived With parameters: 0'
<< Send response With parameters: True'
>> Method SystemStatusChanged arrived With parameters: 0'
<< Send response With parameters: True'
>> Method SystemStatusChanged arrived With parameters: 0'
<< Send response With parameters: True'
```

Satrun Reader: OK TMS: OK Payment Server: OK

3.2 Installation on Linux

Mono installation

The Kiosk core requires Mono v4.4 to be installed.

To install Mono, send the following commands from Linux command line interface (CLI):

```
1. sudo apt-key adv --keyserver hkp://keyserver.ubuntu.com:80 --
   recv-keys 3FA7E0328081BFF6A14DA29AA6A19B38D3D831EF

2. echo "deb http://download.mono-project.com/repo/debian wheezy
   main" | sudo tee /etc/apt/sources.list.d/mono-xamarin.list

3. sudo apt-get update

4. sudo apt-get install mono-complete
```

User credentials – serial port access

The following command will allow the user to access serial devices:

```
sudo usermod -a -G dialout <user>
```

Kiosk core installation

The kiosk installation package can be found on:

.\Linux**otiKiosk**_x_x_x.zip

Note: the file named '**core**_x_x_x.zip' can be used for updating the core remotely via TMS using the 'CON_PROG' command.

1. Run install.sh with parameters:
    - –f: The zip file.
    - –p: The path to install otiKiosk.
    - Example: bash install.sh –f otiKiosk_2_2_4.zip –p ~/otiKiosk
2. To run the test application, send the following command:

```
cd '<path>/otiKiosk Test Application/bin/Debug/'
sudo mono 'otiKiosk Test Application.exe'
```

   Note: the test application will start the otiKiosk core automatically.

3.3 Using the Test application

The Test Application demonstrates the usage of otiKisok APIs.

The test application is provided as Windows installation and as Visual Studio project and solution:

- The installation is explained in the previous chapter.
- The source code and Visual Studio solution can be found at:
    - Windows: C:\otiKiosk\Sample code\otiKiosk Test Application.sln
    - Linux:

      .\otiKiosk For Linux\otiKiosk Test Application**otiKiosk Test Application.sln**

      Note: The solution can be opened using Monodevelop tool for Linux.

NOTE: The Linux test application does not launch the core automatically; you must run the Kiosk core before using the test application

To use the test application, you must:

1. Consult with OTI support for getting the correct firmware and configuration for your reader. If required, you may use the Saturn Management Utility to update the reader firmware and configuration:
    a. The Saturn Management Utility installation (Windows only) can be found at: otiKiosk_SDK**Utilities\OTI Saturn Management Utility**
    b. The reader firmware and configuration can be found at: otiKiosk_SDK **Utilities\Reader FW and Config**
2. Connect your OTI Reader to your PC.
3. Install and run the application as explained in the previous chapter.
4. Make sure your reader is registered to the otiKiosk system.
5. Press the application buttons to call the different Kiosk APIs.

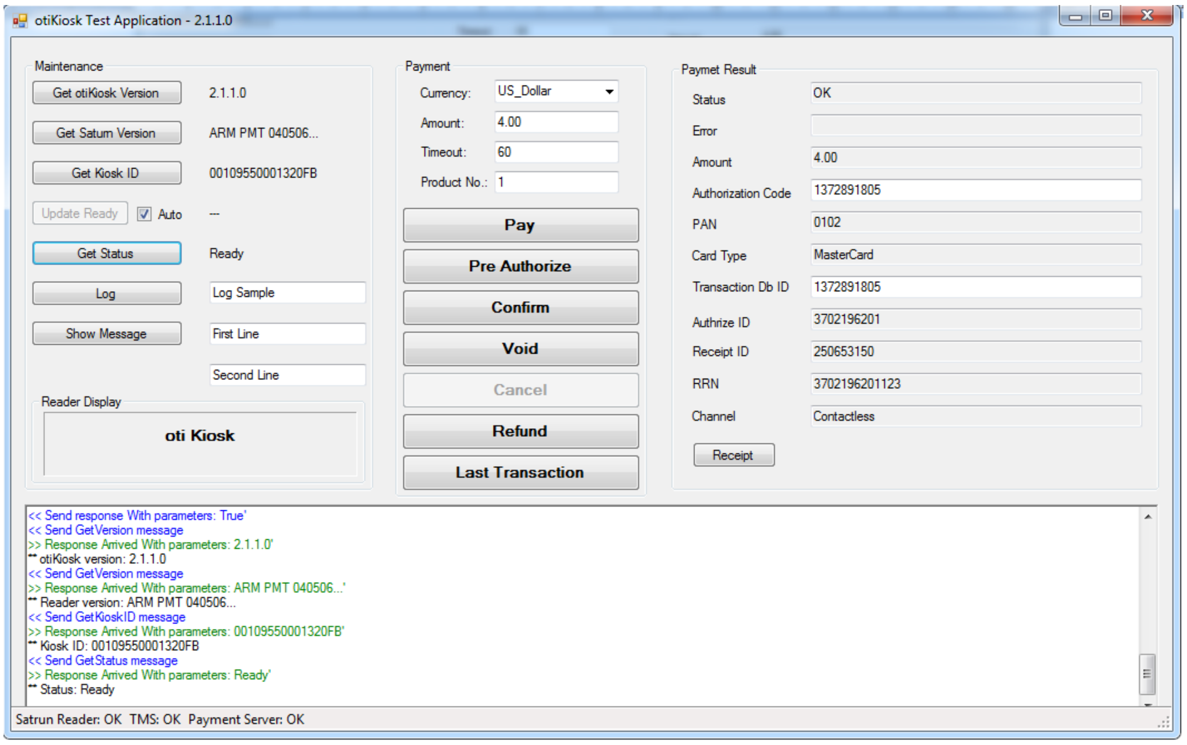
NOTE: The Reader must be registered to otiKiosk system – if not, you will not be able to complete the payment transactions. To register your Kiosk, please provide its Kiosk ID to your OTI account manager.

See more details in the '[Getting the Kiosk ID](#)' chapter.

Payment transaction

To perform payment transaction with the test application:

1. Enter the payment parameters (Currency, Amount, Timeout and Product ID)
2. Press the **Pay** button for one step payment or the **Pre-Authorize** button for 2 step payment.
3. Tap or swipe your payment card.
4. You may **Cancel** a transaction before it was completed.
5. When the reader shows "Approved" on the display, you can:
    a. Complete the transaction by pressing **Confirm** in case of 2 step payment.
    b. Cancel the transaction by pressing **Void**.



Get otiKiosk Version	2.1.1.0
Get Satum Version	ARM PMT 040506...
Get Kiosk ID	00109550001320FB
Update Ready ☑ Auto	--
Get Status	Ready
Log	Log Sample
Show Message	First Line
	Second Line
Reader Display	oti Kiosk

Currency:	US_Dollar
Amount:	4.00
Timeout:	60
Product No.:	1

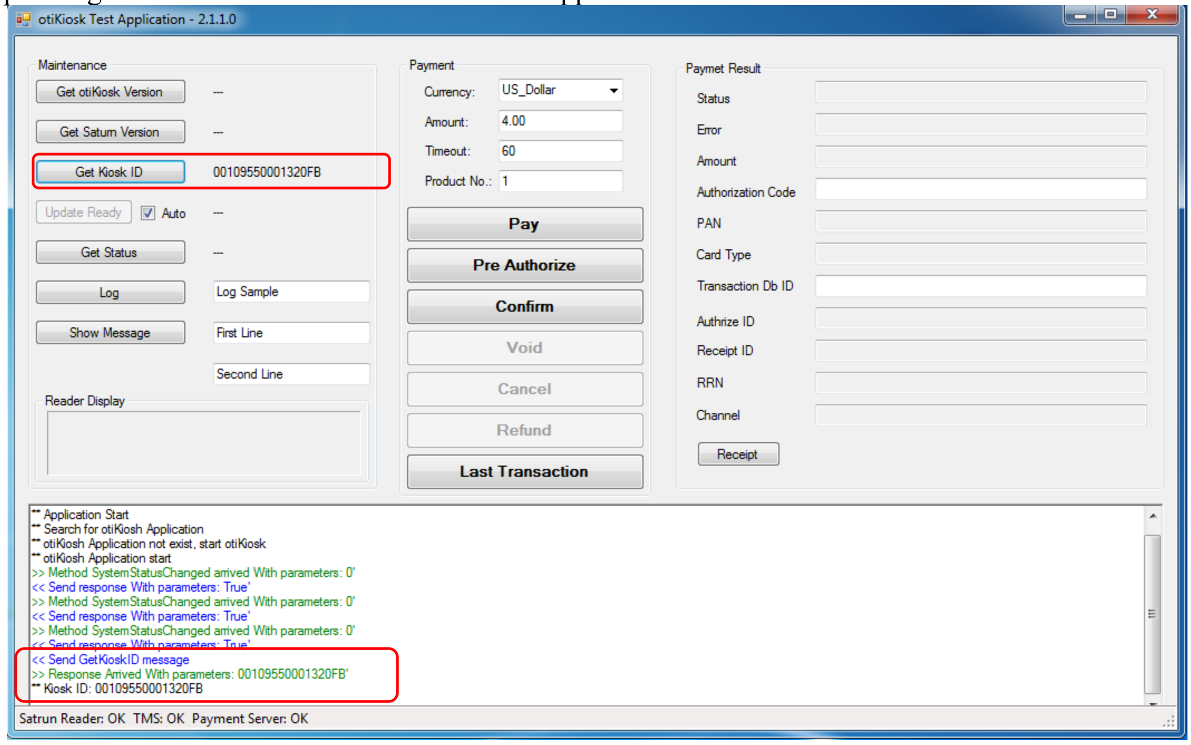
Status	OK
Error	
Amount	4.00
Authorization Code	1372891805
PAN	0102
Card Type	MasterCard
Transaction Db ID	1372891805
Authrize ID	3702196201
Receipt ID	250653150
RRN	3702196201123
Channel	Contactless

```
<< Send response With parameters: True'
<< Send GetVersion message
>> Response Arrived With parameters: 2.1.1.0'
** otiKiosk version: 2.1.1.0
<< Send GetVersion message
>> Response Arrived With parameters: ARM PMT 040506...'
** Reader version: ARM PMT 040506...
<< Send GetKioskID message
>> Response Arrived With parameters: 00109550001320FB'
** Kiosk ID: 00109550001320FB
<< Send GetStatus message
>> Response Arrived With parameters: Ready'
** Status: Ready
```

Satrun Reader: OK TMS: OK Payment Server: OK

3.4 Getting the Kiosk ID

The Kiosk ID is taken from the SATURN Reader Serial Number (SN) and can be obtained by pressing the “Get Kiosk ID” button in the test application:



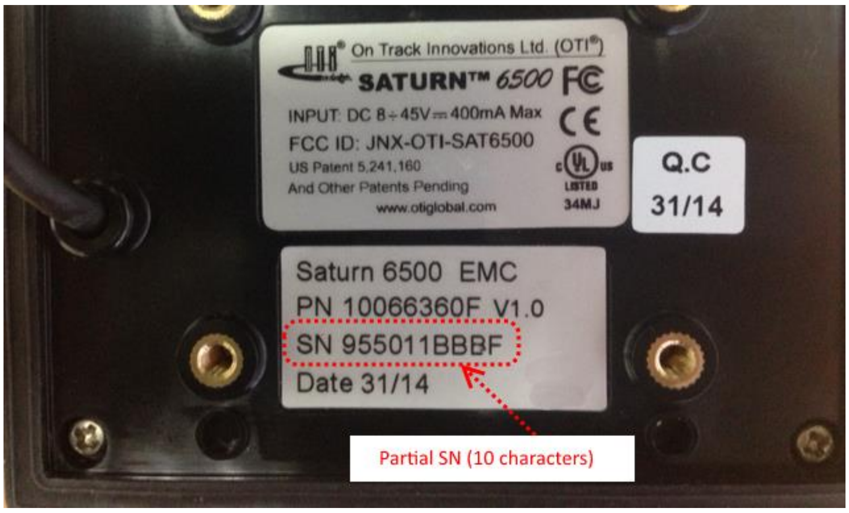
Get otiKiosk Version: ---
Get Saturn Version: ---
Get Kiosk ID: 00109550001320FB
Log: Log Sample
Show Message: First Line / Second Line

Currency: US_Dollar
Amount: 4.00
Timeout: 60
Product No.: 1

```
** Application Start
** Search for otiKiosh Application
** otiKiosh Application not exist, start otiKiosk
** otiKiosh Application start
>> Method SystemStatusChanged arrived With parameters: 0'
<< Send response With parameters: True'
>> Method SystemStatusChanged arrived With parameters: 0'
<< Send response With parameters: True'
>> Method SystemStatusChanged arrived With parameters: 0'
<< Send response With parameters: True'
<< Send GetKioskID message
>> Response Arrived With parameters: 00109550001320FB'
** Kiosk ID: 00109550001320FB
```

Satrun Reader: OK TMS: OK Payment Server: OK

Note that in some of the old readers, the Kiosk ID is not identical to the Reader SN printed on the back of the reader, in such readers the printed SN is showing partial SN (10 hex characters):



The complete SN used as the Kiosk ID includes 16 hex characters and can be obtained from the reader in one of the following ways:

1. Use the Kiosk test application as explained above.
2. Use Saturn Management Utility (PC tool provided with the SDK) and send the "Get SN" command to the reader:

   Send: 02 09 00 3D DF 4D 00 A4 03
   Receive: 02 08 00 3D 00 00 37 03 02 14 00 3D FF 01 0B DF 4D 08 00 10 95 50 00 11 BB BF 84 03

3.5 Kiosk Configuration File

Configuration file contains configure parameters and initial settings.
These parameters may change from customer to customer.

- **Location:**
  - **Windows:** C:\Users\<user>\AppData\Local\otiKiosk\otiKiosk.cfg
  - **Linux:** /home/<user>/.local/share/otiKiosk's
- **Format:**
  - <Parameter Name>=<Value>
  - Example: TMS_POLL_INTERVAL=180

Configuration Parameters

The following parameters are supported by the configuration:

Parameter	**Default value**	**Description**
TMS_POLL_INTERVAL	180	Time interval (in **seconds)** that the kiosk core approaches to the TMS server for updates on its first connection (firmware, config. files, versions...). After first connection, kiosk core receives the poll interval from TMS 'Timeout Interval in Seconds' field.
TMS_SVC_URL	http://tms.otiglobal.com	TMS server URL
PAYMENT_SVC_URL	https://oti-apriva.azurewebsites.net	Payment server URL
MSG_UPDATING1	UPDATING...	Display message when system is updating by TMS – 1st line
MSG_UPDATING2	PLEASE WAIT	Display message when system is updating by TMS – 2nd line
INIT_CMD		Initialization commands for Saturn Reader.
MSG_SWIPEALL1	Insert, swipe or	Display message presented when transaction is terminated, and the user can try another interface or card.
MSG_SWIPEALL2	Try another card	
UPDATE_INTERVAL	10	Interval time (in **minutes**) for TMS update (firmware or configuration) in case that the host did not send the `start Update()` command. The core resets this timeout counting with each transaction to ensure that the update is done in system idle time.
COMM_PORT		Serial COM Port used by the Kiosk core to find the reader.
Linux platform only parameters:		
KIOSK_SOCKET_PATH	/home/<user>/.local/share/otiKiosk/**kiosk sockets**	otiKiosk core socket file path (Core send messages and receive responses on this file)
HOST_SOCKET_PATH	/home/<user>/.local/share/otiKiosk/**pos_sockets**	POS socket file path (POS send messages and receive responses on this file)

4 Error handling and troubleshooting

In case there are errors or unexpected behavior, please provide the following information to OTI:

1. Kiosk ID
2. Kiosk configuration file
3. Kiosk log file
4. In case there are transaction errors:
    a. Date and time of the transaction.
    b. The reader configuration (can be retrieved with the Saturn Management Utility)

4.1 Log files

The Kiosk core saves some information in a log file for debug purpose.
The logs are saved in the following location:

- **Windows:** C:\Users\<user>\AppData\Local\otiKiosk\Logs
- **Linux:** /home/<user>/.local/share/logs

4.2 Troubleshooting sequence

1. Check SDK version
    a. Uninstall old versions
    b. Install latest version
2. Check and update configuration:
    a. Kiosk
    b. Reader
3. Verify correct registration:
    a. OTI TMS
    b. Apriva
    c. Note that Test terminals and Live terminals require different registration.
4. Try transaction with Valid MagStripe card
5. Make sure communication is not blocked by firewall or similar mechanism.

4.3 Common problems troubleshooting

otiKiosk Errors:

- Positive sign errors are from Apriva system (e.g. 99)
- Negative sign errors are coming from OTI system (e.g. -997)

Note that minus sign errors such as “-995 declined by reader” can also be the result of ‘online decline’ by the PSP.

4.3.1 Reader not responding

- **Possible cause:** COM Port issues
- **Suggested steps:**
    - Close kiosk test application and core
    - Make sure **otikiosk.exe** process is closed in **task manager**
    - Disconnect and connect the reader
    - Check COM Port number in device manager and set COM port number in kiosk config file: `C:\Users\<user>\AppData\Local\otiKiosk\otiKiosk.cfg`
    - If all the above steps don’t help:
        - Disable the COM port from Device manager and enable it, then try again.
        - Restart the PC

4.3.2 Terminal registration issues

- **Symptoms:**
  - Status = Not ready (Press the 'Get Status' button in test application)
  - Error: "Failed getting terminal operator …code: -997"
- **Possible cause:** Terminal is not registered properly in TMS
- **Suggested steps:**
  - Make sure your terminal ID appears in TMS under the correct operator; consult with OTI support if required.
  - Check kiosk configuration file, line:
    ```
    MS_SVC_URL=http://tms.otiglobal.com
    ```

- **Symptoms:**
  - Transaction ends with error:

    Method TransactionComplete arrived With parameters: Declined' 1' Invalid Device' {…
- **Suggested steps:**
  - Check kiosk configuration file, line:
    ```
    PAYMENT_SVC_URL=https://oti-apriva.azurewebsites.net
    ```
  - Verify the terminal ID is properly registered at Apriva system – consult with OTI or Apriva support team.

4.3.3 Kiosk Core or Test application not updated

- **Symptoms:**
  - Some operations fail with the following error:
    ```
    !! Failed with error: Object reference not set to an instance of an object.
    !! Error Arrived - Code -1000 Message Object reference not set to an instance of an object.
    ```
- **Possible cause:** otiKiosk Core or Test application is not updated to the latest version.
- **Suggested steps:**
  - Close otiKiosk Test Application and Core.
  - Make sure otiKiosk.exe process is not running in the background.
  - Follow otiKiosk installation instructions.

4.3.4 Encryption Keys Issues (DUKPT)

- **Symptoms:**
  - Transaction returned with errors:

    Decrypt Read Head Data Failed:Error in TokenManager(RA). code: 99

    Or

    Error 1 (can be related to other issues as well)
- **Possible cause:** Wrong Encryption keys (DUKPT):
- **Suggested steps:**
  - Check log for "**Ksn**":

    Transaction result >> **Ksn** (DF 81 6A): 00 00 0A FE 01 64 1F 60 00 02
  - Byte 4 in KSN: FE = Apriva Test

    FF, FD = OTI Test (can be changed with DUKPT utility)

    03 = Apriva production (can't be changed)

5 Release Notes

5.1 SDK Version 2.2

5.1.1 Updates & Fixed issues from previous version of the SDK

1. Fixed issue with Currency code format that was showing 'HUFFt' instead of 'USD'.
2. Linux: fixed synchronization issues between core and test application that caused JSON errors.
3. The POS doesn't have to close the PreAuthorize session (with Confirm or Void/Cancel) before it starts a new PreAuthorization session – see more details in the Message Sequences section.
4. All Boolean parameters fixed to be **true** (lower case letters, no quotation marks), in previous versions some messages were using **"True"** as parameter.
5. The PC Test application shows detailed log of the communication between the kiosk core and the POS application.
6. The SDK includes the following Reader configuration files:

Reader Configuration file	File name
EMV without display messages	Apriva_EMV_45.sbc
EMV with display messages	Apriva_EMV_45**d**.sbc
MagStripe Only without display messages	Apriva_MSR_05.sbc
MagStripe Only with display messages	Apriva_MSR_05**d**.sbc

   These configuration files can be found at the SDK folder:
   .\Utilities\Reader FW and Config

   See more details about display messages at the Reader Message Event description.

7. Reader firmware and configuration updated to support the new Card Token feature.
   For this feature to work, 2 new tags were added ot the '**Clear results tag list**' (DF 81 66):
   Tag **DFA502** – Card token strong (based on Partial PAN, Expiry date and Card holder name)
   Tag **DFA503** – Card token weak (based on Partial PAN and Expiry date)

5.1.2 Known issues in SDK Version 2.2

1. Duplicate STAN error – When moving the cashless reader from one Kiosk PC to another, you may get transactions declined due to "Duplicate STAN" error. In such case, you should ask Apriva support to reset the STAN in their system, or workaround this issue by manipulating the STAN counter on the Kiosk PC. The STAN counter is kept on "C: users\current user\AppData\Local\otiKiosk\Meta.dat"
   You may edit this file with text editor and set it to a number that is higher than the server STAN.
2. The SDK was not tested with UNO-6 (Saturn 6700).

5.2 SDK Version 2.1

5.2.1 New features in SDK Version 2.1

1. Support **EMV CDA mode 1** (based on EMVCo v4.3 specifications)
2. **Send Event Message to POS with the Reader Display messages index**:
    a. otiKiosk core can send Reader Display messages to the POS application.
    b. These events can be used to synchronize the kiosk application with the Reader state .
    c. NOTE: For this feature to work, the reader must be configured for sending External Display message events (Tag DF 46 with index 01 - Send only Preset Message Index).
    d. The SDK includes Reader configuration files with and without these events:

Reader Configuration file	XML format	Binary format
EMV without display messages	Apriva_EMV_41.fc	Apriva_EMV_41.sbc
EMV with display messages	Apriva_EMV_41**d**.fc	Apriva_EMV_41**d**.sbc
MagStripe Only without display messages	Apriva_MSR_04_USA.fc	Apriva_MSR_04_USA.fc
MagStripe Only with display messages	Apriva_MSR_04**d**_USA.fc	Apriva_MSR_04**d**_USA.fc

These configuration files can be found at the SDK folder: **.\Utilities\Reader FW and Config**

See more details at the Reader Message Event description.

3. **Improved installation process:**
    a. The setup file takes care of the complete otiKiosk package installation including the otiKiosk Core, Test application and Sample code.
    b. Setup files can be found at the SDK package sub folders:
        i. Windows: .\Windows**otiKiosk v2.1.0 Setup.msi**
        ii. Linux: .\Linux**install.sh**

5.2.2 Updates & Fixed issues from previous version of the SDK

8. Fixed issue with Currency code format.

5.2.3 Known issues in SDK Version 2.1

3. Duplicate STAN error – When moving the cashless reader from one Kiosk PC to another, you may get transactions declined due to "Duplicate STAN" error. In such case, you should ask Apriva support to reset the STAN in their system, or workaround this issue by manipulating the STAN counter on the Kiosk PC. The STAN counter is kept on “C: users\current user\Update\Local\otiKiosk\Meta.dat”
   You may edit this file with text editor and set it to a number that is higher than the server STAN.
4. The SDK was not tested with UNO-6 (Saturn 6700).

5.3 SDK Version 2.0

5.3.1 New features in SDK Version 2.0

1. **New Kiosk Core for Linux (v1.5.x):**
    a. The interface between Kiosk Core and POS application on Linux has been changed. **The new Interface method is not backward compatible with previous version.** See details in the Linux platform Transport rules section.
    b. Core sends the payment interface (MagStripe or Contactless) to the PSP (Data Chanel tag DF70) and to the Kiosk application via the Transaction completed event (Authorization details | Channel field)
    c. Added Refund Transaction API
    d. Kiosk core can use TLS for Payment server communication
    e. The kiosk core will force update in case the Kiosk user application doesn't call StartUpdate command within UPDAE_INTERVAL (default is 10 minutes) from the UpdateIsPending event.
2. **Test application improvements:**
    a. Added a label to show the Payment results 'Data Chanel' (MagStripe / Contactless / Contact)
    b. Added a button to open a receipt sample.

5.3.2 Updates & Fixed issues from previous version of the SDK

9. Saturn Management Utility was updated to version 2.15
10. Windows Test application automatically closes the Core when exited.
11. Fixed issue with kiosk core on Linux erasing the configuration file.

5.3.3 Known issues in SDK Version 2.0

5. Duplicate STAN error – When moving the cashless reader from one Kiosk PC to another, you may get transactions declined due to "Duplicate STAN" error. In such case, you should ask Apriva support to reset the STAN in their system, or workaround this issue by manipulating the STAN counter on the Kiosk PC. The STAN counter is kept on "C: users\current user\AppData\Local\otiKiosk\Meta.dat"
You may edit this file with text editor and set it to a number that is higher than the server STAN.

5.4 SDK Version 1.9

5.4.1 New features in SDK Version 1.9

2. **New Kiosk Core (v1.3.2.8):**
    - c. DLLs are signed with Strong names
    - d. Core sends the payment interface (MagStripe or Contactless) to the PSP (Data Chanel tag DF70) and to the Kiosk application via the Transaction completed event (Authorization details | Channel field)
    - e. Added Refund Transaction API
    - f. Kiosk core can use TLS for server communication
3. **TMS improvements:**
    - a. Added support for UPLOAD_LOG command sent from TMS
    - b. The kiosk core will force update in case the Kiosk user application doesn't call StartUpdate command within 10 minutes from the UpdateIsPending event.
4. **Test application improvements:**
    - a. Added a checkbox for **automatic TMS updates -** when checked, there is no need to press the 'Update' button and TMS pending updates will be done immediately.
    - b. Added a button for performing the new Refund command.
    - c. Added a label to show the Payment results 'Data Chanel' (MagStripe / Contactless / Contact)

5.4.2 Updates & Fixed issues from previous version of the SDK

12. Updated TRIO firmware and configuration (TRIO_6500_SDK_v4.3.Oef, Apriva_MSR_04_USA.scf)
13. Updated Saturn Management Utility (v2.3.4.21)
14. Calling the Cancel command after Authorize will Void the last transaction

5.4.3 Known issues in SDK Version 1.9

6. Duplicate STAN error – When moving the cashless reader from one Kiosk PC to another, you may get transactions declined due to "Duplicate STAN" error. In such case, you should ask Apriva support to reset the STAN in their system, or workaround this issue by manipulating the STAN counter on the Kiosk PC. The STAN counter is kept on "C: users\current user\AppData\Local\otiKiosk\Meta.dat"
   You may edit this file with text editor and set it to a number that is higher than the server STAN.