

SDK DOCUMENT

C#

VERSION 1.4.6

Serino Inc

Revision History

Date	Revision	Author	Description
2022-12-07	1.4.6	Mark Olimberio	Changed encoding from ASCII to ISO-8859-1
2022-11-23	1.4.5	Mark Olimberio	Updated to reset timeout when sending a new command
2022-10-14	1.4.4	Mark Olimberio	Added TableId in REP Field Added Handling of new REP Field
2022-08-23	1.4.3	Mark Olimberio	Changed the version to 1.4.3 to align with the SDK library version
2022-03-04	2.2.1	Winston Carmelo	Updated the flow for the timeout under section 3.2. And updated the diagram of figure 5.
2021-12-17	2.2.0	Winston Carmelo	Added AutoTicket support for transactions.
2021-07-12	2.1.0	Michael Angelo Natanawan	Add Support for Multiple Ingenico Terminal Device for Pay@Table Mode under section 2.3 Multiple Client.
2021-04-23	2.0.2	Michael Natanawan Steven Lester Tan Xedrick Camba	Added External Referral Management EOT Fix
2020-11-26	2.0.1	Ariel Lagonsin Steven Lester Tan Xedrick Camba	Added Timeout Handling
2020-10-19	2.0.0	Ariel Lagonsin Steven Lester Tan Xedrick Camba	Added Pay@Table section Added Pay@Table methods and enums More detailed definition of commands, responses, classes and enums
2020-04-30	1.0.1	Steven Lester Tan Rommel Vallejo	Addition of Java sample codes Addition of sample code for Serial terminal configuration
2020-02-28	1.0.0	Peter Yanong Steven Lester Tan	Starting document

Content

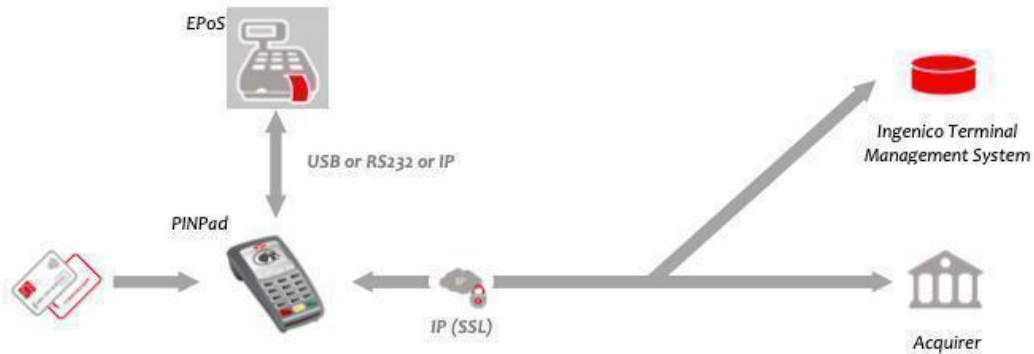
Revision History	2
Content	3
Introduction	5
Initializing a connection from the EPoS to the terminal	6
Setting up a Connection Configuration	6
Creating a Device	6
Multiple Client	7
Request Message from EPoS to the Terminal	8
Message Construction	8
Timeout Handling	8
Timeout Error Messages	9
Response data from the Terminal to the EPoS	10
IngenicoTerminalResponse	10
REP Fields	10
ReportResponse	11
IngenicoTerminalReportResponse	11
StateResponse	11
POSIentifierResponse	12
Handling of broadcast messages	12
Commands	12
Payment Management	12
Sale	13
Refund	14
Hotel Pre-Authorization	15
Hotel Completion	16
Account Verification	17
Tax Free Cash Refund	18
Tax Free Credit Card Refund	19
External Referral Management	20
Transaction Management	21
Cancel	21
Duplicate	21
Reverse	22
Reverse with Transaction ID	23
Report Management	24
EOD	24

Banking	25
XBAL	25
BAL	26
XML Management	26
Report	26
Ticket	27
SplitR	27
Tax Free	28
Terminal Management	28
Call TMS	28
Logon	29
Reset	29
State	30
PID	30
Pay@Table Mode	32
Communication Flow / Initiating a Transaction	32
Handling Terminal Requests	33
PATRequest	33
Responding to Terminal Requests	34
Confirmation Response	34
XML Response	35
Request Types	35
Table List	36
Table Lock	36
Receipt Request	37
Transaction Outcome	38
Additional Message	38
Split Sale Report Data	39
Final Report Data	40
Ticket	40
Transfer Data	41
EOD Report	41
Table Unlock	42
Appendix	42
ConnectionConfig	42
IDeviceInterface	43
TerminalAuthBuilder	46
TerminalManageBuilder	47
IngenicoTerminalResponse	47
IngenicoTerminalReportResponse	48
Enums	49
REFERENCES	51

Introduction

The EPoS interface is built on top of the SDK and follows a communication flow as illustrated in Figure 1. The SDK provides an integrator with the capability necessary to handle this process. It enables the integrator to establish a connection with the terminal, perform transactional requests and receive response data.

The communication flow of EPoS Interface with a PINPad is as follows:



The message flow for EPoS Interface with a PINPad is as follows:

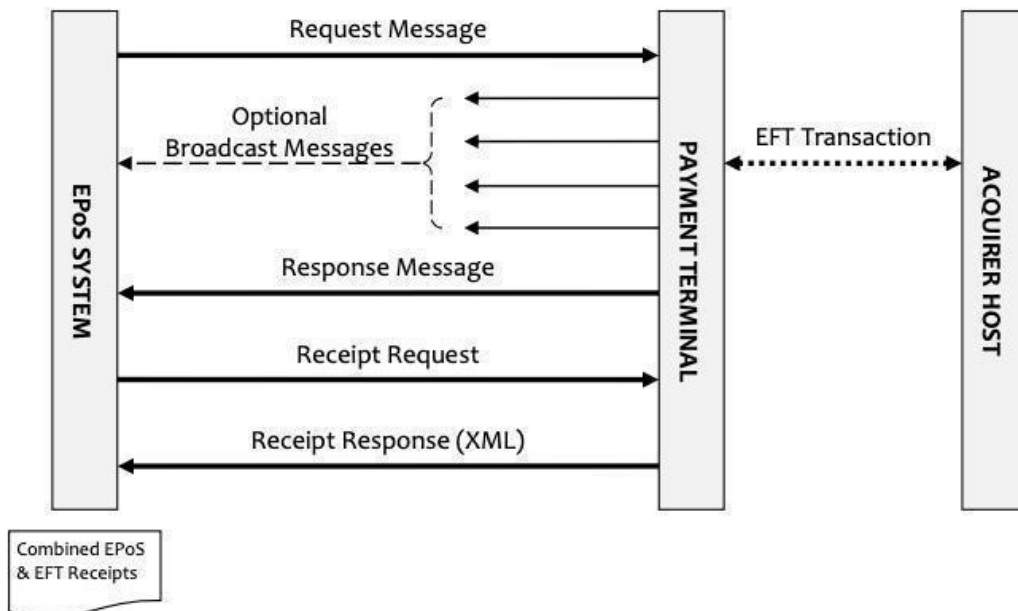


Figure 1: Referenced from Ingenico Documentation (Section 2, Page 9)

- See References section for more details on the reference document hereinafter referred to as “Ingenico Documentation”.
- The SDK supports commands for receipt printing on PINPad terminals (Refer to Ingenico Documentation section 2.1.2 for a list of terminal hardware) - see section (5.4) for more details.
- Display of optional broadcast messages are also supported - see section (4.6) for more details.

1. Initializing a connection from the EPoS to the terminal

A connection from the EPoS to the terminal needs to be initialized before we can perform requests. To set this up, import or use the following headers or namespaces `GlobalPayments.Api.Terminal`, `GlobalPayments.Api.Services`, `GlobalPayments.Api.Entities` in your class then setup your connection by referring to the sample code below.

1.1 Setting up a Connection Configuration

See appendix 7.1 for more details on `ConnectionConfig` class.

TCP / IP Connection

```
ConnectionConfig config = new ConnectionConfig();
config.DeviceType = DeviceType.INGENICO;
config.ConnectionMode = ConnectionModes.TCP_IP_SERVER;
config.Port = "18101";
config.Timeout = 65000;
```

Serial Connection

```
ConnectionConfig config = new ConnectionConfig();
config.DeviceType = DeviceType.INGENICO;
config.Port = 9;
config.ConnectionMode = ConnectionModes.SERIAL;
config.BaudRate = BaudRate.r9600;
config.DataBits = DataBits.Seven;
config.Parity = Parity.Even;
config.StopBits = StopBits.One;
config.HandShake = HandShake.None;
config.Timeout = 65000;
```

Pay@Table

```
ConnectionConfig config = new ConnectionConfig();
config.DeviceType = DeviceType.INGENICO;
config.ConnectionMode = ConnectionModes.PAY_AT_TABLE;
config.Port = "18101";
```

1.2 Creating a Device

Use this connection configuration to create a "Device" object which represents the terminal's interface. This device object will later on be used to perform transactions/requests. See appendix 7.2 for more details on `IDeviceInterface` class.

```
IDeviceInterface device = DeviceService.Create(config);
```

Figure 2: Device initialization

Assuming the proper configuration was provided, calling the method `DeviceService.Create()` initializes the communications interface (IP Port / Serial Port) then listens and waits for a terminal to connect. It successfully returns as soon as a connection is established. The method however, throws an exception and returns nil if there are errors

regarding the configuration or device initialization. For TCP/IP Communication, the method immediately returns a response regardless if the terminal is connected. To check if there is a connected device a method `device.ConnectedDevicesCount()` can be called. This method returns the number of connected devices to the SDK.

1.3 Multiple Client

The SDK supports multiple device connection only on Pay@Table Mode. The SDK can receive multiple PATRequests from different devices which can be received thru `PayAtTableRequestEventHandler` for processing. `PATRequest` class has a `TcpClient` class member which would be a reference for each device. Please note that terminal should have a unique IP Address, in the event of 2 or more devices shares the same IP Address the 1st device that connects that will be recognized. The recommended max client connections should be 20 terminals.

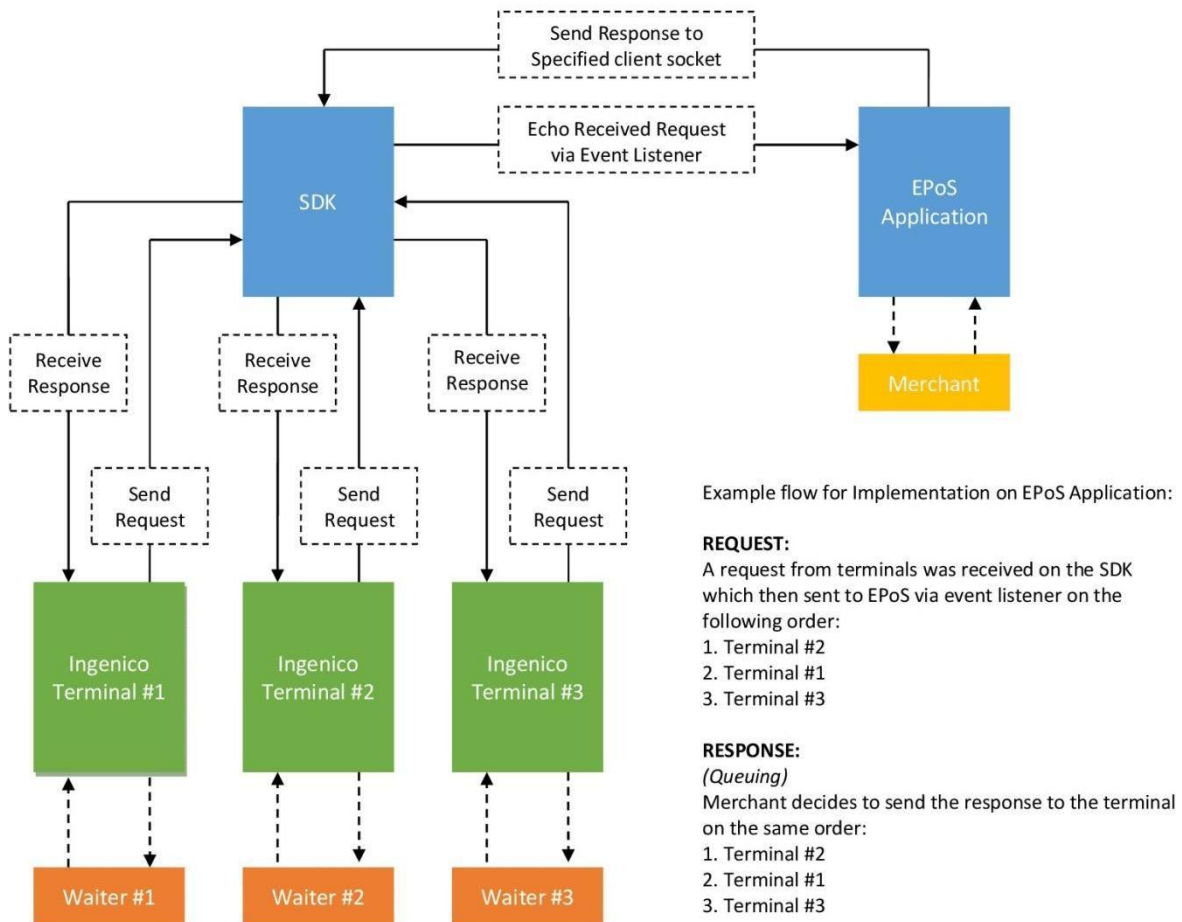


Figure 3: Multiple Device Communication Flow

2. Request Message from EPoS to the Terminal

2.1 Message Construction

Sending of request messages can now be performed from the EPoS to the terminal using the initialized device object from the previous section. To demonstrate, please refer to the simple Sale command/request below:

```
try {
    IngenicoTerminalResponse response = new
    IngenicoTerminalResponse ();
    response = (IngenicoTerminalResponse) device.Sale(12.5m)
                .WithReferenceNumber(1)
                .WithPaymentMode(PaymentMode.APPLICATION)
                .WithCurrencyCode("826")
                .Execute();
} catch (ApiException e) {
    throw e;
}
```

Figure 4: Sample Sale request

From the above example, we can get an idea of the basic anatomy of a request message. It comes with only two parts, a “build” and an “execute” phase.

The method `device.Sale()` returns an object of type `TerminalAuthBuilder` which is built on by other `TerminalAuthBuilder` methods through method chaining syntax. These methods provide a way for arguments to be passed onto our request. In this example, `WithReferenceNumber`, `WithPaymentMode` and `WithCurrencyCode` were used.

However, not all `TerminalAuthBuilder` methods can be applied to a command/request. To see which ones are applicable for each command, see section (5).

For more details on the `TerminalAuthBuilder` class in general see section (7.3).

Now that we’ve “built” our request message. We simply call on `Execute()` to send our request to the terminal. The transaction is executed on the terminal as soon as it’s received. A card may now be presented, inserted or swiped as appropriate.

Once the transaction is completed, the `Execute()` method returns an `IngenicoTerminalResponse` object.

The handling for this object is discussed further in the next section.

Request messages can also be surrounded with a try-catch statement as illustrated in Figure 4. The `Execute()` method throws exceptions such as invalid input for the amount, builder errors, connection timeout during an ongoing transaction etc. which can help with troubleshooting.

2.2 Timeout Handling

A timeout timer is always initiated upon successful connection. An error will be thrown in the catch block (Refer to Figure 4) if the amount of idle time (no response/message from the terminal) exceeds what was set for the timeout property upon device initialization. (Unless the value was set to 0 in which case timeout is disabled)

The timer is reset every time the SDK sends a request message, receives data from the terminal such as final response and broadcast messages. Broadcast messages can be disabled essentially leaving the SDK clueless as to when it would initiate the timeout if it were to follow Figure 5 (right), so the SDK developers opted for the current approach as illustrated in Figure 5 (left).

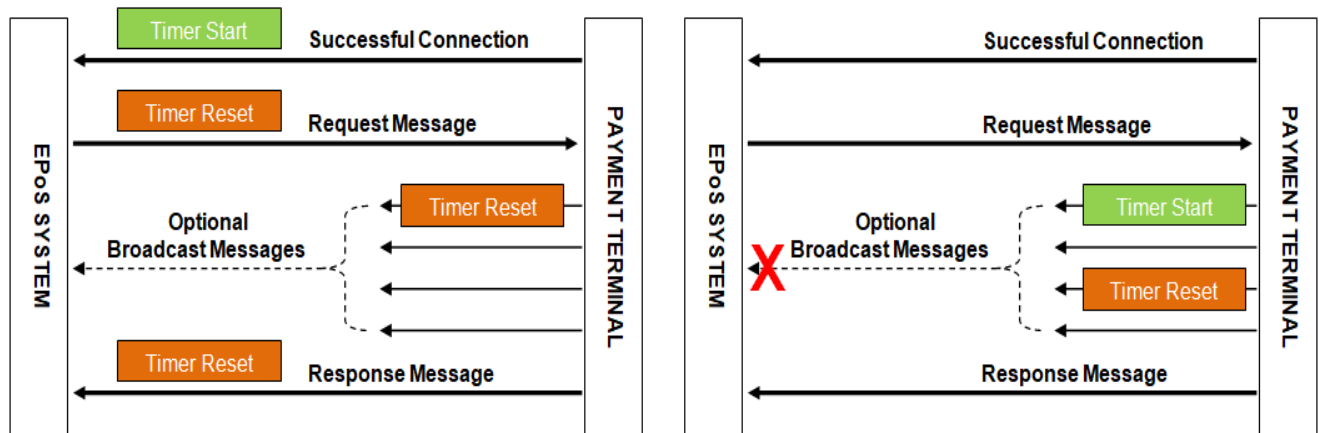


Figure 5: Timeout is initiated upon successful connection (left).
Timeout is initiated only when the SDK receives a response from the terminal (right).

This approach (Figure 5, left) may have implications in certain transaction flows such as when gratuity is enabled for example. Broadcast messages are not yet thrown by the terminal before the gratuity is performed so a timeout may occur if the user takes too long to respond since the timer would not be reset on the SDK. This would then trigger a timeout error leaving the SDK unable to receive responses while the terminal carries on with the transaction.

The handling for these types of cases are beyond the scope of the SDK's capability. It is therefore the integrator's responsibility to properly handle scenarios such as these and provide a reasonable amount of timeout interval to accommodate their needs. The interval can also be set to 0 as mentioned previously so that a timeout wouldn't occur. The SDK currently does not support automatic reversal of transactions so any that might have gone through without the SDK's intervention would be up to the integrator to deal with accordingly whether it is to perform a manual reversal, void, etc.

If the connection between EPOS and terminal during idle time needs to be maintained, the EPOS IP Keep Alive configuration parameter in the terminal must be enabled and the EPOS IP Keep Alive Timer configuration parameter value must be equal or less than the SDK timeout.

2.2.1 Timeout Error Messages

Timeout is returned as an error message which can be printed using the exception object in Figure 4.

Error messages for the following connection modes are listed below:

TCP – “Unable to read data from the transport connection: A connection attempt failed because the connected party did not properly respond after a period of time, or the established connection failed because the connected host failed to respond.”

Serial – “Terminal did not respond within timeout.”

3. Response data from the Terminal to the EPOS

Response data is returned at the end of every transaction. The type of response data from the terminal depends on the type of request. Request messages are grouped into modules which will be discussed in section 5. Here, we specify which modules (including its commands) fall under a particular type of response data and give an example for each type of response.

3.1 IngenicoTerminalResponse

This response is returned by the following modules:

- Payment Management (Sale with/without Cashback, Refund, Hotel Pre-auth, Hotel Completion, Account Verification)
- Transaction Management (Cancel, Duplicate, Reverse, Reverse with ID)
- Terminal Management (Call TMS, Logon, Reset)

Its base response values can be extracted from the following properties/methods (Refer to appendix 7.6 for more details):

```
string refNum = response.ReferenceNumber;  
string status = response.Status;  
decimal amount = response.TransactionAmount;  
PaymentMode paymentMode = response.PaymentMode;  
string currencyCode = response.CurrencyCode;  
string privateData = response.PrivateData;  
string receipt = response.Receipt;
```

3.1.1 REP Fields

The REP field in the response message is used to advise the EPOS system of specific details regarding the transaction (such as the Auth Code and how the transaction amount is broken down). (Refer to Ingenico Documentation section 7.2)

```
string authCode = response.AuthorizationCode;  
decimal cashBackAmount = response.CashBackAmount;  
decimal gratuityAmount = response.TipAmount;  
decimal finalAmount = response.FinalAmount;  
decimal availableAmount = response.BalanceAmount;  
string paymentMethod = response.PaymentType;  
TransactionSubTypes txnSubType = response.TransactionSubType;  
decimal splitSaleAmount = response.SplitSaleAmount;  
DynamicCurrencyStatus dccStatus = response.DccStatus;  
decimal dccAmount = response.DccAmount;  
string dccCurrency = response.DccCurrency;  
string tableId = response.TableId;
```

Not all properties/methods will return a value. These will vary according to the command. For an in-depth discussion of a command's request/response, see section (5).

See appendix 7.8 for more info on enum types such as `PaymentMode`, `TransactionSubTypes` and `DynamicCurrencyStatus`.

3.2 ReportResponse

This is a subclass of `IngenicoTerminalResponse` so it has the same properties and methods. It is used for handling commands under:

- o Report Management (EOD, Banking, XBAL, ZBAL)

Its values can be extracted from the following properties/methods: (same as `IngenicoTerminalResponse`)

3.3 IngenicoTerminalReportResponse

This response type returns the XML data from a receipt request. As such, requests with this return type fall under:

- o XML Management (Report, Ticket, SplitR, TaxFree)

The xml data can be accessed using the `ToString()` method on the response object (see `Ingenico Documentation` section 6.3.7 for sample data).

```
string xmlData = response.ToString();
```

3.4 StateResponse

This response type is used specifically for the command `State` under the `Terminal Management` module.

Data can be accessed from the following properties/methods:

(`StateResponse` is a subclass of `IngenicoTerminalResponse`. It only has base response values and no `REP Field` data.)

Additional methods:

```
string terminalStatus = response.TerminalStatus;  
string salesMode = response.SalesMode;  
decimal terminalCapabilities = response.TerminalCapabilities;  
string addlTerminalCapabilities =  
response.AdditionalTerminalCapabilities;  
string appVersion = response.AppVersionNumber;  
string handsetNumber = response.HandsetNumber;
```

```
string terminalID = response.TerminalId;
```

3.5 POSIdentifierResponse

This response type is used specifically for the command PID under the Terminal Management module.

Data can be accessed from the following properties/methods:

(POSIdentifierResponse is a subclass of IngenicoTerminalResponse. It only has base response values and no REP Field data.)

Additional methods:

```
string serialNumber = response.SerialNumber;
```

3.6 Handling of broadcast messages

Aside from the response data that's received at the end of a transaction, the SDK is also capable of receiving broadcast messages which are sent by the terminal during a live transaction. Refer to the sample code below.

```
device.OnBroadcastMessage += new  
BroadcastMessageEventHandler((code, message) => {  
    Invoke((MethodInvoker) delegate {  
        Console.WriteLine(string.Format("{0} | {1}", code, message));  
    });  
});
```

The above can be declared as soon as you have initialized the device. The event OnBroadcastMessage will only receive messages during an ongoing transaction. For a list of possible broadcast message values, you may refer to Ingenico Documentation section 6.3.9

4. Commands

This section discusses the request and response for each command in more detail. Commands are grouped into modules as mentioned previously and can be executed by our initialized device object from section 3. Below is a summary of supported commands for each module along with their corresponding request/ response examples.

4.1 Payment Management

These commands utilize TerminalAuthBuilder methods to build their request message except for Hotel Completion which uses TerminalManageBuilder. You can read more about these two classes in appendix 7.3 and 7.4 respectively. The return type of these commands is IngenicoTerminalResponse (see appendix 7.6).

4.1.1 Sale

4.1.1.1 Request

```
IngenicoTerminalResponse response = new  
IngenicoTerminalResponse ();  
response = (IngenicoTerminalResponse) device.Sale(100m)  
    .WithReferenceNumber(1)  
    .WithPaymentMode(PaymentMode.APPLICATION)  
    .WithCashback(10m)  
    .WithCurrencyCode("826")  
    .WithTicket(true)  
    .Execute();
```

Builder Type: TerminalAuthBuilder

Additional Methods:

Required
WithReferenceNumber
WithCurrencyCode

Optional
WithCashBack
WithPaymentMode (Default = PaymentMode.APPLICATION)
WithTableNumber
WithTicket

4.1.1.2 Response

Response Type: IngenicoTerminalResponse

Properties:

With Values	
ReferenceNumber	CurrencyCode
Status	PrivateData
TransactionAmount	PaymentType
PaymentMode	BalanceAmount
AuthorizationCode	FinalTransactionAmount

Conditional Values	
TipAmount	Will have a value if gratuity amount is entered on the terminal
CashBackAmount	Will have a value if request has WithCashBack method
Receipt	Will have a value if AutoTicket is enabled. Note: If TICKET is unsuccessful, Receipt will be empty.

4.1.2 Refund

4.1.2.1 Request

```
IngenicoTerminalResponse response = new
IngenicoTerminalResponse(); response =
(IngenicoTerminalResponse) device.Refund(100m)
    .WithReferenceNumber(1)
    .WithPaymentMode(PaymentMode.APPLICATION)
    .WithCurrencyCode("826")
    .WithTicket(true)
    .Execute();
```

Builder Type: TerminalAuthBuilder

Additional Methods:

Required
WithReferenceNumber
WithCurrencyCode

Optional
WithPaymentMode (Default = PaymentMode.APPLICATION)
WithTableNumber
WithTicket

4.1.2.2 Response

Response Type: IngenicoTerminalResponse

Properties:

With Values	
ReferenceNumber	CurrencyCode
Status	PrivateData
TransactionAmount	PaymentType
PaymentMode	BalanceAmount
AuthorizationCode	FinalTransactionAmount

Conditional Values	
Receipt	<p>Will have a value if AutoTicket is enabled</p> <p>Note: If TICKET is unsuccessful, Receipt will be empty.</p>

4.1.3 Hotel Pre-Authorization

4.1.3.1 Request

```

IngenicoTerminalResponse response = new
IngenicoTerminalResponse ();
response = (IngenicoTerminalResponse) device.Authorize(100m)
    .WithReferenceNumber(1)
    .WithTableNumber("1WE3464Y")
    .WithCurrencyCode("826")
    .WithTicket(true)
    .Execute();

```

Builder Type: TerminalAuthBuilder

Additional Methods:

Required
WithReferenceNumber
WithCurrencyCode

Optional
WithTableNumber
WithTicket

4.1.3.2 Response

Response Type: IngenicoTerminalResponse

Properties:

With Values	
ReferenceNumber	CurrencyCode
Status	PrivateData
TransactionAmount	PaymentType
PaymentMode	BalanceAmount
AuthorizationCode	FinalTransactionAmount

Conditional Values	
TipAmount	Will have a value if gratuity amount is entered on the terminal
CashBackAmount	Will have a value if fallback is triggered. Cashback is entered on the terminal.
Receipt	Will have a value if AutoTicket is enabled. Note: If TICKET is unsuccessful, Receipt will be empty.

4.1.4 Hotel Completion

4.1.4.1 Request

```
IngenicoTerminalResponse response = new  
IngenicoTerminalResponse();  
response = (IngenicoTerminalResponse) device.Capture(100m)  
    .WithReferenceNumber(1)  
    .WithAuthCode("123456")  
    .WithCurrencyCode("826")  
    .WithTicket(true)  
    .Execute();
```

Builder Type: TerminalAuthBuilder

Additional Methods:

Required
WithReferenceNumber
WithCurrencyCode

Optional
WithAuthCode
WithTicket

4.1.4.2 Response

Response Type: IngenicoTerminalResponse

Properties:

With Values	
ReferenceNumber	CurrencyCode
Status	PrivateData
TransactionAmount	PaymentType
PaymentMode	BalanceAmount
AuthorizationCode	FinalTransactionAmount

Conditional Values	
TipAmount	Will have a value if gratuity amount is entered on the terminal
CashBackAmount	Will have a value if fallback is triggered. Cashback is entered on the terminal
Receipt	<p>Will have a value if AutoTicket is enabled.</p> <p>Note: If TICKET is unsuccessful, Receipt will be empty.</p>

4.1.5 Account Verification

4.1.5.1 Request

```
IngenicoTerminalResponse response = new
IngenicoTerminalResponse ();
response = (IngenicoTerminalResponse) device.Verify()
    .WithReferenceNumber(1)
    .WithCurrencyCode("826")
    .WithTicket(true)
    .Execute();
```

Builder Type: TerminalAuthBuilder

Additional Methods:

Required
WithReferenceNumber
WithCurrencyCode

Optional
WithTicket

4.1.5.2 Response

Response Type: IngenicoTerminalResponse

Properties:

With Values	
ReferenceNumber	CurrencyCode
Status	PrivateData
TransactionAmount	PaymentType
PaymentMode	

Conditional Values	
Receipt	<p>Will have a value if AutoTicket is enabled.</p> <p>Note: If TICKET is unsuccessful, Receipt will be empty.</p>

4.1.6 Tax Free Cash Refund

4.1.6.1 Request

```
IngenicoTerminalResponse response = new
IngenicoTerminalResponse ();
response = (IngenicoTerminalResponse) device.Refund(100m)
    .WithReferenceNumber(1)
    .WithPaymentMode(PaymentMode.APPLICATION)
    .WithCurrencyCode("826")
    .WithTaxFree(TaxFreeType.CASH)
    .Execute();
```

Builder Type: TerminalAuthBuilder

Additional Methods:

Required

WithReferenceNumber
WithCurrencyCode
WithTaxFree

Optional
WithPaymentMode (Default = PaymentMode.APPLICATION)

4.1.6.2 Response

Response Type: IngenicoTerminalResponse

Properties:

With Values	
ReferenceNumber	CurrencyCode
Status	PrivateData
TransactionAmount	PaymentType
PaymentMode	

4.1.7 Tax Free Credit Card Refund

4.1.7.1 Request

```
IngenicoTerminalResponse response = new
IngenicoTerminalResponse ();
response = (IngenicoTerminalResponse) device.Refund(100m)
    .WithReferenceNumber(1)
    .WithPaymentMode(PaymentMode.APPLICATION)
    .WithCurrencyCode("826")
    .WithTaxFree(TaxFreeType.CREDIT)
    .Execute();
```

Builder Type: TerminalAuthBuilder

Additional Methods:

Required
WithReferenceNumber
WithCurrencyCode
WithTaxFree

Optional

WithPaymentMode (Default = PaymentMode.APPLICATION)

4.1.7.2 Response

Response Type: IngenicoTerminalResponse

Properties:

With Values	
ReferenceNumber	CurrencyCode
Status	PrivateData
TransactionAmount	PaymentType
PaymentMode	

4.1.8 External Referral Management

4.1.8.1 Request

```
IngenicoTerminalResponse response = new  
IngenicoTerminalResponse ();  
response =  
(IngenicoTerminalResponse) device.ReferralConfirmation()  
    .WithReferenceNumber(1)  
    .WithCurrencyCode("826")  
    .WithAuthCode("123456")  
    .WithTicket(true)  
    .Execute();
```

Builder Type: TerminalManageBuilder

Additional Methods:

Required
WithReferenceNumber
WithCurrencyCode

Optional
WithAuthCode
WithTicket

4.1.8.2 Response

Response Type: IngenicoTerminalResponse

Properties:

With Values	
ReferenceNumber	CurrencyCode
Status	PrivateData
TransactionAmount	PaymentType
PaymentMode	BalanceAmount
AuthorizationCode	FinalTransactionAmount

Conditional Values	
Receipt	Will have a value if AutoTicket is enabled. Note: If TICKET is unsuccessful, Receipt will be empty.

4.2 Transaction Management

These commands are used to handle various transaction management tasks. Their return type is `IngenicoTerminalResponse`. Refer to Ingenico Documentation section 5.2 for more details.

4.2.1 Cancel

4.2.1.1 Request

Does not utilize a builder. Executed upon method call.

```
IngenicoTerminalResponse response = new  
IngenicoTerminalResponse ();  
response = (IngenicoTerminalResponse) device.Cancel ();
```

4.2.1.2 Response

Response Type: `IngenicoTerminalResponse`

Properties:

With Values	
ReferenceNumber	CurrencyCode
Status	PrivateData
TransactionAmount	PaymentMode

4.2.2 Duplicate

4.2.2.1 Request

Does not utilize a builder. Executed upon method call.

```
IngenicoTerminalResponse response = new  
IngenicoTerminalResponse ();  
response = (IngenicoTerminalResponse) device.Duplicate(true);
```

4.2.2.2 Response

Response Type: IngenicoTerminalResponse

Properties:

With Values	
ReferenceNumber	CurrencyCode
Status	PrivateData
TransactionAmount	PaymentMode

Conditional Values	
Receipt	Will have a value if AutoTicket is enabled. Note: If TICKET is unsuccessful, Receipt will be empty.

4.2.3 Reverse

4.2.3.1 Request

```
IngenicoTerminalResponse response = new  
IngenicoTerminalResponse ();  
response = (IngenicoTerminalResponse) device.Reverse(100m)  
    .WithReferenceNumber(1)  
    .WithTicket(true)  
    .Execute();
```

Builder Type: TerminalMangeBuilder

Additional Methods:

Required
WithReferenceNumber

Optional
WithPaymentMode (Default = PaymentMode.APPLICATION)

WithTicket

4.2.3.2 Response

Response Type: IngenicoTerminalResponse

Properties:

With Values	
ReferenceNumber	CurrencyCode
Status	PrivateData
TransactionAmount	PaymentMode

Conditional Values	
Receipt	<p>Will have a value if AutoTicket is enabled.</p> <p>Note: If TICKET is unsuccessful, Receipt will be empty.</p>

4.2.4 Reverse with Transaction ID

4.2.4.1 Request

```
IngenicoTerminalResponse response = new
IngenicoTerminalResponse ();
response = (IngenicoTerminalResponse) device.Reverse (100m)
    .WithReferenceNumber (1)
    .WithTransactionId ("8153")
    .WithTicket (true)
    .Execute ();
```

Builder Type: TerminalMangeBuilder

Additional Methods:

Required
WithReferenceNumber
WithTransactionId

Optional
WithTicket

4.2.4.2 Response

Response Type: IngenicoTerminalResponse

Properties:

With Values	
ReferenceNumber	CurrencyCode
Status	PrivateData
TransactionAmount	PaymentMode

Conditional Values	
Receipt	Will have a value if AutoTicket is enabled. Note: If TICKET is unsuccessful, Receipt will be empty.

4.3 Report Management

These commands utilize TerminalReportBuilder (see appendix 7.5) to build their request message. Their return type is ReportResponse (see section 4.2). Refer to Ingenico Documentation section 5.3 for more details.

4.3.1 EOD

4.3.1.1 Request

```
ReportResponse response = new ReportResponse();  
response = (ReportResponse) device.GetReport(ReportType.EOD)  
    .Execute();
```

Builder Type: TerminalReportBuilder

Additional Methods: None

4.3.1.2 Response

Response Type: IngenicoTerminalResponse

Properties:

With Values	
ReferenceNumber	CurrencyCode

Status	PrivateData
TransactionAmount	PaymentMode

4.3.2 Banking

4.3.2.1 Request

```
ReportResponse response = new ReportResponse();
response =
    (ReportResponse) device.GetReport(ReportType.BANKING)
        .Execute();
```

Builder Type: TerminalReportBuilder

Additional Methods: None

4.3.2.2 Response

Response Type: IngenicoTerminalResponse

Properties:

With Values	
ReferenceNumber	CurrencyCode
Status	PrivateData
TransactionAmount	PaymentMode

4.3.3 XBAL

4.3.3.1 Request

```
ReportResponse response = new ReportResponse();
response = (ReportResponse) device.GetReport(ReportType.XBAL)
    .Execute();
```

Builder Type: TerminalReportBuilder

Additional Methods: None

4.3.3.2 Response

Response Type: IngenicoTerminalResponse

Properties:

With Values	
ReferenceNumber	CurrencyCode
Status	PrivateData
TransactionAmount	PaymentMode

4.3.4 BAL

4.3.4.1 Request

```
ReportResponse response = new ReportResponse();  
response = (ReportResponse) device.GetReport(ReportType.ZBAL)  
    .Execute();
```

Builder Type: TerminalReportBuilder

Additional Methods: None

4.3.4.2 Response

Response Type: IngenicoTerminalResponse

Properties:

With Values	
ReferenceNumber	CurrencyCode
Status	PrivateData
TransactionAmount	PaymentMode

4.4 XML Management

These commands utilize TerminalReportBuilder to build their request message. Their return type is IngenicoTerminalReportResponse (see section 4.3). Refer to Ingenico Documentation section 5.1 for more details.

4.4.1 Report

4.4.1.1 Request

```
IngenicoTerminalReportResponse response = new  
IngenicoTerminalReportResponse();  
response = (IngenicoTerminalReportResponse) device  
    .GetLastReceipt(ReceiptType.REPORT)  
    .Execute();
```

Builder Type: TerminalReportBuilder

Additional Methods: None

4.4.1.2 Response

Response Type: IngenicoTerminalReportResponse

Methods:

With Values
ToString

4.4.2 Ticket

4.4.2.1 Request

```
IngenicoTerminalReportResponse response = new
IngenicoTerminalReportResponse();
response = (IngenicoTerminalReportResponse)device
    .GetLastReceipt(ReceiptType.TICKET)
    .Execute();
```

Builder Type: TerminalReportBuilder

Additional Methods: None

4.4.2.2 Response

Response Type: IngenicoTerminalReportResponse

Methods:

With Values
ToString

4.4.3 SplitR

4.4.3.1 Request

```
IngenicoTerminalReportResponse response = new
IngenicoTerminalReportResponse();
response = (IngenicoTerminalReportResponse)device
    .GetLastReceipt(ReceiptType.SPLITR)
    .Execute();
```

Builder Type: TerminalReportBuilder

Additional Methods: None

4.4.3.2 Response

Response Type: IngenicoTerminalReportResponse

Methods:

With Values
ToString

4.4.4 Tax Free

4.4.4.1 Request

```
IngenicoTerminalReportResponse response = new
IngenicoTerminalReportResponse ();
response = (IngenicoTerminalReportResponse) device
    .GetLastReceipt (ReceiptType.TAXFREE)
    .Execute ();
```

Builder Type: TerminalReportBuilder

Additional Methods: None

4.4.4.2 Response

Response Type: IngenicoTerminalReportResponse

Methods:

With Values
ToString

4.5 Terminal Management

These commands don't utilize a builder for their request messages and are executed upon method call. Special response types were created to handle specific commands such as State and PID namely, StateResponse and POSIdentifierResponse (see sections 4.4 and 4.5). Refer to Ingenico Documentation section 5.4 for more details.

4.5.1 Call TMS

4.5.1.1 Request

```
IngenicoTerminalResponse response = new
IngenicoTerminalResponse ();
response
= (IngenicoTerminalResponse) device.GetTerminalConfiguration ()
;
```

4.5.1.2 Response

Response Type: IngenicoTerminalResponse

Properties:

With Values	
ReferenceNumber	CurrencyCode
Status	PrivateData
TransactionAmount	PaymentMode

4.5.2 Logon

4.5.2.1 Request

```
IngenicoTerminalResponse response = new
IngenicoTerminalResponse ();
response =
(IngenicoTerminalResponse) device.TestConnection(true);
```

4.5.2.2 Response

Response Type: IngenicoTerminalResponse

Properties:

With Values	
ReferenceNumber	CurrencyCode
Status	PrivateData
TransactionAmount	PaymentMode

Conditional Values	
Receipt	<p>Will have a value if AutoTicket is enabled.</p> <p>Note: If TICKET is unsuccessful, Receipt will be empty.</p>

4.5.3 Reset

4.5.3.1 Request

```
IngenicoTerminalResponse response = new
IngenicoTerminalResponse ();
response = (IngenicoTerminalResponse) device.Reboot();
```

4.5.3.2 Response

Response Type: IngenicoTerminalResponse

Properties:

With Values	
ReferenceNumber	CurrencyCode
Status	PrivateData
TransactionAmount	PaymentMode

4.5.4 State

4.5.4.1 Request

```
IngenicoTerminalResponse response = new  
IngenicoTerminalResponse ();  
response =  
(IngenicoTerminalResponse) device.GetTerminalStatus ();
```

4.5.4.2 Response

Response Type: StateResponse

Properties:

With Values	
ReferenceNumber	PrivateData
Status	PaymentMode
TransactionAmount	TerminalStatus
SalesMode	TerminalCapabilities
AdditionalTerminalCapabilities	AppVersionNumber
HandsetNumber	TerminalId

4.5.5 PID

4.5.5.1 Request

```
IngenicoTerminalResponse response = new  
IngenicoTerminalResponse ();  
response = (IngenicoTerminalResponse) device.Initialize ();
```

4.5.5.2 Response

Response Type: POSIdentifierResponse

Properties:

With Values	
ReferenceNumber	CurrencyCode
Status	PrivateData
TransactionAmount	PaymentMode
SerialNumber	

5. Pay@Table Mode

In addition to standard/normal mode, the terminal also supports Pay@Table mode which enables it to initiate transactions with the EPOS to retrieve check/bill information. For more info regarding this mode and how to configure the terminal specifically to enable it, refer to Ingenico Documentation section 7.2 and 7.2.1 respectively.

5.1 Communication Flow / Initiating a Transaction

In contrast with the communication flow of standard/normal mode given by Figure 1 - Section 1 of this document, the flow of Pay@Table follows a different process and starts off with the terminal initiating the transaction instead of the EPOS as illustrated below in Figure 6.

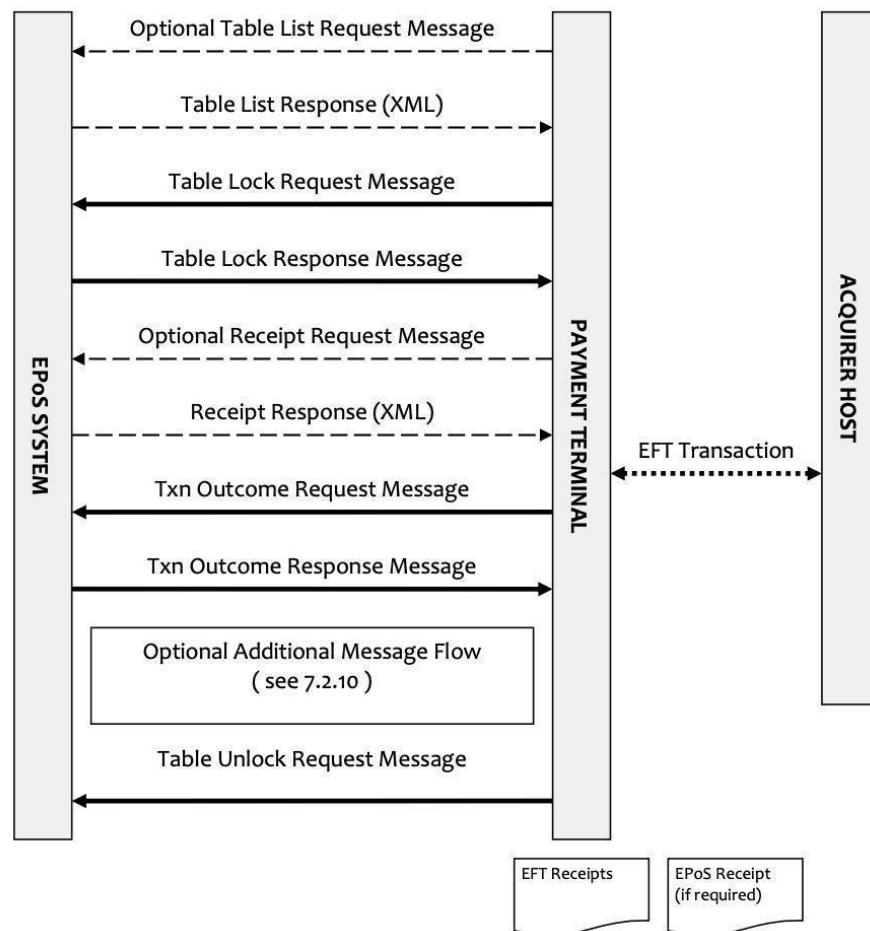


Figure 6: Referenced from Ingenico Documentation (Section 7.2.3, Page 68)

A transaction can be initiated by pressing MENU > SALE on a properly configured terminal. This action causes the terminal to send a request to the EPOS which is subsequently handled by an event as will be discussed further in the next section.

5.2 Handling Terminal Requests

Before proceeding to initiate a transaction, we must first make sure that our EPOS is “listening” - ready to receive requests from the terminal. The SDK provides a way to do this by means of an event handler somewhat similar in declaration to that of broadcast messages.

```
device.OnPayAtTableRequest += new
PayAtTableRequestEventHandler((request) => {
});
```

To reiterate, Pay@Table uses an entirely different communication flow to that of standard/normal mode. As such, the device object illustrated above must be created using the proper ConnectionConfig specific to Pay@Table as stated in section 2.1. The code above can be declared once the device object is properly initialized.

5.2.1 PATRequest

An object of type PATRequest is returned by the event handler which is the class responsible for parsing data from the terminal’s raw request. Depending on the type of request and terminal configuration, values are returned from the following methods:

```
TcpClient client = request.Client;
string waiterID = request.WaiterId;
string tableID = request.TableId;
string terminalID = request.getTerminalId;
string terminalCurrency = request.TerminalCurrency;
string xmlData = request.XMLData;
PATRequestType requestType = request.RequestType;
TransactionOutcome txnOutcome = request.TransactionOutcome;
```

5.2.1.1 PATRequestType

This enum specifies the type of request which can be used later on for handling responses.

Refer to section 7.8 for enum values.

```
PATRequestType requestType = request.RequestType();
if (requestType == PATRequestType.TableLock) {
    //Send Table Lock response
}
```

5.2.1.2 TransactionOutcome

This object is used specifically to handle a request of type `TransactionOutcome`. Values are returned from the following methods:

```
string status = txnOutcome.Status.ToString();
string amount = txnOutcome.Amount;
string currencyCode = txnOutcome.CurrencyCode;
string privateData = txnOutcome.PrivateData;
DataResponse repFields = txnOutcome.RepField;
```

Values in the `RepField` object can be accessed similarly to section 4.1.1.

```
string authCode = response.AuthorizationCode;
decimal finalAmount = repFields.FinalAmount;
string paymentMethod = response.PaymentType;
(...)
```

5.2.1.3 XMLData

For terminal request types which send XML data to the EPoS, only the method below returns a value (aside from `RequestType`).

```
String xmlData = request.XMLData;
```

5.3 Responding to Terminal Requests

As mentioned previously, we can use the request type in order to determine the response that needs to be sent back to the terminal. For each request, a specific response must be provided in order to proceed with the communication flow as illustrated in Figure 5. We will be discussing each request type (in order) in the next section but for now, we will only be focusing on response types.

A method is provided in `IDeviceInterface` specifically for responding to `Pay@Table` requests - `device.PayAtTableResponse(client)`. This is further built upon by `TerminalAuthBuilder` methods similar to how request messages are constructed in standard/normal mode.

5.3.1 Confirmation Response

For request types which require a confirmation response, the following additional `TerminalAuthBuilder` methods must be appended. Given below is an example.

```
if (requestType == PATRequestType.TableLock) {
    try {
```

```

        _device.PayAtTableResponse(client)
            .WithPayAtTableResponseType(PATResponseType.CONF_OK)
            .WithAmount(100m)
            .WithPaymentMode(PATPaymentMode.NO_ADDITIONAL)
            .WithCurrencyCode("826")
            .Execute();

    } catch (ApiException e) {
        throw e;
    }
}

```

This type of response can be used to send a positive (PATResponseType.CONF_OK) or negative (PATResponseType.CONF_NOK) confirmation. Request for additional data can also be specified using this response (PATPaymentMode.USE_ADDITIONAL). Refer to section 7.3 for more info.

5.3.2 XML Response

For request types which require an XML response, the following additional TerminalAuthBuilder methods must be appended. Given below is an example.

```

if (requestType == PATRequestType.TableList) {
    try {
        device.PayAtTableResponse(client)
            .WithXML("C:\\Users\\john.doe\\Desktop\\sample.xml")
            .Execute();
    } catch (ApiException e) {
        throw e;
    }
}

```

The SDK provides a way to send XML data to the terminal through the WithXML() method. The file path to be provided must be a valid xml file in order for the contents to be sent properly. Refer to section 7.3 for more info.

Unlike request messages in standard/normal mode, no specific object (like IngenicoTerminalResponse for example) is returned by the execute method. In Pay@Table mode, we are simply responding to terminal requests. We just have to make sure that our event handler (section 6.2) is always listening so we can respond accordingly. Responses can also be surrounded with try catch statements for error handling (in terms of message construction).

5.4 Request Types

In this section, we will specify the values that are returned for each request type as well as the response that must be sent back to the terminal. This section follows an order which goes through each step in the communication flow described in Figure 5.

5.4.1 Table List

5.4.1.1 Request

Request Type: TableList

Properties:

Required
RequestType

Conditional Values	
WaiterId	Will have a value if Waiter ID is entered on the terminal

5.4.1.2 Response

Response Type: XML Response

XML Content: A list of table ID's listed in XML format. Refer to Ingenico Documentation section 7.2.7 for sample list content.

Notes:

- o "If the Waiter ID is "00" (or empty) then it is expected that a list of all open table is returned, otherwise it is expected that a list of open tables against that Waiter ID is returned" (Referenced from Ingenico Documentation section 7.2.5, Page 71).
- o We recommend using less than 10 characters for Table ID names in the XML file for the sake of compatibility with future API versions of Ingenico Terminals. Special handling was made on the SDK to accommodate Table ID names with 10 characters or more due to issues with length formatting on the current API version.

5.4.2 Table Lock

5.4.2.1 Request

Request Type: TableLock

Properties:

Required
RequestType
TableId

Conditional Values	
WaiterId	Will have a value if Waiter ID is entered on the terminal
TerminalId	Will have a value if enabled in terminal configuration. Refer to Ingenico Documentation section 7.2.1
TerminalCurrency	Will have a value if enabled in terminal configuration. Refer to ingenico Documentation section 7.2.1

5.4.2.2 Response

Response Type: Confirmation Response

Additional Methods:

Required
WithPayAtTableResponseType
WithAmount
WithPaymentMode
WithCurrencyCode

5.4.3 Receipt Request

5.4.3.1 Request

Request Type: ReceiptMessage

Properties:

Required
RequestType

5.4.3.2 Response

Response Type: XML Response

XML Content: Refer to Ingenico Documentation section 7.2.8 for a guide on receipt response construction.

5.4.4 Transaction Outcome

5.4.4.1 Request

Request Type: TransactionOutcome

Properties:

Required
RequestType
TransactionOutcome (Refer to section 6.2.1.2)

5.4.4.2 Response

Response Type: Confirmation Response

Additional Methods:

Required
WithPayAtTableResponseType
WithAmount
WithPaymentMode
WithCurrencyCode

If additional message option is enabled on the terminal, you can set USE_ADDITIONAL on WithPaymentMode to signal the terminal to send additional data to the EPOS.

5.4.5 Additional Message

This type refers to the terminal's request for an additional message list. See Ingenico Documentation section 7.2.10 for more details on Additional Message flow in general.

5.4.5.1 Request

Request Type: ADDITIONAL_MESSAGE

Properties:

Required
RequestType
XMLData

5.4.5.2 Response

Response Type: XML Response

XML Content: Refer to Ingenico Documentation section 7.2.10.2 for a guide on additional message list construction

5.4.6 Split Sale Report Data

Will only be returned if Split Sale has been included in the additional data list.

5.4.6.1 Request

Request Type: SplitSaleReport

Properties:

Required
RequestType
XMLData

5.4.6.2 Response

Response Type: Confirmation Response

Additional Methods:

Required
WithPayAtTableResponseType
WithAmount
WithPaymentMode
WithCurrencyCode

5.4.7 Final Report Data

Will only be requested if Final Report has been included in the additional data list. Same request/response as section 6.4.3

5.4.8 Ticket

Will only be returned if Ticket has been included in the additional data list.

5.4.8.1 Request

Request Type: TICKET

Properties:

Required
RequestType
XMLData

5.4.8.2 Response

Response Type: Confirmation Response

Additional Methods:

Required
WithPayAtTableResponseType
WithAmount
WithPaymentMode
WithCurrencyCode

5.4.9 Transfer Data

Refers to a transfer data request just before sending the EOD report. See Ingenico Documentation section 7.2.12 for more details.

5.4.9.1 Request

Request Type: TransferData

Properties:

Required
RequestType
XMLData

5.4.9.2 Response

Response Type: Confirmation Response

Additional Methods:

Required
WithPayAtTableResponseType
WithAmount
WithPaymentMode
WithCurrencyCode

5.4.10 EOD Report

Sent after a Transfer Data request is received.

5.4.10.1 Request

Request Type: EndOfDayReport

Properties:

Required
RequestType
XMLData

5.4.10.2 Response

Response Type: Confirmation Response

Additional Methods:

Required
WithPayAtTableResponseType
WithAmount
WithPaymentMode
WithCurrencyCode

5.4.11 Table Unlock

5.4.11.1 Request

Request Type: TableUnlock

Properties:

Required
RequestType
XMLData

5.4.11.2 Response

Response Type: No response

6. Appendix

6.1 ConnectionConfig

ConnectionConfig class is used to setup a connection to the terminal.

Methods:

Method Name	Description
Baudrate	Connection Baud Rate
ConnectionMode	Connection Mode (See 7.8 for enum values)
DataBits	Connection Data Bits
DeviceType	Type of Device to be used. (See 7.8 for enum values)
Parity	Connection Parity

Port	Port of device.
StopBits	Connection Stop Bits
Timeout	Set a timeout (in milliseconds) for the connection.
ConnectionConfig	Method used to connect the current connection.
Validate	Used to validate connection.

6.2 IDeviceInterface

IDeviceInterface is a class that contains methods which are required to communicate with the terminal.

Some of the methods like Sale, Refund, etc. will be used to instruct the terminal to process transaction.

Methods:

Method Name	Module	Description
Sale (decimal amount)	Payment Management	Perform Sale (Payment Type) transaction into terminal
Refund (decimal amount)		Perform Refund (Payment Type) transaction into terminal
Capture(decimal amount)		Perform Hotel Mode Completion (Payment Type) transaction into terminal
Authorize (decimal amount)		Perform Hotel Mode Pre-Authorization (Payment Type) transaction into terminal
Verify()		Perform Account Verification (Payment Type) transaction into terminal

Method Name	Module	Description
Cancel()	Transaction Management	The terminal will immediately cancel the current transaction
Duplicate(boolean isAutoTicketEnabled)		The terminal will immediately initiates a duplicate of the last completed transaction

Reverse(string transactionId)		This command has two types that immediately start a reversal of the last completed transaction. Implicit Reversal – requires a transaction number. Explicit Reversal – transaction number is not needed. That’s why the parameter for this method has a default value of 0. Please consider that both reversals may only be completed within 80 seconds of the transaction being completed.
GetLastReceipt(ReceiptType type)	XML Management	The terminal returns the XML data for the last completed transaction receipt that is stored in the terminal’s memory. Receipt Type: REPORT TICKET SPLITR TAXFREE
GetReport(ReportType type)	Report Management	The Report method is used to request the XML data for the last completed report (be it an End of Day, Banking, X or Z Balance) that is stored in the terminal’s memory. BANKING - Banking report for all Acquirers. EOD - End of Day report for all Acquirers XBAL - X Balance report ZBAL - Z Balance report

Method Name	Module	Description
GetTerminalConfiguration()	Terminal Management	The terminal immediately initiates a TMS Call to pick up the latest configuration for the terminal.
TestConnection(boolean isAutoTicketEnabled)		The terminal immediately initiates a Logon report to test communications with all Acquirers.
Reboot()		The terminal immediately initiates power cycles of the terminal hardware.
GetTerminalStatus()		The terminal returns the current terminal status

Initialize()		The Initialize (PID) command is used to return the POS identifier stored in the terminal which is limited to 10 characters. This is an optional function on the terminal and if not enabled the terminal will return NO ID
PayAtTableResponse(TcpClient client)	Pay@Table	Command used for responding to terminal requests in Pay@Table mode. TcpClient is a mandatory parameter which is a reference to specify which device to send the response.

Events:

Method Name	Description
OnBroadcastMessage	Event listener for broadcast data sent from the terminal.
OnPayAtTableRequest	Event listener for terminal requests in Pay@Table mode

Event Parameters:

Method Name	Description
BroadcastMessageEventHandler(string code, string message)	Interface method which handles broadcast data from the broadcast message
PayAtTableRequestEventHandler(PATRequest request)	Interface method which handles parsing of terminal requests in Pay@Table mode

6.3 TerminalAuthBuilder

To be used if there are additional options required for the request message to be complete.

Property Name	Description
WithReferenceNumber(int value)	This command allows addition of Epos Number in request message.
WithCashback(decimal amount)	Indicates the Cashback value of currency selected in implied decimal format
WithCurrencyCode(string currencyCode)	Indicates the currency code for the transaction. This is defined by ISO 4217. Using values which aren't support Indicates the currency code for the transaction. This is defined by ISO 4217. Using values which aren't supported by the terminal and EFT application will lead to the transaction failing. Default value: GBP
WithPaymentMode(PaymentMode value)	Type of Payment Method, this is usually for Refund transaction. See section 7.8 for PaymentMode values
WithTableNumber(string value)	This method allows addition of a reference number for the transaction receipt. (Table Number in receipt) Refer to Ingenico Documentation section 5.2.3 for more details.
WithTaxFree(TaxFreeType value)	Used to specify the type of Tax Free transaction. See section 7.8 for TaxFreeType values
WithAmount(decimal amount)	Allows the addition of an amount to a confirmation response. (Pay@Table mode)
WithPayAtTableResponseType(PATResponseType type)	Used to specify a positive (CONF_OK) or negative (CONF_NOK) confirmation. (Pay@Table mode)
WithPaymentMode(PATPaymentMode mode)	Tells the terminal whether additional message is being requested (USE_ADDITIONAL) or not (NO_ADDITIONAL). (Pay@Table mode)
WithTicket(bool? isEnabled)	This method allows the user to get a combination of transaction response and receipt.
Execute()	Execute the function in accordance to previous method. This method should be the last to call for executing a request. The return type will be IngenicoTerminalResponse (none for Pay@Table mode)

6.4 TerminalManageBuilder

To be used if there are additional options required for the request message to be complete.

Property Name	Description
WithAuthCode(string authCode)	The authorization code received from the transaction that has been processed.
WithTransactionId(string value)	This method allows the addition of a transaction number for a reversal. Refer to Ingenico Documentation section 5.2.4 for more details.
WithTicket(bool? isEnabled)	This method allows the user to get a combination of transaction response and receipt.
Execute()	Execute the function in accordance to previous method. This method should be the last to call for executing a request. The return type will be IngenicoTerminalResponse

6.5 TerminalReportBuilder

Property Name	Description
GetLastReceipt(ReceiptType type)	Automatically executed upon method call in IDeviceInterface (section 7.2).
GetReport(ReportType type)	Automatically executed upon method call in IDeviceInterface (section 7.2)
Execute()	Execute the function in accordance to previous method. This method should be the last to call for executing a request. The return type will be IngenicoTerminalReportResponse

6.6 IngenicoTerminalResponse

Base response data

Method Name	Data Type	Description
ReferenceNumber	string	Terminal / EPOS reference number
Status	string	Status / Response Code of Transaction. Refer to section 7.8 for return values.
TransactionAmount	decimal	Amount of Transaction
CurrencyCode	string	The currency used for the transaction.

PaymentMode	string	The mode of Payment. Refer to section 7.8 for return values.
PrivateData	string	Returns various private data sent by the EPOS/ Terminal.
Receipt	string	Returns receipt of the last transaction in xml format.
IsReceiptSuccess	boolean	Returns true if Receipt is not empty and false if Error or Exception occur during AutoTicket
ReceiptErrorMessage	string	Contains error message if Exception occur in AutoTicket

REP Field data

Method Name	Data Type	Description
AuthorizationCode	string	Terminal / EPOS reference number
CashBackAmount	decimal	Cashback Amount from request message.
TipAmount	decimal	Amount of Gratuity
FinalAmount	decimal	Total amount including Cashback and Gratuity
BalanceAmount	decimal	The available amount of transaction.
DccCode	string	The currency of Dynamic currency conversion (DCC).
DccStatus	enum	The status of dynamic currency conversion (DCC).
DccAmount	decimal	The amount of dynamic currency conversion (DCC).
PaymentType	string	The method of Payment used. Refer to section 7.8 for return values.
TableId	string	The tableID provided by the ECR in the request.

6.7 IngenicoTerminalReportResponse

Method Name	Data Type	Description
ToString()	string	Returns string value of the raw response data / xml receipt from the terminal.

6.8 Enums

ConnectionModes

Values
SERIAL
TCP_IP_SERVER
PAY_AT_TABLE

Status / Response Codes

Values
SUCCESS (0)
REFERRAL (2)
CANCELLED_BY_USER (6)
FAILED (7)
RECEIVED (9)

PaymentMode

Values
APPLICATION (0)
MAIL ORDER (1)

PaymentMethod

Values
KEYED (1)
SWIPED (2)
CHIP (3)
CONTACTLESS (4)

TransactionSubTypes

Values
SPLIT_SALE_TXN (0x53)
DCC_TXN (0x44)
REFERRAL_RESULT (0x82)

DynamicCurrencyStatus

Values
CONVERSION_APPLIED (1)
REJECTED (0)

TaxFreeType

Values
CREDIT (4)
CASH (5)

ReportType

Values
EOD
BANKING
XBAL
ZBAL

ReceiptType

Values
TICKET
SPLITR
TAXFREE
FREE

PATRequestType

Values
TABLE_LOCK (1)
TABLE_UNLOCK (2)
RECEIPT_MESSAGE (3)
TABLE_LIST (4)
TRANSACTION_OUTCOME (5)
ADDITIONAL_MESSAGE (6)
TRANSFER_DATA (7)
SPLITSALE_REPORT (8)
TICKET (9)

EOD_REPORT (10)

PATResponseType

Values
CONF_OK
CONF_NOK

PATPaymentMode

Values
USE_ADDITIONAL
NO_ADDITIONAL

REFERENCES

O'Donnell, H. (2019) EPoS interface for TMS telium terminals (semi-integration). Kirkcaldy, UK.